本资源来自数缘社区

http://www.mathmagic.cn

欢迎来到数缘社区。本社区是一个高等数学及密码学的技术性论坛，由山东大学数学院研究生创办。在这里您可以尽情的遨游数学的海洋。作为站长，我诚挚的邀请您加入，希望大家能一起支持发展我们的论坛，充实每个版块。把您宝贵的资料与大家一起分享！

数学电子书库

每天都有来源于各类网站的与数学相关的新内容供大家浏览和下载，您既可以点击左键弹出网页在线阅读，又可以点右键选择下载。现在书库中藏书1000 余本。如果本站没有您急需的电子书，可以发帖说明，我们有专人负责为您寻找您需要的电子书。

密码学论文库

国内首创信息安全专业的密码学论文库，主要收集欧密会（Eurocrypt）、美密会（Crypto）、亚密会（Asiacrypt）等国内外知名论文。现在论文库中收藏论文4000余篇（包括论文库版块700余篇、论坛顶部菜单“密码学会议论文集”3000 余篇）。如果本站没有您急需的密码学论文，可以发帖说明，我们有专人负责为您寻找您需要的论文。

提示：本站已经收集到1981－2003 年欧密会、美密会全部论文以及1997 年－2007 年七大会议全部论文（欧密会、美密会、亚密会、PKC、FSE、RSA、CHES）。

数学综合讨论区

论坛管理团队及部分会员来源于山东大学数学院七大专业（基础数学、应用数学、运筹学、控制论、计算数学、统计学、信息安全），在数学方面均为思维活跃、成绩优秀的研究生，相信会给您的数学学习带来很大的帮助。

密码学与网络安全

山东大学数学院的信息安全专业师资雄厚，前景广阔，具有密码理论、密码技术与网络安全技术三个研究方向。有一大批博士、硕士及本科生活跃于本论坛。本版块适合从事密码学或网络安全方面学习研究的朋友访问。

网页公式编辑器

数缘社区公式编辑器采用Latex语言，适用于任何支持图片格式的论坛或网页。在本论坛编辑好公式后，您可以将自动生成的公式图片的链接直接复制到你要发的帖子里以图片的形式发表。

如果您觉得本站对您的学习和成长有所帮助，请把它添加到您的收藏夹。如果您对本论坛有任何的意见或者建议，请来论坛留下您宝贵的意见。

附录A：本站电子书库藏书目录

http://www.mathmagic.cn/bbs/dispbbs.asp?boardID=18&ID=2285

附录B：版权问题

Lecture Notes in Computer Science 2523

Edited by G. Goos, J. Hartmanis, and J. van Leeuwen

Springer
Berlin
Heidelberg
New York
Barcelona
Hong Kong
London
Milan
Paris
Tokyo

Burton S. Kaliski Jr. Çetin K. Koç
Christof Paar (Eds.)

Cryptographic Hardware and Embedded Systems – CHES 2002

4th International Workshop
Redwood Shores, CA, USA, August 13-15, 2002
Revised Papers

Springer

Volume Editors

Burton S. Kaliski Jr.
RSA Laboratories
174 Middlesex Turnpike, Bedford, MA 01730, USA
E-mail: bkaliski@rsasecurity.com

Çetin K. Koç
Oregon State University
Corvallis, Oregon 97330, USA
E-mail: koc@ece.orst.edu

Christof Paar
Ruhr-Universität Bochum
44780 Bochum, Germany E-mail: cpaar@crypto.rub.de

Cataloging-in-Publication Data applied for

A catalog record for this book is available from the Library of Congress
Bibliographic information published by Die Deutsche Bibliothek
Die Deutsche Bibliothek lists this publication in the Deutsche Nationalbibliografie; detailed bibliographic data is available in the Internet at <http://dnb.ddb.de>.

CR Subject Classification (1998): E.3, C.2, C.3, B.7.2, G.2.1, D.4.6, K.6.5, F.2.1, J.2

ISSN 0302-9743
ISBN 3-540-00409-2 Springer-Verlag Berlin Heidelberg New York

Springer-Verlag Berlin Heidelberg New York
a member of BertelsmannSpringer Science+Business Media GmbH

http://www.springer.de

Printed in Germany

Typesetting: Camera-ready by author, data conversion by PTP-Berlin, Stefan Sossna e. K.
Printed on acid-free paper SPIN 10873104 06/3142 5 4 3 2 1 0

Preface

These are the proceedings of CHES 2002, the Fourth Workshop on Cryptographic Hardware and Embedded Systems. After the first two CHES Workshops held in Massachusetts, and the third held in Europe, this is the first Workshop on the West Coast of the United States. There was a record number of submissions this year and in response the technical program was extended to 3 days.

As is evident by the papers in these proceedings, there have been again many excellent submissions. Selecting the papers for this year's CHES was not an easy task, and we regret that we could not accept many contributions due to the limited availability of time. There were 101 submissions this year, of which 39 were selected for presentation. We continue to observe a steady increase over previous years: 42 submissions at CHES '99, 51 at CHES 2000, and 66 at CHES 2001. We interpret this as a continuing need for a workshop series that combines theory and practice for integrating strong security features into modern communications and computer applications. In addition to the submitted contributions, Jean-Jacques Quisquater (UCL, Belgium), Sanjay Sarma (MIT, USA) and a panel of experts on hardware random number generation gave invited talks.

As in the previous years, the focus of the Workshop is on all aspects of cryptographic hardware and embedded system security. Of special interest were contributions that describe new methods for efficient hardware implementations and high-speed software for embedded systems, e.g., smart cards, microprocessors, DSPs, etc. CHES also continues to be an important forum for new theoretical and practical findings in the important and growing field of side-channel attacks.

We hope to continue to make the CHES Workshop series a forum for intellectual exchange in creating the secure, reliable, and robust security solutions of tomorrow. CHES Workshops will continue to deal with hardware and software implementations of security functions and systems, including security for embedded wireless ad hoc networks.

We thank everyone whose involvement made the CHES Workshop such a successful event. In particular we would like to thank André Weimerskirch (Ruhr-University, Bochum) for his help again with the website and Gökay Saldamlı and Colin van Dyke (Oregon State University) for their help on registration and local organization.

August 2002

Burton S. Kaliski Jr.
Çetin K. Koç
Christof Paar

Acknowledgements

The organizers express their thanks to the program committee, the external referees for their help in getting the best quality papers selected, and also the companies which provided support to the workshop.

The program committee members for CHES 2002:

- Beni Arazi, arazi@ee.bgu.ac.il
  Ben Gurion University, Israel
- Jean-Sébastien Coron, coron@clipper.ens.fr
  Gemplus Card International, France
- Kris Gaj, kgaj@gmu.edu
  George Mason University, USA
- Craig Gentry, cgentry@docomolabs-usa.com
  DoCoMo Communications Laboratories, USA
- Jim Goodman, jimg@engim.com
  Engim Canada, Canada
- M. Anwar Hasan, ahasan@ece.uwaterloo.ca
  University of Waterloo, Canada
- David Jablon, dpj@theworld.com
  Phoenix Technologies, USA
- Peter Kornerup, kornerup@imada.sdu.dk
  University of Southern Denmark, Odense, Denmark
- Pil Joong Lee, pjl@postech.ac.kr
  Pohang Univ. of Sci. & Tech., Korea
- Preda Mihailescu, preda@uni-paderborn.de
  University of Paderborn, Germany
- David Naccache, david.naccache@gemplus.com
  Gemplus Card International, France
- Bart Preneel, Bart.Preneel@esat.kuleuven.ac.be
  Katholieke Universiteit Leuven, Belgium
- Erkay Savaş, savas@ece.orst.edu
  Oregon State University, USA
- Joseph Silverman, jhs@math.brown.edu
  Brown University and NTRU Cryptosystems, Inc., USA
- Jacques Stern, Jacques.Stern@ens.fr
  Ecole Normale Superieure, France
- Berk Sunar, sunar@ece.wpi.edu
  Worcester Polytechnic Institute, USA
- Colin Walter, colin@comodo.net
  Comodo Research Labs, UK

The external referees:

- Murat Aydos (Oregon State University, USA)
- Vittorio Bagini (Gemplus, Italy)

- Lejla Batina (KU Leuven, ESAT/COSIC, Belgium / SafeNet, The Netherlands)
- Sıddıka Berna Örs (KU Leuven, ESAT/COSIC, Belgium)
- Eric Brier (Gemplus, France)
- Marco Bucci (Gemplus, France)
- Jaewook Chung (University of Waterloo, Canada)
- Christophe Clavier (Gemplus International, France)
- Nora Dabbous (Gemplus, France)
- Jean-François Dhem (Gemplus, France)
- Itai Dror (M-Systems Flash Disk Pioneers, Israel)
- Nevine Ebeid (University of Waterloo, Canada)
- Levent Ertaul (Oregon State University, USA)
- Lijun Gao (Bermai Inc., USA)
- Johann Großschädl (IAIK, Graz University of Technology, Austria)
- Frank K. Gürkaynak (Swiss Federal Institute of Technology, Zurich, Switzerland)
- Pascal Guterman (Gemplus, France)
- Helena Handschuh (Gemplus, France)
- Kouichi Itoh (Fujitsu Laboratories Ltd., Japan)
- Marc Joye (Gemplus, France)
- Vangelis Karatsiolis (Technical University of Darmstadt / Fraunhofer Institute of Secure Telecooperation, Germany)
- Chong Hee Kim (Pohang Univ. of Sci. & Tech., Korea)
- Volker Krummel (University of Paderborn, Germany)
- Manuel Leone (Telecom Italia Lab, Italy)
- Albert Levi (Sabancı University, Turkey)
- Pierre-Yvan Liardet (STMicroelectronics, France)
- Renato Menicocci (Gemplus, France)
- Bodo Möller (Technical University of Darmstadt, Germany)
- Olaf Mueller (University of Paderborn, Germany)
- Gerardo Orlando (General Dynamics / WPI, USA)
- Elisabeth Oswald (IAIK TU-Graz, Austria / KU Leuven, Belgium)
- Christof Paar (Ruhr-University, Bochum, Germany)
- Pascal Paillier (Gemplus, France)
- Dong Jin Park (Pohang Univ. of Sci. & Tech., Korea)
- Stephanie Porte (Gemplus, France)
- Vincent Rijmen (Cryptomathic, Belgium)
- Francisco Rodriguez-Henriquez (CINVESTAV-IPN, Mexico)
- Gökay Saldamlı (Oregon State University, USA)
- Tom Schmidt (Oregon State University, USA)
- Jasper Scholten (KU Leuven, ESAT/COSIC, Belgium)
- Stefaan Seys (KU Leuven, ESAT/COSIC, Belgium)
- Jamshid Shokrollahi (University of Paderborn, Germany)
- Sang Gyoo Sim (Pohang Univ. of Sci. & Tech., Korea)
- Tsuyoshi Takagi (Technical University of Darmstadt, Germany)
- Masahiko Takenaka (Fujitsu Laboratories Ltd., Japan)
- Alexandre F. Tenca (Oregon State University, USA)
- Georgi Todorov (Oregon State University, USA)

- Elena Trichina (Gemplus, Italy)
- Christophe Tymen (Gemplus/ENS, France)
- Johannes Wolkerstorfer (Graz University of Technology, Austria)
- Thomas Wollinger (Ruhr-University, Bochum, Germany)
- Yiqun Lisa Yin (NTT MCL, USA)

The companies which provided support to CHES 2002:

- Intel - `http://www.intel.com`
- NTRU Cryptosystems, Inc. - `http://www.ntru.com`
- RSA Security, Inc. - `http://www.rsasecurity.com`

CHES Workshop Proceedings

- Ç.K. Koç and C. Paar (Editors). *Cryptographic Hardware and Embedded Systems*, Lecture Notes in Computer Science No. 1717, Springer-Verlag, Berlin, Heidelberg, New York, 1999.
- Ç.K. Koç and C. Paar (Editors). *Cryptographic Hardware and Embedded Systems - CHES 2000*, Lecture Notes in Computer Science No. 1965, Springer-Verlag, Berlin, Heidelberg, New York, 2000.
- Ç.K. Koç, D. Naccache, and C. Paar (Editors). *Cryptographic Hardware and Embedded Systems - CHES 2001*, Lecture Notes in Computer Science No. 2162, Springer-Verlag, Berlin, Heidelberg, New York, 2001.
- B. Kaliski Jr., Ç.K. Koç, and C. Paar (Editors). *Cryptographic Hardware and Embedded Systems - CHES 2002*, Lecture Notes in Computer Science No. 2523, Springer-Verlag, Berlin, Heidelberg, New York, 2002. (These proceedings).

Table of Contents

AES and AES Candidates

Tamper Resistance

RSA Implementation

Finite Field and Modular Arithmetic II

Elliptic Curve Cryptography II

Random Number Generation

Invited Talk

New Primitives

Finite Field and Modular Arithmetic II

Elliptic Curve Cryptography III

Hardware for Cryptanalysis

Template Attacks

Suresh Chari, Josyula R. Rao, and Pankaj Rohatgi

No Institute Given

Author Index

CHES: Past, Present, and Future

Jean-Jacques Quisquater

UCL Crypto Group
Universite Catholique de Louvain
Louvain-La-Neuve, Belgium
quisquater@dice.ucl.ac.be

CHES is (coming to be) a very interesting conference thanks to the excellent submitted papers, the new results about embedded systems, the coprocessors, and the new use of FPGAs.

But my talk will be about the nice locations for the CHES conference:

- First, it was Worcester and I'll speak about Vernam.
- Next, it was Paris and I'll speak about the Rose-Sainte-Croix company and, more importantly, the principle of Kerckhoffs (with a curious story about Napoleon).
- And now it is Redwood City: here is the whole of the public-key crypto is near or just there (Diffie, Hellman, Merkle, El Gamal, RSA Data Security, the RSA conferences at this hotel, ...). And a surprise: a big and secure embedded system: from Lockheed and it is the SeaShadow; it is time to think about James Bond, ..., and, finally, smart cards).
- the travel in the time and the space is finished and I'll propose some ideas for next location.

B.S. Kaliski Jr. et al. (Eds.): CHES 2002, LNCS 2523, p. 1, 2003.

Optical Fault Induction Attacks

Sergei P. Skorobogatov and Ross J. Anderson

University of Cambridge, Computer Laboratory,
15 JJ Thomson Avenue, Cambridge CB3 0FD, United Kingdom
{sps32, rja14}@cl.cam.ac.uk

Abstract. We describe a new class of attacks on secure microcontrollers and smartcards. Illumination of a target transistor causes it to conduct, thereby inducing a transient fault. Such attacks are practical; they do not even require expensive laser equipment. We have carried them out using a flashgun bought second-hand from a camera store for $30 and with an $8 laser pointer. As an illustration of the power of this attack, we developed techniques to set or reset any individual bit of SRAM in a microcontroller. Unless suitable countermeasures are taken, optical probing may also be used to induce errors in cryptographic computations or protocols, and to disrupt the processor's control flow. It thus provides a powerful extension of existing glitching and fault analysis techniques. This vulnerability may pose a big problem for the industry, similar to those resulting from probing attacks in the mid-1990s and power analysis attacks in the late 1990s.
We have therefore developed a technology to block these attacks. We use self-timed dual-rail circuit design techniques whereby a logical 1 or 0 is not encoded by a high or low voltage on a single line, but by (HL) or (LH) on a pair of lines. The combination (HH) signals an alarm, which will typically reset the processor. Circuits can be designed so that single-transistor failures do not lead to security failure. This technology may also make power analysis attacks very much harder too.

1 Introduction

Secure microcontrollers and smartcards are designed to protect both the confidentiality and the integrity of sensitive information. It is not sufficient to prevent an attacker from finding out the value of a stored cryptographic key; she must also be unable to set part of the key to a known value, or to induce errors in the computation that enable sensitive information to be deduced. These errors may be data errors, such as an incorrect digital signature that leaks the value of the signing key [3], or errors in the code, such as a missed conditional jump that reduces the number of rounds in a block cipher [1]. Until now, the most widely known technique for inducing such errors was glitching – the introduction of voltage transients into the power or clock line of the target chip. Many chips are now designed to resist glitch attacks.

A review of the tamper-resistance of smartcard and secure microcontroller chips may be found in [2]. Attacks tend to be either invasive, using chip testing equipment such as probing stations and focused ion beam workstations to

B.S. Kaliski Jr. et al. (Eds.): CHES 2002, LNCS 2523, pp. 2–12, 2003.

extract data from the chip directly, or else non-invasive processes involving the exploitation of unintentional electromagnetic emissions, protocol design flaws, and other vulnerabilities that manifest themselves externally. Either type of attack may be passive or active. The standard passive invasive attack involves using microprobes to monitor a smartcard's bus while a program is executing; in an active attack, signals may be also injected, the classic example being the use of a grounded microprobe needle on the clock line to the instruction latch to disable jump instructions. A passive non-invasive attack is analyzing the electromagnetic field in the neighborhood of the device under test [10], while glitching is the classic example of an active attack.

Until now, invasive attacks involved a relatively high capital investment for lab equipment plus a moderate investment of effort for each individual chip attacked. Non-invasive attacks such as power analysis require only a moderate capital investment, plus a moderate investment of effort in designing an attack on a particular type of device; thereafter the cost per device attacked is low. Non-invasive attacks are thus particularly attractive where they exist.

Unfortunately for the attacker, many chipmakers have now implemented defenses against the most obvious non-invasive attacks. These defenses include random clock jitter to make power analysis harder, and circuits that react to glitches by resetting the processor. Meanwhile invasive attacks are becoming constantly more demanding and expensive, as feature sizes shrink and device complexity increases, We therefore set out to find new, more powerful, ways of attacking chips.

We describe our new class of attacks as 'semi-invasive'. By this, we mean that, like invasive attacks, they require depackaging the chip to get access to the chip surface. But the passivation layer of the chip remains intact – semi-invasive methods do not require electrical contact to the metal surface so there is no mechanical damage to the silicon.

Semi-invasive attacks are not entirely new. The electromagnetic analysis of [10] is best performed on a naked chip, and the old EPROM-hacking trick of exposing the write protect bit of a microcontroller to UV light usually entails depackaging it. Semi-invasive attacks could in theory be performed using such tools as UV light, X-rays, lasers, electromagnetic fields and local heating. They could be used individually or in conjunction with each other. However, this field has hardly been explored.

We will now show that extremely powerful attacks can be carried out quickly using very cheap and simple equipment.

2 Background

Once the semiconductor transistor had been invented, it was found to be more sensitive to ionizing radiation – whether caused by nuclear explosions, radioactive isotopes, X-rays or cosmic rays – than the thermionic valves (vacuum tubes) used previously. In the middle sixties, during experiments with pulsed lasers, it was

found that intensive light causes some similar phenomena. Lasers started to be used to simulate the effects of ionizing radiation on semiconductors [4].

Since then the technology has been improved dramatically. Expensive inert-gas-based lasers and solid-state lasers have been replaced with low-cost semiconductor lasers. As a result, the technology has moved from the laboratory all the way down to consumer electronics.

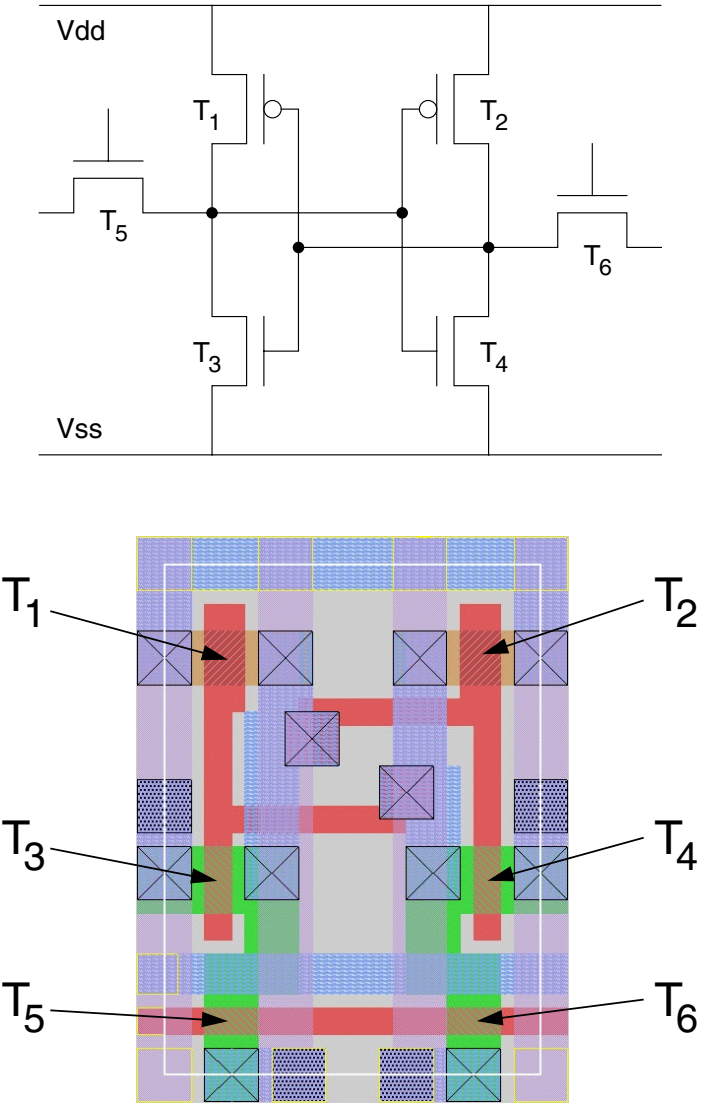


Fig. 1. Circuit structure and layout of a six-transistor SRAM cell

Laser radiation can ionize an IC's semiconductor regions if its photon energy exceeds the semiconductor band gap. Laser radiation with 1.06 μm wavelength (1.17 eV photon energy) used in [5] has a penetration depth of about 700 μm and provides good spatial ionization uniformity for silicon devices. However, its focusing is restricted by dispersion to several micrometers, and this is not precise enough for modern semiconductor devices. However, when moving from infrared to visible light, photon absorption dramatically increases [7], and it has become possible to use red and green lasers as the transistors in modern chips became thinner. Smaller devices also mean that less energy is required to achieve the same level of ionization.

In the case of CMOS devices, there is a danger of latching up the circuit, causing a short circuit that can result in permanent damage. So the use of radiation with CMOS structures must be done with appropriate precautions.

Although there are many publications about using pulsed lasers to simulate ionizing radiation, we could find no published information about using them to control or change the behavior of integrated circuits. So we decided to apply an intense light source to a semiconductor chip, and particularly to CMOS logic, to see whether it would be possible to change the state of a memory cell and how easy, or difficult, it might be.

Our first experiments targeted SRAM. The structure of a standard six-transistor SRAM cell is shown in Fig. 1 [8].

Two pairs of p- and n-channel transistors create a flip-flop, while two other n-channel transistors are used to read its state and write new values into it. The layout of the cell is shown on the right of Fig. 1 [9]. The transistors T_1 and T_3 create the CMOS inverter; together with the other similar pair, they create the flip-flop which is controlled by the transistors T_5 and T_6.

If the transistor T_3 could be opened for a very short time by an external stimulus, then it could cause the flip-flop to change state. By exposing the transistor T_4, the state of the cell would be changed to the opposite. The main difficulties we might anticipate are focusing the ionizing radiation down to several μm^2 and choosing the proper intensity.

3 Experimental Method

For our experiments we chose a common microcontroller (Microchip PIC16F84), which has 68 bytes of SRAM memory on chip (Fig. 2). A standard depackaging procedure was applied to the chip and the result of this operation is shown as well in Fig. 2.

The SRAM memory array is located in the centre of the bottom section of the chip. This area is shown with 80× magnification on Fig. 4.

Because we had a very limited equipment budget, and the laser we had appeared unsuitable, we decided to use a cheap photoflash lamp (a Vivitar 550FD, bought secondhand from a camera shop for $30). Although the luminosity of a flashlamp is much less than that of a pulsed laser, with appropriate magnification the necessary level of ionization might be achieved. We used duct tape to fix

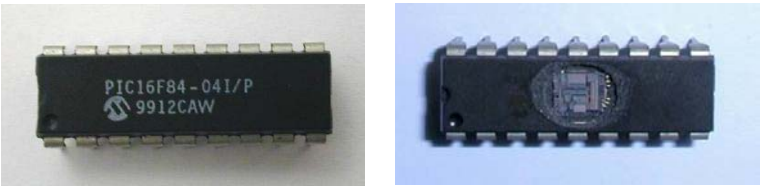


Fig. 2. Microcontroller PIC16F84 original and depackaged

the photoflash lamp on the camera port of a Wentworth Labs MP-901 manual probing station (Fig. 3). Magnification was set to the maximum – 1500×.

The microcontroller was programmed, such that its memory could be uploaded and downloaded via a serial interface connection. By filling the whole memory with constant values, exposing it to the flash light, and downloading the result, we could observe which cells changed their state. We used the TTL-level control input of the flash to remote control it from a connected PC and changing the capacitor recharge time allowed us to control the energy output. The output power of the lamp was set to the maximum possible in this experiment.

By shielding the light from the flash with an aperture made from aluminum foil, we succeeded in changing the state of only one single SRAM cell. The final state of the cell depended on the area exposed to the flash. This confirmed our intuition that it would be possible to change the contents of SRAM using a low cost semi-invasive attack.

4 Results

We found we could change any individual bit of an SRAM array. The array, under maximum magnification, is shown in Fig. 5. Focusing the light spot from the lamp on the area shown by the white circle caused the cell to change its state from 1 to 0, with no change if the state was already 0. By focusing the spot on the area shown by black circle, the cell changed its state from 0 to 1 or remained in state 1.

It can be seen from Fig. 4 that the SRAM array is divided into eight equal blocks. By exposing cells in different blocks, we found that each block corresponds to one bit plane of information. The result of this operation is shown in Fig. 6.

We built a map of the addresses corresponding to the physical location of each memory cell by exposing each cell in turn to the photoflash light. The result is presented in Fig. 7, with the left edge corresponding to the bottom side of the block. It can be seen that the addresses are not sequential, but divided into three groups.

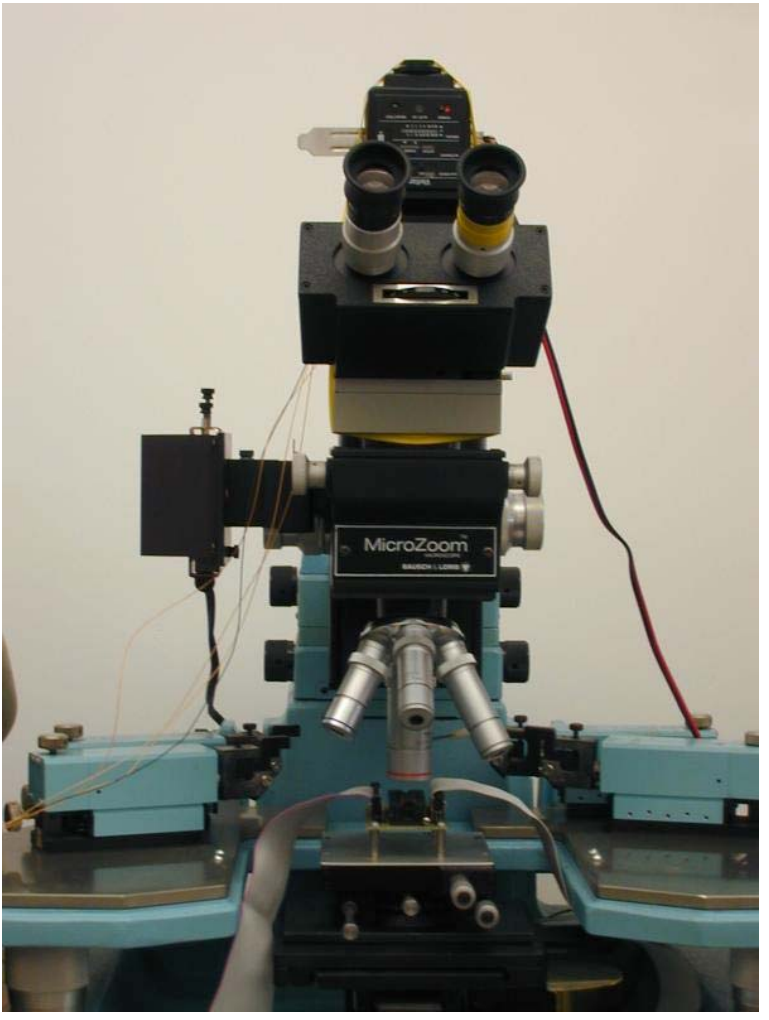


Fig. 3. Wentworth Labs MP-901 manual prober with Vivitar 550FD photoflash lamp mounted on top

This shows how simple semi-invasive attack methods can be used for reverse engineering a memory address map. The only limitation is that the flash does not produce even and monochromatic light, so it is very difficult to control the area where the spot of the light will be applied. This problem can be solved by replacing the flash with a suitable laser.

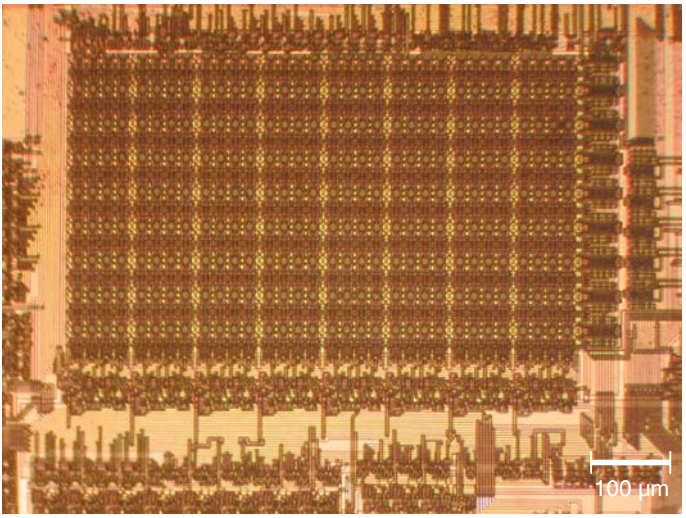


Fig. 4. SRAM memory array with magnification

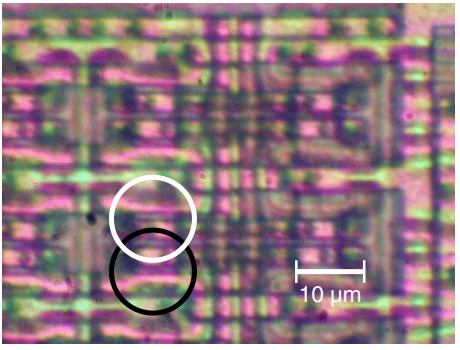


Fig. 5. SRAM memory array with maximum magnification

5 Implications and Further Work

This work shows that optical probing attacks are possible using low-cost equipment. We have repeated the experiments using a laser pointer (Fig. 8), which we bought on a local market for $8, and a motorized stage. The same results were achieved, but there were several practical differences. On the one hand that we could probe the chip surface automatically, and at a rate which we estimate could be driven as high as 100 flashes per second. On the other hand we had to be more careful with alignment because of the narrower aperture and lower power. The pointer was designed as a Class II laser device (< 1 mW), but we operated it with a supply current that should result in up to 10 mW light out-

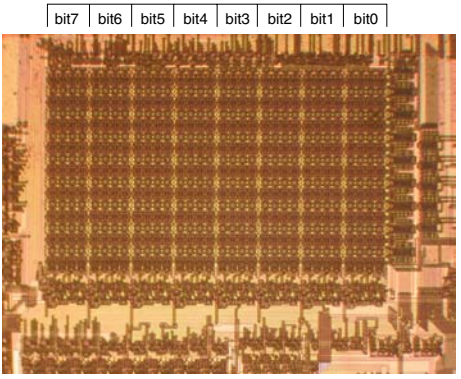


Fig. 6. Allocation of data bits in the SRAM memory array

30h	34h	38h	3Ch	40h	44h	48h	4Ch	10h	14h	18h	1Ch	20h	24h	28h	2Ch	0Ch
31h	35h	39h	3Dh	41h	45h	49h	4Dh	11h	15h	19h	1Dh	21h	25h	29h	2Dh	0Dh
32h	36h	3Ah	3Eh	42h	46h	4Ah	4Eh	12h	16h	1Ah	1Eh	22h	26h	2Ah	2Eh	0Eh
33h	37h	3Bh	3Fh	43h	47h	4Bh	4Fh	13h	17h	1Bh	1Fh	23h	27h	2Bh	2Fh	0Fh

Fig. 7. Allocation of addresses in each bit block of SRAM memory array

put. We can focus it to around 1 μm on the chip surface and its wavelength is around 650 nm.

We used our automated probing equipment to implement attacks on a number of semiconductor devices. The best designed of the modern secure microcontrollers are not vulnerable to attacks using single laser flashes, as their protection state depends on a number of bits of physical storage. However, a number of designs can be unprotected by changing the state of the flip-flop that latches the read-protect state. We strongly recommend that designers of ICs should study their designs carefully to ensure that there are no single-transistor failures that can subvert the chip's security policy.

Attack experiments have been conducted on smartcards too. It may be helpful at this point to recall some of the earlier literature on fault analysis. In [3], Boneh, Demillo and Lipton pointed out that the faulty computation of an RSA digital signature leaks the signing key. For example, when doing an RSA signature the secret computation $S = h(m)^d \pmod{pq}$ is carried out mod p, then mod q, and the results are then combined, as this is significantly faster. However, if the card returns a defective signature S_p which is correct modulo p but incorrect modulo q, then we will have $p = \gcd(pq, S_p^e - h(m))$.

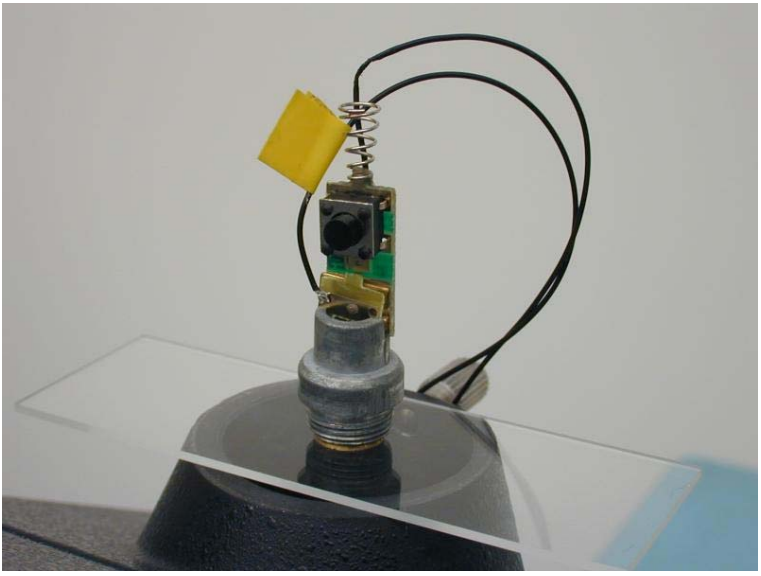


Fig. 8. Disassembled laser pointer mounted to the microscope camera port

In [1], Anderson and Kuhn pointed out that interference with jump instructions is an even more powerful and general attack: an attacker who can cause conditional branches in the smartcard code to be taken wrongly may, for example, reduce the number of rounds in a block cipher to one or two, making key recovery straightforward. The first of these two types of attack has been implemented successfully using our technique, but an NDA prevents us from giving further information.

Further scientific work in our plan includes a fuller investigation of the potential for attacks by an opponent with a moderately resourced laboratory, by which we mean a modern probing station with a multiple wavelength laser. We are commissioning such equipment and plan to use it to explore the potential for fault induction through the rear of the chip using infrared light. We have also obtained access to a suitable X-ray source and will investigate whether it can be used to induce useful faults. The significance of this is that X-rays can penetrate top-layer metal, as well as most types of protective packaging likely to be encountered in practice.

6 Countermeasures

The optical probing attack described above is a new and devastating technique for attacking smartcards and other security processors. We anticipate that, like the power analysis attacks reported by Kocher in [6], it could have a significant

commercial effect on the industry, in that it will force a thorough reappraisal of security claims and the introduction of new defensive technology.

Following Kocher, we decided to delay the announcement of our attack until proper defenses were available. Existing high-end chip-defense techniques, such as top-layer metal shielding and bus encryption, may make an attack using these techniques more complicated, but are not enough. A sophisticated attacker can defeat metal shielding by using infrared light or X-rays, while bus encryption can be defeated by attacking registers directly.

The defensive technology that we have developed uses self-timed dual-rail logic. Conventional digital logic uses a clock to synchronize activities; but the cost of clocking rises as devices become more complex, and this has led to a surge of interest in design techniques for self-timed, or asynchronous, circuits – circuits that do not use clocks. Such circuits need some mechanism whereby functional components in a circuit can signal that they are ready to receive data, or are done. One way of doing this is to introduce redundancy into the data path.

In dual-rail logic, a 0 or 1 is signaled not by a low or high voltage on a single wire, but by a combination of signals on a pair of wires. For example, 0 may be 'LH' and 1 may be 'HL'. When used in self-timed circuits, 'LL' signals quiescence. The principal drawback of this simple arrangement is fragility: bugs tend to cause the emergence of the unwanted 'HH' state, which propagates rapidly throughout the circuit and locks it.

Our innovation was to turn this fragility to advantage, by making 'HH' into an error signal. This signal can be raised deliberately by tamper sensors, causing the device to lock [12]. Of more interest here is the fact that matters can be so arranged that single device failures cause are unlikely to cause the output of sensitive information [11]. We believe that such robustness will be a requirement for many high-security devices in future.

The engineering details are non-trivial. For example, an obvious concern is that almost any undetected malfunction could be exploited by the attack of Boneh et al. on RSA signatures. Colleagues have therefore developed a modular multiplication unit using our technology. Similarly, although bus encryption can remove the need to protect on-chip memory arrays, there remains the risk of attacks on the program counter and other registers. Other colleagues have therefore developed registers, and a memory management unit, that use our technology [11].

7 Conclusion

Standard CMOS circuitry is extremely vulnerable to attack using optical probing. By exposing a transistor to a laser beam, or even the focused light from a flash, it can be made to conduct. This gives rise to many effects that can be used by an attacker. We have described here how the illumination of a certain area of an SRAM cell can be used to set it to either 0 or 1. Other memory technologies, such as EPROM, EEPROM and Flash, can also be manipulated in various ways.

However, this is only the beginning. Given only moderately expensive equipment, an attacker may be able to induce a fault in a CMOS integrated circuit, in any targeted transistor, and at precisely the clock cycle of her choice. This is quite devastating. Hardware countermeasures will be necessary.

References

1. R.J. Anderson, M.G. Kuhn, "Low Cost Attacks on Tamper Resistant Devices", in M. Lomas et al. (ed.), Security Protocols, 5th International Workshop, Paris, France, April 7–9, 1997
2. R.J. Anderson, "Security Engineering – A Guide to Building Dependable Distributed Systems", Wiley 2001
3. D. Boneh, R.A. DeMillo, R.J. Lipton, "On the Importance of Checking Cryptographic Protocols for Faults, Advances in Cryptology – Eurocrypt 97", Springer LNCS vol 1233 pp 37–51
4. D.H. Habing, "Use of Laser to Simulate Radiation-induced Transients In Semiconductors and Circuits", IEEE Trans. Nuc. Sci., Vol NS-12, No 6, pp 91–100, Dec. 1965
5. A.H. Johnston, "Charge Generation and Collection in p-n Junctions Excited with Pulsed Infrared Lasers", IEEE Trans. Nuc. Sci., Vol NS-40, No 6, pp 1694–1702, 1993
6. P. Kocher, "Differential Power Analysis", Advances in Cryptology – Crypto 99, Springer LNCS vol 1666 pp 388–397
7. "Handbook of Optical Constants of Solids", edited by Edward D. Palik, Orlando: Academic Press, 1985, pp 547–569
8. J.M. Rabaey, "Digital Integrated Circuits: A Design Perspective", Prentice-Hall, 1995
9. K. Yun, "Memory", UC San Diego, Adapted from EE271 notes, Stanford University, `http://paradise.ucsd.edu/class/ece165/notes/lecC.pdf`
10. J.J. Quisquater, D. Samyde, "ElectroMagnetic Analysis (EMA): Measures and Countermeasures for Smart Cards", International Conference on Research in Smart Cards, E-smart 2001, Cannes, France, pp 200–210, Sept. 2001
11. S.W. Moore, R.J. Anderson, P. Cunningham, R. Mullins, G. Taylor, "Improving Smartcard Security using Self-Timed Circuits", Asynch 2002, proceedings published by IEEE Computer Society Press
12. S.W. Moore, R.J. Anderson, M.G. Kuhn, "Improving Smartcard Security using Self-Timed Circuit Technology", Fourth AciD-WG Workshop, Grenoble, ISBN 2-913329-44-6, 2000

Template Attacks

Suresh Chari, Josyula R. Rao, and Pankaj Rohatgi

IBM Watson Research Center,
P.O. Box 704
Yorktown Heights, NY 10598
{schari,jrrao,rohatgi}@us.ibm.com

Abstract. We present *template attacks*, the strongest form of side channel attack possible in an information theoretic sense. These attacks can break implementations and countermeasures whose security is dependent on the assumption that an adversary cannot obtain more than one or a limited number of side channel samples. They require that an adversary has access to an identical experimental device that he can program to his choosing. The success of these attacks in such constraining situations is due manner in which noise within each sample is handled. In contrast to previous approaches which viewed noise as a hindrance that had to be reduced or eliminated, our approach focuses on precisely modeling noise, and using this to fully extract information present in a single sample. We describe in detail how an implementation of RC4, not amenable to techniques such as SPA and DPA, can easily be broken using template attacks with a single sample. Other applications include attacks on certain DES implementations which use DPA–resistant hardware and certain SSL accelerators which can be attacked by monitoring electromagnetic emanations from an RSA operation even from distances of fifteen feet.

1 Introduction

In the past few years, side channel attacks [13,12] have shown to be extremely effective as a practical means for attacking implementations of cryptographic algorithms. Adversaries can obtain sensitive information from side channels such as timing of operations[13], power consumption [12], electromagnetic emanations [19,9,20] etc. In constrained devices such as chip–cards, straightforward implementations of cryptographic algorithms can be broken with minimal work.

Since Paul Kocher's original paper [12], a number of devastating attacks, such as Simple Power Analysis (SPA) and Differential Power Analysis (DPA) have been reported on a wide variety of cryptographic implementations [15,18, 6,11,17,8,3,4,10,16,7,5,21]. In SPA, keying information is easily extracted from a single sample due to leakage from the execution of key dependent code and/or the use of instructions which leak substantial information in the side channel over the noise. When the leakage relative to noise is much less, statistical techniques such as DPA are applicable. DPA relies on a statistical analysis of a *large number of samples* where the same keying material is used to operate on different data. A large number of samples is used to reduce noise by averaging.

B.S. Kaliski Jr. et al. (Eds.): CHES 2002, LNCS 2523, pp. 13–28, 2003.

In this paper, we show that these attacks are not optimal as they do not take advantage of all information available in *each* side channel sample. Consequently, some implementations believed to be immune to side channel attacks simply because the adversary is limited to one or at most a few compromising samples, can in reality be broken by harnessing all available information.

Consider an implementation of the RC4 stream cipher. While there are recent reports of cryptanalytic results highlighting minor statistical weakness, there are no major statistical biases to be easily exploited by side–channel attacks. To our knowledge, no successful side channel attack on a reasonable RC4 implementation has been reported[1]. Initializing the 256-byte internal state of RC4 using the secret key is simple enough to be implemented in a key independent manner. While implementations of this will certainly leak some information about the key, the individual steps do not leak *enough* information. Thus, simple side channel attacks such as SPA are not possible. After initialization, the rapidly evolving internal state of the stream cipher, independent of adversarial action (due to the absence of any external inputs), offers innate defense against statistical attacks such as DPA. One can at most hope to obtain a *single* sample of the side channel leakage during the key initialization phase of RC4. Figure 1 is based on side channel samples from the RC4 key initialization phase: the upper trace is the difference between two single power samples when the keys are the same, the lower trace when they are different. Contrary to expectation, the first case shows larger differences. This ambiguity exists even when one looks at differences of averages of upto five invocations (as shown in Figure 3 in Section 3). Clear and consistent differences emerge only when one considers averages of several tens of samples. Therefore, it would appear that such a carefully coded RC4 implementation cannot be attacked using only *one* available sample.

Consider a smart card which has a fast[2] hardware DES engine. These have become very popular especially because they have been evaluated and shown to be highly resistant to side channel attacks. In conjunction with protocols limiting adversaries to only few DES invocations with the card's secret key, it would appear that this is immune to side channel attacks. The third case we consider is where an adversary is able to position sensitive (and bulky) electromagnetic (EM) eavesdropping equipment in the proximity of a server with a commercial RSA accelerator inside. In this environment, due to a risk of detection, it is likely that only a few samples can be obtained.

In all these cases, the adversary has to work with far fewer signals than is believed is necessary for side channel attacks based on known techniques. The *template attacks* introduced in this paper can break all of these implementations. In fact, as we will see, the template attack extracts all possible information available in each sample and is hence the strongest form of side channel attack possible in an information theoretic sense given the few samples that are available.

A key requirement for the template attack is that the adversary has an identical experimental device which can be programmed. While such an assumption

[1] IEEE 802.11 uses RC4 in a mode which makes implementations vulnerable to DPA.
[2] Typically, such engines perform the entire DES in a few cycles.

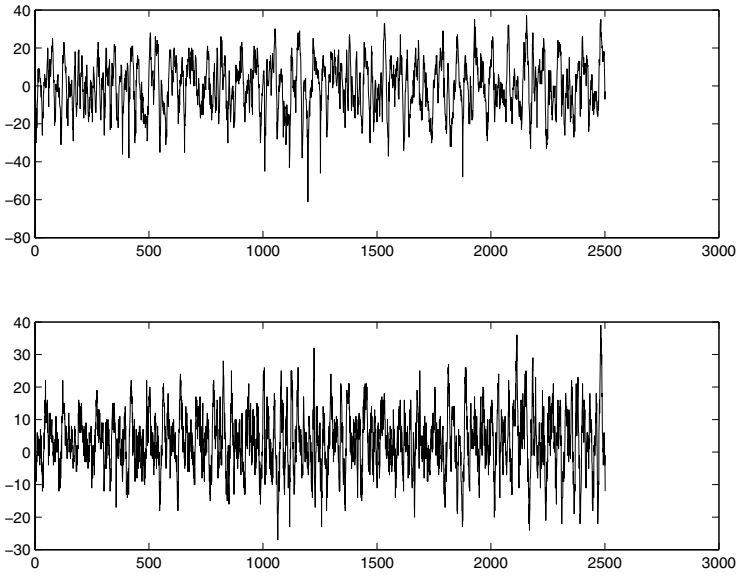


Fig. 1. Differences of side channel samples: upper figure is for the same key while the lower is for two different keys

is limiting, it holds in many cases and has been used in other side channel attacks [8,18] before. The template attack derives its power from using the experimental device to derive a precise multivariate characterization of the noise. In sharp contrast, prior approaches focussed on eliminating noise by averaging. We argue that, especially for cryptographic algorithms implemented in CMOS devices, the use of such a characterization is an extremely powerful tool to classify even a single sample. The situation is analogous to the manner in which very weak signals are extracted in signal communications. Even though the received signal strength is very weak, it can be extracted by a receiver who has a very good characterization of the signal and the ambient noise.

We refer to the precise, detailed models of the signal and noise as the *template* of the computation. The concept of a template is based on Signal Detection and Estimation Theory and in particular, the use of information theoretic techniques such as likelihood ratios for hypothesis testing. Although other techniques such as DPA, can also be viewed as coarse approximations of likelihood ratios, the use of multivariate noise statistics is key to extracting the maximum information from a single sample. Empirically, we have observed that in several situations, univariate statistics are not sufficient and yield poor results.

The template attack works by a process of iterative classification. At each step, we unroll one more segment of the sample which uses more bits of the unknown key. Correspondingly, larger templates are used to prune the space of possible hypotheses for the values of key bits, while controlling error probability.

While minor differences in the keys can possibly confuse a classifier, this attack is effective on cryptographic algorithms because the natural diffusion properties of cryptographic algorithms actually aid in eliminating precisely such mistakes.

The paper is organized as follow: Section 2 introduces the theory behind template attacks. Section 3 describes the application of template attacks to extract keys from an implementation of RC4 using a single sample. Section 4 describes two other cases where template attacks are feasible. In Section 5, we describe the implications of template attacks and discuss potential countermeasures.

2 Theory

In this section, using Signal Detection and Estimation Theory we derive the template attack and describe some heuristics to make the attacks practical. Essentially, we have a device performing one of K possible operation sequences, $\{O_1, \ldots, O_K\}$: for example, these could be to execute the same code for different values of key bits. An adversary who can sample the side channel during this operation wishes to identify which of the operations is being executed or to significantly reduce the set of possible hypotheses for the operation.

In signal processing, it is customary to model the observed sample as a combination of an intrinsic signal generated by the operation and noise which is either intrinsically generated or ambient. Whereas the signal component is the same for repeated invocations of the operation, the noise is best modeled as a random sample drawn from a noise probability distribution that depends on the operating and other ambient conditions. It is well known[22] that the optimal approach for the adversary, who is trying to find the right hypothesis given a single sample S, is to use the maximum likelihood approach: The best guess is to pick the operation such that, the probability of the observed noise in S is maximized. Computing this probability requires the adversary to model both the intrinsic signal and the noise probability distribution for each operation accurately.

Template attacks meld this basic principle with details of the cryptographic operation being attacked. The adversary uses an experimental device, identical to the device under test, to identify a small section of the sample S depending only on a few unknown key bits. With experimentation, he builds templates corresponding to each possible value of the unknown key bits. The template consist of the mean signal and noise probability distributions. He then uses these templates to classify that portion of S and limit the choices for the key bits to a small set. This is then repeated with a longer prefix of S involving more key bits. We will retain only a small number of possibilities for the portion of the key considered thus far. Thus template attacks essentially use an extend-and-prune strategy directed by the single sample S to be attacked: we use increasingly longer prefixes of S and the corresponding templates to prune the space of possible key prefixes. The success critically depends on how effectively the pruning strategy reduces the combinatorial explosion in the extension process.

Template attacks are particularly effective on implementations of cryptographic algorithms on CMOS devices due to their *contamination* and *diffusion*

properties. Contamination refers to key dependent leakages which can observed over multiple cycles in a section of computation. In CMOS devices, direct manipulation of the key bits makes them part of the device state and these state leakages can persist for several cycles. Additionally, other variables affected by the key, such as key dependent table indices and values, cause further contamination at other cycles. The extent of contamination controls the success of the pruning of the fresh key bits introduced in the expansion phase. It is to be expected that if two keys are almost the same, that even with the effects of contamination, pruning at this stage, may not be able to eliminate one of them. Diffusion is the well-known cryptographic property wherein small differences in key bits are increasingly magnified in subsequent portions of the computation. Even if certain candidates for key bits were not eliminated due to contamination effects, diffusion will ensure that closely spaced keys will be pruned rapidly.

The implementation of an algorithm on a particular device inherently places theoretical bounds on the success of the template attack. The best any adversary can do to approach this theoretical bound is to have extremely good and accurate characterizations of the noise. While one get elaborately sophisticated with such characterizations, in practice approximations such as a multivariate Gaussian model for the noise distributions yields very good results.

2.1 The Multivariate Gaussian Model Approach

The steps in developing a Gaussian model are as follows:

1. Collect a large number L (typically one thousand) of samples on the experimental device for each of the K operations, $\{O_1, \ldots, O_K\}$.
2. Compute the average signal $M_1, \ldots, M_K$ for each of the operations.
3. Compute pairwise differences between the average signals $M_1, \ldots, M_K$ to identify and select only points $P_1, \ldots, P_N$, at which large differences show up. The Gaussian model applies to these N points. This optional step significantly reduces the processing overhead with only a small loss of accuracy.
4. For each operation O_i, the N–dimensional noise vector for sample T is $N_i(T) = (T[P_1] - M_i[P_1], \ldots, T[P_N] - M_i[P_N])$. Compute, the *noise covariance matrix* between all pairs of components of the noise vectors for operation O_i using the noise vectors N_is for all the L samples. The entries of the covariance matrix Σ_{N_i} are defined as:

$$\Sigma_{N_i}[u, v] = cov(N_i(P_u), N_i(P_v))$$

Using this we compute the templates (M_i, Σ_{N_i}) for each of the K operations. The signal for operation O_i is M_i and the noise probability distribution is given by the N–dimensional multivariate Gaussian distribution $p_{\mathbf{N_i}}(\cdot)$ where the probability of observing a noise vector $\mathbf{n}$ is:

$$p_{\mathbf{N_i}}(\mathbf{n}) = \frac{1}{\sqrt{(2\pi)^N |\Sigma_{N_i}|}} \exp(-\frac{1}{2}\mathbf{n}^T \Sigma_{N_i}^{-1}\mathbf{n}), \quad \mathbf{n} \in \mathcal{R}^N \tag{1}$$

where $|\Sigma_{N_i}|$ denotes the determinant of Σ_{N_i} and $\Sigma_{N_i}^{-1}$ is its inverse.

In this model, the optimal technique to classify a sample S, is as follows: for each hypothesized operation O_i, compute the probability of observing S if indeed it originated from O_i. This probability is given by first computing the noise $\mathbf{n}$ in S using the mean signal M_i in the template and then computing the probability of observing $\mathbf{n}$ using the expression for the noise probability distribution and the computed Σ_{N_i} from the template. If the noise was actually Gaussian, then the approach of selecting the O_i with the highest probability is optimal.The probability of making errors in such a classification is also computable. If we use this approach to distinguish two operations O_1 and O_2 with the same noise characterization Σ_N, the error probability is given by:

Fact 1 *[22]For equally likely binary hypotheses, the error probability of error of maximum likelihood test is*

$$P_\epsilon = \frac{1}{2}\,\mathrm{erfc}\Big(\frac{\Delta}{2\sqrt{2}}\Big) \tag{2}$$

where $\Delta^2 = (M_1 - M_2)^T \Sigma_N^{-1}(M_1 - M_2)$ *and* $\mathrm{erfc}(x) = 1 - \mathrm{erf}(x)$.

To implement the pruning process of the template attack, we deal with multiple hypotheses and bound the probability of classification errors by judiciously selecting a small subset of possible operations as most likely candidates.

2.2 The Pruning Process

In the extend-and-prune paradigm of the template attack, each extension results in several hypotheses about the operation being performed. For the attack to be tractable, the pruning process has to reduce the set of possible hypotheses to a very small number while ensuring with high probability that the correct hypothesis is *not* discarded. To achieve this we ensure that the cumulative probability of the hypothesis not retained is within the desired error bound. While this can be done exactly given the precise noise characterizations, several heuristic methods are easier to implement and give good results.

One approach that works well is to scale the probabilities so that the noise probabilities under all of the hypotheses add up to one. We then discard those hypotheses with the lowest scaled probabilities till the cumulative probability of error due to the discarded hypotheses reaches the desired error bound.

Another heuristic that is easy to implement is to fix a constant factor c and only retain those hypotheses in the pruned set whose noise vector probabilities are within this constant fraction c of the highest noise probability: that is, if P_{max} is the maximum noise probability, we keep all hypotheses whose noise probability is at least $\frac{P_{max}}{c}$. We refer to this pruning process as the *ball approach*. The intuition for this heuristic is that if the noise characterization is approximately the same for all hypotheses then the logarithm of the noise probability for a hypothesis is a measure of the distance between the received signal and that hypothesis. The misclassification error is an inverse exponential function of the distance between hypotheses. The heuristic is to create a ball centered at

the received sample whose radius is $d_{min} + \log(c)$, where d_{min} is the shortest distance between the sample and the nearest hypothesis. We then retain only those hypotheses that fall into this ball. Under the assumption of approximately similar noise characterizations, the worst case probability of error can be shown to be bounded by $\mathcal{O}(\frac{\infty}{\sqrt{c}})$ [22]. In practice, the error is much better. In subsequent sections, we will illustrate how this theory can be applied to a number of case studies including an RC4 implementation.

3 Case Study: RC4

We describe a template attack on an implementation of RC4. RC4 is a stream cipher operating on a 256-byte state table. The state table is used to generate a pseudo-random stream of bytes that is then XOR'ed with the plaintext to give the ciphertext. It is a popular choice in a number of products and standards. RC4 uses a variable key length (from 1 to 256 key bytes) to update the 256-byte state table (initially fixed) using the pseudo code below:

```
index1 = index2 = 0;
for (counter = 0; counter < 256; counter++)  {
  index2 = (key[index1] + state[counter] + index2) % 256;
  swap_byte(&state[counter], &state[index2]);
  index1 = (index1 + 1) % key_data_len;
}
```

A portion of the corresponding side channel sample, in this case the power consumption, is shown in Figure 2. The repeated structure observed is exactly five successive iterations of the loop. As described in Figure 1, two keys cannot be distinguished on the basis of a single sample. In fact, this remains true if one were to consider the averages of five samples as illustrated in Figure 3. However, significant and widespread differences become apparent when examining averages of several dozen samples (see Figure 4).

A well-designed system using RC4 is unlikely to permit an attacker to repeatedly obtain samples of identical state initialization computations[3]. Thus the real challenge is to break this implementation using a single sample. The figures clearly show that traditional attacks like SPA will not work and DPA is clearly not an option since we can not obtain more than a single sample.

RC4 is, however, an ideal candidate for template attacks. It is evident from inspecting the code snippet above, that the key byte used in each iteration causes substantial contamination. The loading of the key byte, the computation of `index2` and the use of `index2` in swapping the bytes of the state table all contaminate the side channel at different cycles. The extent of this contamination is easily visible as significant and widespread once averages of a large number of samples are taken. Further, the use of `index2` and the state in subsequent

[3] Note that in devices implementing the 802.11 standard the key initialization is done repeatedly with a fixed secret key and a variable part.

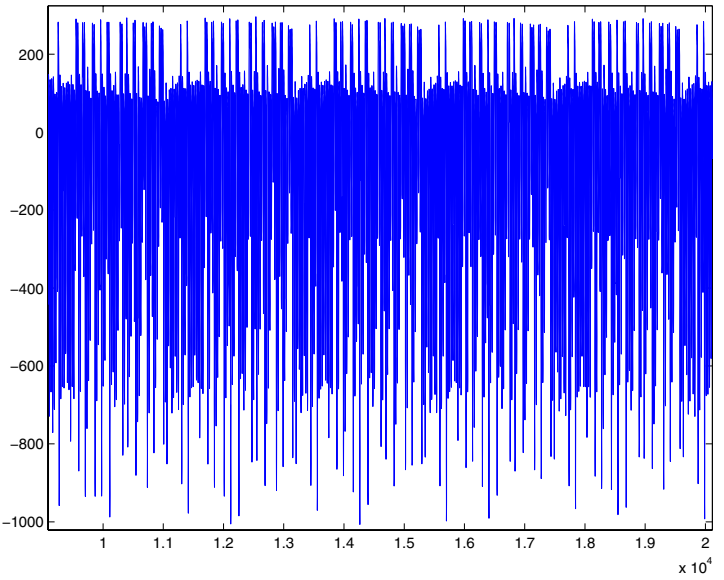


Fig. 2. Power sample during first 5 iterations of RC4 state initialization loop.

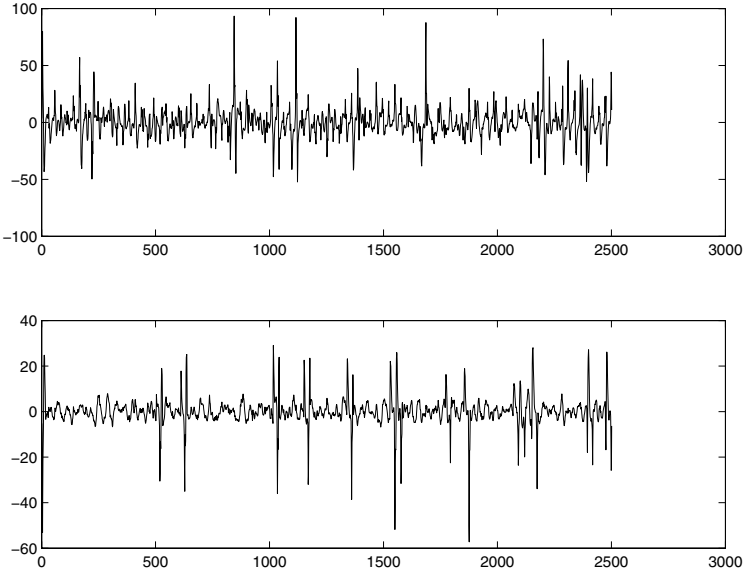


Fig. 3. Differences of averages of 5 side channel samples: upper figure is for the same key while the lower is for two different keys

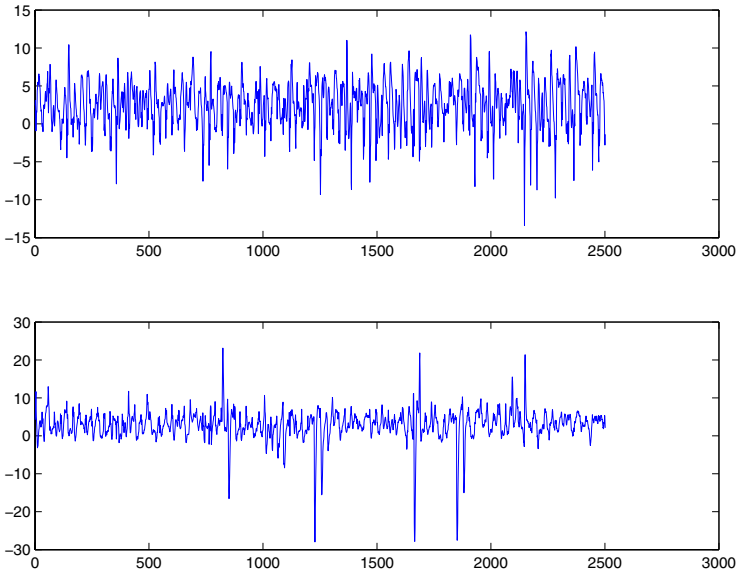


Fig. 4. Differences of averages of 50 side channel samples: upper figure is for the same key while the lower is for two different keys

iterations, and the fact, that RC4 is a well-designed stream cipher, quickly propagates small key differences to cause diffusion. Thus, one expects that templates corresponding to different choices of key bytes are very different and can be used to efficiently and effectively classify a single sample.

3.1 Template Attack on RC4

Inspecting the averaged RC4 side channel samples using several different keys, we identified 42 points in the side channel sample for each iteration of the loop for classification purposes. Our first attempt used statistical measures that treated these 42 points independently, i.e, we only looked at means and standard deviations of the samples at each of the points. Although encouraging, the results have high classification errors (as much as 35%) for pairs of keys with few bit differences. Some empirical classification results of samples with five different keys using this approach are shown in Fig 5. When key bits are very different, even this simplistic approach gives us 100% success rate. However, in general, this approach is unsuitable for an extend-and-prune attack due to high errors. In the worst case, a large number of keys close to the actual key would be retained. Empirically, we have observed that this could be of the order of a few tens of keys. This is because even if keys have the same Hamming weight, the state table addressing part distinguishes keys with different higher order bits. We use the approach described in Section 2 to launch an attack based on multivariate

Key Byte	1111 1110	1110 1110	1101 1110	1011 1110	0001 0000
1111 1110	0.86	0.04	0.07	0.03	0.00
1110 1110	0.06	0.65	0.10	0.20	0.00
1101 1110	0.08	0.16	0.68	0.09	0.00
1011 1110	0.10	0.11	0.08	0.71	0.00
0001 0000	0.00	0.00	0.00	0.00	1.00

Fig. 5. Classification Probability of 5 competing hypotheses using univariate statistics. Entry (i,j) is probability of classifying samples with key i as one with key j.

statistics with the Gaussian noise. For our experiment, we used 10 choices for the first key byte, as shown in Fig. 6. They are carefully chosen to be very close and yielded poor results with the univariate statistics. For each key byte, we computed the mean of 2000 samples of the side channel. We used the same 42 points of interest as in the univariate experiment. The templates consisted of the means and the noise covariance at these points. To obtain statistics on how well this approach would work, we used the templates to classify tens of thousands drawn using one of the 10 choices as the first key byte. Figs. 6 and 7 summarize the results of the classification experiments for this set of 10 key choices. Since the values were carefully chosen to reflect the worst case, these results can be extrapolated to the case of 256 different values of the key byte. Fig. 8 is an extrapolation of our results for the case of 256 different templates by making pessimistic assumptions about the number of "close" keys. In practice the actual results should be much better.

Our first classification heuristic was to retain only the most likely hypothesis i.e. with highest likelihood probability. Even with such a drastic pruning approach, average classification success probability was 99.3% with these 10 hypotheses and worst–case probability was 98.1%. Detailed results are described in column 1 of the Fig. 6. We can get reasonable results even if we use this extensive pruning strategy in each iteration of the extend-and-prune approach. Extrapolating, as shown in Fig. 8 we expect average error probability of the closest hypothesis approach to be about $5-6\%$ when we consider all 256 possible values, since we pessimistically expect around $50-60$ keys to be "close" to any key. Bounding the error probability over many iterations by the sum of error is in each iteration we note that when the number of key bytes is small this can be used to extract all key bytes. For example, we can do better than 50% for about 8 bytes of key material.

With a little more effort, much better results can be obtained by using the *ball approach* to pruning as shown in columns 2, 3 and 4 of Fig. 6 showing success probability of retaining the correct hypothesis for balls with different values of ball size. Average success probability has improved and is better than 99.8% and the worst–case probability is 99.5%, for this set of samples. As shown in Fig. 7 the average number of hypotheses that we retain is still close to 1 for balls of size e^6 and e^{12}. Again, using an estimate of about $50-60$ close keys, we can extrapolate these results as done in Fig. 8. For example, choosing the ball size e^6, with good probability, at the end of one iteration we expect to retain at

Key Byte	Ball Size $c = 1$	Ball Size $c = e^6$	Ball Size $c = e^{12}$	Ball Size $c = e^{24}$
1111 1110	98.62	99.46	99.88	99.94
1110 1110	98.34	99.82	99.88	99.88
1101 1110	99.16	100.00	100.00	100.00
1011 1110	98.14	99.52	99.82	100.00
0111 1110	99.58	99.76	99.89	99.94
1111 1101	99.70	99.94	99.94	99.94
1111 1011	99.64	99.82	99.82	99.89
1111 0111	100.00	100.00	100.00	100.00
1110 1101	99.76	99.82	99.88	99.88
1110 1011	99.94	100.00	100.00	100.00
Average	99.29	99.81	99.91	99.95

Fig. 6. Percentage of samples for which the correct hypothesis is retained under different ball sizes with 10 competing hypotheses

most 2 hypotheses, yet we are guaranteed to retain the correct hypothesis with probability at least 98.67%. Using this approach independently in each iteration, we can correctly classify keys of size n bytes with expected probability around $(100 - 1.33n)\%$ and the number of remaining hypotheses would grow no more than $(1.5)^k$, which is substantially than the 2^{8k} (the entropy of the key). The next subsection describes experiments where we use larger templates comprising multiple iterations to get better pruning.

Ball Size $c = 1$	Ball Size $c = e^6$	Ball Size $c = e^{12}$	Ball Size $c = e^{24}$
1	1.041	1.158	1.842

Fig. 7. Expected number of hypotheses retained under different ball sizes for 10 competing hypothesis.

	Ball Size $c = 1$	Ball Size $c = e^6$	Ball Size $c = e^{12}$	Ball Size $c = e^{24}$
Success Prob.	95.02	98.67	99.37	99.65
Retained Hypotheses	1	1.29	2.11	6.89

Fig. 8. Extrapolated results for 256 competing hypotheses.

Iteration. Instead of using the above attack *independently* on subsequent portions of the sample which use the next key byte, it is advantageous to consider the basic attack on the whole prefix of the sample including previous iterations. In our RC4 implementation, we now use 84 points of interest in the sample spread over two loop iterations to prune possible candidates for the first two key bytes. After pruning hypotheses for the first byte (as described earlier), we extend extend each remaining hypothesis by all 256 possible values for the second

key byte. 84–point templates are created for this set of possibilities. Using these larger templates, we classify the sample and retain only those hypotheses for the first two bytes which remain in the ball around the sample.

Key Byte	Ball Size $c = 1$	Ball Size $c = e^6$	Ball Size $c = e^{12}$	Ball Size $c = e^{24}$
1101 1110 1111 1110	99.16	99.58	99.70	99.94
1101 1110 1110 1110	98.10	99.53	99.88	99.94
1101 1110 1101 1110	99.46	99.88	99.94	100.00
1101 1110 1011 1110	98.80	99.64	99.89	100.00
1101 1110 0111 1110	99.33	99.70	99.88	99.94
1101 1110 1111 1101	99.88	99.94	99.94	99.94
1101 1110 1111 1011	99.03	99.58	99.70	100.00
1101 1110 1111 0111	99.94	100.00	100.00	100.00
1101 1110 1110 1101	99.82	99.82	99.88	99.88
1101 1110 1110 1011	99.94	100.00	100.00	100.00
1011 1110 1111 1110	96.81	99.05	99.65	100.00
1011 1110 1110 1110	99.76	99.88	99.88	99.94
1011 1110 1101 1110	98.57	99.82	100.00	100.00
1011 1110 1011 1110	97.87	99.76	100.00	100.00
1011 1110 0111 1110	98.25	99.28	99.52	99.82
Average	98.98	99.70	99.86	99.96

Fig. 9. Percentage of samples for which the correct hypothesis is retained for 2 iterations and 15 competing hypotheses

Ball Size $c = 1$	Ball Size $c = e^6$	Ball Size $c = e^{12}$	Ball Size $c = e^{24}$
1	1.04	1.141	1.524

Fig. 10. Expected number of hypotheses retained under different ball sizes for 15 competing hypotheses.

To verify that this is better than using just 42 points of the second iteration, we performed the following experiment: We considered 15 possible combinations of the first and second key bytes. There were two possibilities for the first key byte which roughly simulates what we expect after the template attack on the first iteration. The classification results for different ball sizes are given in Figs. 9 and 10. To compare these results to the earlier case, we must scale the results from the tables for 10 hypotheses. Since there are 14 wrong hypothesis instead of 9 before comparison each figure from the earlier tables must be scaled by a factor of 14/9. This is indeed the case for the error probability of not retaining the correct hypothesis for ball sizes of 1,e^6 and e^{12}. The error probability is better for ball size of e^{24} when we use longer templates. More importantly, note that the average number of hypotheses retained is substantially better uniformly for each and every ball size. Thus, we retain fewer number of candidates by using a longer template. Empirically, we have observed that with a longer template, with

extremely high probability, all of the hypotheses remaining have the correct first key byte. After 2 iterations, we could only find 1 sample amongst 16000 where a hypothesis with the wrong first key byte was retained.

4 Other Case Studies

We briefly describe two other examples where template attacks can be used. The first example is a smart card with fast DES hardware. Use of such hardware is currently very popular, since they highly resistant to power attacks. The only exposure for such engines is the loading of the key bytes from EEPROM which usually leaks the hamming weight. However, implementors also have to worry about Differential Fault Attacks [2,1], and a card that we looked at addressed this problem using a checksum on key bytes, verified before the key was loaded in the DES engine. The checksum was calculated accessing key bytes in circular order starting from a random offset. This can be easily fixed using signal processing. Fig. 11 shows an averaged signal depicting the checksum computation loop in more detailfollowed by the enabling of the DES engine (the region with large current consumption). In the checksum calculation, the key bytes are loaded from EEPROM, placed in RAM and are then used as operands in the checksum computation, thus creating a large contamination. For example, Figure 12 shows that even key bytes with the same hamming weight can be distinguished by taking averages of a few samples. Such types of signals are prime candidates for a template attack.

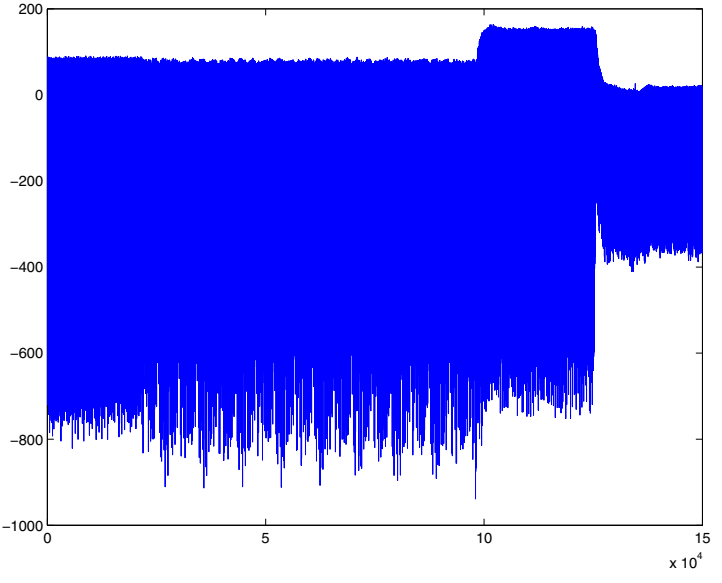


Fig. 11. Average signal showing details of checksum calculation for 8 key bytes.

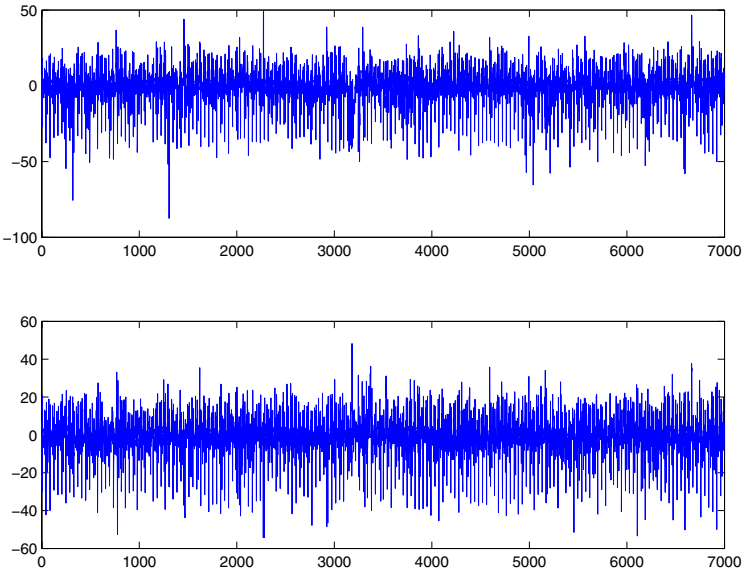


Fig. 12. Differences in average signals: lower figure is for same key and upper figure is for different keys but with same hamming weight.

The second example is a EM signal we collected from a distance of 15 feet away from an SSL accelerator inside a closed server. We programmed the SSL accelerator to do a 2048 bit exponentiation with a single-nibble exponent. The fact that the exponent leaks from this computation follows from Figure13 which shows the signal differences between exponentiation with two different nibble exponents B and D after taking averages of few signals and some signal processing. Thus template attacks are very feasible for this case as well. In this example, other attacks such as MESD [18] can be possible as well, if one can collect a large number of EM samples.

5 Implications and Countermeasures

In principle, the template attack described is the strongest side channel attack possible from an information theoretic sense. The information present in each portion of the side channel signal is fully used for classification. This makes it a very powerful tool for attacking a wide range of cryptographic implementations. However, the effort required, in terms of creating a large number templates in an adaptive manner, make the task daunting. It also presumes the availability of an identical test device which can be programmed to the adversary's whim. Since the attack requires only a single invocation of the test device, all countermeasures to side channel attacks that rely on limiting the number of samples that an adversary can take with a fixed key may be vulnerable depending on the extent

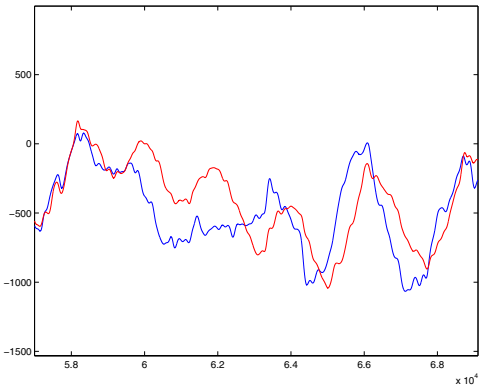


Fig. 13. Average processed EM signals for two diffrent private exponents.

of contamination. Such countermeasures include high level protocols to limit key usage and non-linear key update techniques of [12].

The requirement of having an identical experimental device is also the weakness of the template approach. Randomization in the computation such as address/data scrambling, blinding/masking of data and key bits and ensuring that the adversary cannot control the choice of randomness in *his own experimental device* is one way this attack can be mitigated. This countermeasure may not be feasible for highly programmable devices such as SSL accelarators. The overriding principle in building in securing implementations against templates is to minimize contamination caused by use of sensitive information in the clear.

References

1. Ross Anderson and Markus Kuhn. Low Cost Attacks on Tamper Resistant Devices. In Proc. Security Protocols, 5th International Workshop, Paris, France, Springer-Verlag LNCS Volume 1361 , pp 125–136, April 1997.
2. Dan Boneh, Richard DeMillo and Richard Lipton. On the Importance of Checking Cryptographic Protocols for Faults. Journal of Cryptology, Springer-Verlag, Volume 14, Number 2, pp 101–119, 2001.
3. Suresh Chari, Charanjit S. Jutla, Josyula R. Rao and Pankaj Rohatgi. Towards Sound Countermeasures to Counteract Power–Analysis Attacks. Proc. Crypto '99, Springer–Verlag, LNCS 1666, August 1999, pages 398–412.
4. Suresh Chari, Charanjit S. Jutla, Josyula R. Rao and Pankaj Rohatgi. A Cautionary Note Regarding the Evaluation of AES Candidates on Smart Cards. Proc. Second AES Candidate Conference, Rome, Italy, March 1999.
5. Christopher Clavier, Jean-Sebastien Coron, and Nora Dabbous. Differential Power Analysis in the Presence of Hardware Countermeasures. In Proc. Workshop on Cryptographic Hardware and Embedded Systems, Aug. 2000, LNCS 1965, Springer-Verlag, pp 252–263.

6. Jean-Sebastien Coron. Resistance against Differential Power Analysis for Elliptic Curve Cryptosystems. In Proc. Workshop on Cryptographic Hardware and Embedded Systems , Aug. 1999, LNCS 1717, Springer-Verlag. pp 292–302.
7. Jean-Sebastien Coron, and Louis Goubin. On Boolean and Arithmetic Masking against Differential Power Analysis. In Proc. Workshop on Cryptographic Hardware and Embedded Systems, Aug. 2000, LNCS 1965, Springer-Verlag. pp 231–237.
8. P.N. Fahn and P.K. Pearson. IPA: A New Class of Power Attacks. In Proc.Workshop on Cryptographic Hardware and Embedded Systems, Aug. 1999, LNCS 1717, Springer-Verlag. pp 173–186.
9. K. Gandolfi, C. Mourtel and F. Olivier. Electromagnetic Attacks: Concrete Results. In Proc. Workshop on Cryptographic Hardware and Embedded Systems, Paris, France, May 2001.
10. L. Goubin and J. Patarin. DES and Differential Power Analysis. In Proc. Workshop on Cryptographic Hardware and Embedded Systems, Aug. 1999, LNCS 1717, Springer-Verlag. pp 158–172.
11. M.A. Hasan. Power Analysis Attacks and Algorithmic Approaches to Their Countermeasures for the Koblitz Curve Cryptosystems. In Proc. Workshop on Cryptographic Hardware and Embedded Systems, Aug. 2000, LNCS 1965, Springer-Verlag. pp 93–108.
12. P. Kocher, J. Jaffe and B. Jun. Differential Power Analysis: Leaking Secrets. In Proc. Crypto '99, Springer-Verlag, LNCS 1666, pages 388–397.
13. P. Kocher. Timing Attacks on Implementations of Diffie-Hellman, RSA, DSS and Other Systems. In Proc. Crypto '96, LNCS 1109, Springer-Verlag, pp 104–113.
14. J. Kelsey, Bruce Schneier, D. Wagner and C. Hall. Side Channel Cryptanalysis of Product Ciphers. Journal of Computer Security, Volume 8, Number 2–3, 2000, pages 141–158.
15. Rita Mayer-Sommer. Smartly Analyzing the Simplicity and the Power of Simple Power Analysis on Smart Cards. In Proc. Workshop on Cryptographic Hardware and Embedded Systems, Worcester, MA, USA, Aug. 2000, LNCS 1965, Springer-Verlag. pp 78–92.
16. Thomas S. Messerges. Securing the AES Finalists Against Power Analysis Attacks. In Proc. Fast Software Encryption Workshop 2000, New York, NY, USA, April 2000, Springer-Verlag.
17. Thomas S. Messerges. Using Second-Order Power Analysis to Attack DPA Resistant Software. In Proc. Workshop on Cryptographic Hardware and Embedded Systems, Aug. 2000, LNCS 1965, Springer-Verlag. pp 238–251.
18. T.S. Messerges, E.A. Dabbish, and R.H. Sloan. Power Analysis Attacks of Modular Exponentiation in Smart Cards. In Proc. Workshop on Cryptographic Hardware and Embedded Systems 1999, Aug. 1999, LNCS 1717, Springer-Verlag. pp 144–157.
19. Jean–Jacques Quisquater and David Samyde. Simple Electromagnetic analysis for Smart Cards: New Results. Rump session talk at Cyrpto 2000.
20. Dakshi Agrawal, Bruce Archambeault, Josyula Rao, Pankaj Rohatgi. The EM Side-Channel(s). In Proc. Workshop on Cryptographic Hardware and Embedded Systems 2002, Aug. 2002,
21. Adi Shamir. Protecting Smart Cards from Power Analysis with Detached Power Supplies. In Proc. Workshop on Cryptographic Hardware and Embedded Systems, Aug. 2000, LNCS 1965, Springer-Verlag. pp 71–77.
22. H. L. Van Trees. Detection, Estimation, and Modulation Theory, Part I. John Wiley & Sons. New York. 1968.

The EM Side–Channel(s)

Dakshi Agrawal, Bruce Archambeault, Josyula R. Rao, and Pankaj Rohatgi

IBM T.J. Watson Research Center
P.O. Box 704
Yorktown Heights, NY 10598
{agrawal,barch,jrrao,rohatgi}@us.ibm.com

Abstract. We present results of a systematic investigation of leakage of compromising information via electromagnetic (EM) emanations from CMOS devices. These emanations are shown to consist of a multiplicity of signals, each leaking somewhat different information about the underlying computation. We show that not only can EM emanations be used to attack cryptographic devices where the power side–channel is unavailable, they can even be used to break power analysis countermeasures.

1 Introduction

Side–channel cryptanalysis has been used successfully to attack many cryptographic implementations [7,8]. Most published literature on side–channels deals with attacks based on timing or power. With the recent declassification of portions of the TEMPEST documents [5], and other recent results [9,6], an awareness of the potential of the EM side–channel is developing. However, some basic questions remain unanswered. For instance, what are the causes and types of EM emanations? How does information leaked via EM emanations compare with leakages from other side–channels? What new devices and implementations are vulnerable to EM side–channel attacks? Can the EM side–channel overcome countermeasures designed to provide protection against other side–channel attacks? With questions such as these in mind, we conducted a systematic investigation of EM side–channel leakage from CMOS devices. In this paper, we address each of these basic questions.

In Section 2, we discuss the causes and types of various EM signals and describe the equipment required to capture and extract these signals. In addition to the direct emanations, EM signals consist of several compromising signals which are unintentional and are found in unexpected places. For instance, researchers have thus far missed the faint, but far more compromising amplitude modulated EM signals present even in the power line.

Section 3 presents experimental results illustrating various types of emanations and Section 4 provides a qualitative comparison of information leakages from EM and power. These results are very instructive. One crucial observation is that even a *single* EM sensor can easily pick up multiple compromising signals of different types, strengths and information content. Moreover, significant amount of compromising information is to be found in very low energy signals.

B.S. Kaliski Jr. et al. (Eds.): CHES 2002, LNCS 2523, pp. 29–45, 2003.

It is therefore critical that signals be separated early in the acquisition process to avoid loss of these low energy signals due to precision limits of signal capturing equipment. A very effective way to achieve such a separation is to exploit unintentionally modulated carriers at higher frequencies where there is less interference and noise rather than focusing on direct emanations in the baseband where the large amount of interference and noise may require techniques such as chip decapsulation and use of carefully positioned micro–antenna [9,6].

Using EM, we launched attacks such as simple and differential electromagnetic attacks (SEMA and DEMA [9]) on straight–forward implementations of DES, RSA and COMP128 on smart cards, cryptographic tokens and SSL accelerators. While the EM side–channel remains the most viable avenue for attacking cryptographic devices where the power side–channel is unavailable, an important question is whether the EM side–channel provides any other advantage when the power side–channel is available. In Section 5, we answer this in the affirmative. We outline an approach that breaks some fielded systems with power analysis countermeasures. The approach is based on the observation that most devices have classes of "bad instructions" whose leakage in some EM side–channel far exceeds the corresponding leakage in the power side–channel and works against two major classes of power analysis countermeasures [8,2,4]. We illustrate this approach by attacking a test implementation[1] of the secret–sharing countermeasures of [2,4]. This approach works in many cases even when the code is unknown.

Despite their effectiveness, our low–cost attacks provide only a glimpse of what is possible: combining leakages from multiple EM signals could yield substantially better attacks. Furthermore, developing countermeasures requires a methodology to assess the net information leakage from all the EM signals realistically available to an adversary. Our work on these aspects of the EM side–channel(s) is described in more detail in [1].

2 EM Emanations and Acquisition

This section describes the origin and types of various compromising EM signals that we have observed[2] and the equipment and techniques to extract them.

2.1 Origin of EM Emanations

EM emanations arise as a consequence of current flows within the control, I/O, data processing or other parts of a device. These flows and resulting emanations may be *intentional* or *unintentional*. Each current carrying component of the device not only produces its own emanations based on its physical and electrical characteristics but also affects the emanations from other components due to coupling and circuit geometry.

[1] To avoid disclosing weaknesses of commercially deployed systems.

[2] While there is an obvious overlap with the declassified TEMPEST documents (NACSIM 5000) [5], we only describe what we have verified in our investigations.

An attacker is typically interested in emanations resulting from data processing operations. In CMOS devices, ideally, current only flows when there is a change in the logic state of a device and this logic state change is controlled by a "square–wave" shaped clock. These currents result in compromising emanations, sometimes, in unintended ways. Such emanations carry information about the currents flowing and hence the events occurring during each clock cycle. Since each active component of the device produces and induces various types of emanations, these emanations provide multiple views of events unfolding within the device at each clock cycle. This is in sharp contrast to the power side–channel where only a single aggregated view of net current inflow is available thus, explaining why the EM side–channel(s) are much more powerful.

2.2 Types of EM Emanations

There are two broad categories of EM emanations:

1. Direct Emanations: These result from *intentional* current flows. Many of these consist of short bursts of current with sharp rising edges resulting in emanations observable over a wide frequency band. Often, components at higher frequencies are more useful to the attacker due to noise and interference prevalent in the lower bands. In complex circuits, isolating direct emanations may require use of tiny field probes positioned very close to the signal source and/or special filters to minimize interference: getting good results may require decapsulating the chip packaging [6,9].

2. Unintentional Emanations: Increased miniaturization and complexity of modern CMOS devices results in electrical and electromagnetic coupling between components in close proximity. Small couplings, typically ignored by circuit designers, provide a rich source of compromising emanations. These emanations manifest themselves as *modulations* of carrier signals generated, present or "introduced" within the device. One strong source of carrier signals is the ubiquitous harmonic–rich "square–wave" clock signal[3]. Other sources include communication related signals. Ways in which modulation occurs include:

a. Amplitude Modulation: Non–linear coupling between a carrier signal and a data signal results in the generation and emanation of an Amplitude Modulated (AM) signal. The data signal can be extracted via AM demodulation using a receiver tuned to the carrier frequency.

b. Angle Modulation: Coupling of circuits also results in Angle Modulated Signals (FM or Phase modulation). For instance, while signal generation circuits should ideally be decoupled from data processing circuits, this is rarely achieved in practice. For example, if these circuits draw upon a limited energy source the generated signal will often be angle modulated by the data signal. The data signal is recoverable by angle demodulation of the generated signal.

[3] Theoretically a symmetric, square clock signal consists of the fundamental frequency and all the odd harmonics with progressively diminishing strengths. In practice, the clock signal is always imperfect.

Exploiting unintentional emanations can be much more effective that trying to work with direct emanations. Some modulated carriers have substantially better propagation than direct emanations. This enables attacks to be carried out without resorting to invasive techniques and even attacks that can be performed at a distance. None of the attacks described in this paper require any invasive techniques or fine grained positioning of probes. Secondly, careful field probe positioning cannot separate two sources of direct emanations in close proximity, while such sources may be easily separable due to their differing interaction with the carriers present in the vicinity.

2.3 Propagation and Capture of EM Signals

EM signals propagate via radiation and conduction, often by a complex combination of both. Thus two classes of sensors are required to capture the signals that emerge. The most effective method for capturing radiated signals is to place near field probes as close as possible to the device or at least in the near field, i.e., no more that a wavelength away. Some of these emanations can also be captured at much larger distances using standard antennas. In our experiments, the most effective near field probes are those made of a small plate of a highly conducting metal like silver or copper attached to a coaxial cable. In the far field, we used biconical and log–periodic wide–band antennas as well as hand–crafted narrow–band, high gain Yagi antennas. Conductive emanations consist of faint currents found on all conductive surfaces or lines attached to the device possibly riding on top of stronger, intentional currents within the same conductors. Capturing these emanations requires current probes. The quality of the received signal improves if the equipment is shielded from interfering EM emanations in the band of interest, though the shielding does not have to be elaborate.

The emanations received by the sensor have to be further processed to extract compromising information. For direct emanations, filters may suffice. For unintentional emanations, which manifest themselves as modulations of carrier signals, a receiver/demodulator is required. For experimental work, a wide bandwidth, wideband tunable receiver such as the R–1550 Receiver from Dynamic Sciences and the 8617 Receiver from Watkins–Johnson is convenient. A cheaper alternative is to use wide–band radio receivers such as the ICOM 7000/8500 which have intermediate frequency outputs and to then perform the demodulating functionality in software. An even cheaper approach is to construct the receiver using commonly available low noise electronic components. At some stage, the signal has to be digitized using digital scope/sampling card as done for power analysis attacks. Equipment such as spectrum analyzers are also useful for quickly identifying carriers and potentially useful emanations. A useful rule–of–thumb is to expect strong carriers at odd harmonics of the clock.

3 Experimental Results

We describe experiments that illustrate the various types and nature of EM emanations.

Experiment 1: Direct Near–Field Emanations: We programmed a recently deployed smart card, called smartcard A (to protect vendor identity[4]), to enter a 13 cycle infinite loop using the externally supplied 3.68MHz clock. A near–field probe (a small metal plate attached to a co–axial cable) was placed near the chip at the back of smart card. After wide–band amplification, 500K sample points (representing approx 284 iterations of the loop) were captured using an 8–bit, 500MHz digital scope. In the time domain, the baseband *direct emanations* signal (band centered at 0MHz), looked like a differentiated form of the external clock and provided *no* visual indication of a loop execution. In the frequency domain, the signal received by the probe consists of the signal of interest, i.e., a periodic signal corresponding to a loop iteration at 283KHz (3.68MHz/13), other signals from the chip and its vicinity such as the clock (periodic with freq 3.68MHz) and aperiodic noise. Capturing the received signal with a limited resolution scope further introduces quantization noise. Figure 1 plots the magnitude[5] of the FFT of the captured baseband signal against the frequency in KHz over the 0–200 MHz band. The large spikes below 100 MHz are the high energy harmonics of the clock signal and tiny spikes sprinkled between them are other types of direct and unintentional emanations which are of interest. Very little signal is noticeable above 125 MHz because these signals have lower strengths and have been overwhelmed by quantization noise. In the linear scale used in Figure 1, the loop execution is not apparent. On a log (base 10) scale, zooming into the region from 0 to 20MHz, as shown in Figure 2, the signal of interest at 283KHz and its harmonics can be seen interspersed between the clock signal and its harmonics. Note that the use of a large time window, i.e., 284 iterations of the loop, helps in detecting this periodic signal since aperiodic noise from the chipcard, environment and quantization gets reduced due to averaging. Since the direct emanations are at least an order of magnitude smaller than interfering signals, exploiting them in the presence of quantization noise, is quite challenging and has been addressed by [6,9]. Our approach focuses on the much easier task of exploiting *unintentional emanations.*

Experiment 2: Unintentional Near–Field AM Emanations: We use the same setup as in Experiment 1, but with the output of the probe connected to an AM receiver, tuned to the 41'st clock harmonic at 150.88 MHz with a band of 50MHz. The demodulated output was sampled with a 12–bit 100MHz scope[6] and 100K sample points representing approximately 284 loop iterations were collected. Figure 3 plots the magnitude of the FFT of this signal against the frequency in KHz. Notice that even in this *linear* scale plot, the signal of interest, i.e., the 283KHz signal corresponding to the loop and its harmonics, is clearly

[4] Smartcard A is 6805–based, uses 0.6 micron triple metal technology with an optional variable internal clock as one defense against DPA.

[5] In all figures, signal magnitudes should be treated as relative quantities: we don't track the absolute values as the signals typically undergo analog processing before being captured by an 8/12–bit scope. The scope sensitivity is set so that the 8/12–bit dynamic range is fully utilized.

[6] The lower bandwidth allows the use of a lower sampling rate with higher precision.

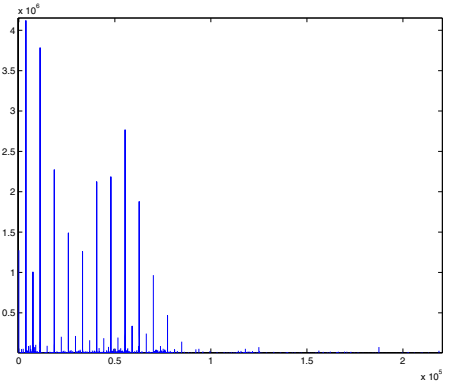


Fig. 1. FFT of baseband signal from Experiment 1 with Smartcard A

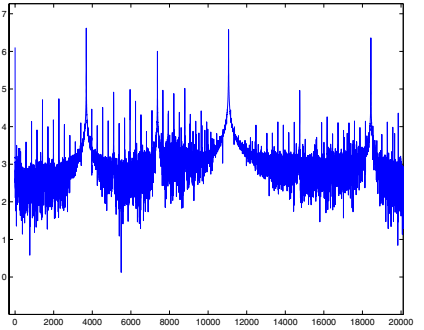


Fig. 2. Log of FFT in the region 0–20MHz from Experiment 1 with Smartcard A

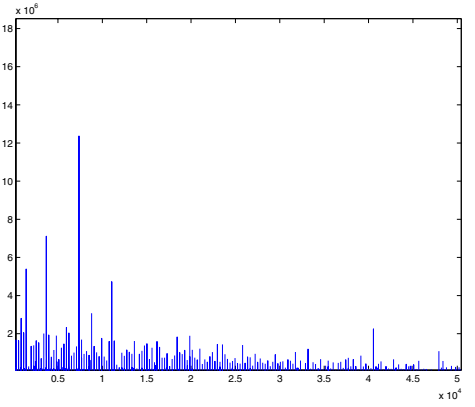


Fig. 3. FFT of demodulated signal (150.88 MHz carrier, 50Mz band) in Experiment 2 with Smartcard A

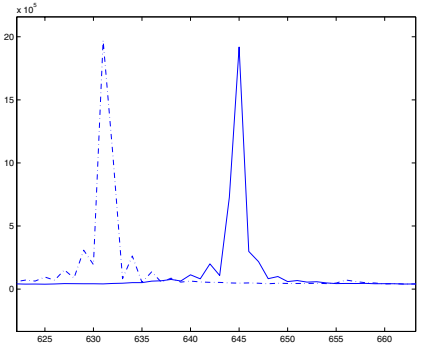


Fig. 4. Two FFTs showing loop frequency differences (LSB 0 and 1) for smartcard A

visible among the clock harmonics. The loop structure is also clearly visible in the time domain. Notice that these greatly improved results were obtained using the same sensor setting as in Experiment 1, and with the same number of loop iterations. Note that we are also operating in a part of the spectrum which showed hardly any signal according to Figure 1; since the signals in this band were overwhelmed by the quantization noise in that experiment.

Experiment 3: Unintentional Near/Far–Field Angle Modulated Emanations: Next we enabled the variable internal clock DPA protection mechanism in Smartcard A and kept everything else the same. One of the instructions in the 13–cycle loop was to load a user supplied byte B from RAM to accumulator. We experimented with different values of the byte B and made the following surprising observation: the average frequency of the 13–byte loop was dependent on the least significant bit (LSB) of B but not on other bits. This is shown in Figure 4, where the magnitude of FFT of the EM output for two different cases is plotted against the frequency in KHz. The first case (shown by a broken line) shows the loop frequency with the $LSB(B) = 1$ and in the second case (shown by a solid line) the loop frequency when the $LSB(B) = 0$. In the first case, the loop runs slower. This is due to coupling between the LSB and the circuitry generating the internal clock. Although the clock frequency itself varies frequently, when there is a 1 bit on the LSB line, the intrinsic variation is biased towards slowing down the clock for a couple of subsequent cycles. We speculate that this is due to the clock circuitry drawing energy from the same source as some other circuitry affected by the LSB. Thus, angle demodulation, e.g., FM demodulation, turns out to be a good avenue for attacking smartcard A using LSB based hypothesis. This effectively transforms a countermeasure into a liability! Another advantage of such an attack is that it can be performed at a distance in the far field since the clock signal is quite strong.

Experiment 4: Unintentional Far–Field AM Emanations: We examined emanations from an Intel–based server containing a commercial, PCI bus based SSL accelerator S[7]. We programmed the server to repeatedly invoke S to perform

[7] S is rated to perform 200, 1024-bit CRT based RSA private key ops/s.

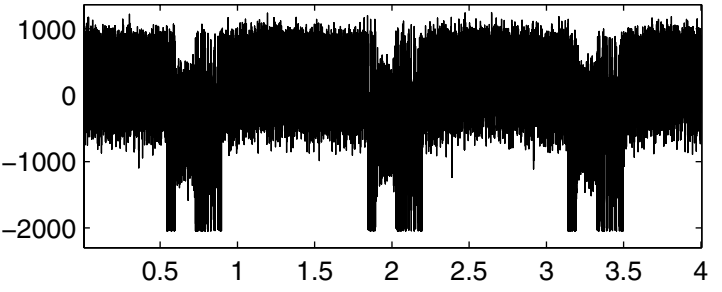


Fig. 5. EM Signal from SSL Accelerator S

a 2048 bit exponentiation with a single–nibble exponent. Several AM modulated carriers (at multiples of the 33MHz PCI clock) containing compromising information propagated to distances upto forty feet. Figure 5 plots a signal (amplitude vs. time in ms) captured by a log–periodic antenna 15 feet away using the 299MHz carrier and 1MHz bandwidth. Three invocations of S are clearly visible as bands where the amplitude goes below -1000. At this resolution, the macro structures of the exponentiation are already visible. At higher resolutions, there is enough information to enable the new class of template attacks [3].

Experiment 5: Conductive Emanations: Conductive emanations appear at unexpected places and are easy to overlook. In fact, if researchers experimenting with power analysis attacks re–analyze the raw signals from their current probes, they will discover that apart from the relatively low frequency, high amplitude power consumption signal, there are faint higher frequency AM modulated carriers representing conductive EM emanations from the device, since the power line is also a conductor. Figure 6 plots one such EM signal (amplitude vs time in 10ns units) extracted from the power line by AM demodulating one such carrier while a smart card (which we call smartcard B[8]) executes 3 rounds of DES. These rounds are clearly visible in the signal.

4 Information Leakage across EM Spectrum

In this section, we provide experimental evidence to reinforce a central theme of this paper, i.e., the output of even a single wide–band EM sensor logically consists of multiple EM signals each carrying qualitatively different compromising information and in some cases, EM leakages can be substantially superior to the power consumption signal.

While the presence of certain types of EM signals (e.g., angle modulated carriers, intermodulated carriers etc) are device dependent, our experiments show that invariably, AM carriers at clock harmonics are a rich and easily accessible source of compromising signals. For smart cards, since the fundamental

[8] Smartcard B is a 6805–based, 0.7micron, double metal technology card with inbuilt noise generators.

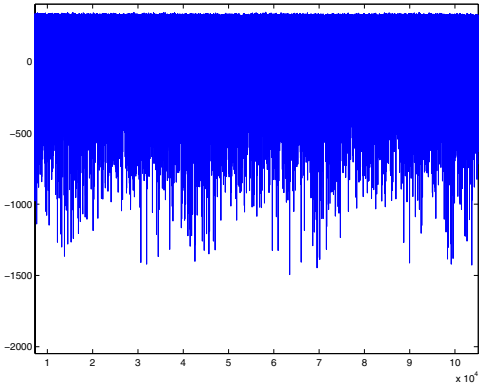


Fig. 6. EM Signal on Power Line for 3 rounds of DES on smartcard B

frequency is low, the intermediate harmonics are usually the best. Lower harmonics suffer from excessive noise and interference and higher harmonics tend to have extremely low signal strength[9].

We now examine the leakage of information from four types of signals obtained from smartcard B when it performed DES in software. No power analysis countermeasures, except for the internal noise generators, were enabled on the card. The smart card ran on the 3.68MHz external clock. Three of these signals were obtained by AM demodulating the output of a near field probe placed as in Experiment 1, at three different carrier frequencies (50MHz bands around 188MHz, 224.5MHz and 262MHz). The fourth signal was the power consumption signal. All signals were collected by a 12–bit, 100MHz digital scope.

It is well known that plotting the results of a differential side channel attack launched against a bit value used in a computation is a good way to assess the leakage of the bit [8]. This is because the plot is essentially the difference between the average of all signals in which the bit is 1 and the average of all signals in which the bit is 0, plotted against time. At points in the computation where this bit is not involved or where the bit is involved but information about it does not leak in the side–channel, the value of the difference is small and not noticeable. At points where the bit is used in the computation *and* this information leaks in the signal, this difference is likely to be large or noticeable.

Figures 7, 8, 9, and 10 show the results of a differential side–channel attack on an S–box output bit in the first cycle of the DES implementation, using the four different signals. Figures 7, 8 and 9, are for the EM signals and Figure 10 is for the power signal. All figures are aligned in time. In all figures, the X–axis shows elapsed time in 10ns units and the Y–axis shows the difference in the averages of signals with bit=0 and bit=1 for 2000 invocations of DES with random inputs. Even at this resolution, it is clear that the leakage results are qualitatively different from each other. There are some gross similarities between the EM leakages in Figures 7 and 8 and between the EM leakage in Figure 9

[9] This is because clock edges are not very sharp in practice.

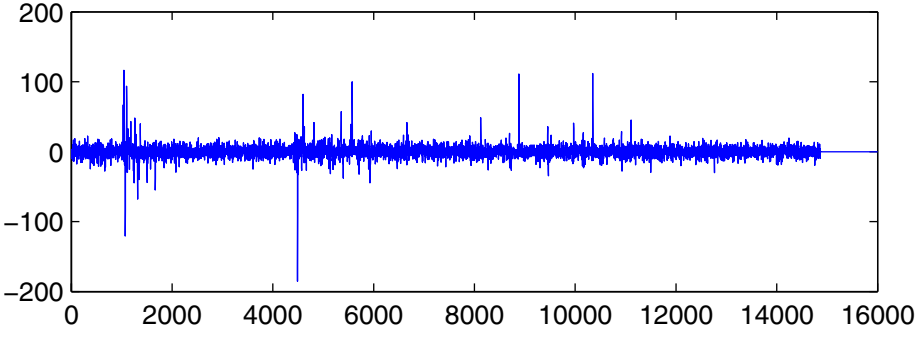


Fig. 7. DEMA attack on DES on smartcard B using the 224.5 MHz carrier

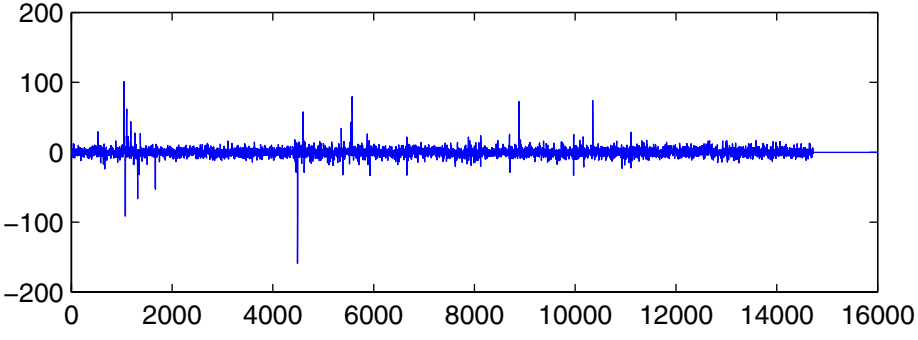


Fig. 8. DEMA attack on DES on smartcard B using the 262MHz carrier

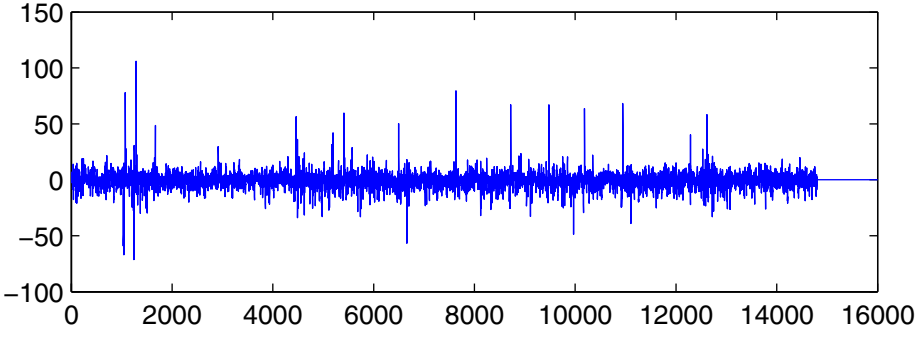


Fig. 9. DEMA attack on DES on smartcard B using the 188MHz carrier

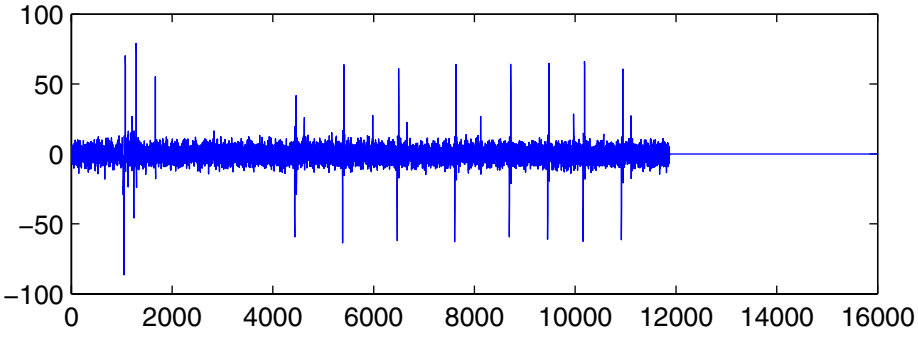


Fig. 10. DPA attack on DES on smartcard B

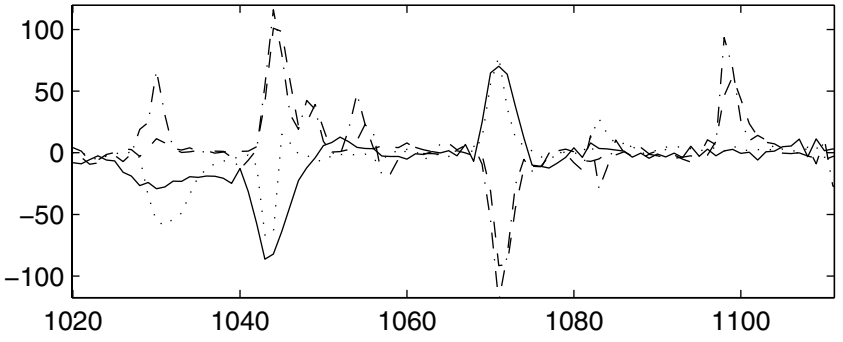


Fig. 11. Comparison of DEMA/DPA Leakages at region 1020–1110

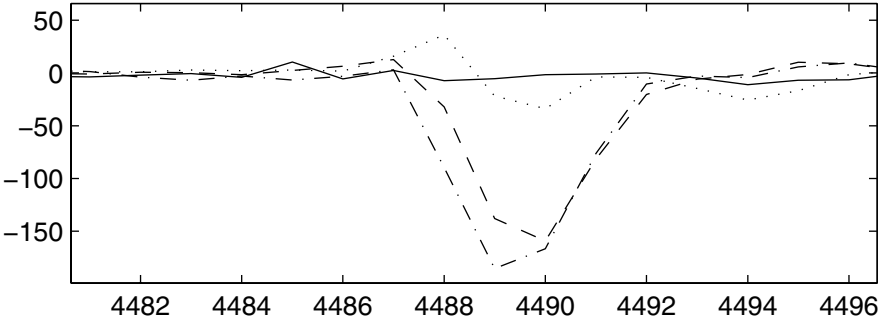


Fig. 12. Comparison of DEMA/DPA Leakages at region 4482–4496

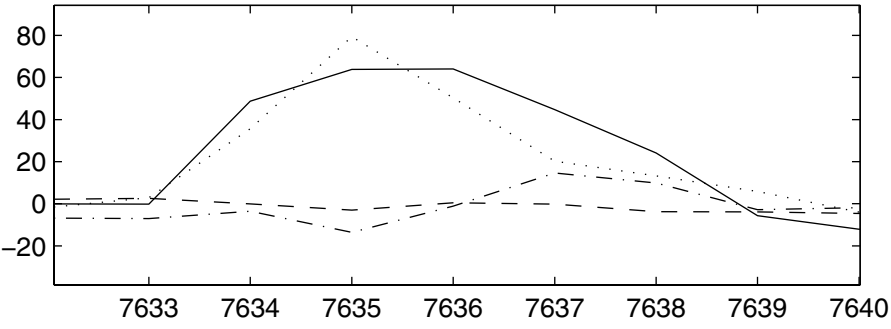


Fig. 13. Comparison of DEMA/DPA Leakages at region 7633–7640

and the power leakage in Figure 10. These leakages can be compared by plotting them together. Figures 11, 12, 13 show some of the regions in such a plot. Each leakage is plotted in a different line–style, with the power leakage being a solid line and the 3 EM leakages plotted in different broken–line styles (188MHz with a dotted line, 224.5MHz with a dashed line and 262MHz with alternate dot and dashes). It is clear from these figures that even though the signals fall into two gross classes at the macro level, there are significant differences even between signals within a class at a cycle level (see Figure 11). Moreover, there are leakages which appear in EM signals (and sometimes excessively so), which do not appear in the power signal (see Figure 12). Such leakages are due to what we will later term as a "bad" instruction. There are also leakages which are large in power, but low in some (but not all) EM signals (see Figure 13).

5 The Power of the EM Side–Channel(s)

Using low–cost EM equipment, which can collect only one signal at time, we have experimented with a wide variety of cryptographic equipment and computing peripherals. We could easily launch attacks such as simple and differential electromagnetic attacks (SEMA and DEMA [9]) on straight–forward implementations of DES, RSA and COMP128 on smart cards, cryptographic tokens and SSL accelerators. While these attacks are interesting, this does not justify why EM side–channel(s) should be used in preference to others. In some cases, e.g., attacking an SSL accelerator from a distance, the only strong side–channel available is EM. We now show that the EM side–channel is extremely useful even in cases where the power side–channel is available, i.e., the EM side channel can be used to break power analysis resistant implementations.

In [8], a suggested countermeasure to power analysis is to use only those instructions whose power leakage is not excessive and to refresh sensitive information, such as a key, after each invocation in a non–linear fashion. This forces the adversary to extract a substantial portion of the key from a *single invocation* since incomplete key information does not help in subsequent invocations. Another class of countermeasures is based on splitting all sensitive information

into shares [2,4]. The basic idea is that uncertainty in the information about each share is exponentially magnified in proportion to the number of shares.

5.1 Bad Instructions Defeat Power Analysis Countermeasures

The key to breaking both classes of countermeasures is to identify instructions, that we term *bad* instructions, which leak much more information in some EM signals as compared to the power signal. If bad instructions are used in power–analysis resistant implementations, the leakage assumptions made the implementation become invalid.

For all chip cards that we examined, there were several bad instructions. In our investigations, we did not find any instruction that leaked in the power side–channel but did not leak in some EM side–channel. This can happen if all critical parts of a chipcard are well–shielded but the power signal is not. We feel that this is unlikely since a designer who shields EM emanations so well is also likely to protect against power signal leakages.

For example, the bit test instruction is very useful for implementing algorithms, such as DES, which involve bit level permutations. For example, it can be used for key expansion and P–permutation. The value of the tested bit is known to have low power leakage characteristics on many smart cards. This is because the power signal is dominated by the larger currents needed to drive bus lines as opposed to the smaller currents within a CPU performing a bit–test. Thus, it is likely to be present in some power analysis resistant implementations.

However, this bit test instruction turned out to be a bad instruction for smartcard B. When the internal noise generators had been turned off, we observed that it leaked information about the tested bit from even a *single* signal sample in the EM side–channel but not in the power side–channel. This is illustrated in Figures 14 and 15 where the amplitudes of two EM signals are plotted against time (in 10ns units). In both figures, the data was collected by a 12–bit, 100MHz scope after demodulating at the 262MHz carrier. Figure 14 shows two EM signals in which the bits tested are both 0: this is seen as a low value in both the signals at the point 18915. Figure 15 shows two EM signals in which one of the bits tested is 0 and the other is 1: this is seen as a low value in one of the signals and a high value in the other at point 18780. These points correspond to the cycle where the value of the bit is tested.

Even with noise generators enabled, it was possible to classify the bit value correctly with high probability by using only a few samples (20–30). We experimentally verified that no such differences were to be found at the corresponding cycle for the power signals. Even after statistical tests involving thousands of power samples, there are no differences at this cycle although they show up at other cycles (such as the point where the byte containing the bit is loaded).

If the bit test instruction was used for implementing permutations in a power analysis resistant implementation of DES, with noise generators off, a SEMA attack would be sufficient to extract the DES key regardless of which class of countermeasures [8,2,4] was used. However, if noise was enabled, then the countermeasure of [8] may still remain immune. However, as we now show, higher order statistical attacks would still defeat the countermeasures of [2,4].

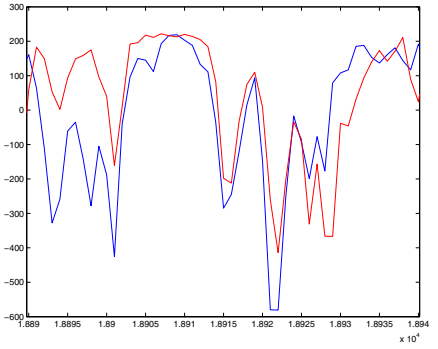


Fig. 14. Two EM Signals where tested bits are 0 (seen as low values at 18915)

Higher Order EM Attacks on Secret–Sharing. The secret–sharing based DPA countermeasure chooses a value for the number of shares based on leakage characteristics and the desired level of resistance against higher order power analysis attacks [2,4], in terms of the number of samples required to break the implementation. If a leakage is superior in an EM signal, then the number of samples for the corresponding higher order EM attack can be substantially lower. The task of an adversary attempting this higher order attack may be complicated by the fact that the code could be unknown. We now outline a general technique to perform higher order EM attacks exploiting bad instructions which can work even when the code is unknown.

Attacks on Unknown Code. Assume a chipcard containing an unknown k–way secret–sharing based DPA protected code for a known algorithm. Further assume that "bad" instructions have already been identified and some of these instructions are used to manipulate shares. These, of course, are necessary conditions for EM attacks to be more effective than power attacks. Let us also assume that it is possible to use signal processing to remove execution sequence and variable clock randomization that has been added as countermeasures to complicate alignment of signals and that each signal can be realigned into a canonical execution sequence[10].

The value of k is usually small. For simplicity, assume that k is 2: the attack generalizes for slightly larger k. Fix a reasonable limit L on the number of EM samples that can be collected. We now show that if k is small and if with knowledge of the code we could have broken the protected code using L samples, then this attack can break the unknown protected code with $O(L)$ samples.

In case of a two–way split, a first step is to identify the two locations where the shares of an algorithmic quantity are being manipulated using bad instructions. If code–execution randomization can be effectively neutralized, then this can be done for many algorithms. Knowing the algorithm, one can provide two different inputs such that the value of the variable is different for these inputs while most

[10] We have found this to be quite feasible, especially since canonicalization with a reasonable probability of correctness suffices.

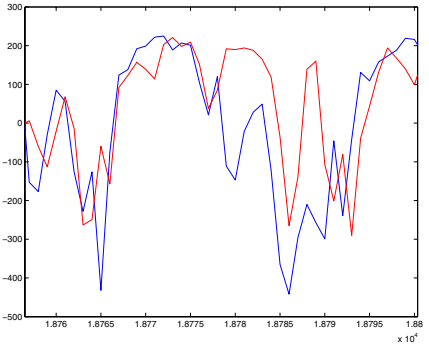


Fig. 15. Two EM Signals where tested bits are 0 and 1 (low and high values at 18780)

of the other variables are the same within the window of interest. For example, in DES if algorithmic quantity is an S–box output, one could choose two inputs which differ only on 1 bit so that only that S–box output is affected.

Take L EM samples for each of these two different inputs. If the exact locations where the two shares were manipulated was known, then there is second order statistic, S, that can be applied to the signal at these two locations to distinguish between the two different inputs, thus enabling hypothesis testing.

Without location information, one can only assume that the two locations are an integral number, D, of clock cycles apart. So the strategy is to compute the statistic S for each point on the signal with respect to a corresponding point D cycles away. This is done for both sets of inputs for all reasonable values of D. If the shares are not manipulated at distance D, then the values of the statistic S at all points will be similar for the two inputs. However, for the right value of D, there will be a significant difference in S exactly at the point where the first share is manipulated and thus the exact location of each share is revealed.

An optimization is to choose the two inputs so that multiple algorithmic variables are different. Then the above exercise will yield candidate locations for the shares for all these variables. Once these locations are identified, second (or higher) order attacks can be applied as if the code were known.

To validate this approach, we implemented a two–way XOR–based secret sharing scheme for bits on smartcard B with noise generators on. The sample code split the input bits into pairs of shares and tested the values of the bits of the shares using the bit test instruction. We confirmed that DPA and DEMA on input bits did not work. In the implementation, the shares of one of the input bits were tested 40 cycles apart. Section 5.1 shows that when a bit is 1, the signal at the bit test instruction is high and when the bit is 0, the signal is low. For a 2–way bit split using an XOR–scheme, the shares of a 0 bit will be (0, 0) or (1, 1) with equal probability and the shares of a 1 bit would be (0, 1) or (1, 0) with equal probability. This suggests that a good statistic S is the correlation coefficient between the corresponding signal points where the shares of bits are being tested. S will be positive when the bit is 0 and negative when the bit is 1.

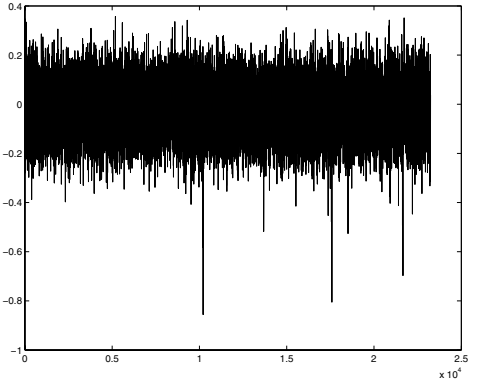


Fig. 16. Difference in correlation statistics for $D = 40$, $L = 500$

We experimented with $L = 500$, for two different inputs, which differed in exactly three bits. Figure 16 shows the difference in the statistic S when the distance D is 40, plotted against elapsed time in 10ns units. The three significant negative peaks were confirmed to be at exactly the points where the first shares of the three bits (that differ) were being manipulated. In fact this attack even worked when $L = 200$. No peaks were seen when D differed from 40. The same experiment when repeated for $D = 40$ for five thousand power signals did not work showing that higher order DPA does not work with five thousand signals.

6 Conclusion and Further Work

This paper, together with other recent work [9,6,1], lays the foundations for a theory of EM leakages during computation in CMOS devices. While a significant amount of information had been publicly available on EM leakages, that work mostly dealt with leakages from displays and other peripherals[10].

Our paper highlights a key aspect of the nature of EM leakage, i.e., the presence of multiple, unintentional, information–bearing signals within this side–channel. In addition, this paper also demonstrates why EM side–channel(s) are so useful: multiple signals with differing leakage characteristics enable a variety of attacks, including attacks against implementations secure against power analysis.

Despite their effectiveness, the single–channel attacks described in this paper provide only a glimpse of what is possible. Combining leakages from multiple EM channels using techniques from Signal Detection and Estimation Theory yield substantially stronger attacks. The existence of such multi-channel attacks highlights a pressing need for models and techniques to assess the net information leakage from all the EM signals realistically available to an adversary. Preliminary results on these aspects of the EM side–channel(s) is described in more detail in [1].

7 Countermeasures

Due to the presence of several unexpected EM leakages, a comprehensive EM vulnerability assessment has to be an integral part of any effort to develop countermeasures against EM attacks on specific implementations. Such countermeasures fall into two broad categories: *signal strength reduction* and *signal information reduction*. Techniques for signal strength reduction include circuit redesign to reduce egregious unintentional emanations and the use of shielding and physically secured zones to reduce the strength of compromising signals available to an adversary relative to ambient thermal noise. Techniques for signal information reduction rely on the use of randomization and/or frequent key refreshing within the computation [7,8,2,4] so as to substantially reduce the effectiveness of statistical attacks using the available signals.

Acknowledgments. This paper has greatly benefitted from the advice of anonymous CHES referees whose comments helped in selecting aspects of our work on the EM side–channel to create a more focussed paper. We would like to thank Helmut Scherzer for key components that enabled this work, in particular, his help with experimental setup, smart card programming and data collection and analysis tools. We would also like to thank Elaine and Charles Palmer for their encouragement and useful comments.

References

1. D. Agrawal, B. Archambeault, J. R. Rao, and P. Rohatgi, EM Side–Channel(s): Attacks and Assessment Methodologies, http://www.research.ibm.com/intsec.
2. S. Chari, C. S. Jutla, J. R. Rao and P. Rohatgi. Towards Sound Countermeasures to Counteract Power–Analysis Attacks. Proc CRYPTO '99, LNCS 1666, pp 398–412.
3. S. Chari, J. R. Rao and P. Rohatgi. Template Attacks, Proc CHES '02.
4. L. Goubin and J. Patarin. DES and Differential Power Analysis. Proc CHES '99, LNCS 1717, pp 158–172.
5. NSA Tempest Series `http://cryptome.org/#NSA--TS`.
6. K. Gandolfi, C. Mourtel and F. Olivier. Electromagnetic Attacks: Concrete Results. Proc CHES '01, LNCS 2162, pp 251–261.
7. P. Kocher. Timing Attacks on Implementations of Diffie-Hellman, RSA, DSS and Other Systems. Proc CRYPTO '96, LNCS 1109, pp 104–113.
8. P. Kocher, J. Jaffe and B. Jun. Differential Power Analysis: Leaking Secrets. Proc CRYPTO '99, LNCS 1666, pp 388–397.
9. J.–J. Quisquater and D. Samyde. ElectroMagnetic Analysis (EMA): Measures and Counter-Measures for Smart Cards. In Smart Card Programming and Security (E-smart 2001), LNCS 2140, pp. 200–210.
10. The complete unofficial TEMPEST web page, http://www.eskimo.com/~joelm/tempest.html.

Enhanced Montgomery Multiplication

Shay Gueron[1,2]

[1] Department of Mathematics, University of Haifa, Haifa 31905, Israel
[2] Discretix Technologies, Netanya, Israel
shay@math.haifa.ac.il

Abstract. We define the Non Reduced Montgomery Multiplication of order s, of A and B, modulo N (odd integer) by $NRMM^s(A,B,N) = \left(AB + N\left(-ABN^{-1} \pmod{2^s}\right)\right)2^{-s}$. Given an upper bound on A and B, with respect to N, we show how to choose the variable s in a way that guarantees that $NRMM^s(A,B,N) < 2N$.
A few applications are demonstrated, showing the advantage of using $NRMM^s$ with an appropriately chosen s, over the classical Montgomery Multiplication.

1 Introduction

Various public key cryptosystems are based on computing modular arithmetic functions (e.g., modular exponentiation and modular multiplication). Thus, methods for fast computation of such operations, particularly in hardware, are of great importance for practical cryptography. One of the most successful methods known today, whose advantage is mostly appreciated in hardware realization, is the Montgomery multiplication [2]. As such, it is a standard approach in hardware implementations of, for example, RSA, digital signature schemes, Diffie-Hellman key exchange, and elliptic curve cryptography (over $GF(p)$).

Despite its advantage, the Montgomery multiplication method suffers from a drawback: the need to reduce the final result to $[0, N)$. This complicates the related implementation and also creates a security problem.

In this paper, we show how to overcome this drawback by using the Non Reduced Montgomery Multiplication of order s, where s is appropriately chosen according to the application at hand. The described method is a generalization of modifications to the Montgomery algorithm (e.g., [4], [1]).

2 Preliminaries and Definitions

Definition 1 (Non Reduced Montgomery Multiplication of order s ($NRMM^s$)).
Let A, B and s be positive integers, and let N be an odd positive integer. The Non Reduced Montgomery Multiplication of order s, of A, B, modulo N is defined by

$$NRMM^s = NRMM^s(A,B,N) = \frac{AB + N\left(-ABN^{-1} \pmod{2^s}\right)}{2^s} \qquad (1)$$

B.S. Kaliski Jr. et al. (Eds.): CHES 2002, LNCS 2523, pp. 46–56, 2003.

Remarks: By looking at the quantity $AB + N\left(-ABN^{-1} \pmod{2^s}\right)$ modulo 2^s, it is clear that $NRMM^s(A, B, N)$ is an integer. Also, considering $AB + N\left(-ABN^{-1} \pmod{2^s}\right)$ modulo N, we can conclude that $NRMM^s(A, B, N) \pmod{N} = AB2^{-s} \pmod{N}$. On the other hand, even when $A, B < N$, the value of $NRMM^s(A, B, N)$ is not necessarily reduced modulo N (i.e., it is not necessarily in the range $[0, N)$). An example for input $A, B < N$ where $NRMM^s(A, B, N)$ is nonreduced is $NRMM^5(17, 18, 19) = 25 > 19$.

Algorithm 1 below describes the bit-level computation of $NRMM^s(A, B, N)$, for s such that $s \geq 1 + [\log_2 A]$. For this description, we denote the binary representation of a number Q by a string $Q = [Q_{m-1}...Q_1Q_0]$ of m bits, where $m = 1 + [\log_2 Q]$, and where Q_0 is the least significant bit of Q.

Algorithm 1: (Non Reduced Montgomery Multiplication of order s)
Input: A, B, N (N is odd), s ($s \geq 1 + [\log_2 A]$)
Output: $NRMM^s(A, B, N)$

$S = 0$
For i from 0 to $s - 1$ do
  1.1 $S = S + A_iB$
  1.2 $S = S + S_0N$
  1.3 $S = S/2$
End For
Return S

Note that although by definition, $NRMM^s$ is symmetric in A and B, this is not the case its bit-level implementation, in particular when A and B have a different number of bits in their binary representation.

It is not difficult to see that the output of Algorithm 1 is $NRMM^s(A, B, N)$, as we now explain. Steps 1.1, 1.3 compute (bit by bit) the quantity $AB2^{-s}$, where step 1.2 is a "parity correction": the odd number N is conditionally added to S whenever it is odd and cannot be straightforwardly divided by 2 (in the subsequent step 1.3). Since $s \geq 1 + [\log_2 A]$, all the bits of A are scanned. Thus, Algorithm 1 actually computes $(AB + KN)/2^s$ for some K. The number K is constructed (bit by bit, in step 1.2) in a way that $2^s|(AB + KN)$. Since K has altogether s bits, $K < 2^s$. The only value of $K < 2^s$ satisfying $AB + KN \equiv 0 \pmod{2^s}$ is $K = -ABN^{-1} \pmod{2^s}$.

Relation to the classical Montgomery multiplication. The classical Montgomery multiplication of A and B modulo N, under the condition $A, B < N$, is defined as $MMUL(A, B, N) = AB2^{-n} \pmod{N}$, where $n = 1 + [\log_2 N]$. Thus, $MMUL(A, B, N)$ can be viewed as the special case of $NRMM^s(A, B, N)$ where $s = 1+[\log_2 N] = n$, but *with a reduced result*, that is: $MMUL(A, B, N) = NRMM^n(A, B, N) \pmod{N}$.

Observing Algorithm 1 for $A, B < N$, it can be shown that $S < 2N$ after step 1.3. The proof is by induction. Starting with $S < 2N$ before step 1.1 (the algorithm is initialized with $S = 0$), S can increase in steps 1.1-1.3 up to $(S +$

$B+N)/2 < (2N+N+N)/2 = 2N$. Since $S < 2N$ in any step of Algorithm 1, we can deduce a bit level algorithm for computing $MMUL(A, B, N)$ (which reduces to the classical algorithm for computing the Montgomery Multiplication): it is the same as Algorithm 1 with the particular choice $s = n$, followed by a single reduction step ($if\ S > N\ then\ S = S - N$) after the end of the For-loop. Algorithm 2 describes this procedure.

Algorithm 2: (Classical Montgomery Multiplication)
Input: A, B, N ($A, B < N$, $n = 1 + [\log_2 N]$, N is odd)
Output: $MMUL(A, B, N)$

$S = 0$
For i from 0 to $n - 1$ do
  2.1 $S = S + A_i B$
  2.2 $S = S + S_0 N$
  2.3 $S = S/2$
End For
2.4 if $S > N$ then $S = S - N$
Return S

Note that unlike Algorithm 1, Algorithm 2 is symmetric in A and B.

The main advantage in using the Montgomery multiplication lies in the hardware implementation of Algorithm 2: it requires only additions and divisions by 2. More important is the special property of step 2.2: only the least significant bit of the cumulative result (S) must be known in order to determine whether or not N should be added, and to carry on to the next step. In fact, this is the key feature that makes Montgomery multiplication superior to standard modular multiplication algorithms, and therefore what made it a de-facto standard in cryptographic hardware.

The Montgomery multiplication is especially efficient when a sequence of MMUL operations is used in a row. For example, consider the task of computing modular exponent $A^E \pmod N$ ($A < N$) which can be carried out as follows. The constant $H = 2^{2n} \pmod N$ (where $n = 1 + [\log_2 N]$) is pre-computed. The input A is transformed to what is called the Montgomery domain (or Montgomery base): $A' = A2^n \pmod N = MMUL(A, H, N)$. Then, the square and multiply (or any other) exponentiation algorithm is applied to A' with the following adaptation that accounts for being in the Montgomery domain: Montgomery multiplications are used instead of regular modular multiplications. The end result is transformed back to the real domain by Montgomery multiplication by 1 (since for any X, we have $MMUL(X', 1, N) = X \pmod N$). These steps are summarized in Algorithm 3.

Algorithm 3: (Modular exponentiation via Montgomery Multiplication)
Input: A, E, N (N is odd, $A < N$), $n = 1 + [\log_2 N], m = 1 + [\log_2 E]$, $H = 2^{2n} \pmod N$
Output: $A^E \pmod N$

Initialization: $A' = MMUL(A, H, N) = A2^n \pmod N$; $T_{m-1} = A'$
$S = 0$

For i from $m-2$ to 0 do
3.1 $T_i = MMUL(T_{i+1}, T_{i+1}, N)$
3.2 if $E_i = 1$ then $T_i = MMUL(T_i, A', N)$
End For
3.3 $T_0 = MMUL(T_0, 1, N)$
Return T_0

The advantage of Algorithm 3 is that expensive modular multiplications in the real domain are replaced by the analogous cheap (hardware wise) Montgomery multiplications in the Montgomery domain. The overhead of the transformation from and to the real domain is insubstantial: the cost is only two additional Montgomery multiplications (by H at the beginning and by 1 at the end).

It is important to note that the reduction step 2.4 is crucial. Without reduction, the output of one $MMUL$ operation is not necessarily a legal input to the subsequent $MMUL$.

Unfortunately, the need for reduction in each $MMUL$ operation substantially complicates hardware realization of Algorithm 2. Dedicated circuitry is required for detecting the cases where the result is greater than N, and for performing the appropriate subtraction. Furthermore, this conditional subtraction exposes the implementation to side channel attacks. Consequently, although the Montgomery multiplication enables efficient hardware implementation of modular arithmetic operations, a method that does not require repeated reductions can be a significant improvement. We show here that the use of $NRMM^s$, together with a proper choice of the parameter s, is a proper solution.

3 The Appropriate Choice of s in the $NRMM^s$ Algorithm

The following Lemma shows that the output of Algorithm 1 can be made smaller than $2N$ for input A and B of any size, provided that the parameter s is properly chosen. It also determines the required size of the accumulator S, when Algorithm 1 is used.

Lemma 1.
Let N be a given positive odd number, and $n = 1 + [\log_2 N]$. Let A, and B be nonnegative integers such that $A \leq 2^k N$ and $B \leq 2^k N$, for some $k \geq 0$. Let s be a positive integer such that $s \geq n + 2k$. Then,
a) $NRMM^s(A, B, N) < 2N$.
b) $s = n + 2k$ is the smallest integer that guarantees that $NRMM^s(A, B, N) < 2N$.
c) $NRMM^s(A, 1, N) \leq N$ with equality only if A is a multiple of N.
d) When applying Algorithm 1, the accumulator S is bounded by $2(1 + 2^k)N$. Thus, the accumulator should be capable of storing $n + k + 2$ bits.

Proof.
(a) Take $s \geq n + 2k$. By definition, $N < 2^n$. Using this bound, we have

$$NRMM^s(A, B, N) = \frac{AB + N\left(-ABN^{-1} \pmod{2^s}\right)}{2^s} \leq$$

$$\frac{2^{2k}N^2 + N(2^s - 1)}{2^{n+2k}} = N\left(\frac{N}{2^n} + \frac{2^s - 1}{2^s}\right) < 2N \tag{2}$$

(b) To show that no smaller s can be chosen, consider $N = 2^n - 1$, $A = B = 2^k N$ and $s = n + 2k - 1$. In this case $-2^{2k}N \pmod{2^s} = (2^{2k} - 2^{n+2k}) \pmod{2^s} = 2^{2k}$. Therefore,

$$NRMM^s(2^k N, 2^k N, N) = \frac{2^{2k}N^2 + N\left(-2^{2k}N \pmod{2^s}\right)}{2^s} =$$

$$\frac{2^{2k}N^2 + N2^{2k}}{2^s} = \frac{2^{2k}N(N+1)}{2^s} = 2N \tag{3}$$

(c) We have

$$NRMM^s(A, 1, N) \leq \frac{2^k N + N(2^s - 1)}{2^s} =$$

$$N + \frac{N}{2^n} \cdot \frac{1}{2^k}(1 - \frac{1}{2^k}) < N + 1 \tag{4}$$

It follows that $NRMM^s(A, 1, N)$ can be at most N.
Since N is odd and $NRMM^s(A, 1, N) \equiv A2^{-s} \pmod{N}$ equality to N can occur only if N divides A.

(d) Suppose that $S < (1 + 2^k)N$ before step 1.1 (note that S is initialized to 0). Then, after step 1.1 we have $S < (1 + 2^k)N + B \leq (1 + 2^k)N + 2^k N$. After step 1.2 we have $S < ((1 + 2^k)N + 2^k N) + N = 2(1 + 2^k)N$, and after step 1.3, we have again $S < (1 + 2^k)N$. Therefore, the maximal possible value for S is $2(1 + 2^k)N$, which can be stored in $n + k + 2$ bits.

Before demonstrating applications of Lemma 1, with different values of $k \geq 1$ and appropriate s, we start with the value $k = 0$. This choice actually deals with the case $A, B < N$ (we ignore equality to N) and $s = n$, i.e., it reduces to the parameters range of the classical Montgomery multiplication. Here, Lemma 1 gives another explanation to the fact that in the computation of $MMUL(A, B)$ (Algorithm 2), a single reduction step (2.4) is sufficient to reduce S to the range $[0, N)$.

Modular Exponentiation via the $NRMM^s$ Method

Taking $k = 1$ in Lemma 1, shows that modular exponentiation can be carried out by choosing $s = n + 2$. We start with Algorithm 4 to describe the procedure.

Algorithm 4: (Modular Exponentiation via the $NRMM^s$ method)
Input: A, E, N ($A < N$, N is odd), $n = 1 + [\log_2 N]$, $m = 1 + [\log_2 E]$, $s =$

$n + 2 = 3 + [\log_2 N]$, $H = 2^{2s} \pmod N$
Output: $A^E \pmod N$
Initialization: $A' = NRMM^s(A, H, N)$; $T_{m-1} = A'$
$S = 0$
For i from $m - 2$ to 0 do
4.1 $T_i = NRMM^s(T_{i+1}, T_{i+1}, N)$
4.2 if $E_i = 1$ then $T_i = NRMM^s(T_i, A', N)$
End For
4.3 $T_0 = NRMM^s(T_0, 1, N)$
Return T_0

The correctness of Algorithm 4 follows from Lemma 1 (using $k = 1$). We show that all inputs to $NRMM^s$ are legal, that is less than $2N$. Since $A < N < 2N$, it is a legal input to $NRMM^s$ (that is, to the initialization step of Algorithm 4), and the resulting A' satisfies $A' < 2N$. Consequently, A' is also a legal input to $NRMM^s$ (step 4.2). For the same reason, $T_{m-1} < 2N$, implying that $T_i < 2N$ for all i, and thus T_i is a legal input as well. Finally, after the transformation back to the real domain (step 4.3) $T_0 < N$. We remark that technically, we have only $T_0 \leq N$, but in practice, the inequality is strict. The reason is that we typically choose N to be either a prime or the product of two primes. Since $A < N$ to begin with T_0 cannot be a multiple of N in any step of Algorithm 4, and thus, we end up with $T_0 < N$.

Example. The following example demonstrates the use of the Algorithm 4, shows that nonreduced intermediate results indeed appear in the calculations, and shows that the bound on the maximal bit-length of S is tight.

We compute here $A^E \pmod N$ with $A = 212$, $E = 240$ $(= [11110000]_2)$, $N = 249$, namely $212^{240} \pmod{249} = 241$.
Here, $m = 1 + [\log_2 E] = 8$ and $n = 1 + [\log_2 N] = 8$. We use $s = n + 2 = 10$. $H = 2^{2s} \pmod N = 2^{20} \pmod{249} = 37$. Pre-computation gives $A' = NRMM^s(A, H, N) = 209$ $(= T_7$ in Algorithm 4). The steps of the square and multiply algorithm are given in Table 1. Column 1 shows the index i (in Algorithm 4) running from $i = m-2 = 6$ down to $i = 0$. Column 2 shows the relevant bit (E_i) of the exponent. Columns 3, and 4 show the inputs to steps 4.1, 4.2, and column 5 holds the result after the square and (conditional) multiply are performed. This value is fed into the subsequent step.

The final exponentiation result is obtained (step 4.3) by back transforming T_0 to the real domain, i.e., by computing $T_0 = NRMM^s(T_0, 1, N) = NRMM^s(25, 1, 249) = 241$.

Note that some results of the intermediate $NRMM^s$ steps are not reduced: step 4.2 in iterations $i = 6$, 5, and 4, and step 4.1 in iteration $i = 3$. However, these nonreduced results (output of $NRMM^s$) never exceed $2N$ $(= 498$ in our case), and are therefore legal input to a subsequent $NRMM^s$ operation.

Table 2 shows the details of the calculation of $NRMM^s$ in iteration $i = 3$ of Table 1, namely the calculation of $NRMM^{10}(319, 319, 249)$.

Table 1. Computing $212^{240} \pmod{249}$. Details are given in the text

i	E_i	T_{i+1}	T_{i+1}	T_i
6	1	209	235	269
5	1	269	121	254
4	1	254	241	296
3	0	296	319	319
2	0	319	175	175
1	0	175	160	160
0	0	160	25	25

Table 2. Computing $NRMM^{10}(319, 319, 249)$. Details are given in the text

i	A_i	$S = S + A_iB$	S_0	$S = S + S_0N$	$S = S/2$
0	1	319	1	568	284
1	1	603	1	852	426
2	1	745	1	994	497
3	1	816	0	816	408
4	1	727	1	976	488
5	1	807	1	1056	528
6	0	528	0	528	264
7	0	264	0	264	132
8	1	451	1	700	350
9	0	350	0	350	175

Here, $A = B = 319 = [100111111]_2$. The result, 175, is reduced in this case. As Lemma 1 predicts indeed, $S < 6N = 1494$ in all steps. However, note that in step $i = 5$, $S = S + S_0N = 1056 > 2^{10}$. This shows that the bound given in Lemma 1 (d) is sharp: for $k = 1$, the accumulator must hold $n + 3$ bits ($= 11$ in this example), and this number of bits cannot be decreased.

We point out that attempting to use $s < n + 2$ in the exponentiation process may fail. With the input under consideration, using $s = n + 1 = 9$ gives the correct result, but if we choose $s = n = 8$, we get 296428706871896110475206. Modulo N, this number is, of course, 241, and this overflow illustrates exactly the danger of choosing an inappropriate s.

For the sake of completeness, we give an example where the choice $s = n + 1$ is not sufficient to assure that the output is bounded by $2N$. Consider $N = 255$, $A = 505$, $B = 507$, $n = 1 + [\log_2 255] = 8$, $s = n + 1 = 9$. Computation shows that $NRMM^s(A, B, N) = 645 > 2N(= 510)$.

4 Choosing s for a Chain of Operations in $\mathbb{Z}_N$

We have already showed that exponentiation requires the choice of $s = n+2$. The reason for the sufficiency of this relatively small value is that the "square and

multiply" exponentiation algorithm involves only multiplications (and squaring), and no other operations in $\mathbb{Z}_N$. When other operations are involved, a higher value of s may be necessary. The following proposition shows that there always exists a suitable value of s, with which the operations can be performed in the Montgomery domain, using $NRMM^s$ instead of modular multiplication.

Proposition 1.
Let N be a given positive odd number, and $n = 1 + [\log_2 N]$. Let $\{A_j, j = 1, 2, \ldots, r\}$ be a set of r inputs (positive integers), such that for all j, $A_j \leq 2^d N$ for some $d \geq 0$.
Suppose that a sequence of operations in $\mathbb{Z}_N$ (modular addition, modular subtraction, modular multiplication) is to be performed, where the inputs to each operation are either taken from $\{A_j\}$, or are the results of a previous operation in the sequence.

Then, there exists a value of s with which the computation can be performed in the following way:
1. Convert the inputs to the Montgomery domain, i.e., set
$A'_j = NRMM^s(A_j, H, N)$, $j = 1, 2, \ldots, r$, where $H = 2^{2s} \pmod N$.
2. Replace each modular multiplication in the sequence by an $NRMM^s$ operation. Replace each modular addition by regular addition. Replace each modular subtraction, $X - Y$, by $2^l N + X - Y$ for l such that $2^l N + X - Y \geq 0$.
3. Perform the modified sequence of operations, using the converted inputs.
4. Convert the desired result, R, to the real domain by $NRMM^s(R, 1, N)$.

Proof. We start with some $s \geq n + 2d$, baring in mind that the final value of s is yet to be determined. Suppose that all the inputs are already converted to the Montgomery domain by the operation $A'_j = NRMM^s(A_j, H, N)$. Note that the choice $s \geq n + 2d$ guarantees that $A'_j \leq 2N$.

We now replace each modular addition and modular subtraction operation by plain addition (ADD) and plain subtraction (SUB). At this point we note that negative results may appear as a result of subtraction. Thus, to avoid dealing with negative integers, we modify all the SUB operations in the following way: instead of $SUB(X, Y) = X - Y$ where the inputs are bounded by $X, Y \leq 2^l N$ for some l, we use $SUBlN(X, Y) = 2^l N + X - Y = SUB(ADD(2^l N, X), Y)$. In this case, $0 \leq SUBlN(X, Y) \leq 2^{l+1} N$. Of course, the result remains unchanged modulo N.

To assess the upper bound of the intermediate addition results, note that if $X, Y \leq 2^l N$ then $ADD(X, Y) = X + Y \leq 2^{l+1} N$.

We now show how to determine an appropriate value of s. Consider the modified sequence of operations, with the additional operation $NRMM(\cdot, 1, N)$ appended to the end of the sequence (used for transforming the result back to the real domain). Suppose that the i-th operation is $NRMM^s(X_i, Y_i, N)$ where the input satisfies $X_i, Y_i \leq 2^{k_i} N$ for some k_i. In general, $k_i \geq d$ because the input can involve intermediate results obtained by previous additions/subtractions. Any choice of $s \geq n + 2k_i$ is appropriate at this point: by Lemma 1, this assures that the input to the $NRMM^s$ is legal, and also assures that the output is bounded by $2N$.

It now follows that we can set $s = n + 2(\max_i k_i)$, where the maximum is taken over all occurrences of $NRMM^s$ in the sequence. This value is appropriate throughout the sequence of operations. In particular, it assures that the returned result is either reduced or equals N. We finally comment that the bound on s, computed here, is not necessarily tight, and that there may be other ways to work out a chain of operations.

The following example shows how Proposition 1 can be used for point doubling on an elliptic curve over $GF(p)$.

Example: point doubling on an elliptic curve over $GF(p)$. Consider the elliptic curve $y^2 = x^3 + ax + b$ over $GF(p)$, where p is an odd prime. Suppose that the curve is represented in projective coordinates. If the point to double is (x_1, y_1, z_1), and we denote $2(x_1, y_1, z_1) = (x_2, y_2, z_2)$ we have

$$\lambda_1 = 3x_1^2 + az_1^4, \quad z_2 = 2y_1z_1, \quad \lambda_2 = 4x_1y_1^2,$$

$$x_2 = \lambda_1^2 - 2\lambda_2, \quad \lambda_3 = 8y_1^4, \quad y_2 = \lambda_1(\lambda_2 - x_2) - \lambda_3.$$

This computation can be implemented in several ways. One method is the following sequence of 18 steps consisting of 5 ADD, 3 $SUBlp$ (with $l = 1, 2$ or 3), and 10 $NRMM^s$ operations.

In cryptographic applications, point doubling is part of a scalar multiplication process which involves repeated instances of point addition and point doubling. In this context, conversion to the Montgomery domain is performed only at the beginning, and conversion back to the real domain is performed only at the end of the computations. For brevity, we suppose here that the input x_1, y_1, z_1 and a are already given in the Montgomery domain, and in the first time doubling is performed, they are bounded by $2p$. The result we return is also given in the same Montgomery domain (we do not transform back to the real domain). The output of the doubling procedure satisfies $x_2 < 8p$, $y_2 < 4p$ and $z_2 < 2p$. As explained below, this assures that this result can be reused as input for a subsequent doubling (or addition) procedure. The sequence of operations (in the Montgomery domain) is the following:

Compute the Montgomery base analog of $\lambda_1 = az_1^4 + 3x_1^2$
1. $R0 = NRMM^s(z_1, z_1, p)$
2. $R0 = NRMM^s(R0, R0, p)$
3. $R0 = NRMM^s(a, R0, p)$
4. $T1 = NRMM^s(x_1, x_1, p)$
5. $T2 = ADD(T1, T1)$
6. $T1 = ADD(T2, T1)$
7. $T1 = ADD(R0, T1)$

Compute the Montgomery base analog of $z_2 = 2y_1z_1$
8. $T2 = ADD(y_1, y_1)$
9. $z_2 = NRMM^s(T2, z_1, p)$

Compute the Montgomery base analog of $\lambda_2 = 4x_1y_1^2$

10. $R0 = NRMM^s(T2, T2, p)$
11. $T2 = NRMM^s(x_1, R0, p)$

Compute the Montgomery base analog of $x_2 = \lambda_1^2 - 2\lambda_2$

12. $T3 = NRMM^s(T1, T1, p)$
13. $T4 = ADD(T2, T2)$
14. $x_2 = SUB2p(T3, T4, p)$

Compute the Montgomery base analog of $\lambda_3 = 8y_1^4$

15. $T3 = NRMM^{s+1}(R0, R0, p)$

Compute the Montgomery base analog of $y_2 = \lambda_1(\lambda_2 - x_2) - \lambda_3$

16. $T2 = SUB3p(T2, x_2, p)$
17. $T1 = NRMM^s(T2, T1, p)$
18. $y_2 = SUB1p(T1, T3, p)$

To determine the required value of s, we look at consecutive $ADD/SUBlp$ operations, and set s to be high enough to assure that with the relevant input, the output of $NRMM^s$ is bounded by $2p$.

After step 5 we have $T2 < 4p$, after step 6 $T1 < 6p$, and after step 7 $T1 < 8p$. In step 8 we have $T2 < 4p$, and thus for the $NRMM^s$ in step 9 we have $k_1 = 2$. The output z_2 is bounded by $2p$. Steps 10 and 11 require $k_2 = 1$. Step 12 requires $k_3 = 3$. In step 13 we have $T4 < 4p$, and this is why $SUB2p$ is used in step 14. Note that $x_2 = SUB2p(T3, T4, p) < 6p < 8p$. In step 15 we square $(4y_1^2)^2 = 16y_1^2$ and divide by 2 (to get $8y_1^2$) by using $NRMM^{s+1}$. In step 15 we have $T2 < 2p$ (from step 11) and $x_2 < 8p$, and this is why we use $SUB3p$ in step 16. Therefore, for the $NRMM^s$ operation in step 17 we use $k_4 = 4$. Finally, in step 18 we compute the output y_2 which is bounded by $4p$.

Consequently, the value $s = n + 2 \max k_i = n + 8$, where $n = 1 + [\log_2 p]$, is an appropriate choice.

In the above procedure we assure that $x_2 < 8p$, $y_2 < 4p$ and $z_2 < 2p$. With our choice of s, these values are proper inputs for a subsequent point doubling (i.e., can the roles of x_1, y_1, z_1). This can be verified by noting that x_1, z_1 are used in $NRMM^s$ operation (steps 1, 4, 11), and y_1 is used in step 8, in an ADD operation. This operation yields $T2$, which is then a legal input to the $NRMM^s$ operation in step 9. Although we do not give here the details we point out that $x_2 < 8p$, $y_2 < 4p$ and $z_2 < 2p$ are legal inputs to the procedure implementing point addition as well.

5 Discussion

Proper choice of the parameter s enables using the $NRMM^s$ in a variety of applications and contexts, while keeping modular reduction not required in any stage. Consequently, using $NRMM^s$ in hardware implementations can save on time and/or hardware requirements. Another important feature of $NRMM^s$ (i.e., of Algorithm 1) is in being homogeneous: its execution does not depend on the input (in contrast with Algorithm 2). Thus, timing attacks as well as other

attacks to which the classical Montgomery multiplication is susceptible [3] do not affect systems using the $NRMM^s$.

We further comment that there may be different ways to implement a given sequence of operations in $\mathbb{Z}_n$, and the flexibility in choosing s may help in optimizing the computations in various contexts.

Finally, we mention that Lemma 1 can be generalized to the situation where $A \leq 2^{\alpha_1}N$, $B \leq 2^{\alpha_2}N$ $\alpha_1, \alpha_2 \geq 0$, and the result of $NRMM^s$ is to be smaller than $2^{\alpha_3}N$ for some $\alpha_3 \geq 1$. In such cases, we need to choose $s = n + \alpha_1 + \alpha_2 + 1 - \alpha_3$. Proof and applications where different values of $\alpha_1, \alpha_2, \alpha_3$ are useful, will be discussed in subsequent publication.

Acknowledgments. I thank Or Zuk for helpful discussions, suggestions, and help in preparing the paper.

References

1. Blum, T., Paar, C.: Montgomery modular exponentiation on reconfigurable hardware. In Proceedings of the 14th symposium on computer arithmetic, (1999) 70–77
2. Montgomery, P. L.: Modular multiplication without trial division. Mathematics of Computation **44** (1985) 519–521
3. Schindler, F.: A timing attack against RSA with Chinese Remainder Theorem. Lecture Notes Comp. Sci. **1965** (2000) 110–124
4. Walter, C. D.: Montgomery exponentiation needs no final subtractions. Electronics Letters **35** (1992) 1831–1832

New Algorithm for Classical Modular Inverse

Róbert Lórencz

Department of Computer Science and Engineering
Faculty of Electrical Engineering, Czech Technical University
Karlovo nám. 13, 121 35 Praha 2, Czech Republic
lorencz@fel.cvut.cz

Abstract. The Montgomery inverse is used in cryptography for the computation of modular inverse of b modulo a, where a is a prime. We analyse existing algorithms from the point of view of their hardware implementation. We propose a new, hardware-optimal algorithm for the calculation of the classical modular inverse. The left-shift binary algorithm is shown to naturally calculate the classical modular inverse in fewer operations than the algorithm derived from the Montgomery inverse.

1 Introduction

The basic arithmetic operations in modular arithmetic where the modulo is prime are a natural and inseparable part of cryptographic algorithms [6], [8], as well as nowadays often used elliptic curve cryptography [9], [10]. Modular inverse is especially important in computations of point operations on elliptic curves defined over a finite field $GF(p)$ [9], in acceleration of the exponentiation operation using the so-called addition-subtraction chain [11], [4], in Diffie-Hellman key exchange method [7], and in decipherment operations in RSA algorithm [6]. The modular inverse of an integer $a \in [1, p-1]$ modulo p, where p is prime, is defined as an integer $r \in [1, p-1]$ such that $a.r \equiv 1 \pmod{p}$, often written as

$$r = a^{-1} \bmod p. \tag{1}$$

This classical definition of the modular inverse and an algorithm for its calculation in a binary form is specified in [4]. Kaliski has extended the definition of the modular inverse to include the so-called Montgomery inverse [2]. The Montgomery inverse is based on the Montgomery multiplication algorithm [1]. The Montgomery inverse of an integer $a \in [1, p-1]$ is b such that

$$b = a^{-1}2^n \bmod p, \tag{2}$$

where p is prime and $n = \lceil \log_2 p \rceil$. In this paper, we present a left-shift binary algorithm for the computation of the classical modular inverse which is more efficient than the algorithm derived from the Montgomery modular inverse algorithm [2], [3] and the ordinary inverse algorithm [4].
Our incentive for the search of effective computation of modular inverse was,

B.S. Kaliski Jr. et al. (Eds.): CHES 2002, LNCS 2523, pp. 57–70, 2003.

besides the above facts, to use it in a modular system for solving systems of linear equations [16], [17]. In the whole paper, we assume that least significant bit (LSB) is the rightmost position.

2 The Classical Modular Inverse in Previous Works

The two commonly used approaches for the computation of ordinary modular inverse are a binary algorithm derived from the Montgomery modular inverse [2], [3] and a binary algorithm for ordinary inverse [4]. Both of these approaches are based on the same algorithmic principle, which is the binary right-shift greatest common divisor algorithm (gcd) [4] that calculates the value for two integers using halving and subtraction. Both of the mentioned algorithms are suitable for implementation in hardware, since the halving operation is equal to a binary right shift.

2.1 The Right-Shift Algorithm for the Classical Modular Inverse

The ordinary modular inverse algorithm described in [4], attributed to M.Penk (see exercise 4.5.2.39 in [4]), calculates the modular inverse $r = a^{-1} \bmod p$ using the extended Euclidean algorithm. We have modified the Penk's algorithm with the aim to enable its easy hardware implementation. The modified Penk's algorithm, called Algorithm I, is given below:

Algorithm I

Input: $a \in [1, p-1]$ and p
Output: $r \in [1, p-1]$ and k, where $r = a^{-1} \bmod p$
and $n \leq k \leq 2n$

```
1.  u := p, v := a, r := 0, s := 1
2.  k := 0
3.  while (v > 0)
4.      if (u is even) then
5.          if (r is even) then
6.              u := u/2, r := r/2, k := k + 1
7.          else
8.              u := u/2, r := (r + p)/2, k := k + 1
9.      else if (v is even) then
10.         if (s is even) then
11.             v := v/2, s := s/2, k := k + 1
12.         else
13.             v := v/2, s := (s + p)/2, k := k + 1
14.     else x := (u − v)
15.         if (x > 0) then
16.             u := x, r := r − s
17.             if (r < 0) then
18.                 r := r + p
```

19. else
20. $v := -x$, $s := s - r$
21. if $(s < 0)$ then
22. $s := s + p$
23. if $(r > p)$ then
24. $r := r - p$
25. if $(r < 0)$ then
26. $r := r + p$
27. return r and k.

Algorithm I continuously halves (shifts to the right) both values, even and odd; if the value is odd, the modulus p which is odd (p is prime) is added to it beforehand. These operations are performed in steps 8 and 13. Any negative values of r and s that result from the subtraction are converted to positive ones in the same residue class in steps 18 and 22 by adding p so that $r, s \in [1, p-1]$. The algorithm outputs two integers, r and k, where k is the number of halvings during the calculation of $\gcd(p, a)$ and it satisfies $n \leq k \leq 2n$.

2.2 The Montgomery Algorithm for the Classical Modular Inverse

In contrast to Algorithm I, Montgomery algorithms for computing modular inverse (in integer or Montgomery domains) split the computation to two phases. In the first phase the so-called Almost Montgomery Inverse $a^{-1}2^k \bmod p$ [3] is calculated in k iterations, where k follows from input values. In case of integer domain, k is taken to be the number of deferred halvings in the second phase [3]. Hence, the modular inverse according to Equation (1) is computed by k hlavings modulo p. This algorithm, called Algorithm II, is given below:

Algorithm II

Phase I
Input: $a \in [1, p-1]$ and p
Output: $y \in [1, p-1]$ and k, where $y = a^{-1}2^k \pmod p$
and $n \leq k \leq 2n$
1. $u := p, v := a, r := 0, s := 1$
2. $k := 0$
3. while $(v > 0)$
4. if (u is even) then
5. $u := u/2$, $s := 2s$, $k := k + 1$
6. else (if v even) then
7. $v := v/2$, $r := 2r$, $k := k + 1$
8. else
9. $x := (u - v)$
10. if $(x > 0)$ then
11. $u := x/2$, $r := r + s$, $s := 2s$, $k := k + 1$
12. else

13. $v := -x/2$, $s := r + s$, $r := 2r$, $k := k + 1$
14. if $(r > p)$ then
15. $r := r - p$
16. return $y := p - r$ and k.

Phase II
Input: $y \in [1, p-1], p$ and k from Phase I
Output: $r \in [1, p-1]$, where $r = a^{-1} \pmod{p}$, and $2k$ from Phase I
17. for $(i = 1$ to $k)$ do
18. if (r is even) then
19. $r := r/2$
20. else
21. $r := (r + p)/2$
22. return r and $2k$.

In case of Montgomery domain, the number of deferred halvings in the second phase is $k - n$, where k is guaranteed to $n \leq k \leq 2n$ [2]. It is interesting to compare Algorithms I and II. The operation of the correction of an odd number before halving performed in steps 8 and 13 of Algorithm I is done in step 21 of Phase II of Algorithm II. Conversion of negative values of r and s is not necessary here, since no subtraction of r or s is performed during calculation. It is clear that the number of iterations in Algorithm II is $2k$, k iterations in Phase I and k iterations in Phase II.

3 New Left-Shift Algorithm for the Classical Modular Inverse

The new approach to the calculation of modular inverse, which is the subject of this paper, avoids the drawbacks of the above algorithms. In Algorithm I, these are especially: high number of tests such as '$u > v$', '$s < 0$', '$r < 0$', which essentially represent a subtraction and also an addition '$r + p$', '$s + p$' if s and r are negative so that $r, s \in [1, p-1]$. In case of the modular inverse calculation using the Montgomery modular inverse, it is necessary to perform the deferred halving in k iterations in Phase II of Algorithm II, including corrections of r if it is odd (step 21 of the algorithm). An algorithm that avoids the mentioned problems is presented below:

Algorithm III

Input: $a \in [1, p-1]$ and p
Output: $r \in [1, p-1]$, where $r = a^{-1} \pmod{p}$, c_u, c_v
and $0 < c\_v + c\_u \leq 2n$
1. $u := p, v := a, r := 0, s := 1$
2. $c\_u = 0$, $c\_v = 0$
3. while$(u \neq \pm 2^{c_u}$ & $v \neq \pm 2^{c\_v})$
4. if (u_n,u_{n-1}= 0) *or* $(u_n, u_{n-1} = 1$ & OR$(u_{n-2}, \ldots, u_0) = 1)$ then

5. if ($c\_u \geq c\_v$) then
6. $u := 2u$, $r := 2r$, $c\_u := c\_u + 1$
7. else
8. $u := 2u$, $s := s/2$, $c\_u := c\_u + 1$
9. else if ($v_n, v_{n-1} = 0$) *or* ($v_n, v_{n-1} = 1$ & OR$(v_{n-2}, \ldots, v_0) = 1$) then
10. if ($c\_v \geq c\_u$) then
11. $v := 2v$, $s := 2s$, $c\_v := c\_v + 1$
12. else
13. $v := 2v$, $r := r/2$, $c\_v := c\_v + 1$
14. else
15. if ($v_n = u_n$) then
16. *oper* = " $-$ "
17. else
18. *oper* = " $+$ "
19. if ($c\_u \leq c\_v$) then
20. $u := u$ *oper* v, $r := r$ *oper* s
21. else
22. $v := v$ *oper* u, $s := s$ *oper* r
23. if ($v = \pm 2^{c\_v}$) then
24. $r := s$, $u_n := v_n$
25. if ($u_n = 1$) then
26. if ($r < 0$) then
27. $r := -r$
28. else
29. $r := p - r$
30. if ($r < 0$) then
31. $r := r + p$
32. return r, c_u, and c_v.

Algorithm III was designed to be easily implemented in hardware (Section 5). Registers Ru, Rv, Rs are $m = n + 1$ bit wide registers and contain individual values of the variables u, v, s. The value of variable r is in $m+1$ bit wide register Rr. Counters Cu and Cv are auxiliary $e = \lceil \log_2 n \rceil$ bit wide counters containing values c_u and c_v. The presented left-shifting binary algorithm computes the modular inverse of a according to Equation (1)) using the extended Euclidean algorithm and shifting the values u and v to the left, that is multiplying them by two. The multiplication is performed as long as the original value multiplied by 2^i is preserved, where i is the number of left shifts. Negative values are represented in the two's complement code. The shift is performed as long as the bits u_n, u_{n-1} or v_n, v_{n-1} are zeros for positive values or ones for negative values, while at least one of the bits u_{n-2}, u_{n-3}, $\ldots u_0$ or v_{n-2}, v_{n-3}, $\ldots v_0$ is not zero - binary 'OR' (steps 4 and 9). With each shift, counters Cu and Cv (values c_u and c_v) that track the number of shifts in Ru, Rv are incremented (steps 6, 8, 11, and 13). Registers Rr and Rs (values r and s) are also shifted to the right (steps 8 and 13) or left (steps 6 and 11) according to conditions in steps 5 and 10. In step 15, addition or subtraction, given variable *oper*, is selected according

to sign bits u_n and v_n for the subsequent reduction of u, v and r, s in steps 20 and 22. Results of these operations are stored either in Ru and Rr (values u and r), if the number of shifts in Ru is less or equal to the number of shifts in Rv, or in registers Rv and Rs (values v and s) otherwise. The loop ends whenever '1' or '-1' shifted by the appropriate number of bits to the left appears in register Ru or Rv. Branch conditions used in steps 4, 9, and 15 are easily implemented in hardware. Similarly, the test in steps 5, 10, and 19 can be implemented by an e bit comparator of values c_u and c_v with two auxiliary single-bit flips-flops u/$\bar{\text{v}}$ and wu (see Section 5). Table 1 shows an example of the calculation of the

Table 1. Example of the calculation

l	*operations*	*values of registers*	*tests*
0		$u^{(0)} = (13)_{10} = (01010.)_2$	$u^{(0)} \neq \pm 2^0$
		$v^{(0)} = (10)_{10} = (01010.)_2$	$v^{(0)} \neq \pm 2^0$
		$r^{(0)} = (0)_{10} = (00000.)_2$	
		$s^{(0)} = (1)_{10} = (00001.)_2$	
1	$u^{(1)} = u^{(0)} - v^{(0)}$	$u^{(1)} = (3)_{10} = (00011.)_2$	$u^{(1)} \neq \pm 2^0$
		$v^{(1)} = (10)_{10} = (01010.)_2$	$v^{(1)} \neq \pm 2^0$
	$r^{(1)} = r^{(0)} - s^{(0)}$	$r^{(1)} = (-1)_{10} = (11111.)_2$	
		$s^{(1)} = (1)_{10} = (00001.)_2$	
2	$u^{(2)} = 4u^{(1)}$	$u^{(2)} = (12)_{10} = (011.00)_2$	$u^{(2)} \neq \pm 2^2$
		$v^{(2)} = (10)_{10} = (01010.)_2$	$v^{(2)} \neq \pm 2^0$
	$r^{(2)} = 4r^{(1)}$	$r^{(2)} = (-4)_{10} = (111.00)_2$	
		$s^{(2)} = (1)_{10} = (00001.)_2$	
3		$u^{(3)} = (12)_{10} = (011.00)_2$	$u^{(3)} \neq \pm 2^2$
	$v^{(3)} = v^{(2)} - u^{(2)}$	$v^{(3)} = (-2)_{10} = (11110.)_2$	$v^{(3)} \neq \pm 2^0$
		$r^{(3)} = (-4)_{10} = (111.00)_2$	
	$s^{(3)} = s^{(2)} - r^{(2)}$	$s^{(3)} = (5)_{10} = (00101.)_2$	
4		$u^{(4)} = (12)_{10} = (011.00)_2$	$u^{(4)} \neq \pm 2^2$
	$v^{(4)} = 4v^{(3)}$	$v^{(4)} = (-8)_{10} = (110.00)_2$	$v^{(4)} \neq \pm 2^2$
	$r^{(4)} = r^{(3)}/4$	$r^{(4)} = (-1)_{10} = (11111.)_2$	
		$s^{(4)} = (5)_{10} = (00101.)_2$	
5	$u^{(5)} = u^{(4)} + v^{(4)}$	$u^{(5)} = (4)_{10} = (001.00)_2$	$u^{(5)} = 2^2$
	$r^{(5)} = r^{(4)} + s^{(4)}$	$r^{(5)} = (4)_{10} = (00100.)_2$	

modular inverse for $p = 13$ and $a = 10$. Therefore, $n = 4$ and $m = 5$. The computed result is $r = a^{-1} \bmod 13 = 4$.

Description of Table 1: l is the iteration number, column *operations* lists the performed arithmetic operations of iteration l and column *tests* lists conditions evaluated in iteration l . The notation $u^{(l)}$ means the actual value u of register Ru in the l-th iteration, etc. The dot in binary representation of values in column

values of registers specifies the reference shift position that is equal to the current position of initial LSB. It represents the value of accumulated left-shift for u and v and left/right shift for r and s.

4 Results and Discussion

A simulation and a quantitative analysis of the number of additions or subtractions ('+/−'), shifts and tests was performed for all the algorithms presented. Simulation of modular inverse computation according to Equation (1) was performed for all integers $a \in [2, p-1]$ and all 1899 prime moduli $p < 2^{14}$ ($n \leq 14$) . A total of 14,580,841 inverses were computed by each method. Simulation results are presented in Table 2. The number of all tests, additions and subtractions are listed in column "+/− & *tests*". The tests include all "greater than" and "less than" comparisons except '$v > 0$' in the main loop, which is essentially a '$v \neq 0$' test that does not require a subtraction. The "+/−" column lists additions and subtractions without tests. The column "*total shift*" indicates the number of all shift operations during the computation. The last column lists the number of shifts minus the number of '+/−' operations, assuming the shift is performed together with storing the result of the '+/−' operation. The columns give minimum and maximum numbers of operations ($min; max$) and their average ($av.$) values.

Table 2. Results for primes less than 2^{14}

Algorithm	+/− & *tests*		+/−		*total shift*		*shift* − (+/−)	
	min; max	*av.*	*min; max*	*av.*	*min; max*	*av.*	*min; max*	*av.*
Algorithm III	-	-	2 - 21	9.9	2 - 26	23.3	1 - 24	13.4
Algorithm II Ph. I	3 - 28	15.7	1 - 15	10.6	3 - 27	19.1	0 - 23	8.5
Algorithm II Ph. II	2 - 25	10.5	2 - 25	10.5	3 - 27	19.1	0 - 24	8.6
Algorithm II	5 - 45	26.2	4 - 40	21.1	6 - 54	38.2	0 - 43	17.1
Algorithm I	9 - 80	40.4	6 - 53	27.1	2 - 26	18.1	0	0

Shift operations are faster in hardware than additions, subtractions, and comparison operations performed with the Ru, Rv, Rr, Rs registers. The comparison operations are about as slow as additions/subtractions, since they cannot be performed in parallel; they depend on data from previous operations. In Algorithm I, they are the conditions in steps 15, 17, and 21. If a suitable code is used to represent negative numbers, this condition can be realized as a simple sign test, avoiding the complicated testing in steps 17 and 21. The Algorithms I and Algorithm II suffer from a large number of additions and subtractions that correct odd numbers before halving in steps 8 and 13 of Algorithm I and step 21 in Algorithm II, and convert negative numbers in steps 18 and 22 of Algorithm I.

Moreover, Algorithm II needs twice the number of shifts compared to Algorithm I, required by the Phase II.

The previous analysis shows that Algorithm III removes drawbacks of Algorithm I and Algorithm II. Let us assume full hardware support for each algorithm and simultaneous execution of operations specified on the same line of the pseudocodes. Let us further assume that no test is needed in step 10 of Algorithm II. Then, we can use the values in columns "$+/-$" and "$total\ shift$" to compare the number of operations. Algorithm III needs half the number of '$+/-$' operations compared to Algorithm II, and 2.7 times less the number of '$+/-$' operations compared to Algorithm I.

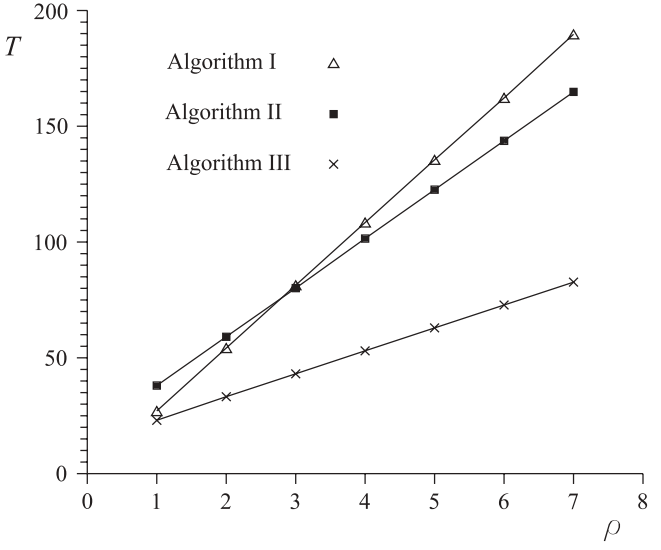


Fig. 1. Average number of execution cycles T of the three algorithms as a function of the ratio ρ

The simulation results from Table 2 for prime moduli less than 2^{14} ($n = 14$) are plotted in Figure 1. It shows the average number of cycles T needed to compute the modular inverse using Algorithms I - III as a function of the ratio ρ, where ρ is defined as the ratio of the critical path length in cycles of the shift and the critical path length in cycles of the adder/subtracter. All (Algorithm I) or a part of (Algorithms II and III) shift operations are included in '$+/-$' operations. Shift operations that are not performed as a part of '$+/-$' operations are performed individually (they are listed in the last column of Table 2).

With an increasing word length, the time complexity of shift operations remains constant. However, the complexity of additions/subtractions increases approximately $\lceil \log_2 m \rceil$ - times, m is the number of bits of a word. For long words,

often used in cryptographic algorithms, the modular inverse computation for individually algorithms is strongly dependent on addition/subtraction operations. That in such cases Algorithm III is twice faster than Algorithm II and 2.7-times faster than Algorithm I.

Table 3. Results of Algorithm III for three cryptographic primes

Primes	n	$+/-$		*total shift*		*inverses*
		min; max	*av.*	*min; max*	*av.*	
$2^{192} - 2^{64} - 1$	192	64 - 182	132.9	343 - 382	380	3,929,880
$2^{224} - 2^{96} + 1$	224	81 - 213	154.8	408 - 446	441	4,782,054
$2^{521} - 1$	521	18 - 472	387.5	999 - 1040	1029	4,311,179

The statistical analysis of Algorithm III (see the previous page) for large integers of cryptographic was performed. The results of the analysis are presented in Table 3. The first column contains values of primes, the second column gives the word length and the last column gives number of inverses. Other columns have the same meaning as columns in Table 2. The average number of '$+/-$' operations grows with n approximately linearly. The multiplicative coefficient is ≈ 0.7 for all three primes. The average number of shifts is nearly equal to $2n$. Similar results hold for primes $p < 2^{14}$.

5 HW Implementation

Algorithm III is optimized in terms of reducing the number of additions and subtractions, which are critical in integer arithmetic due to carry propagation in long computer words. Other optimization criteria included making the evaluation of tests during the calculation as simple as possible and minimizing data dependencies to enable calculation in parallel calculations. Figure 2 shows the circuit implementing the computation of classical modular inverse . Only data paths for computing the classical modular inverse according to Algorithm III are shown. The system consists of three basic parts. The first two parts form the well-known "butterfly" [14], [15], typical for algorithms based on the extended Euclidean algorithm; the third part consists of the controller and support circuitry. "Master" half of the "butterfly" calculates the $\gcd(m, a)$ and consists of two m bit registers Ru, Rv, m bit adder/subtracter ADD1, multiplexer MUX1, and left-shift logic SHFT1. "Slave" half of the butterfly consists of $(m + 1)$ bit register Rr and m bit register Rs, m bit adder/subtracter ADD2, multiplexers MUX2, MUX3, MUX4, and right/left-shift logic SHFT1. The controller unit controls the operation of the entire system. The controller part also includes an m bit mask register Rm with test logic provided test in step 3, two e bit counters Cu, Cv with the comparator d and single-bit flip-flops u/$\bar{\text{v}}$ and wu.

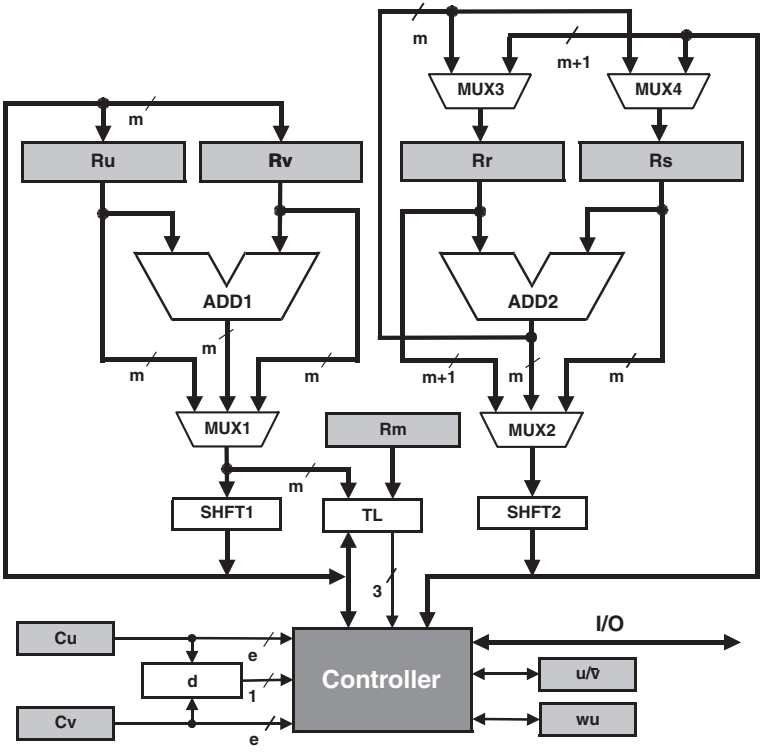


Fig. 2. The circuit implementation of Algorithm III

6 Conclusion

A new algorithm (Algorithm III) for classical modular inverse was presented and its HW implementation has been proposed. A mathematical proof (see the Appendix) that the proposed algorithm really computes classical modular inverse was performed. A statistical analysis of the new algorithm for large cryptographic integers was carried out. Computation of the modular inverse using the new algorithm is always faster and in the case of long words at least twice faster than other algorithms currently in use. The principles of the presented algorithm will also be used in a modular system for solving systems of linear equations without rounding errors [17].

References

1. P. L. Montgomery: Modular Multiplication Without Trial Division. Mathematics of Computation **44** No. 170 (1985) 519–521
2. B. S. Kaliski Jr.: The Montgomery Inverse and Its Application. IEEE Transaction on Computers **44** No. 8 (1995) 1064–1065

3. E. Savaş and Ç. K. Koç: The Montgomery Modular Inverse - Revisited. IEEE Transaction on Computers **49** No. 7 (2000)
4. D. E. Knuth: The Art of Computer Programming **2** / Seminumerical Algorithms. Addison-Wesley, Reading, Mass. Third edition (1998)
5. Ç. K. Koç: High-Radix and Bit Recoding Techniques For Modular Exponentiation. Int'l J. Computer Mathematics **40** (1991) 139–156
6. J.-J. Quisquarter and C. Couvreur: Fast Decipherment Algorithm for RSA Publickey Cryptosystem. Electronics Letters **18** No. 21 (1982) 905–907
7. W. Diffie and M. E. Hellman: New Directions in Cryptography. IEEE Transactions on Information Theory **22** (1976) 644–654.
8. Nat'l Inst. of Standards and Technology (NIST). FIPS Publication 186: Digital Signature Standard (1994)
9. N. Koblitz: Elliptic Curve Cryptosystem. Mathematics of Computation **48** No. 177 (1987) 203–209
10. A. J. Menezes: Elliptic curve Public Key Cryptosystem. Kluwer Academic Publishers, Boston, MA (1993)
11. Ö. Eğecioğlu and Ç. K. Koç: Exponentiation Using Canonical recoding. Theoretical Computer Science **129** No. 2 (1994) 407–717
12. R. T. Gregory and E. V. Krishnamurthy: Methods and Applications of Error-free Computation. Springer-Verlag, New York, Berlin, Heidelberg, Tokyo (1984)
13. K. H. Rosen: Elementary Number Theory and Its Applications. Addison-Wesley, Reading, Massachusetts (1993)
14. J. D. Dworkin, P. M. Glaser, M. J. Torla, A. Vadekar, R. J. Lambert, S. A. Vanstone: Finite Field Inverse Circuit. US Patent 6,009,450 (1999)
15. B. Bruner, A. Curiger, M. Hofstetter: On Computing Multiplicative Inverse in GF(2^m). IEEE Trans. Computer **42** (1993) 1010–1015
16. M. Morháč and R. Lórencz: A Modular System for Solving Linear Equations Exactly, I. Architecture and Numerical Algorithms. Computers and Artificial Intelligence **11** No. 4 (1992) 351–361
17. R. Lórencz and M. Morháč: Modular System for Solving Linear Equations Exactly, II. Hardware Realization. Computers and Artificial Intelligence **11** No. 5 (1992) 497–507

7 Appendix: The Mathematical Proof of Proposed Algorithm

Algorithm III has similar properties as the Euclidean Algorithm and algorithms derived from it, introduced in [4], [12], [13], where methods for their verification are also presented. By using similar proof techniques we have carried out a proof that Algorithm III computes correctly the classical modular inverse.

For computing multiplicative inverse of an integer a in the finite field GF(p), where p is a prime, the following lemma is important.

Lemma 1. *If* $\gcd(p, a) = 1$ *and if*

$$1 = px + ab,$$

then

$$a^{-1} \bmod p = b \bmod p.$$

The proof of the lemma is in [12]. By finding a pair of integers x and b that satisfy equations in Lemma 1, we prove that Algorithm III computes classical inverse in GF(p). Individual iterations of Algorithm III can be described by following system 3 of recurrent equations and guarding conditions for quotients,

$$
\begin{array}{llllll}
r_1 & = p - aq_1 & 0 < r_1 < aq_1 & q_1 & = 2^{(\langle p\rangle - \langle a\rangle)} & q_1 > 1 \\
r_2 & = |r_1| - aq_2 & 0 < r_2 < aq_2 & q_2 & = 2^{(\langle r_1\rangle - \langle a\rangle)} & q_2 > 1 \\
 & \vdots & & & & \\
r_j & = |r_{j-1}| - aq_j & 0 < |r_j| < a & q_j & = 2^{(\langle r_{j-1}\rangle - \langle a\rangle)} & q_j > 1 \\
\hline
r_{j+1} & = a - |r_j|q_{j+1} & 0 < |r_{j+1}| < |r_j|q_{j+1} & q_{j+1} & = 2^{(\langle a\rangle - \langle r_j\rangle)} & q_{j+1} > 1 \\
r_{j+2} & = |r_{j+1}| - |r_j|q_{j+2} & 0 < |r_{j+2}| < |r_j|q_{j+2} & q_{j+2} & = 2^{(\langle r_{j+1}\rangle - \langle r_j\rangle)} & q_{j+2} > 1 \\
 & \vdots & & & & \\
r_k & = |r_{k-1}| - |r_j|q_k & 0 < |r_k| < |r_j| & q_k & = 2^{(\langle r_{k-1}\rangle - \langle r_j\rangle)} & q_k > 1 \\
\hline
r_{k+1} & = |r_j| - |r_k|q_{k+1} & 0 < |r_{k+1}| < |r_k|q_{k+1} & q_{k+1} & = 2^{(\langle r_j\rangle - \langle r_k\rangle)} & q_{k+1} > 1 \\
r_{k+2} & = |r_{k+1}| - |r_k|q_{k+2} & 0 < |r_{k+2}| < |r_k|q_{k+2} & q_{k+2} & = 2^{(\langle r_{k+1}\rangle - \langle r_k\rangle)} & q_{k+2} > 1 \\
 & \vdots & & & & \\
r_l & = |r_{l-1}| - |r_k|q_l & 0 < |r_l| < |r_k| & q_l & = 2^{(\langle r_{l-1}\rangle - \langle r_k\rangle)} & q_l > 1 \\
\hline
 & \vdots & & & & \\
\hline
 & \vdots & & & & \\
r_m & = \ldots & & & & \\
\hline
r_{m+1} & = \ldots & & & & \\
 & \vdots & & & & \\
r_n & = |r_{n-1}| - |r_m| & 0 < |r_n| < |r_{n-1}| & q_n & = 2^{(\langle r_{n-1}\rangle - \langle r_m\rangle)} & q_n = 1 \\
\hline
r_{n+1} & = |r_{n-1}| - |r_n|q_{n+1} & 0 < |r_{n+1}| < |r_n|q_{n+1} & q_{n+1} & = 2^{(\langle r_{n-1}\rangle - \langle r_n\rangle)} & q_{n+1} > 1 \\
r_{n+2} & = |r_{n+1}| - |r_n|q_{n+2} & 0 < |r_{n+2}| < |r_n|q_{n+2} & q_{n+2} & = 2^{(\langle r_{n+1}\rangle - \langle r_n\rangle)} & q_{n+2} > 1 \\
 & \vdots & & & & \\
r_o & = |r_{o-1}| - |r_n|q_o & |r_o| = 1 & q_o & = 2^{(\langle r_{o-1}\rangle - \langle r_n\rangle)} & q_o > 1 \\
\hline
0 & = |r_n| - |r_o|q_{o+1}, & & & &
\end{array}
\tag{3}
$$

where $r_1, r_2, \ldots, r_{o+1}$ are remainders, $q_1, q_2, \ldots, q_{o+1}$ are quotients, $\langle r_i\rangle$ is the number of bits needed for binary representation $|r_i|$. If the recursive definition of r_i is unrolled up to p and a, each r_i can be expressed by a Diophantine equation $r_i = pf_i + aq_i$, where $f_i = f_i(q_1, q_2, \ldots, q_i)$, and $g_i = g_i(q_1, q_2, \ldots, q_i)$.

The last non-zero remainder equal r_o fulfils the following theorem:

Theorem 1. $\gcd(p, a) = 1$ *iff* $|r_o| = 1$.

Proof.

$$\begin{aligned}
\gcd(p,a) &= \gcd(r_1,a) = \gcd(r_2,a) = \ldots = \gcd(r_{j-1},a)\\
&= \gcd(a,|r_j|) = \gcd(|r_{j+1}|,|r_j|) = \ldots = \gcd(|r_{k-1}|,|r_j|)\\
&= \gcd(|r_j|,|r_k|) = \gcd(|r_{k+1}|,|r_k|) = \ldots = \gcd(|r_{l-1}|,|r_k|)\\
&= \gcd(|r_k|,|r_l|) = \ldots\\
&\vdots\\
&= \ldots = \gcd(|r_{n-1}|,|r_m|) = \gcd(|r_n|,|r_m|) = \gcd(|r_{n-1}|,|r_n|)\\
&= \gcd(|r_{n-1}|,|r_n|) = \gcd(|r_{n+1}|,|r_n|) = \ldots = gcd(|r_{o-1}|,|r_n|)\\
&= \gcd(|r_n|,|r_o|) = \gcd(|r_o|,0)\\
&= |r_o| = 1.
\end{aligned}$$

□

The previous statement assumed the following trivial properties of gcd:

$$\begin{aligned}
&\gcd(0,d) = |d| \text{ for } d \neq 0,\\
&\gcd(c,d) = \gcd(d,c),\\
&\gcd(c,d) = \gcd(|c|,|d|),\\
&\gcd(c,d) = \gcd(c+ed,d),
\end{aligned}$$

where c, d, and e are integers. The description of the properties are introduced in [12], [13].

The fact that the guarding conditions for quotients q_i guarantee correct values of remainders r_i follows from the Lemma 2:

Lemma 2. *Let c and d be positive integers with binary representations $c = 2^i + c_{i-1}2^{i-1} + \ldots + c_0$ and $d = 2^j + d_{j-1}2^{j-1} + \ldots + d_0$. Assume $i \geq j$. Let $q = 2^{(i-j)}$ and $e = c - qd$. Then:*

$$|e| < qd \text{ and } |e| < c.$$

Proof. Follows easily from identity

$$\sum_{k=1}^{i} \frac{1}{2^k} = 1 - \frac{1}{2^i},$$

which is proven for example in [13]. □

The computation specified in Equations (3) can be expressed in the form of Table 4, which gives the expressions for integer values f_i and g_i.

Since $r_o = pf_o + ag_o$, it follows that:

if $(r_o = 1)$ then
  $x = f_o$, $b = g_o$, and $a^{-1} \bmod p = g_o \bmod p$,
if $(r_o = -1)$ then
  $x = -f_o$, $b = -g_o$, and $a^{-1} \bmod p = (-g_o) \bmod p$.

Table 4. The computation of f_o, g_o, and r_o

i	r_i	f_i	g_i
1	r_1	1	$-q_1$
2	r_2	± 1	$\pm q_1 - q_2$
3	r_3	± 1	$\pm q_1 \pm q_2 - q_3$
$\vdots$	$\vdots$	$\vdots$	$\vdots$
j	r_j	± 1	$\pm q_1 \pm q_2 \pm \ldots - q_j$
$j+1$	r_j	$\pm q_{j+1}$	$1 \pm q_{j+1}(\pm q_1 \pm q_2 \pm \ldots - q_j)$
$\vdots$	$\vdots$	$\vdots$	$\vdots$
k	r_k	f_k	g_k
$\vdots$	$\vdots$	$\vdots$	$\vdots$
o	r_o	f_o	g_o
$o+1$	0	f_{o+1}	g_{o+1}.

□

It is the last step of the proof that Algorithm computes classical modular inverse. Finally, we take note of the principal features of the proposed Algorithm III from the point of view of the presented proof. The algorithm employs two's complement code for additions or subtractions. Therefore, the test whether an addition or subtraction is to be performed becomes a simple sign test. If the signs of both operands are equal, we subtract one from the other and in the opposite case we add both operands. Equations in (3) respect this rule when using absolute values of operands. According to Lemma 2, successive values r_i of remainders decrease in such a way that $p > a > |r_1| > |r_2| > \ldots |r_o|$.

The selection of operands which will be rewritten with a new value of the computed remainder is based on a simple test. The write is performed into the operand which needs more bits for its binary representation. This fact is respected in (3) by the value q_i. If $q_i = 1$, the write is into one of operands without respect on their absolute values. This case is demonstrated for remainder r_n. The conditions given by inequalities of Lemma 2 for r_n are fulfilled, too. Hence it holds $p > a > |r_1| > |r_2| > \ldots > |r_{n-1}| > |r_n| > |r_{n+1}| > \ldots > |r_o|$.

Increasing the Bitlength of a Crypto-Coprocessor

Wieland Fischer and Jean-Pierre Seifert

Infineon Technologies
Security & ChipCard ICs
CC TI Concepts & Innovations
D-81609 Munich
Germany
{wieland.fischer,jean-pierre.seifert}@infineon.com

Abstract. We present a novel technique which allows a virtual increase of the bitlength of a crypto-coprocessor in an efficient and elegant way. The proposed algorithms assume that the coprocessor is equipped with a special modular multiplication instruction. This instruction, called `MultModDiv`(A, B, N) computes $A * B \bmod N$ and $\lfloor (A * B)/N \rfloor$. In addition to the doubling algorithm, we also present two conceivable economic implementations of the `MultModDiv` instruction: one hardware and one software realization. The hardware realization of the `MultModDiv` instruction has the same performance as the modular multiplication presented in the paper. The software realization requires two calls of the modular multiplication instruction. Our most efficient algorithm needs only six calls to an n-bit `MultModDiv` instruction to compute a modular $2n$-bit multiplication. Obviously, special variants of our algorithm, e.g., squaring, require fewer calls.

Keywords: Arithmetical coprocessor, Hardware architecture, Modular multiplication, Hardware/Software codesign.

1 Introduction

Fast modular multiplication algorithms have been extensively studied [Ba,DQ, HP1,HP2,Knu,Mo,Om,Pai,Q,Sed,WQ,Wa]. This is due to the fact that large integer arithmetic is essential for public-key cryptography. Recently, we have seen some progress of integer factorization [C+] which demands for higher RSA bit lengths. On the other hand, for low cost and low power devices (e.g., in Smartcards, PDAs, Cellular Phones, etc.) one has to use hardware which does not provide sufficient bitlengths.

Unfortunately, these two requirements lead to a burden of the system issuer, e.g., the card industry. The source of this burden is the fact that, say 2048-bit RSA, cannot be handled efficiently on a 1024 bit device. Only with some work-around this problem becomes a manageable task. Namely, as it is now commonly known, one can use the Chinese Remainder Theorem for the RSA signature, see

B.S. Kaliski Jr. et al. (Eds.): CHES 2002, LNCS 2523, pp. 71–81, 2003.

[CQ]. To keep the RSA verification also relatively simple, most often the fourth Fermat number is used as public exponent. Only recently it was shown how to efficiently reduce such modular 2048-bit multiplications to 1024-bit modular multiplications, see [HP1,HP2,Pai]. Pailler [Pai] initiated this doubling research topic and formulated the following research problem:

Problem. Find an nk-bit modular multiplication algorithm using a minimal number of n-bit modular operations.

His algorithm needs nine modular multiplications for the case $k = 2$, see [Pai]. In this paper we will provide an answer to his question by presenting a novel doubling algorithm. Our most general and most efficient algorithm needs only six n-bit modular operations to compute a $2n$-bit modular multiplication. The idea for our family of algorithms is based on the fact that the coprocessor is equipped with a special modular multiplication instruction. This instruction is called `MultModDiv` and defined within the next section. An optimal realization is clearly achieved using an enhanced hardware modular multiplication instruction. Nevertheless, an efficient software realization of this instruction is possible. The software realization requires two calls to the modular multiplication instruction. Both realizations of this `MultModDiv` instruction will be presented.

The present paper is organized as follows: The next section gives the necessary definitions of `MultModDiv`. Section 3 explains our basic doubling algorithm, an enhanced version and also our special purpose variants. In section 4 we introduce the simple software emulation of the `MultModDiv` instruction. Finally, in section 5 we show how to realize the `MultModDiv` instruction in hardware.

2 Preliminaries

2.1 The Instructions `MultMod` and $\texttt{MultModInit}_n$

The following definition is the usual modular multiplication.

Definition 1. *For numbers A, B and N, $N > 0$, the* `MultMod` *instruction is defined as*

$$R = \texttt{MultMod}(A, B, N)$$

with

$$R := (A * B) \bmod N.$$

The following extension of the modular multiplication is already a feature of today's existing crypto coprocessors.

Definition 2. *For a fixed integer n and numbers A, B, C and N, $N > 0$, the* $\texttt{MultModInit}_n$ *instruction is defined as*

$$R = \texttt{MultModInit}_n(A, B, C, N)$$

with

$$R := (A * B + C * 2^n) \bmod N.$$

2.2 The Instructions MultModDiv and MultModDivInit$_n$

The following definition is a natural extension of the usual modular multiplication.

Definition 3. *For a fixed integer n and numbers A, B and N, $N > 0$, the* MultModDiv *is defined as*

$$(Q, R) = \texttt{MultModDiv}(A, B, N)$$

with

$$Q := \left\lfloor \frac{A * B}{N} \right\rfloor \quad \text{and} \quad R := (A * B) - Q * N.$$

Definition 4. *For a fixed integer n and numbers A, B, C and N, $N > 0$, the* MultModDivInit$_n$ *instruction is defined as*

$$(Q, R) = \texttt{MultModDivInit}_n(A, B, C, N)$$

with

$$Q := \left\lfloor \frac{A * B + C * 2^n}{N} \right\rfloor \quad \text{and} \quad R := (A * B + C * 2^n) - Q * N.$$

3 The Doubling Algorithm

3.1 Modular Multiplication without Initialization

We will start with the easiest of our algorithms, which needs 7 MultModDiv instructions on an n-bit processor.

Theorem 1. *There exists an algorithm to compute $A * B \bmod N$ using seven* MultModDiv *instructions of length n, provided that $2^{2n-1} \leq N < 2^{2n}$ and $0 \leq A, B < N$.*

Proof. We will first present the algorithm.

Basic Doubling Algorithm:
input: $N = N_t 2^n + N_b$ with $0 \leq N_b < 2^n$,
$A = A_t 2^n + A_b$ with $0 \leq A_b < 2^n$,
$B = B_t 2^n + B_b$ with $0 \leq B_b < 2^n$

$(Q^{(1)}, R^{(1)}) := \texttt{MultModDiv}(B_t, 2^n, N_t)$
$(Q^{(2)}, R^{(2)}) := \texttt{MultModDiv}(Q^{(1)}, N_b, 2^n)$
$(Q^{(3)}, R^{(3)}) := \texttt{MultModDiv}(A_t, R^{(1)} - Q^{(2)} + B_b, N_t)$
$(Q^{(4)}, R^{(4)}) := \texttt{MultModDiv}(A_b, B_t, N_t)$
$(Q^{(5)}, R^{(5)}) := \texttt{MultModDiv}(Q^{(3)} + Q^{(4)}, N_b, 2^n)$
$(Q^{(6)}, R^{(6)}) := \texttt{MultModDiv}(A_t, R^{(2)}, 2^n)$
$(Q^{(7)}, R^{(7)}) := \texttt{MultModDiv}(A_b, B_b, 2^n)$

$Q := (R^{(3)} + R^{(4)} - Q^{(5)} - Q^{(6)} + Q^{(7)})$
$R := (R^{(7)} - R^{(6)} - R^{(5)})$
make final reduction on $(Q * 2^n + R)$

output: $Q * 2^n + R$

We will prove that $(R^{(3)}+R^{(4)}-Q^{(5)}-Q^{(6)}+Q^{(7)})*2^n+(R^{(7)}-R^{(6)}-R^{(5)})$ is indeed congruent to $A*B$ modulo N. This can easily be seen from the following, where we use $Z = 2^n$ as abbreviation.

$$
\begin{aligned}
& (A_t Z + A_b) * (B_t Z + B_b) \\
&= A_t B_t Z Z + A_t B_b Z + A_b B_t Z + A_b B_b \\
&= A_t (Q^{(1)} N_t + R^{(1)}) Z + A_t B_b Z + A_b B_t Z + A_b B_b \\
&\equiv A_t R^{(1)} Z - A_t Q^{(1)} N_b + A_t B_b Z + A_b B_t Z + A_b B_b \\
&= A_t R^{(1)} Z - A_t (Q^{(2)} Z + R^{(2)}) + A_t B_b Z + A_b B_t Z + A_b B_b \\
&= A_t (R^{(1)} - Q^{(2)} + B_b) Z - A_t R^{(2)} + A_b B_t Z + A_b B_b \\
&= (Q^{(3)} N_t + R^{(3)}) Z - A_t R^{(2)} + A_b B_t Z + A_b B_b \\
&= (Q^{(3)} N_t + R^{(3)}) Z - A_t R^{(2)} + (Q^{(4)} N_t + R^{(4)}) Z + A_b B_b \\
&\equiv (R^{(3)} + R^{(4)}) Z - (Q^{(3)} + Q^{(4)}) N_b - A_t R^{(2)} + A_b B_b \\
&= (R^{(3)} + R^{(4)}) Z - (Q^{(5)} Z + R^{(5)}) - A_t R^{(2)} + A_b B_b \\
&= (R^{(3)} + R^{(4)}) Z - (Q^{(5)} Z + R^{(5)}) - (Q^{(6)} Z + R^{(6)}) + A_b B_b \\
&= (R^{(3)} + R^{(4)}) Z - (Q^{(5)} Z + R^{(5)}) - (Q^{(6)} Z + R^{(6)}) + (Q^{(7)} Z + R^{(7)}) \\
&= (R^{(3)} + R^{(4)} - Q^{(5)} - Q^{(6)} + Q^{(7)}) Z + (R^{(7)} - R^{(6)} - R^{(5)}) \bmod N
\end{aligned}
$$

The two congruences above are based on the fact that $N_t Z \equiv -N_b \bmod N$. Apart from the fact that this result still has to be reduced modulo N, this completes the proof. □

Practical Implementation Issues

1. Observe that in steps three and five negative numbers may occur. This can be resolved by the fact that for positive numbers A, B and N the equation $(Q, R) = \texttt{MultModDiv}(A, B, N)$ implies $(-Q - 1, N - R) = \texttt{MultModDiv}(A, -B, N)$, if $R \neq 0$.
2. It is possible that the intermediary output (Q, R) is not reduced, i.e., $0 \leq R < 2^n$ and $0 \leq Q < N_t$ is not fulfilled. In this case one has to do a final reduction: first, do $(Q, R) \leftarrow (Q \pm N_t, R \pm N_b)$ until Q is reduced modulo N_t. Then, do $(Q, R) \leftarrow (Q \pm 1, R \mp 2^n)$ until R is reduced modulo 2^n.
3. Using two parallel n-bit processors one only needs the time of four `MultModDiv` instructions.
4. If the given module N has an odd bitlength, then one has to compute with $2 * N$.

3.2 Modular Multiplication with Initialization

By using a $\texttt{MultModDivInit}_n$ instruction we can reduce the number of steps to six.

Theorem 2. *There exists an algorithm to compute* $A * B \bmod N$ *using five* `MultModDiv` *and one* $\texttt{MultModDivInit}_n$ *instruction of length* n*, provided that* $2^{2n-1} \leq N < 2^{2n}$ *and* $0 \leq A, B < N$.

Proof. We first present the algorithm.

Enhanced Basic Doubling Algorithm:
input: $N = N_t 2^n + N_b$ with $0 \leq N_b < 2^n$,
$A = A_t 2^n + A_b$ with $0 \leq A_b < 2^n$,
$B = B_t 2^n + B_b$ with $0 \leq B_b < 2^n$

$(Q^{(1)}, R^{(1)}) := \texttt{MultModDiv}(A_t, B_t, N_t)$
$(Q^{(2)}, R^{(2)}) := \texttt{MultModDivInit}_n(N_b, -Q^{(1)}, R^{(1)}, N_t)$
$(Q^{(3)}, R^{(3)}) := \texttt{MultModDiv}(A_t, B_b, N_t)$
$(Q^{(4)}, R^{(4)}) := \texttt{MultModDiv}(A_b, B_t, N_t)$
$(Q^{(5)}, R^{(5)}) := \texttt{MultModDiv}(A_b, B_b, 2^n)$
$(Q^{(6)}, R^{(6)}) := \texttt{MultModDiv}(Q^{(2)} + Q^{(3)} + Q^{(3)}, N_b, 2^n)$

$Q := (R^{(2)} + R^{(3)} + R^{(4)} + Q^{(5)} - Q^{(6)})$
$R := (R^{(5)} - R^{(6)})$
make final reduction on $(Q * 2^n + R)$

output: $Q * 2^n + R$

We will prove that $(R^{(2)} + R^{(3)} + R^{(4)} + Q^{(5)} - Q^{(6)}) * 2^n + (R^{(5)} - R^{(6)})$ is indeed congruent to $A * B$ modulo N. This can be seen from the following, where we use $Z = 2^n$ as abbreviation.

$$
\begin{aligned}
&(A_tZ + A_b) * (B_tZ + B_b) \\
&= A_tB_tZZ + A_tB_bZ + A_bB_tZ + A_bB_b \\
&= (Q^{(1)}N_t + R^{(1)})ZZ + A_tB_bZ + A_bB_tZ + A_bB_b \\
&\equiv (R^{(1)}Z - Q^{(1)}N_b)Z + A_tB_bZ + A_bB_tZ + A_bB_b \\
&= (Q^{(2)}N_t + R^{(2)})Z + A_tB_bZ + A_bB_tZ + A_bB_b \\
&\equiv (R^{(2)}Z - Q^{(2)}N_b) + A_tB_bZ + A_bB_tZ + A_bB_b \\
&= (R^{(2)}Z - Q^{(2)}N_b) + (Q^{(3)}N_t + R^{(3)})Z + A_bB_tZ + A_bB_b \\
&= (R^{(2)}Z - Q^{(2)}N_b) + (Q^{(3)}N_t + R^{(3)})Z + (Q^{(4)}N_t + R^{(4)})Z + A_bB_b \\
&= (R^{(2)}Z - Q^{(2)}N_b) + (Q^{(3)}N_t + R^{(3)})Z + (Q^{(4)}N_t + R^{(4)})Z + (Q^{(5)}Z + R^{(5)}) \\
&\equiv (R^{(2)} + R^{(3)} + R^{(4)} + Q^{(5)})Z - (Q^{(2)} + Q^{(3)} + Q^{(4)})N_b + R^{(5)} \\
&= (R^{(2)} + R^{(3)} + R^{(4)} + Q^{(5)})Z - (Q^{(6)}Z + R^{(6)}) + R^{(5)} \\
&= (R^{(2)} + R^{(3)} + R^{(4)} + Q^{(5)} - Q^{(6)})Z + (R^{(5)} - R^{(6)}) \bmod N
\end{aligned}
$$

The three congruences above are based on the fact that $N_tZ \equiv -N_b \bmod N$. Apart from the fact that this result still has to be reduced modulo N, this completes the proof. □

Practical Implementation Issues

1. Observe that in steps two and six negative numbers may occur. This can be resolved as shown above.
2. It is possible that the intermediary output (Q, R) is not reduced. This can be resolved as shown above.
3. Using two parallel n-bit processors one only needs the time of three `MultModDiv` instructions.
4. Again, if the given module N has an odd bitlength, then one has to compute with $2 * N$.

3.3 Optimized Special Purpose Variants

Now the basic strategy of our algorithms should be clear. Therefore, we will present the results for special purpose variants.

Squaring

Theorem 3. *There exists an algorithm to compute* $A^2 \bmod N$ *using six* `MultModDiv` *instructions of length* n*, provided that* $2^{2n-1} \leq N < 2^{2n}$ *and* $0 \leq A < N$.

If we consider the algorithm of section 3.2 for the case $A = B$, we see that steps three and four are identical. Therefore, we get the following result:

Theorem 4. *There exists an algorithm to compute* $A^2 \bmod N$ *using four* `MultModDiv` *and one* $\texttt{MultModDivInit}_n$ *instruction of length* n*, provided that* $2^{2n-1} \leq N < 2^{2n}$ *and* $0 \leq A < N$.

Precomputation

If the factor B is known in advance (e.g., square and *multiply* for exponentiation), then the first and second computation of the algorithm of section 3.1 can be carried out in advance. Therefore, the multiplication can be done in five steps.

Theorem 5. *There exists an algorithm to compute* $A * B \bmod N$ *using five* `MultModDiv` *instructions of length* n*, provided that* $2^{2n-1} \leq N < 2^{2n}$ *and* $0 \leq A, B < N$*, where* B *is known in advance.*

Using a completely different idea, one needs only six `MultModDiv` steps. This time, one uses a special representation of A and B. Namely, $A = A_t * N_t + A_b$ and $B = B_t * N_t + B_b$, where $N_t := \lfloor \sqrt{N} \rfloor$.

Theorem 6. *There exists an algorithm to compute* $A * B \bmod N$ *using six* `MultModDiv` *instructions of length* n*, provided that* $2^{2n-1} \leq N < 2^{2n}$ *and* $0 \leq A, B < N$*, where* N *is known in advance..*

4 Software Realization of the `MultModDiv` and $\texttt{MultModDivInit}_n$ Instructions

This section presents a software emulation of the `MultModDiv` and $\texttt{MultModDivInit}_n$ instructions.

Theorem 7. *There exists an algorithm to compute* $\texttt{MultModDiv}(A, B, N)$ *using two* `MultMod` *instructions, provided that* $0 \leq A, B < N$*.*

Proof. We present the simple algorithm.

Simulation of `MultModDiv`:
input: A, B, N with $0 \leq A, B < N$

$R := \texttt{MultMod}(A, B, N)$
$N' := N + 1$
$R' := \texttt{MultMod}(A, B, N')$
$Q := R - R'$
if $(Q < 0)$ **then**
  $Q := Q + N'$
fi

output: (Q, R)

We will prove that the former algorithm correctly computes the `MultModDiv` instruction. For given inputs A, B and N there exists some Q and R with

$$A * B = Q * N + R \quad \text{and} \quad R = (A * B) \bmod N,$$

where $0 \leq R < N$ and $0 \leq Q < N - 1$. Equivalently, we also have

$$A * B = Q * (N + 1) + (R - Q).$$

For $(R-Q) \geq 0$ this means $(R-Q) = (A*B) \bmod (N+1)$ and for $(R-Q) < 0$ this means $(R-Q)+(N+1) = (A*B) \bmod (N+1)$. Thus, for Q we have

$$Q = ((A*B) \bmod N) - ((A*B) \bmod (N+1))$$

or, if this value is less than zero

$$Q = ((A*B) \bmod N) - ((A*B) \bmod (N+1)) + (N+1).$$

This completes the proof. □

In a similar way the $\texttt{MultModDivInit}_n$ instruction is emulated by the $\texttt{MultModInit}_n$ instruction.

Theorem 8. *There exists an algorithm to compute* $\texttt{MultModDivInit}_n$ (A, B, C, N) *using two* $\texttt{MultModInit}_n$ *instructions, provided that* $2^{n-1} \leq N < 2^n$ *and* $0 \leq A, B, C < N$.

Proof. We present the algorithm.

Simulation of $\texttt{MultModDivInit}_n$:
input: A, B, C, N with $0 \leq A, B, C < N$

$R := \texttt{MultModInit}_{n+2}(2A, 2B, C, 4N)$
$N' := 4N + 1$
$R' := \texttt{MultModInit}_{n+2}(2A, 2B, C, N')$
$Q := R - R'$
if $(Q < 0)$ **then**
  $Q := Q + N'$
fi

output: $(Q, R/4)$

The proof is a derivation of the former one, leaving the modifications to the reader. However, we note that bounding the size of the quotient Q is the crucial point. □

Both algorithms can be extended to algorithms also working for non-reduced A, B and C. This is necessary for our doubling algorithms.

5 Hardware Realization of the MultModDiv and $\texttt{MultModDivInit}_n$ Instructions

We will now sketch how an algorithm for the MultMod instruction can be extended into an algorithm for the MultModDiv instruction. We first consider the textbook MultMod implementation.

Textbook `MultMod` implementation:
input: A, B, N with $0 \leq A, B < N$, and $A = (A_{n-1}, \ldots, A_0)$
$i := n; Z := 0$
repeat
  $i := i - 1$
  case A_i **is**
    0**:** $Z := 2 * Z$
    1**:** $Z := 2 * Z + B$
  end case
  if $(Z \geq N)$ **then**
    $Z := Z - N$
    if $(Z \geq N)$ **then**
      $Z := Z - N$
    fi
  fi
until $(i = 0)$
output: Z

The extension is rather trivial. During the modular multiplication we simply have to "count" the number of subtracted N's. Observe that during a modular multiplication this implicit information is always known to the algorithm/hardware.

`MultModDiv` implementation:
input: A, B, N with $0 \leq A, B < N$ and $A = (A_{n-1}, \ldots, A_0)$
$i := n; Z := 0$
repeat
  $i := i - 1$
  case A_i **is**
    0**:** $Z := 2 * Z$
    1**:** $Z := 2 * Z + B$
  end case
  $Q_i := 0;\ Q'_i := 0$
  if $(Z \geq N)$ **then**
    $Z := Z - N$
    $Q_i := 1;\ Q'_i := 0$
    if $(Z \geq N)$ **then**
      $Z := Z - N$
      $Q_i := 1;\ Q'_i := 1$
    fi
  fi
until $(i = 0)$
$Q := Q + Q'$
output: (Q, Z)

In the paper's full version we will actually show how the former algorithm can be simply integrated into the modular multiplication algorithm due to H. Sedlak [Sed].

The $\texttt{MultModDivInit}_n$ and $\texttt{MultModInit}_n$ are derived from the former ones essentially by exchanging the step $Z := 0$ with $Z := C$.

6 Conclusion

In this paper we have introduced new efficient algorithms to compute $2n$-bit modular multiplications using only n-bit modular multiplications. Using the $\texttt{MultModDiv}$ and $\texttt{MultModDivInit}_n$ instructions we were able to improve the results presented by Pailler [Pai]. The question of what is the minimal number of multiplications is still open, as we currently have no proof of the optimality of our algorithm.

Acknowledgments. We would like to thank Holger Sedlak for several valuable discussions on this topic.

References

[Ba] P. Barret, "Implementing the Rivest, Shamir and Adleman public-key encryption algorithm on a standard digital signal processor", *Proc. of CRYPTO '86*, Springer LNCS, vol. 263, pp. 311–323, 1987.

[C+] S. Cavallar *et alii*, "Factoring a 512 bit RSA modulus", *Proc. of EUROCRYPT '00*, Springer LNCS, vol. 1807, pp. 1–19, 2000.

[CQ] C. Couvreur, J.-J. Quisquater, "Fast decipherment algorithm for RSA public-key cryptosystem", *Electronics Letters* **18**(21):905–907, 1982.

[DQ] J.-F. Dhem, J.-J. Quisquater, "Recent results on modular multiplication for smart cards", *Proc. of CARDIS '98* Springer LNCS vol. 1820, pp. 336–352, 1998.

[HP1] H. Handschuh, P. Pailler, "Smart Card Crypto-Coprocessors for Public-Key Cryptography", *CryptoBytes* **4**(1):6–11, 1998.

[HP2] H. Handschuh, P. Pailler, "Smart Card Crypto-Coprocessors for Public-Key Cryptography", *Proc. of CARDIS '98* Springer LNCS vol. 1820, pp. 372–379, 1998.

[Knu] D. E. Knuth, *The Art of Computer Programming, Vol.2: Seminumerical Algorithms*, 3rd ed., Addison-Wesley, Reading MA, 1999.

[MvOV] A. J. Menezes, P. van Oorschot, S. Vanstone, *Handbook of Applied Cryptography*, CRC Press, New York, 1997.

[Mo] P. L Montgomery, "Modular Multiplication without Trial Division", *Math. of Computation*, vol. **44**, pp. 519–521, 1985.

[NMR] D. Naccache, D. M'Raihi, "Arithmetic co-processors for public-key cryptography: The state of the art", *IEEE Micro*, pp. 14–24, 1996.

[Om] J. Omura, "A public key cell design for smart card chips", *Proc. of IT Workshop*, pp. 27–30, 1990.

[Pai] P. Pailler, "Low-cost double size modular exponentiation or how to stretch your cryptocoprocessor", *Proc. of Public Key Cryptography '99*, Springer LNCS, vol. 1560, pp. 223–234, 1999.

[Q] J.-J. Quisquater, "Encoding system according to the so-called RSA method, by means of a microcontroller and arrangement implementing this system", U.S. Patent #5,166,979, Nov. 24, 1992.

[RSA] R. Rivest, A. Shamir, L. Adleman, "A method for obtaining digital signatures and public-key cryptosystems", *Comm. of the ACM* **21**:120–126, 1978.

[Sed] H. Sedlak, "The RSA cryptographic Processor: The first High Speed One-Chip Solution", *Proc. of EUROCRYPT '87*, Springer LNCS, vol. 293, pp. 95–105, 198.

[WQ] D. de Waleffe, J.-J. Quisquater, "CORSAIR, a smart card for public-key cryptosystems", *Proc. of CRYPTO '90*, Springer LNCS, vol. 537, pp. 503–513, 1990.

[Wa] C. Walter, "Techniques for the Hardware Implementation of Modular Multiplication", *Proc. of 2nd IMACS Internat. Conf. on Circuits, Systems and Computers*, vol. **2**, pp. 945–949, 1998.

Enhancing Simple Power-Analysis Attacks on Elliptic Curve Cryptosystems*

Elisabeth Oswald

Institute for Applied Information Processing and Communications
Graz University of Technology
Inffeldgasse 16a, A-8010 Graz, Austria
Elisabeth.Oswald@iaik.at

Abstract. Recent applications of lattice attacks against elliptic curve cryptosystems have shown that the protection of ephemeral keys in the ECDSA is of greatest importance. This paper shows how to enhance simple power-analysis attacks on elliptic-curve point-multiplication algorithms by using Markov models. We demonstrate the attack on an addition-subtraction algorithm (fixing the sequence of elliptic-curve operations) which is similar to the one described by Morain et al. in [MO90] and apply the method to the general addition-subtraction method described in ANSI X9.62 [ANS99].

1 Introduction

Elliptic curve cryptosystems (ECC) have been introduced in 1985 by Miller and Koblitz and are widely accepted. Since there are no sub-exponential algorithms known for the elliptic-curve discrete-logarithm problem (ECDLP), the keys can be much smaller in elliptic curve cryptography than in other public-key cryptosystems. Consequently, elliptic-curve cryptography offers significant advantages in many practical aspects. Due to their practical advantages, elliptic curve cryptosystems can be expected to be incorporated in many future cryptographic applications and protocols. The most effective cryptanalytic attacks on implementations of elliptic curve cryptosystems nowadays are the power attacks [KJJ99][1]. They use the power consumption of a device performing an elliptic-curve scalar point-multiplication as a side-channel. Both power-analysis variants, the simple power analysis (SPA) and the differential power analysis (DPA), are effective against unprotected implementations of an elliptic-curve scalar point-multiplication. Due to the importance of elliptic curve cryptosystems, many articles related to power analysis and elliptic curve cryptography have recently been published. We want to mention one article especially. This article [RS01] (which is based on [HGS01]) describes how to compute the secret key of the ECDSA, if a few bits of the ephemeral key for several ECDSA

* The work in this paper was partially done while the author visited COSIC, KU Leuven, Belgium, 2002.

[1] EM attacks appear to become increasingly powerful as well, see for example [QS01]

B.S. Kaliski Jr. et al. (Eds.): CHES 2002, LNCS 2523, pp. 82–97, 2003.

signatures are known. Consequently the protection of ephemeral keys is a very important aspect that cannot be neglected. Therefore the implementation of an elliptic-curve scalar point-multiplication algorithm should be resistant against simple power-analysis attacks even if it is used only for signatures.

The main contribution of this paper is the development of a new and more powerful simple power-analysis attack which is even applicable to elliptic-curve scalar point-multiplication algorithms that do not fix the sequence of elliptic-curve operations. This attack shows that certain attempts to counteract simple power-analysis attacks by only obscuring[2] the ephemeral key, fail.

This paper is organized as follows. Section 2 is dedicated to related work. In section 3, the relationship between Markov models and point-multiplication algorithms is established, and the general idea for the enhanced simple power-analysis attack is presented. Finally, in section 4, we apply this method to a addition-subtraction algorithm and to the scalar point-multiplication algorithm defined in ANSI X9.62 ([ANS99] and IEEE P1363a [IEE99]). We also reason about the applicability of this method to the randomized algorithms as presented in [OA01].

2 Related Work

Finding efficient countermeasures to protect implementations of elliptic-curve scalar point-multiplication against power attacks has proven to be a difficult and challenging task. This is due to the fact that different constraints have to be taken into account for an actual implementation. For example, legal issues such as avoiding patents or implementation constraints. Implementations of elliptic curve cryptosystems usually make use of so called EC-accelerator modules that are very often connected via a slow bus to the rest of the IC. Depending on the specific hardware architecture that is used, an algorithm may or may not lead to an efficient (fast, small, or flexible, etc ...) implementation. Another constraint for countermeasures, is due to the fact that NIST[3] published a set of recommended curves [NIS99] which can be used as 'named curves' in certificates and protocols. These curves have one common property. They all have a cofactor of 2. Because of this specific choice of the cofactor, none of these curves has a Montgomery form. Therefore, countermeasures using the Montgomery form cannot be applied to them. On the other hand, these curves have been given OIDs and the set of elliptic-curve parameters can be replaced by these OIDs in certificates. This is certainly a big advantage since certificate sizes can be significantly reduced. Consequently, countermeasures should be applicable to all recommended curves.

[2] Obscuring means in this context, that there is a more complex relationship between the bits of the ephemeral key and the performed elliptic-curve operations. We will discuss this in more detail in section 3.1

[3] There exists a second set of 'named curves' which has been selected by the SECG [Cer00].

2.1 Previous Results

The basic principles of how to apply power analysis attacks on elliptic curve cryptosystems have been discussed in [Cor99]. To counteract both simple power-analysis attacks and (first order) differential power-analysis attacks there are basically two things that have to be done. Firstly, one has to randomize the expressions (i.e. the coordinates) of calculated points. This can be done by using randomized projective coordinates (DPA countermeasure). Secondly, one has to conceal the ephemeral key. It would be optimal if there would be no statistical relationship between the sequence of elliptic-curve operations and the bits of the ephemeral key (SPA countermeasure). Countermeasures applicable to arbitrary curves fixing the sequence of elliptic-curve operations have been presented by Coron [Cor99], Möller [Möl01] and Izu et al. [IT02]. Countermeasures applicable to arbitrary curves not fixing the sequence of elliptic-curve operations have been presented by Oswald et al. [OA01] and Brier et al. [BJ02]. Countermeasures applicable to special curves fixing the sequence of elliptic-curve operations have been presented by Hasan [Has00] and by Okeya et al. [OS00]. Countermeasures applicable to special curves not fixing the sequence of elliptic-curve operations have been presented by Liardet et al. [LS01] and Joye et al. [JQ01].

In none of the papers it was ever tried to extend the obvious simple power-analysis attack to more general point-multiplication algorithms, i.e. algorithms that also use elliptic-curve point-subtraction or do not fix the sequence of elliptic-curve operations.

3 An Attack Based on a Markov Model for the Elliptic-Curve Scalar Point-Multiplication Algorithm

In a simple power-analysis attack, the adversary is assumed to be able to monitor the power consumption of one scalar point-multiplication, $Q = kP$, where Q and P are points on an elliptic curve E, and $k \in \mathbb{Z}$ is a scalar. The attacker's goal is to learn the key using the information obtained from carefully observing the power trace of a complete scalar point-multiplication. Such a scalar point-multiplication consists of a sequence of point-addition, point-subtraction and point-doubling operations. Each elliptic-curve operation itself consists of a sequence of elementary field-operations. The sequence of elementary field-operations in an elliptic-curve point-addition operation differs from the sequence of elementary field-operations in elliptic-curve point-doubling operation. Every elementary field-operation has its unique power-consumption trace. Hence, the sequence of elementary field-operations that form the point-addition operation has a different power-consumption pattern than the sequence of elementary field-operations that form the point-doubling operation. Because point addition and point subtraction only differ slightly, they can be implemented in such a way that they are indistinguishable for an attacker. This is why we will not distinguish between these two operations in the subsequent sections.

3.1 Elliptic-Curve Scalar Point-Multiplication Algorithms

The simplest way of performing a scalar point-multiplication is the binary algorithm (see table 1 for the bottom-up version).

Table 1. Bottom-up version of the binary algorithm

binalg(P,M,k)
$Q = M$
if $k_0 = 1$ then $P = M$ else $P = 0$
for $i = 1$ to $n - 1$
$\quad Q = Q * Q$
$\quad$ if $(k_i == 1)$ then
$\quad\quad P = P * Q$
return P

For validity and explanation see [Knu98]. In table 1 the operator $*$ denotes the general elliptic-curve point-addition operation. The expression $P * Q$ denotes the point-addition operation (short A) which adds two distinct points P and Q on the elliptic curve, while $P * P$ denotes the point-doubling operation (short D) which adds P to itself.

What makes this algorithm so vulnerable to SPA is the strong relation between the multiplier bits (i.e. the k_i) and the performed operation (i.e. the point addition) in the conditional branch (see table 1). If and only if the i-th bit of k is set, a point-addition operation is performed. Another way of saying this is that the conditional probability that k_i is non-zero equals 1 under the assumption that an elliptic-curve point-addition operation has been observed. An attacker can simply look for two different patterns in the power trace of the scalar point-multiplication algorithm. One pattern corresponds to the point-addition operation and the other pattern corresponds to the point-doubling operation. Since only the point-addition operation can be induced from a non-zero bit, the attacker learns where the non-zero bits in the binary representation of k are. With this information the attacker has to test at most two values (this is because the attacker does not know which of the two observed patterns corresponds to the point addition-operation and which corresponds to the point-doubling operations) to determine the ephemeral key.

The usage of more sophisticated versions of the binary algorithm, like the window-method, signed representations like the non-adjacent form (short NAF), etc. ... (see [Gor98] for an excellent survey), can obscure the private multiplier to a certain extent. The main goal of these sophisticated algorithms is to speed up the scalar point-multiplication. This is usually accomplished by recoding[4]

[4] Well known methods referring to the same basic principle are for example Booth recoding, or Canonical recoding or NAF.

the multiplier k. The recoded form k' leads to fewer operations that have to be performed in the scalar point-multiplication. The arithmetic of elliptic curves makes it possible to use signed representations, i.e. use digits $-1, 0, 1$, because point subtraction is almost the same operation as point addition. The assumption we made in the beginning of this section, namely that we cannot distinguish between point addition and point subtraction, introduces an additional difficulty in the task of the attacker. Observing a point-addition operation gives the attacker less information, since a point addition corresponds to both -1 and 1 in the digit-expansion of k. Having more difficult relations between certain occurrences of operations in the power trace and specific bits (or maybe combination of bits) is what is meant by "obscuring" in this context. In general it can be said that whenever we don't have an "if and only if" relationship between bits and operations, we are not able to mount a simple power-attack in the way we described it for the standard binary algorithm. However, we show in this paper how an ordinary simple power-analysis attack can be enhanced in order to mount an efficient simple power-analysis attack on certain types of these algorithms.

3.2 The Attacker's Task

The attacker has the ability to observe a sequence of elliptic curve operations, thus, the attacker's aim is to calculate and exploit the probabilities of certain sequences of bits given an observed sequence of elliptic curve operations.

Using the information of such conditional probabilities, the key-space that has to be searched to find the correct ephemeral key, can be significantly reduced. This is because certain combinations of patterns in the power trace and certain combination of digits are less likely than the others (or even not possible at all). The attacker's task can be stated in a more formal way.

Let X be a random variable that denotes a sequence of elliptic-curve operations and $|X|$ the length of X (i.e. the number of elliptic-curve operations in this sequence). For example, X="DDD" (i.e. the realization of the random variable X consists of three consecutive elliptic-curve point-double operations) thus $|X| = 3$, or X="DAD" (i.e. the realization of the random variable X consists of an elliptic-curve point-double operation, an elliptic-curve point-addition operation and an elliptic-curve point-double operation) thus $|X| = 3$.

Let Y be a random variable that denotes a sequence of digits in the digit representation of k and $|Y|$ the length of Y (i.e. the number of digits). For example Y = "000" (i.e. the realization of the random variable Y consists of three consecutive zeros) thus $|Y| = 3$, or Y = "01" (i.e. the realization of the random variable Y consists of a zero and an one digit) thus $|Y| = 2$.

Then the attackers goal is to calculate and exploit the conditional probability

$$P(Y = y | X = x) = \frac{P(Y = y \cap X = x)}{P(X = x)} \tag{1}$$

for many different realizations x of X and y and Y. Equation 1 is the mathematical definition for the conditional probability.

It is an important observation that the calculation of the right hand side of (1) requires the knowledge of the probability to be in a specific state of the point-multiplication algorithm (the terminology used here will be explained in the next section). This is because in order to calculate the probabilities $P(X = x)$, one has to calculate the sum of the probabilities of all possible sequences of digits that lead to the pattern x. Since such a sequence can basically start from any state of the algorithm, the probabilities are dependent on the probability of the starting-state. These probabilities can be calculated by using Markov models which we are going to introduce in the following section.

3.3 Markov Models

The general assumption for the rest of this paper is that the multiplier bits k_i, are independently drawn and identically distributed. We can see a point-multiplication algorithm as a Markov process (see for example [GS92]). A Markov process in general can be used to analyze random, but dependent events. In a Markov process, the next state (or event) is only dependent on the present state but is independent of the way in which the present state arose from the states before. This is often referred to as "memoryless" process. The transitions between the states are determined by a random variable and occur with certain probabilities, that are either known or have to be estimated. A common way to work with Markov processes is to visualize them in so called transition graphs. For example, figure 1 shows the transition graph for the binary algorithm as presented in table 1.

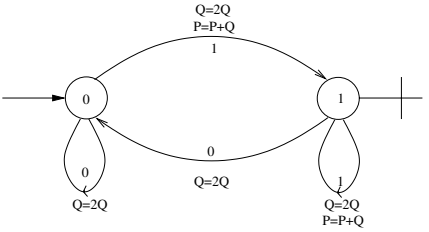


Fig. 1. Transition-graph of the Binary Algorithm

According to the description given for table 1, P and Q denote elliptic-curve points. Circles represent states (0 and 1 in this case) and the arrows between the circles represent transition between the states. Output paths are marked by an additional bar. In the binary algorithm, transitions are triggered by the bits k_i. Consequently, in figure 1 the values next to the arrows, correspond to the possible values of k_i (i.e. 0 and 1). The expressions $Q = 2Q$ (point double) and $P = P + Q$ (point addition) indicate what operation is triggered by the value of k_i.

This graph or Markov process, respectively, has two important properties. The first property is that the graph is *irreducible* in the sense that all states can be reached from all other states with a finite number of steps. The second important property is that the graph is *aperiodic*, in the sense that all states are aperiodic. A state is aperiodic, if the period (that is the greatest common divisor of the set of times a chain has a positive probability of returning to the same state) is equal to 1. These two properties are conditions for the main theorem of Markov theory. This theorem states basically that for Markov processes having the properties of being aperiodic and irreducible a *steady state* always exists[5]. The row-vector $\pi = (\pi_1, \ldots, \pi_n)$ representing the steady state has the following two properties:

$$\sum_{i=1}^{n} \pi_i = 1, \tag{2}$$

$$\pi T = \pi. \tag{3}$$

The variable T in the second equation is a matrix (subsequently referred to as *transition matrix*) containing the transition probabilities. (3) is the formal way of saying that the distribution has become steady. (2) makes clear that we are actually dealing with a probability distribution for the states (since all the individual probabilities sum up to 1). The entries of the row-vector π are simply the probabilities of the states the algorithm can be in. What is important for the attack presented in this paper is that π depends solely on the transition matrix T and can be obtained by calculating the eigenvectors (with the associated eigenvalues) of the transition matrix (this follows directly from (3)). The transition matrix itself can be obtained in a straightforward way. The transition probabilities associated with the transition variables can be written in the transition matrix

$$T = \begin{pmatrix} P(s_{i+1} = 0 | s_i = 0) & P(s_{i+1} = 0 | s_i = 1) \\ P(s_{i+1} = 1 | s_i = 0) & P(s_{i+1} = 1 | s_i = 1) \end{pmatrix} = \begin{pmatrix} 0.5 & 0.5 \\ 0.5 & 0.5 \end{pmatrix}.$$

T contains the probabilities to get from one state to another state. The random variable s_i denotes the state the algorithm is in. For example, the matrix entry in the first row and the second column is the probability to get from state 1 to state 0. Calculating $\pi = (1/2, 1/2)$ leads to the probabilities for being in state 0 and state 1, respectively.

Additionally, we can deduce the number of elliptic-curve operations that need to be executed. We know that every transition requires the calculation of a point-doubling operation, but only a transition leading from state 0 to state 1 or a transition leading from state 1 to state 1 requires the computation of a point-addition operation. Putting this all together and assuming that n denotes

[5] There are several more expressions for the term "steady state". Amongst others, the most common terms seem to be "stationary distribution" and "invariant distribution".

the bit-length of k, we get n point-doubling operations and $n/2$ point-addition operations. So, $3n/2$ elliptic-curve operations have to be performed.

In this section, we have established a relationship between elliptic-curve scalar point-multiplication algorithms and Markov processes. The most important observation was that the main theorem for an important class of Markov processes, namely the irreducible and aperiodic Markov processes, also gives a solution to the problem of calculating the conditional probability defined in (1). Furthermore we have shown how to calculate the number of elliptic-curve operations that need to be performed in a point-multiplication algorithm, with the aid of Markov models.

4 Results

In order to demonstrate that the observations presented in the previous section lead to an effective attack, we apply the technique on well known point-multiplication algorithms. The first algorithm we attack was introduced in the article of Morain et al. [MO90] and belongs to the class of addition-subtraction algorithms. We attack the modified version as given in [OA01]. The second algorithm we discuss is the *addition-subtraction method* or *NAF-method* which is used in important standards such as the ANSI X9.62 and the Annex A of IEEE 1363.

4.1 Analysis of a Double-Add and Subtract Algorithm

We demonstrate how this method can be used to construct a known-ciphertext attack on the example of a modification ([OA01]) of the first algorithm given by Morain et al. [MO90].

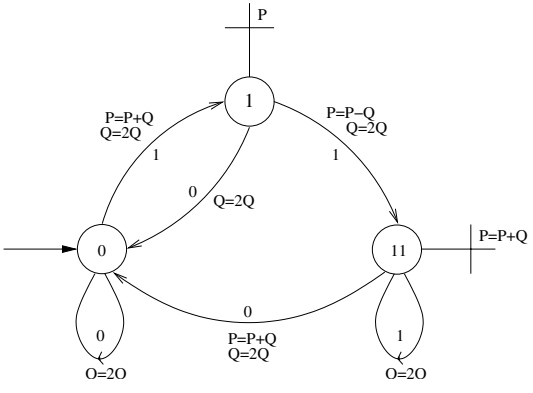


Fig. 2. Transition-graph of Double-Add and Subtract as proposed in [OA01]

The idea of this point-multiplication algorithm is basically to replace a block of at least two consecutive 1's in the binary representation of the multiplier k, by a block of 0's and a $-1 : 1^a \mapsto 1\,0^{a-1}-1$. The original proposal is modified in such a way that for every transition a point-doubling operation or a point-doubling and a point-addition operation has to be performed. Mounting a standard simple power-attack is not possible since there is no "if and only if" relation between the bits and the elliptic curve operations. For both, zero and non-zero bits, elliptic-curve point-addition and elliptic-curve point-doubling operations can occur.

The transition graph of the finite state machine in figure 2 (the notation used here is the same as used before in figure 1) visualizes this algorithm. From this graph one can easily deduce the transition matrix

$$T = \begin{pmatrix} 0.5 & 0.5 & 0.5 \\ 0.5 & 0 & 0 \\ 0 & 0.5 & 0.5 \end{pmatrix}$$

and therefrom the steady-state vector which is $(1/2, 1/4, 1/4)$. With this information the conditional probabilities for many realizations of X and Y can be calculated. With a modest computational effort, one can do this up to $|Y| = 16$. The results we obtained show how poor the secret key is obscured. We found out that for a chosen X and for fixed and small $|Y|$, only **three** bit-patterns are possible, or there is no bit-pattern at all. We derived this result from our computer program, which checked all conditional probabilities up to $|Y| = 12$. For example, table 2 shows some of the results for small values of $|Y|$.

Table 2. Non-zero conditional probabilities. In this table we use an abbreviated notation, i.e. we write $p(000|DDD)$ instead of $p(Y = 000|X = DDD)$. We use the LSB first representation.

$p(000\|DDD) = 1/2$	$p(01\|DAD) = 1/2$	$p(11\|ADAD) = 1/2$
$p(100\|DDD) = 1/4$	$p(10\|DAD) = 1/4$	$p(10\|ADAD) = 1/4$
$p(111\|DDD) = 1/4$	$p(11\|DAD) = 1/4$	$p(01\|ADAD) = 1/4$
$p(001\|DDAD) = 1/2$	$p(000\|ADDD) = 1/4$	$p(110\|ADADAD) = 1/2$
$p(101\|DDAD) = 1/4$	$p(100\|ADDD) = 1/2$	$p(101\|ADADAD) = 1/4$
$p(110\|DDAD) = 1/4$	$p(111\|ADDD) = 1/4$	$p(011\|ADADAD) = 1/4$

The probabilities in table 2 can be derived with the help of the transition matrix T and the Markov model for the point-multiplication algorithm. We illustrate how to derive them on a simple example.

Example 1. Assume we want to calculate the probability that two consecutive elliptic-curve point-doubling operations occur (under the assumption that we only look at two transitions, i.e. $|Y| = 2$) when performing the scalar point-multiplication algorithm depicted in figure 2. Table 3 lists all possible transitions between two states of algorithm 2 in the leftmost column. The column in

the middle lists the to the sequence of transitions corresponding elliptic-curve operations. The rightmost column indicates the occurrence of two consecutive point-doubling operations.

Table 3. The calculation of the probability $p(X = DD)$ and $p(Y = 00|X = DD)$ can be done with this table. The leftmost part of the table shows the transitions, the column in the middle shows the corresponding elliptic-curve operations and the rightmost column indicates the occurrences of $X = DD$.

$0 \to 0 \to 1$	DAD	
$0 \to 0 \to 0$	DD	$\Leftarrow$
$0 \to 1 \to 0$	ADD	$\Leftarrow$
$0 \to 1 \to 11$	ADAD	
$1 \to 0 \to 0$	DD	$\Leftarrow$
$1 \to 0 \to 1$	DAD	
$1 \to 11 \to 11$	ADD	$\Leftarrow$
$1 \to 11 \to 0$	ADAD	
$11 \to 11 \to 11$	DD	$\Leftarrow$
$11 \to 11 \to 0$	DAD	
$11 \to 0 \to 0$	ADD	$\Leftarrow$
$11 \to 0 \to 1$	ADAD	

For each row of table 3 the corresponding probability can be calculated. This can be done by using the steady state vector and the probabilities which we already derived for the transition matrix. For example, the first row defines a sequence of transitions that starts in state 0, then stays in state zero and ends up in state 1. We know that the probability to be in state 0 is 1/2 and the probability to go from state 0 to state 0 or state 1 is 1/2. Thus, the probability that row one occurs is $(\frac{1}{2})^3$. To calculate $p(X = DD)$ one has to calculate the sum of the probabilities of the six rows which are marked with an arrow. To calculate the conditional probability $p(Y = 00|X = DD)$ one has to calculate the numerator of equation 1. This numerator can be calculated by summing up the probabilities of those three rows of the six marked rows that assume two consecutive zero bits (these are row two, row five and row eleven).

The Attack. The concrete attack works as follows.

1. **Precomputation phase:** Find the Markov model, i.e. the transition matrix and the steady state vector, for the given point-multiplication algorithm. Calculate the conditional probabilities for all combinations of X and Y up to a suitable $|Y|$.
2. *Data collection phase:* Deduce from the power trace of an elliptic-curve scalar point-multiplication operation the sequence of point additions and point doublings. This stage is in fact the same as in an ordinary simple power-analysis attack.

3. **Data analysis phase:** Split this sequence into a number of sub-sequences, whereby each sub-sequence must be chosen in a way that it can be produced by a well defined sequence of digits. The number of digits in the sequence have to sum up to the total number of digits of k to ensure a valid partitioning in sub-sequences. *Remark:* After this step one knows for each sub-sequence the number of digits that produced this subsequence. Of course in general, there are several valid possibilities for a such a partitioning, but it is only important to choose and fix one.
4. **Key testing phase:** Check all combinations of bit-patterns that have a non-zero probability to occur, with the known pair of plain- and ciphertext. This will lead finally to the secret key. *Remark:* One should check the combinations of bit-patterns with the highest probabilities first.

We now illustrate the attack on a toy example.

Example 2. In this example, we use the scalar point-multiplication algorithm depicted in figure 2 which we discussed on beforehand. The scalar k which serves as ephemeral key in this example is 560623. The first row shows the sequence of observed point-doubling and point-addition operations. In the second row the same sequence is split into sub-sequences that contain patterns for which we already calculated the conditional probabilities (see table 2). In the third, fourth and fifth rows the possible bit-patterns are listed according to their conditional probabilities. From all possible bit-pattern combinations the attacker can determine the correct one with the aid of the known plaintext-ciphertext pair.

Table 4. Example : $k = 1111011101100010001$, LSB first representation

ADADDDADADADDDADADADADDDADDDDAD							
ADAD	DDAD	ADAD	DDAD	ADAD	ADDD	ADDD	DAD
11	001	**11**	001	**11**	100	**100**	**01**
10	101	10	101	10	**000**	000	10
01	**110**	01	**110**	01	111	111	11

Effectiveness. We denote the ephemeral secret key with k, it's bit-length by n and the length of the sub-sequences by l. Then, in the worst case where we only take the non-zero conditional probabilities into account but not their actual values, we have to test $3^{3n/2l}$ keys on average (this is because $3n/2$ elliptic-curve operations are calculated on average in this point-multiplication algorithm). If we take into account, that one of the three non-zero conditional probabilities is always $1/2$, we deduce that we only have to test $2^{3n/2l}$ keys on average. In the case of $n = 163$ and $l = 16$ (the size of n is chosen according to the smallest recommend curve by NIST) we can deduce that we only have to test $2^{15.28}$ keys on average.

4.2 Application to the NAF-Method

This algorithm calculates the NAF of the multiplier k. The NAF of k has the property that no two consecutive digits in the expansion of k are non-zero. As a consequence, the average number of non-zero digits is reduced and fewer elliptic-curve operations have to be performed.

Table 5. The NAF algorithm

NAF-mult(P,M,k)
$Q = P$
Set $h = 3k$. Let h_l denote the most significant bit of h.
for $i = l - 1$ down to 1
$Q = Q * Q$
if $(h_i == 1)$ and $(k_i = 0)$ then $Q = Q + P$.
if $(h_i == 0)$ and $(k_i = 1)$ then $Q = Q - P$.
return Q

Table 5 gives a brief description of the NAF algorithm. The finite-state machine describing this algorithm is depicted in figure 3.

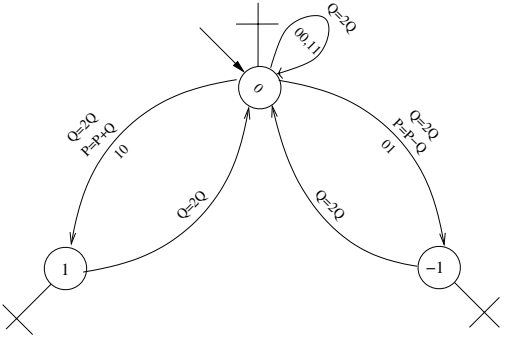


Fig. 3. Transition graph of the NAF algorithm. The label "00","01", "10" and "11" correspond to the values of h_i and k_i in table 5.

Like in the previous section, we can derive the transition matrix from this transition graph quite easily. The NAF property (i.e. that there are no consecutive non-zero digits allowed) is clearly visible in both the transition graph and the transition matrix. The steady-state vector for the transition matrix

$$T = \begin{pmatrix} 0.50 & 1 & 1 \\ 0.25 & 0 & 0 \\ 0.25 & 0 & 0 \end{pmatrix}$$

is $(4/6, 1/6, 1/6)$.

We derived once again a well known result about the distribution of zero and non-zero digits in the NAF-representation, which is, that about 2/3 of all NAF-digits are zeros. We also know immediately from figure 3 that we have an almost perfect and known statistical relationship between the NAF-digits and the elliptic-curve operations. If and only if a non-zero NAF-digit occurs, an elliptic-curve point-addition or an elliptic-curve point-subtraction, has to be performed.

This means, that we not need to calculate conditional probabilities to mount an attack. An attack can be mounted in the very straightforward way by just searching the power trace for occurrences of point-addition patterns. Each such pattern can be produced by either a point-addition or a point-subtraction operation. Since we know that about 1/3 of the NAF-digits are non-zero on average, we know that we would have to test about $2^{n/3}$ keys on average (n denotes the number of digits of the ephemeral key k).

4.3 A General Comment on the Key Testing Phase

The speed with which we can test all the possible ephemeral keys is also certainly an important issue for the whole attack. The scalar point-multiplications which we have to perform in the key testing phase can be done much faster than scalar point-multiplications with random points. This is because the point which is multiplied with a scalar is fixed, and due to elliptic curve protocols, almost never changed. This means that one can use window-methods combined with the NAF-method, for example, to speedup the key testing phase. We already know that because of the NAF property, not all of the bit-patterns (and digit-patterns resp.) are possible. In fact, when using a window-method for the point-multiplication in the key testing phase, one can use rather large windows, since not all, but only a very few points have to be precomputed. For example, for a window with length ten only 144 points need to be calculated. An analysis of window NAF-methods can be found for example in [BHLM01]. This means that in an EC-accelerator module which is designed especially for the key testing phase, one could possibly use such methods.

We now give some rough estimates for testing the number of possible keys if the NAF-method was used on a 163-bit curve. Our analysis in section 4.2 showed that we would have to test about $2^{54,3}$ possible ephemeral keys to determine the correct ephemeral key. Based on the performance data given in [OP00] it turns out that a single device needs appr. 2^{17} years to test all $2^{54,3}$ possible ephemeral keys. Modifying the architecture of [OP00] in such a way that it heavily makes use of precomputed points and thus achieves a further speed-up should be possible. Considering all this, we are concerned about the long-term security margin of the NAF-method in the case of a 163−bit curve.

4.4 A Note on the Application to Randomized Algorithms

In [OA01] the usage of randomized addition-subtraction chains was proposed as a countermeasure against simple and differential power-analysis attacks. We analyzed these randomized algorithms as well. It is clear that the attack as it is described in this paper can directly be applied on the randomized algorithms as well. One can derive the transition matrix for the randomized algorithms in exactly the the same way as we demonstrated it in this paper. The steady state can then be derived from the transition matrix and therefrom the conditional probabilities can be calculated. Due to the limitations of the number of pages for this paper we cannot present a full analysis, but in our investigations it turned out that the Markov method on the randomized algorithms is not better than on the NAF-method. Thus, we consider the randomized algorithms as more secure than the NAF-method with respect to the attack presented in this paper.

5 Conclusion

We presented an enhanced simple power-analysis attack on elliptic-curve point-multiplication algorithms. The method basically uses the information about the conditional probabilities of observed sequences of elliptic-curve operations, and bit-patterns that form the secret key. The method can be considered as an enhancement because it works even in cases where the standard simple-power attack fails. The approach is a general method in the sense that it can be used to attack arbitrary elliptic-curve point-multiplication algorithms that do not even necessarily fix the sequence of instructions. We demonstrated that it can be effectively applied on the example of a modification of a point-multiplication algorithm originally proposed by Morain and Olivos where the standard simple power-analysis attack fails. We also analyzed the security of the NAF-method which is often used in standards. We pointed out that this method applies to randomized algorithms as well. Based on the analysis we conjecture that methods that only use three digits for encoding the ephemeral key are susceptible to this attack. To summarize the results, the method is new and more efficient than the standard simple power-analysis attack and poses a serious threat against certain algorithms that only try to obscure the ephemeral key. Our analysis also indicates that the long-term security margin of the NAF-method in the case of a 163-bit curve is not too large.

Acknowledgments. We'd like to thank Mathijs Coster for his comments on Markov chains. Additionally, we'd like to thank the anonymous reviewers for their helpful comments and Vincent Rijmen for proofreading the final version of this paper.

References

[ANS99] ANSI. ANSI X9.62 Public Key Cryptography for the Financal Services Industry:The Elliptic Curve Digital Signatur Algorithm (ECDSA), 1999.

[BHLM01] Michael Brown, Darrel Hankerson, Julio Lopez, and Alfred Menezes. Software Implementation of the NIST Elliptic Curves Over Prime Fields. In *Progress in Cryptology – CT-RSA*, volume 2020 of *Lecture Notes in Computer Science*, pages 250–265. Springer, 2001.

[BJ02] E. Brier and M. Joye. Weierstrass Elliptic Curves and Side-Channel Attacks. In *Public Key Cryptography*, volume 2274 of *Lecture Notes in Computer Science*, page 335 ff. Springer, 2002.

[Cer00] Certicom Research. Standards For Efficient Cryptography – SECG 2: Recommended Elliptic Curve Cryptography Domain Parameters. Version 1.0, 2000. Available from http://www.secg.org/.

[Cor99] J.-S. Coron. Resistance against Differential Power Analysis for Elliptic Curve Cryptosystems. In *Workshop on Cryptographic Hardware and Embedded Systems – CHES 1999*, volume 1717 of *Lecture Notes in Computer Science*, pages 292–302. Springer, 1999.

[Gor98] D. M. Gordon. A survey of fast exponentiation methods. *Journal of Algorithms*, (27): pp. 129–146, 1998.

[GS92] Geoffrey Grimmett and David Stirzaker. *Probability and Random Processes.* Oxford University Press, 2nd edition, 1992. ISBN: 0198536658.

[Has00] M. A. Hasan. Power Analysis Attacks and Algorithmic Approaches to Their Countermeasures for Koblitz Cryptosystems. In *Cryptographic Hardware and Embedded Systems – CHES 2000*, volume 1965 of *Lecture Notes in Computer Science*, pages 93–108. Springer, 2000.

[HGS01] N. Howgrave-Graham and N. Smart. Lattice attacks on digital signature schemes. *Designs, Codes and Cryptography*, (23):283–290, August 2001.

[IEE99] IEEE. Standard Specifications for Public Key Cryptography, Annex A, D13, 1999.

[IT02] T. Izu and T. Takagi. Fast Parallel Elliptic Curve Multiplications Resistant to Side Channel Attacks. In *to appear in International Workshop on the Practice and Theory of Public Key Cryptography (PKC2002)*, Lecture Notes in Computer Science (LNCS). Springer, 2002.

[JQ01] M. Joye and J.-J. Quisquater. Hessian Elliptic Curves and Side-Channel Attacks. In *Cryptographic Hardware and Embedded Systems – CHES 2001*, volume 2162 of *Lecture Notes in Computer Science*, pages 402–410, 2001.

[KJJ99] P. C. Kocher, J. Jaffe, and B. Jun. Differential Power Analysis. In *Advances in Cryptology-CRYPTO 1999*, volume 1666 of *Lecture Notes in Computer Science*, pages 388–397. Springer, 1999.

[Knu98] D. E. Knuth. *The Art of Computer Programming. Seminumerical Algorithms*, volume 2. Addison-Wesley, 3rd edition, 1998.

[LS01] P.-Y. Liardet and N.P. Smart. Preventing SPA/DPA in ECC Systems Using the Jacobi Form. In *Cryptographic Hardware and Embedded Systems – CHES 2001*, volume 2162 of *Lecture Notes in Computer Science*, pages 391–401, 2001.

[MO90] F. Morain and J. Olivos. Speeding up the computation on an elliptic curve using addition-subtraction chains. *Inform. Theory Appl.*, (24):531–543, 1990.

[Möl01] B. Möller. Securing Elliptic Curve Point Multiplication against Side-Channel Attacks. In *Information Security – 4th International Conference, ISC 2001*, volume 2200 of *Lecture Notes in Computer Science (LNCS)*, page 324 ff. Springer, 2001.

[NIS99] NIST. Recommended Elliptic Curves For Federal Government Use, 1999.

[OA01] E. Oswald and M. Aigner. Randomized Addition-Subtraction Chains as a Countermeasure against Power Attacks. In *Cryptographic Hardware and Embedded Systems – CHES 2001*, volume 2162 of *Lecture Notes in Computer Science (LNCS)*, pages 39–50. Springer, 2001.

[OP00] G. Orlando and Ch. Paar. A high performance reconfigurable elliptic curve processor for gf(2^m). In *Cryptographic Hardware and Embedded Systems – CHES 2000*, volume 1965 of *Lecture Notes in Computer Science*, pages 41–56. Springer, 2000.

[OS00] K. Okeya and K. Sakurai. Power Analysis Breaks Elliptic Curve Cryptosystems even Secure against the Timing Attack. In *Progress in Cryptology – INDOCRYPT 2000*, volume 1977 of *Lecture Notes in Computer Science (LNCS)*, pages 178–190. Springer, 2000.

[QS01] Jean-Jacques Quisquater and David Samyde. ElectroMagnetic Analysis (EMA): Measures and Counter-Measures for Smart Cards. In *Smart Card Programming and Security, E-smart 2001*, volume 2140 of *Lecture Notes in Computer Science*, pages 200–210. Springer, 2001.

[RS01] T. Römer and J.-P. Seifert. Information leakage attacks against Smart Card implementations of the Elliptic Curve Digital Signature Algorithm. In *Smart Card Programming and Security (E-Smart 2001)*, volume 2104 of *Lecture Notes in Computer Science (LNCS)*, page 211 ff. Springer, 2001.

Implementation of Elliptic Curve Cryptography with Built-In Counter Measures against Side Channel Attacks

Elena Trichina[1] and Antonio Bellezza[2]

[1] Cryptographic Design Center, Gemplus Technology R & D
Via Pio Emanuelli 1, 00143 Rome, Italy
elena.trichina@gemplus.com
[2] University of Rome "La Sapienza", Rome, Italy
abellezza@tiscalinet.it

Abstract. Many software implementations of public key cryptosystems have been concerned with efficiency. The advent of side channel attacks, such as timing and power analysis attacks, force us to reconsider the strategy of implementation of group arithmetic. This paper presents a study of software counter measures against side channel attacks for elliptic curve cryptosystems.
We introduce two new counter measures. The first is a new implementation technique, namely, homogeneous group operations, which has the property that addition and doubling on elliptic curves cannot be distinguished from side channel analysis. Being inexpensive time-wise, this technique is an alternative to a well-known Montgomery ladder. The second is a non-deterministic method of point exponentiation with precomputations. Although requiring rather large ROM, it provides an effective resistance against high-order power analysis attacks for the price of index re-computations and ROM accesses.
An experimental implementation of NIST-recommended elliptic curves over binary fields with a balanced suite of counter measures built-in in group arithmetic is presented, and the penalty paid is analyzed. The results of the implementation in C on an AMD Duron 600 MHz running Linux are included in the paper.

1 Introduction

With the advance of *side channel attacks* both hard- and software implementations of cryptosystems have to take into account various protection techniques. It has been claimed that all "naive" implementations can succumb to attacks by power analysis. In this paper we do not detail the principles and techniques of different side channel attacks; it had been done elsewhere [17,18,22,10,23]. Instead, we survey and systematize different software counter measures against such attacks, including a couple of new techniques, and suggest a balanced approach to their implementation in the context of elliptic curve cryptosystems (ECC) in binary fields. Giving a due to other excellent papers on practical realizations of ECC [29,31,19], we claim that the contribution of our paper is an

B.S. Kaliski Jr. et al. (Eds.): CHES 2002, LNCS 2523, pp. 98–113, 2003.

attempt to extend the work in [11] to an implementation in which a balanced suite of counter measures against side channel attacks have been built-in on a group operation level.

An implementation of an elliptic curve cryptosystem involves many choices; not only domain parameters (e.g., underlying finite fields, field representations, particular elliptic curves) and group arithmetic, but also field arithmetic, which, in its turn, includes many different methods and algorithms. It is not surprising therefore that although numerous papers on various aspects of ECC implementations had been written, there are only a handful of those that constitute an extensive and careful study that includes all factors involved in the efficient software implementation, notably [11] and [4]. These papers present C implementations on the Pentium 400 MHz workstation of the NIST-recommended elliptic curves over binary and prime fields, respectively.

All published so far complete implementations of ECC have been concerned with efficiency. The advent of side channel attacks force us to reconsider the development cycle. As had been demonstrated in a break-through work by Kocher et al. [17,18], for real life implementations an attacker can be able to perform measurements on the device. If the different operations present different characteristics detectable from the outside (like power consumption profile or duration), the sequence of operations can be discovered by the attacker.

In the most widely used cryptographic systems, an *exponentiation* with a secret exponent is required at some stage. "Exponentiation" in this context is the repeated application of a group operation to the same element. Efficient exponentiation algorithms involve a sequence of squarings, multiplications and possibly divisions (respectively doublings, additions and possibly subtractions in additive notation). Finding out the sequence of operations gives information on the exponent and in some cases uniquely determines it. A detailed systematic description of power analysis attacks on modular exponentiation algorithms can be found in the work of Messerges et al. [22].

Coron [7] generalized DPA attacks to elliptic curve cryptosystems and had shown how to extend DPA to any scalar multiplication algorithm.

Since then, a plethora of papers suggesting various counter measures both, in a general setting and for particular classes of elliptic curves, has been published. We set upon identifying a set of sufficiently general protection techniques that can be employed in a real life scenario. In what follows we survey algorithmic counter measures suggested in the literature, and introduce some original methods. After which an outline of the ECC software implementation in binary fields with a balanced choice of counter measures against side channel attacks is discussed. A description of the experiment with a C implementation on a PC with AMD Duron 600 MHZ running Linux, concludes the paper.

2 Preliminaries and Notation

Since we are interested in elliptic curves, the notation is additive. Thus the basic operations are doubling, addition and subtraction. We focus on elliptic curves over finite fields of characteristic 2, which have an affine equation of the form

$$y^2 + xy = x^3 + ax^2 + b$$

for two elements a and b in the base field. A point on the curve is either identified by a couple of affine coordinates (P_x, P_y) or is the special point P_∞. If P and Q are points on the curve, then an addition $\oplus$ is defined so that $P \oplus Q$ is also a point on the curve and P_∞ is the neutral element. Thus one can also define a subtraction $P \ominus Q$. In what follows, notation from [2] is used.

Let $P_1 = (x_1, y_1)$ and $P_2 = (x_2, y_2)$ be points on the curve. Assume $P_1, P_2 \neq P_\infty$ and $P_1 \neq -P_2$. The sum $P_3 = (x_3, y_3) = P_1 \oplus P_2$ is computed as follows.

$$x_3 = \lambda^2 + \lambda + x_1 + x_2 + a, \qquad y_3 = (x_1 + x_3)\lambda + x_3 + y_1,$$

where $\lambda = (y_1 + y_2)/(x_1 + x_2)$ if $P_1 \neq P_2$, and $\lambda = (y_1)/(x_1) + x_1$ if $P_1 = P_2$.

Each point has also a projective expression of the form (P_X, P_Y, P_Z). If the point is P_∞ then its projective coordinates are $(1, 1, 0)$. Else they are related to the affine coordinates via the transformation rules

$$(P_x, P_y) \mapsto (P_x, P_y, 1)$$
$$(P_X, P_Y, P_Z) \mapsto (P_X/P_Z^2, P_Y/P_Z^3).$$

A curve then is specified by the equation $Y^2 + XYZ = X^3 + aX^2Z^2 + bZ^6$. Doubling formulas $(X_3, Y_3, Z_3) = 2(X_1, Y_1, Z_1)$ for the projective equation is:

$$Z_3 = X_1^2 Z_1^2, \qquad X_3 = X_1^4 + bZ_1^4, \qquad Y_3 = bZ_1^4 Z_3 + X_3(aZ_3 + Y_1^2 + bZ_1^4).$$

Addition $(X_3, Y_3, Z_3) = (X_1, Y_1, Z_1) + (X_2, Y_2, 1)$ in mixed coordinates is:

$$A = Y_2 Z_1^2, B = X_2 Z_1 + X_1, C = Z_1 B, D = B^2(C + aZ_1^2), Z_3 = C^2, E = AC,$$

$$X_3 = A^2 + D + E, \quad F = X_3 + X_2 Z_3, \quad G = X_3 + Y_2 Z_3, \quad Y_3 = EF + Z_3 G.$$

ECC is based on the general problem of exponentiation in Abelian groups. Various algorithms for exponentiation in the context of cryptography are presented in [21,2]. The survey [8] describes fast methods, including those specialized for elliptic curves. Due to a short bit-length of the exponent, and that it is often generated on-the-fly, sophisticated techniques do not show their advantage. However, certain idiosyncrasies of elliptic curves (e.g., subtraction and addition are almost identical), prompt us to consider together with a classical binary method signed and non-adjacent form (NAF) representation of the exponent [21].

Some algorithms described in this paper work with different exponentiation procedures. In order to be able to interchange them, we rely on a routine that transforms an integer into a sequence indicating the order of operations to perform. Possible contents of the resulting sequence elements are D (double), A (add), S (subtract), suitably encoded.

3 Taxonomy of Software Counter Measures

Three main techniques developed by Kocher et al., are timing attacks [17], and simple (SPA) and differential power analysis (DPA) attacks [18]. Among them,

the DPA is the most sophisticated; nevertheless, there are references in the literature when a DPA-protected implementation could be SPA-vulnerable [9]. Moreover, for ECC it seems more difficult to prevent SPA than DPA, as there are good solutions for the latter[7,15,22].

Side channel attacks work because there is a correlation between the physical measurements taken during computations (e.g., power consumption, computing time, EMF radiation, etc.) and the internal state of the processing device, which is itself related to the secret key. While some authors insist on so-called *provable resistance* [6,9] to specific classes of attacks, we share an attitude of [5] and state that, unlike in classical cryptology, in a case of side-channel attacks the definition of an adversarial model is not absolute. Thus, rather than having a *proof* of security, one can have only a strong *evidence* that the countermeasure resists an attack. Note that the security is assessed at the implementation level.

Three general algorithmic techniques to defy side channel attacks are:

1. *Indistinguishability*, which basically means that the only strategy for an adversary is to guess at random. It can be partially achieved by eliminating disparity between group operations involved in exponentiation.
2. *Random splitting of every bit* of the original computation into k shares where each share is equiprobably distributed. Computation then will be carried securely by performing computation only on shares, without reconstructing the original bit [6,9]. In practice, an implementation of this technique for ECC amounts to blinding an exponent or/and a base point with some random mask generated for every run.
3. *Randomization of the execution sequence* which, in order to be successful, must be carried out extensively [6] and, ideally, supported by hardware [27].

In what follows we systematically describe a wide range of counter measures proposed so far for ECC in a general setting.

3.1 Indistinguishability

The first requirement to an effective protection against *timing and SPA* attacks is to ensure that an instruction performed during an execution of a cryptographic algorithm should not be directly dependent on the data being processed.

Double-and-Add Always. Introduced in [7], this counter measure eliminates branch instructions, conditioned by secret data.

```
Algorithm 1 (Double-and-add) ==> Algorithm 2 (Double-and-add always)
Input: P, d=(d_m-1,...,d_0)      Input: P, d=(d_m-1,...,d_0)
  Q <- P                           Q[0] <- P
  for i from m-1 down to 0 do      for i from m-2 down to 0 do
    Q <- 2Q                          Q[0] <- 2Q[0]
    if d_i=1 then Q <-- Q + P        Q[1] <- Q[0] + P
  output Q                           Q[0] <- Q[d_i]
                                   output Q[0]
```

This method increases by 33% the amount of field operations by necessitating "dummy" computations. As has been shown in [28], it makes it susceptible to a new type of attacks, where an attacker induces any temporary random fault into the ALU during the execution of $Q[1] \leftarrow Q[0] + P$ at iteration i. Then, according to whether the final result is incorrect or not, the attacker may deduce if the d_i bit of the exponent is 0 or 1!

The Universal Addition on Elliptic Curves. This method was suggested in [20] for *projective coordinates* for a subclass of elliptic curves in finite fields of characteristic greater than 3. The approach involves a representation of an elliptic curve as the intersection of two quadric surfaces in P^3. A non-standard definition of point addition has the advantage that the set of equations holds when two points are equal, thus, naturally eliminating the difference between point addition and doubling. The price to pay is that every point addition requires 16 field multiplications and 3 squarings. Even after various optimizations (that involve the Jacobi representation and a windowing method, and require additional memory), a total increase in computation costs is 70% in comparison with a standard projective coordinate method.

Independently, a similar approach was developed for Hessian curves in [14], where due to the high symmetry of the Hessian parameterization, the same algorithm can be used for point addition and point doubling. An implementation of point addition requires 12 field multiplications, providing 33% improvement over Jacobian curves.

Unfortunately, both Jacobian and Hessian parameterizations are not fully general. Brier and Joye later extended the technique in [3] to general Weierstrass elliptic curves and come up with the unified formulae, which requires 7 multiplications, 1 inversion and 3 squarings for affine coordinates and 18 multiplications for projective. This is very expensive in comparison with the method proposed in this paper while providing the same protection.

A Montgomery Ladder. The advantages of Montgomery scalar multiplication [24] in the context of elliptic curves were re-discovered several times [19,25, 13,16]. The Montgomery ladder is based on the binary method and the observation that the x-coordinate of the sum of two points whose difference is known, can be computed in terms of only x-coordinates of the involved points. Incidentally, it is also the most promising candidate for a"side channel indistinguishable" exponentiation. The Montgomery ladder protected with a "no-branches" technique is given below.

```
Algorithm 3 (Montgomery method) => Algorithm 4 (Modified Montgomery)
Input: P, d=(d_m-1,...,d_0)        Input: P, d=(d_m-1,...,d_0)
Output: dP                         Output: dP
  P[1] <- P                          P[1] <- P
  P[0] <- 2P                         P[0] <- 2P
  for i from m-2 down to 0 do        for i from m-2 down to 0 do
    if d_i=1 then                      b <- (1-d_i)
```

```
    P[1] <- P[1]+P[0]                P[b] <- P[0]+P[1]
    P[0] <- 2P[0]                    P[d_i] <- 2P[d_i]
  else
    P[0] <- P[1]+P[0]              return P[1]
    P[1] <- 2P[1]
return P[1]
```

The advantage of this method is perfectly symmetric, memory-efficient computations. The disadvantage is increase in computation costs since both, point addition and point doubling, are computed in each iteration.

An optimization of the Montgomery method based on a new formulae for computing x-coordinate for addition of two points, was suggested in [19]. In affine case, a new formulae requires 2 field inversions, 2 multiplications, 2 squarings and 4 additions per iteration; for projective coordinates, it involves only 6 field multiplications, 6 squarings, and 3 additions per iteration. This is comparable cost-wise with our homogeneous group operation method described later, but has an advantage of using less memory, since y-coordinate is not used.

Universal Exponentiation Algorithm. Introduced in [5], this is an elegant attempt to design a *provably* SPA-resistant exponentiation method. Using a representation of the exponent with addition chains, the authors "transfer" the security of the exponentiation method actually implemented in the exponent itself (i.e., in a secret data). The requirement here is that an exponent must be represented as an addition chain in a secure environment. The algorithm works with virtually all exponentiation methods. It reads triplets of values $(\gamma(i) : \alpha(i) : \beta(i))$ meaning that the content of register $R[\alpha(i)]$ must be multiplied by (added to) the content of register $R[\beta(i)]$ and the result must be written into register $R[\gamma(i)]$. The exponent d then is represented by the register sequence $\Gamma(d) = \{(\gamma(i) : \alpha(i), \beta(i))\}_{0 \le i \le m-1}$. In order to break the algorithm, an adversary must be able to differentiate among the triplets and to recover all their values. The advantage of this method is that it introduces only a small amount of extra computations; the disadvantage is large memory usage.

3.2 Splitting Variables

In practice, the general encoding proposed in [6,9] amounts to blinding an exponent and a base point with some random mask to prevent *DPA* attacks.

Blinding an Exponent. Randomization of the exponent is inexpensive and useful technique. It comes in few flavors.

An additive mask is usually used in the ECC context [7]. It consists in adding a multiple of the element order to the exponent. If N is the group order, then $(kN)P = 0$ for any integer k and for any element P in the group. To compute dP, where d is the secret exponent and P is a base point, one first blinds d

with kN, where k is a (small) random number generated for every run, and N is the group order. Hence, $Q = (d + kN)P = dP + (kN)P = dP$. As d usually is the size of the group order, choosing a t-bit integer k increases the size of the exponent by about t bits.

An exponent splitting [5] is a variant of this technique, where d is represented as a sum of a random k and $d-k$, and exponentiation is carried out in two steps: $R \rightarrow kP; \quad Q \rightarrow (d-k)R$. The values of both, k and $d-k$ are required to recover the value of d, which makes it less attractive than the method above.

A multiplicative mask is a multiplicative analogue of this idea. Let N be the group order. If k is a unit in the multiplicative group $(\mathbf{Z}/\mathbf{Z}N, \times)$ and k^{-1} its inverse, then $d = k(k^{-1}d)(\mod N)$ for any exponent d. This means that calling $d' := k^{-1}d(\mod N)$, for any group element P one has the following equivalent exponentiations:

$$(1) \quad P \overset{d}{\mapsto} dP \qquad\qquad (3) \quad P \overset{d'}{\mapsto} d'P \overset{k}{\mapsto} k(d'P)$$

$$(2) \quad P \overset{k}{\mapsto} kP \overset{d'}{\mapsto} d'(kP) \qquad\qquad (4) \quad P \overset{kd'}{\mapsto} (kd')P$$

Sequence (1) is a straightforward exponentiation; sequence (4) is a sub-case of the additive blinding technique. Sequences (2) and (3) are performed in two phases. We focus on (2). The overhead is given by the exponentiation with exponent k, since both d and d' are on average the size of N. To keep overhead low, one can choose k to yield a fast exponentiation, for instance by choosing it randomly among the elements of $(\mathbf{Z}/\mathbf{Z}N)^*$ of at most t bits, with a small t. As in the additive blinding, one can trade-off the degree of randomization (and thus the security) for speed.

If sequence (2) is used, the element fed to the second phase is not controlled by an attacker. The first phase, on the other hand, doesn't leak any information whatsoever on the secret exponent, depending uniquely on the input element and the random integer k. Thus one gets at the same time additional bonuses of a change in the sequence of point operations and of randomization of the input point. Cost-wise, the method is comparable with the additive mask, although precomputations are somewhat slower.

Blinding a Base Point. Blinding a base point is necessary if we assume that an attacker knows how points are represented in memory during computations [7]. An accepted technique to mask the input by applying some bijective function for which it is easy to compute the final correction.

Adding to the input a "perturbation point" [7] involves storing a couple of "random" points $(R, -dR)$ and updating it at each run by doubling both elements. Exponentiation of an input P with an exponent d is performed in three steps: $(1)P' \leftarrow P + R, \quad (2)Q' \leftarrow dP' = dP + dR, \quad (3)Q \leftarrow Q' + (-dR) = dP.$

As the argument $P + R$ of the exponentiation phase does not depend exclusively on the input, it is not possible to choose points with particular properties. However, if d changes at each run, computing $-dR$ for final correction doubles group operations, and must be protected as well.

Randomizing projective coordinates of a point [7] is achieved almost for free. Projective coordinates of a point is not unique because $(X, Y, Z) = (kX, kY, kZ)$ for every $k \neq 0$ in the finite field. Hence, before each new execution of dP, the projective coordinates of P can be randomized with a random k. It makes it impossible for an attacker to predict any specific bit of the binary representation of P, thus rending a DPA attack infeasible.

3.3 Randomization of the Execution Sequence

Non-deterministic execution of the sequence of instructions, although does not provide perfect protection against statistical techniques, increases the number of measurements.

Randomized Addition/Subtraction chain. [26] exploits the fact that dP can be computed with different execution sequences; for instance, for $9P$: $P \to 2P \to 4P \to 8P \to 9P$ or $P \to 2P \to 4P \to 5P \to 10P \to 9P$. Randomization can be achieved by inserting a random decision that can rearrange a sequence of additions, subtractions and doubling in a signed binary algorithm. The idea is to randomly substitute a long chain of 1's (which correspond to "double-and-add") in the signed binary representation of the exponent by a block of zeros ("double") followed by -1 (subtract). Similar substitution is applied to isolated 0's inside a block of 1's. The algorithm needs some 9% more additions. The authors claim that it makes DPA attacks really difficult.

4 Homogeneous Group Operations

In the choice of doubling and addition algorithms for elliptic curves, it is often advisable to work in projective coordinates, which allows us to avoid costly inversions in the field. For elliptic curves over fields of characteristic 2, complexities for doubling performed on projective coordinates and addition and subtraction on mixed affine-projective coordinates with projective result, are: Here $Mult$ stands for multiplication, $Squar$ for squaring, and Add for addition.

The problem we address here is the relevant computational difference between addition/subtraction and doubling, which in implementation reflects in differences detectable by an attacker. Observing that the complexity for point doubling is about half of the complexity for addition/subtraction, one can think of splitting the complex subroutines into two parts, each one approximately equivalent to a point doubling. This is actually possible, as we show below.

When choosing an elliptic curve, setting $a = 1$ makes the total number of products in addition and subtraction exactly twice the number for doubling,

Table 1. Complexities of group operations in projective coordinates

	General case			**Case $a = 1$**		
Curve operation	Mult	Squar	Add	Mult	Squar	Add
Doubling	5	5	4	5	5	4
Addition	11	3	7	10	3	7
Subtraction	11	3	8	10	3	8

thus avoiding dummy operations. We present atomic units of computations in the case $a = 1$. A general case slightly differs in the addition/subtraction part, while the doubling subroutine is the same.

Before proceeding, we observe that computing the subtraction $P \ominus Q$ of two points corresponds to computing $P \oplus (-Q)$. If Q has affine coordinates (Q_x, Q_y), then $-Q$ has affine coordinates $(Q_x, Q_y + Q_x)$. Thus, the algorithms for addition and subtraction only differ by a field addition, and we will consider them at the same time. Furthermore, if the affine point to be subtracted is the same throughout a loop, one can compute $Q_x + Q_y$ once. Here are the subroutines.

Doubling. The input is given by the coordinates (A_X, A_Y, A_Z) of the point.

$$
\begin{array}{lll}
\lambda_1 \longleftarrow A_Z^2 & \lambda_6 \longleftarrow \widetilde{b}\lambda_1 & \lambda_{11} \longleftarrow \lambda_7^4 \\
\lambda_2 \longleftarrow A_X\lambda_1 & \lambda_7 \longleftarrow A_X + \lambda_6 & \lambda_{12} \longleftarrow \lambda_{10}\lambda_{11} \\
\lambda_3 \longleftarrow A_Y A_Z & \lambda_8 \longleftarrow \lambda_4^2 & \lambda_{13} \longleftarrow \lambda_9 + \lambda_{12} \\
\lambda_4 \longleftarrow A_X^2 & \lambda_9 \longleftarrow \lambda_8\lambda_2 & \\
\lambda_5 \longleftarrow \lambda_2 + \lambda_4 & \lambda_{10} \longleftarrow \lambda_5 + \lambda_3 &
\end{array}
$$

Here $\widetilde{b}$ is such that $\widetilde{b}^4 = b$. It can be computed once and for all when the curve is set. The output point has projective coordinates $(\lambda_{11}, \lambda_{13}, \lambda_2)$. The sequence of operations is SMMSAMASMASSMA.

Addition/Subtraction – Phase I (Case $a = 1$). The input is given by projective coordinates (A_X, A_Y, A_Z) and affine coordinates (B_x, B_y).

$$
\begin{array}{lll}
\mu_1 \longleftarrow \lambda_7^2 & \lambda_4 \longleftarrow B_y \quad (addition) & \lambda_7 \longleftarrow A_Z\lambda_6 \\
\lambda_1 \longleftarrow A_Z^2 & \quad or & \lambda_8 \longleftarrow A_Y + \lambda_5 \\
\lambda_2 \longleftarrow \lambda_1 B_x & \lambda_4 \longleftarrow B_y + B_x \quad (subtraction) & \lambda_9 \longleftarrow \lambda_8 B_x \\
\lambda_3 \longleftarrow \lambda_1 A_Z & \lambda_5 \longleftarrow \lambda_4\lambda_3 & \lambda_{11} \longleftarrow \lambda_7 + \lambda_8 \\
 & \lambda_6 \longleftarrow A_X + \lambda_2 &
\end{array}
$$

The sequence of operations is SMMAMAMAMA.

Addition/Subtraction – Phase II (Case $a = 1$). This subroutine is executed right after the previous one, of which it uses some of the partial results.

$$\begin{array}{lll} \mu_1 \longleftarrow \lambda_7^2 & \mu_5 \longleftarrow \mu_1 + \mu_3 & \mu_9 \longleftarrow \lambda_9 + \mu_2 \\ \mu_2 \longleftarrow \lambda_7\lambda_4 & \mu_6 \longleftarrow \lambda_6\mu_4 & \mu_{10} \longleftarrow \mu_9\mu_1 \\ \mu_3 \longleftarrow \lambda_8\lambda_{11} & \mu_7 \longleftarrow \mu_5 + \mu_6 & \mu_{11} \longleftarrow \mu_8 + \mu_{10} \\ \mu_4 \longleftarrow \lambda_6^2 & \mu_8 \longleftarrow \mu_7\lambda_{11} & \end{array}$$

The output point has projective coordinates $(\mu_7, \mu_{11}, \lambda_7)$.The sequence of operations is SMMSAMAMAMA.

The shortest operation sequence (including a couple of dummy operations) for all three subroutines is $SMMSAMASMASSMA$.

Table 2. Complexities of homogeneous vs normal group operations

	General case			**Case** $a = 1$		
Curve operation	Mult	Squar	Add	Mult	Squar	Add
Doubling (old)	5	5	4	5	5	4
Doubling (new)	6	5	4	5	5	4
Addition (old)	11	3	7	10	3	7
Addition (new)	12	10	8	10	10	8
Subtraction (old)	11	3	8	10	3	8
Subtraction (new)	12	10	8	10	10	8

Table 2 gives the complexities for various options. Observe that squarings and additions in the field of characteristic 2 are very cheap. Thus we are wasting little more than one field multiplication per curve operation in the general case, and just a few additions and squarings in the case $a = 1$.

An Exponentiation Algorithm Using Homogeneous Point Operations. The classical algorithms can be used to implement an exponentiation that shows to the outside world a sequence of homogeneous rounds. The subroutines for doubling, addition-phase I, subtraction-phase I and addition/subtraction-phase II have been implemented in C. In order to achieve homogeneity, they are invoked with the same kind of parameters, i.e. with 5 field elements representing the projective and affine coordinates of two curve points. The result is a point in projective coordinates, i.e. a triplet of field elements.

Phase I of addition and subtraction returns the first input point (the one in projective coordinates). The subroutines are called F_i with i being 0 for doubling, 1 for addition/I, 2 for subtraction/I and 3 for addition-subtraction/II. F_1 and F_2 also have side-effects, namely they store in memory partial results which are used by F_3. We suppose that at the precomputation step, the exponent is transformed in a secure environment into a sequence b[1], ..., b[d] corresponding to procedure calls.

```
Algorithm 5 (Exponentiation with homogeneous group operations)
Input: point P=(P_x,P_y) in affine coordinates,
       sequence b[1],...,b[d] of atomic unit calls for exponent d
Output: point dP in affine coordinates
  (S_X,S_Y,S_Z) <- (1,1,0) (initialise the result)
   for i from 1 to d-1 do
      (S_X,S_Y,S_Z) <- F_{b[i]}(S_X,S_Y,S_Z,I_x,I_y)
Final conversion
    S_x <- S_X/S_Z^2
    S_y <- S_Y/S_Z^3
Return the point with affine coordinates (S_x,S_y)
```

5 Non-deterministic Right-to-Left Method with Precomputations

Non-deterministic right-to-left method with precomputations had been actually implemented in our experiment. It is very efficient but requires a lot of ROM.

An algorithm is based on the observation that $dP = d_{m-1}2^{m-1}P + ... + d_1 2P + d_0 P$, where $(d_{m-1}, ..., d_1, d_0)$ is a binary representation of the exponent d, d_0 being the LSB, and d_{m-1} the MSB of d. Thus, if the multiples of point P are precomputed in advance, it does not matter in which order the bits of the exponent d are scanned as long as one can associate the bit d_i with the precomputed multiple $2^i P$. Here is our solution.

```
Algorithm 6 (Non-deterministic right-to-left with precomputations)
Input: array A=[2^{m-1}P,...,2P, P] of precomputed multiples of P
       d = (d_{m-1},...,d_1, d_0) bits of an exponent
Output: dP
  M <- 0                      -- initialise M to zero
  W <- permute([m-1, ...,0]) -- W contains randomly permuted indices
  for all elements j from W do
    M <- M + d_{W[j]}*A[W[j]]
  return M
```

Instead of permutation, one can just rotate A and d simultaneously j positions to the left or to the right; $0 \leq j \leq m-1$ is a random updated at each run. To enhance the randomization, the method of random splitting and rotating can be applied recursively to the sub-arrays of indices. This method is beneficial for protocols where the base point P does not change often. However, it requires additional ROM to store precomputed multiples of P. Considering that the size of field elements for the ECC is around 200 bits, the total amount of memory in question is $25 \times \log N$ bytes, where N is a group order.

6 A Point of a Small Order as a Perturbation Point

A point of a small order on the curve as a perturbation point is a variation on the idea of blinding a base point. Elliptic curves of use for cryptographic

purposes have a group of points with order a big prime N times a cofactor h, where h is usually small. We suggest choosing as the perturbation point R a point of order h. We suppose the cofactor not to be too small (a cofactor of 2 would be no good) but such that its bit-length l_h is not bigger than, say, 1/5 of bit-length l_N of N.

This presents some disadvantages but allows on-the-fly computation of the point and the correction at each run with much fewer group operations. To be useful, a cofactor h has to be at least 4. Although [1] contains an example with the bit-length $l_h = 16$ and $l_N = 161$, the trend to have a very small cofactor weakens our method.

We compute a random point of order h as the multiple of Q by a random factor r between 0 and $h - 1$. The final correction to apply is $T := -drQ$. We can actually compute it as $(-dr \mod h)Q$. The factor is again between 0 and $h - 1$. As the bit-length of h is considerably less than the bit-length of N, the exponentiations to get R and T are faster than the main exponentiation and require some l_h additions. The computation of rQ and $(-dr \mod h)Q$ must be masked. As r changes after each run, one has to take care mostly of single-run analysis.

One can speed up and simultaneously mask computations of rQ and $(-dr \mod h)Q$ by storing $2Q, 4Q, \ldots, 2^{l_h-1}Q$ and using fast right-to-left binary algorithm in a non-deterministic manner, as described in the previous chapter. This technique had actually been implemented.

7 Computer Experiments

We implemented some of the ideas explained in the previous chapters. For finite fields, we implemented in C basic operations from scratch following [11].

Field Arithmetics. As polynomial basis, we followed [11] and [1], choosing the following irreducible polynomials for the fields in our experiments.

$$\mathbf{F}_{2^{163}} = \mathbf{F}_2[x]/(x^{163} + x^7 + x^6 + x^3 + 1)$$
$$\mathbf{F}'_{2^{163}} = \mathbf{F}_2[x]/(x^{163} + x^8 + x^2 + x + 1)$$
$$\mathbf{F}_{2^{233}} = \mathbf{F}_2[x]/(x^{233} + x^{74} + 1)$$
$$\mathbf{F}_{2^{283}} = \mathbf{F}_2[x]/(x^{283} + x^{12} + x^7 + x^5 + 1)$$

For multiplication right to left and left to right comb methods, plus a base 16 left to right comb method were implemented. As the inversion algorithm we chose the extended Euclidean algorithm. Sample timings taken on an AMD Duron 600 MHz running Linux for selected fields are given in Table 3.

Exponentiation on Elliptic Curves. We implemented both binary and signed NAF exponentiation for elliptic curves over $F_{2^{163}}$, choosing a curve E_1 with $a = 1$ and a general curve E_2 with the parameters specified below.

Table 3. Timing for field operations in μs for various fields

Operation	$\mathbf{F}_{2^{163}}$	$\mathbf{F}'_{2^{163}}$	$\mathbf{F}_{2^{233}}$	$\mathbf{F}_{2^{283}}$
Addition	0.16	0.16	0.17	0.16
Squaring	0.34	0.37	0.38	0.53
Product: RL comb	7.00	7.13	9.86	18.76
Product: LR comb	10.04	10.10	14.93	18.72
Product: Base 16 LR comb	2.78	2.80	3.61	4.59
Inversion	36.02	36.05	60.50	77.97

Point operations were performed using the standard sequence of operations and the homogeneous versions. Table 4 summarizes the penalty for a side-channel protection. To get the following timings, the generating points were multiplied by a random exponent both with standard and homogeneous operations. For E_1 the fact that $a = 1$ was taken into account in both versions.

As for a real-life scenario, a suite of built-in counter measures included:

- indistinguishability via homogeneous operations (3-17% overhead);
- a multiplicative exponent masking, where the first step (multiplication with a small random k) had been implemented via a non-deterministic right-to-left binary method with precomputations, providing an additional benefit of blinding a base point in a process (15% overhead).

The total costs amounted to 2133 field multiplications, 1773 squarings, 1416 additions and 3 inversions, while a non-protected version would have had 1683 multiplications, 1121 squarings, 1200 additions and 1 inversion. In other words, some 30% increase in computation time constitutes the total penalty.

Elliptic curve parameters	
E_1	
F	$\mathbf{F}_2[x]/(x^{163} + x^7 + x^6 + x^3 + 1)$
a	01
b	02 0A601907 B8C953CA 1481EB10 512F7874 4A3205FD
N	04 00000000 00000000 000292FE 77E70C12 A4234C33
h	02
Q_x	03 FB02D922 0A5E7980 D9C7C192 AFC7EDC4 19B261E4
Q_y	05 F8692B70 5F82AAF2 7E41D4D3 82D9E359 98979F99
E_2	
F	$\mathbf{F}_2[x]/(x^{163} + x^8 + x^2 + x + 1)$
a	07 2546B543 5234A422 E0789675 F432C894 35DE5242
b	00 C9517D06 D5240D3C FF38C74B 20B6CD4D 6F9DD4D9
N	04 00000000 00000000 000292FE 77E70C12 A4234C33
h	02
Q_x	07 AF699895 46103D79 329FCC3D 74880F33 BBE803CB
Q_y	06 434AB98E 1F769093 2FA04BCA 9ED0479D 4B5FC954

8 Conclusion

From our experiment it follows that the cost of counter measures against side channel attacks need not to be prohibitively high. With a choice of well-balanced suite of complementary counter measures the total overhead can be as low as 30%. We want to emphasize that the objective of the experiment was an

Table 4. Exponentiation timings in μs

	binary	**NAF**		**binary**	**NAF**
E_1			E_2		
standard	5355	4699	standard	5642	4932
homogeneous	5512	4847	homogeneous	6581	5770
overhead	2.9%	3.1%	overhead	16.6%	17.0%

"average-case" analysis, rather than finding the fastest or the most secure solution. This explains, for example, the choice of the field $GF(2^n)$ and the field representation. It is well-known that operations in $GF(2^n)$ can be efficiently implemented in both, hardware and software, and that, although the normal basis representation offers a very fast squaring, the polynomial representation is more widely used.

The choice of projective coordinates for ECC allows us to use homogeneous group operations, thus achieving SPD resistance with only 3–17% overhead. One can argue that using projective coordinates is a good idea only in prime fields, but not in $GF(2^n)$ [30]. However, the choice of coordinates must take into account not only performance, but also security of the implementation. Performance-wise, it makes sense to use projective coordinates when the ratio inversion/ multiplication is greater than 3. Since this ratio depends very much on the chosen multiplication algorithm (see the table in the previous section), the choice of coordinates cannot be made solely on the criteria of the field.

Projective coordinates offer actually two benefits when considering side channel attack resistance. They can be used to defy DPA attacks by randomizing coordinates as in [7] as well as SPA and timing attacks using our homogeneous group operations method.

The experiment made us also wiser. If we were to give an advise on an all-round suite of counter measures against side channel attacks, including timing, SPA, DPA, and fault attacks, we would recommend the following.

- Blinding the base point by randomizing projective coordinates.
- Randomization of the exponent can be done using both, additive or multiplicative masks; the latter has an additional benefit of randomization of the execution sequence, and partially, of a base point.
- Use the securized Montgomery ladder with optimized group operations (on x-coordinates only) as suggested in [19]. The overhead is comparable with the one achieved in our experiment, but the algorithm is naturally more regular and thus, more secure.

References

1. ANSI X9.62: *The elliptic curve digital signature algorithm (ACDSA)*, draft, July 1997.
2. I. Blake, G. Seroussi, N. Smart. *Elliptic Curves in Cryptography.* Cambridge University Press (1999).
3. Brier, E., Joye,: Weierstrass elliptic curves and side-channel attacks. Proc. *Public Key Cryptography (PKC 2002)*, LNCS **2274** (2002) 335–345
4. Brown, M., Hankerson, D., Lopez, J., Menezes, A.: Software implementation of the NIST elliptic curves over prime fields. Proc. *CT-RSA 2001*, LNCS **2020** (2001) 250–265
5. Clavier, C., Joye, M.: Universal exponentiation algorithm: A first step towards provable SPA-resistance. Proc. *Cryptographic Hardware and Embedded Systems (CHES 2001)*, LNCS **2162** (2001) 300–308
6. Chari, S., Jutla, C., Rao, J., Rohatgi, P.: Towards sound approaches to counteract power-analysis attacks. Proc. *Advances in Cryptology – Crypto'99*, LNCS **1666** (1999) 398–412
7. Coron, J.-S.: Resistance against differential power analysis attacks for elliptic curve cryptosystems. Proc. *Cryptographic Hardware and Embedded Systems (CHES'99)*, LNCS **1717** (1999) 292–302
8. Gordon, D. M.: A survey of fast exponentiation methods. J. *Algorithms* **27** (1998) 129–146
9. Goubin, L., Patarin, J.: DES and differential power analysis. Proc. *Cryptographic Hardware and Embedded Systems (CHES'99)*, LNCS **1717** (1999) 158–172
10. Hasan, M. A.: Power analysis attacks and algorithmic approaches to their countermeasures for Koblitz cryptosystems. Proc. *Cryptographic Hardware and Embedded Systems (CHES 2000)*, LNCS **1965** (2000) 93–108
11. Hankerson, D., Hernandez, J. L., Menezes, A.: Software implementation of elliptic curve cryptography over binary fields. Proc. *Cryptographic Hardware and Embedded Systems (CHES 2000)*, LNCS **1965** (2000)
12. IEEE Std 1363-2000. *IEEE Standard Specification for Public-Key Cryptography.* IEEE Computer Society, August 29, 2000
13. Izu, T., Takagi, T.: A fast parallel elliptic curve multiplication resistant against side channel attacks. Proc. *Public Key Cryptography (PKC 2002)*, LNCS **2274** (2002) 280–296
14. Joye, M., Quisquater, J-J.,: Hessian elliptic curves and side-channel attacks. Proc. *Cryptographic Hardware and Embedded Systems (CHES 2001)*, LNCS **2162** (2001) 402–410
15. Joye, M., Tymen, Ch.: Protection against Differential Power Analysis for elliptic curve cryptography - An algebraic approach. Proc. *Cryptographic Hardware and Embedded Systems (CHES 2001)*, LNCS **2162** (2001) 377–390
16. Joye, M., Yen, S-M.: The Montgomery powering ladder. In these proceedings.
17. Kocher, P.: Timing attacks on implementations of Diffie-Hellmann, RSA, DSS, and other systems. Proc. *Advances in Cryptology – Crypto'96*, LNCS **1109** (1996) 104–113
18. Kocher, P., Jaffe, J., Jun, B.: Differential power analysis. Proc. *Advances in Cryptology – Crypto'99*, LNCS **1666** (1999) 388–397
19. Lopez, J., Dahab, R.: Fast multiplication on elliptic curves over $GF(2^n)$ without precomputations. Proc. *Cryptographic Hardware and Embedded Systems (CHES'99)*, LNCS **1717** (1999) 316–327

20. Liardet, P.-Y., Smart, N. P.: Preventing SPA/DPA in ECC systems using the Jacobi form. Proc. *Cryptographic Hardware and Embedded Systems (CHES 2001)*, LNCS **2162** (2001) 391–401
21. Menezes, A. J., van Oorschot, P. C., Vanstone, S. A.: *Handbook in Applied Cryptography.* CRC Press, 1997
22. Messerges, T. S., Dabbish, E. A., Sloan, R. H.: Power analysis attacks on modular exponentiation in smartcards. Proc. *Cryptographic Hardware and Embedded Systems (CHES'99)*, LNCS **1717** (1999)
23. Messerges, T. S., Dabbish, E. A., Sloan, R. H.: Examining smart-card security under the threat of power analysis attacks. *IEEE Transactions on Computers*, **51** (5) (2002) 541–552
24. Montgomery, P.: Speeding the Pollard and elliptic curve methods of factorization. *Mathematics of Computations*, **48** (1987) 243–264
25. Okeya, K., Sakaurai, K.: Power analysis breaks Elliptic Curve cryptosystems even secure against the timing attacks. Proc. *INDOCRYPT 2000*, LNCS **1977** (2000) 178–190
26. Oswald, E., Aigner, M.: Randomized addition-subtraction chains as a countermeasure against power attacks. Proc. *Cryptographic Hardware and Embedded Systems (CHES 2001)*, LNCS **2162** (2001) 39–50
27. May, D., Muller, H.L., Smart, N.P.: Non-deterministic processors. Proc. *Cryptographic Hardware and Embedded Systems (CHES 2001)*, LNCS **2162** (2001) 28–38
28. Yen, S-M., Kim, S-J., Lim, S-G., Moon, S-J: A counter-measure against one physical cryptanalysis may benefit another attack. Proc. *Information Security and Cryptology (ICISC 2001)*, LNCS **2288** (2002) 414–427
29. De Win,. E., Bosselaers, A., Vandenberghe, S., De Gersem, P., Vandewalle, J.: A fast software implementation for arithmetic operations in $GF(2^n)$. Proc. *Asiacrypt '96*, LNCS **1163** (1996) 65–76
30. De Win,. E., Mister, S., Preneel, B., Wiener M.: On the performance of signature schemes based on elliptic curves. Proc. *An Algorithmic Number Theory Symposium*, LNCS **1423** (1998)
31. Woodbury, A., Bailey, D., Paar, Ch.: Elliptic curve cryptography on smart cards without coprocessors. Proc. *The Forth Smart Card Research and Advanced Applications Conf.*, September 20–22, 2000 Bristol, UK

Secure Elliptic Curve Implementations: An Analysis of Resistance to Power-Attacks in a DSP Processor

Catherine H. Gebotys[1] and Robert J. Gebotys[2]

[1] Department of Electrical and Computer Engineering, University of Waterloo, Waterloo Ontario Canada N2L 3G1
cgebotys@optimal.vlsi.uwaterloo.ca
[2] Wilfrid Laurier University, Waterloo Ontario Canada

Abstract. With the popularity of wireless communication devices a growing new important dimension of embedded systems design is that of security. This paper presents exploration of power attack resistance, using a statistical approach for identifying regions of the power trace which pose a possible security threat. Unlike previous power analysis research, a new metric supporting small timing shifts and complex processor architectures is presented. This research helps to identify how to create secure implementations of software. Elliptic curve point multiplications using the Weierstrass curve and Jacobi form over 192-bit prime fields were implemented and analyzed. Over 60 real measured power traces of elliptic curve point multiplications running at 100MHz on a DSP VLIW processor core were analyzed. Modification of power traces through software design was performed to maximize resistance to power attacks in addition to improving energy dissipation and performance by 44% with a 31% increase in code size. This research is important for industry since efficient yet secure cryptography is crucial for wireless communication embedded system devices and future IP enabled smart cards.

1 Introduction

Security is increasingly becoming important in current design methodologies for embedded systems which concentrate on high performance, low cost, low power and low energy. Design for security involves secure protocol implementation and power analysis in addition to algorithm design. In fact power dissipation has a large impact on security as well as cost and reliability of an embedded system. Not only must cryptographic algorithms be high performance and low energy, but more importantly they must be secure or safe from side channel attacks. A side channel attack[1,11] involves obtaining useful information from the cryptographic application which may lead to the revelation of the secret key. Useful information includes the amount of time it takes to perform various operations or the variation of power dissipation during key computations. In the later case this is known as a power attack[1]. As an example, an attacker who has obtained the secret key is able to eavesdrop on a confidential wireless com-

B.S. Kaliski Jr. et al. (Eds.): CHES 2002, LNCS 2523, pp. 114–128, 2003.

munication between two parties. Consider Mary, the victim, who wishes to encrypt a conversation with Bob. She first has to set up a session key. Mary computes the session key as: xP in elliptic curve cryptography (or $y^x \ mod \ n$ in RSA technology) where x is the secret key and this computation is performed various times for different P's (or y's). The attacker may know P (or y) and in addition may obtain the computations times or may be able to monitor power dissipation. With this additional information the attacker may eventually be able to compute the secret key, x [1]. When the attacker has obtained the secret key, communication between Mary and Bob is not secure. Alternatively, obtaining the secret key in other applications, such as smart cards, allows one to illegally use phone or digital TV services. In power attacks, the dynamic power of the processor is measured over time and is called a power trace [1,11,12,13]. In elliptic curve cryptography (ECC), the analysis of the power trace may reveal when a point doubling occurs (or calculation of $2P$), and when two points are being added (such as $2P+P$) in the computation of xP, thus revealing the secret key.

In SOC (systems on a chip) platforms, multiple DSP processor cores on the same chip are common. Often these core processors run at different voltages, and use separate power pins, thus secure implementations of cryptography on SOCs is important. Cryptography on DSP processor cores is an important lower cost and lower power dissipation alternative to cryptographic processor cores and general purpose processor cores respectively. This paper for the first time presents a new metric for quantizing security and design exploration for ECC on a DSP processor core. Results are based upon over 60 real measured dynamic power traces performing point multiplication with different keys. Additionally tradeoffs in code size, performance, and energy dissipation for security are explored.

Currently research in power attacks of smart cards, have utilized general purpose processors [1,11,12,13]. Typically smart card applications are not time critical and energy dissipation is not a major concern since power is obtained from the card reader (or ATM machine, etc). Power attacks of more sophisticated processors with parallel instruction execution have not been reported in the literature. The measurement of power while a processor is executing an application (or a power trace) has been used in power-attacks of cryptographic devices, such as smart cards [1]. In particular the analysis of the variation of power, and computations on a number of power traces can be used to detect data and algorithmic dependencies [1]. This research studied the correlation of power variation with data values being manipulated and instruction sequencing. In the former case, known as differential power attacks, encryption applications were analyzed [1]. In the later case, known as a simple power attack (SPA), it was concluded that the correlation was significant and techniques such as random sequencing of instructions have since been investigated. Researchers have also investigated the use of DSP processors for encryption [2,5] as well as elliptic curve implementation [9], however their resistance to power attacks has not been addressed. Researchers addressing smart card application have suggested security against power attacks be achieved through 70% increase in computational cost [3], assuming a 192-bit prime number and signed window method with r=5[14], using 16 multiplications for both doubling and summing compared to 8 multiplications for a doubling and 16 for a summing. This is achieved through using different forms of the curve, such as

the Jacobi form, where mathematically the sum and the double of a point use the same formula 3,8]. Other researchers have investigated the cubic form of the EC, known as the Hessian form of the curve [8,10]. In this research a 33% improvement in performance overheads is achieved, since only 12 multiplications are required for both summing and doubling.

This paper will present a new metric for analyzing implementation security against power attacks, the implementation security index (*ISI*), and design exploration of performance, energy dissipation, code size in addition to security. It will be demonstrated on a complex VLIW DSP processor core, the Star*core (SC140), developed by Motorola and Lucent[4]. An elliptic curve cryptographic application is analyzed for resistance to power-attacks trading off low energy dissipation, high performance, and small code size. The uncertainty of security from power-attacks is explored with real current measurements of the DSP hardware VLSI core in a chip. A previously suggested power-attack resistant technique, the Jacobi form of the EC is also implemented and analyzed for comparisons in implementation security.

2 Elliptic Curves and Software Implementation

This section will review the application, elliptic curve point multiplication in prime fields and introduce the methodology used to develop a security index for measuring resistance to power attacks. Prime field cryptography involves a significant number of integer multiplies which can be performed very efficiently on DSP cores. In addition to a chosen key length, there are many different fields, projective coordinates, and types of elliptic curves that can be implemented. For added security portable devices should be able to support numerous curves and fields. However it is important for the designer to be able to choose which sets to implement on an embedded device, to tradeoff performance, code size, energy dissipation, and security against power attacks. The application, point multiplication, will be introduced in this section, followed by a discussion of implementation methods.

Point multiplication was implemented using the Weierstrass equation of the elliptic curve and the Jacobi form of the elliptic curve. Implementation details are provided in appendix A, B respectively. All curves were implemented with 192-bit field operations, using prime polynomial $x^{192}-x^{64}-1$. The Weierstrass equation of the curve was $y^2=x^3-3x^2+b$ over 192-bit prime fields[6] (see Appendix A for details). However to avoid the long latency of inversion in prime fields (Jacobi) projective coordinates [7,8] were chosen. In our DSP processor core the field multiplication to inversion ratio was 0.014 (330 cycles / 23146 cycles) using a worst case time for inversion. The equations for doubling and summing points in prime fields with Jacobi coordinates are given below[8]. These coordinates were used in the SC140 implementation and correspond to the cycle counts of the point double and sum for prime fields in table 1. Given point $P=(x,y,z)$, the point $2P = (x_{12},y_{12},z_{12})$ is given below (point doubling) and the point $(x_1,y_1,z_1)+(x_2,y_2,z_2) = (x_3,y_3,z_3)$ is also given below for point summing.

$$x_{12} = 9(x^2 - z^4)^2 - 8y^2x$$
$$y_{12} = 3(x^2 - z^4)(4y^2x - (3x^2 + z^4) + 8y^2x) - 8y^4$$
$$z_{12} = 2yz$$
$$x_3 = (y_2(z_1)^3 - y_1(z_2)^3)^2 - (x_2(z_1)^2 + x_1(z_2)^2)(x_2(z_1)^2 - x_1(z_2)^2)^2$$
$$y_3 = (y_2(z_1)^3 - y_1(z_2)^3)(x_1(z_2)^2(x_2(z_1)^2 - x_1(z_2)^2)^2 - x_3)$$
$$- y_1(z_2)^3(x_2(z_1)^2 - x_1(z_2)^2)^3$$
$$z_3 = z_1z_2(x_2(z_1)^2 - x_1(z_2)^2)$$

The clock cycle counts for the prime field EC codes running on SC140 along with the code size and average power measurements are given in table 1. The first row of table 1 provides clock cycle counts for point multiplication using key $13 (where $ indicates hexadecimal notation), for illustration purposes, with a signed NAF implementation (no windowing). Using these codes with a 192-bit key point multiplication can be performed in under 3ms running the DSP processor core at 300MHz (assuming a signed NAF sliding window with r=5 as in [3]).

Table 1. Original Prime Field Code Implementation Characteristics on SC140

192-bit Prime Fields	Clock Cycles	Code Size(bytes)	Average Power (mA)
Point Operations			
Point Multiplication (k=$13)	28,897	8,070	48.8
Double	3,177	5,008	47.9
Sum	5,554	4,920	49.8
Field Operations			
Multiplication	330	1,270	49.6
Squaring	213	2,212	51.6
Mod reduction	60	460	45.9
Addition	33	320	45
Subtraction	29	248	43.8

The modification of *sum* and *double* routines for security against power attacks was explored by inserting redundant operations into the *double* and *sum* routines so that the order and type of field operation were identical. Figure 1 illustrates the sum and double routine modifications. Since the point summation was almost double the execution time of the point double routine (see Table 1 cycle counts), it was split into two routines, *sum1* and *sum2* shown in each column in figure 1. Redundant Operations are identified by __ or underscores preceding their operation in the table. Table 2 illustrates the original and modified EC codes with respect to the field operation counts. The shifts are implemented for coefficient multiplications (ie. multiplies by 2,4,8). Approximately 25 clock cycles are required on average to implement a shift of x number of bits (where x = 2,4,8). The only field operations which had variable clock cycle

counts were the modular reductions which may or may not be required after additions, subtractions, or shifts. Also the modular reductions for the result of squarings and

Double	Sum1	Sum2
b1 = y ^2	z2s = z2 ^2	hs = h ^2
e1 = z ^2	z1s = z1 ^2	om = j ^2
b2 = b1 * b1	z2c = z2s * z2	al = th * hs
b = b2<<3	__al=y2<<3	__th=y2<<3
__z2=y2 * x2	f = z1s * x2	la = h * hs
z31 = y * z	g = z2s * x1	th = la * i
e2 = x - e1	___th=x1-z1s	x3 = om - al
e3 = x + e1	___ga=x1+z1s	___be=al + om
e = e2 * e3	z1c = z1s * z1	z3 = ga * h
z12 = z31<<1	___om=g<<1	___al=hs<<1
c1 = e<<1	___ga=z1c<<1	___be=z3<<1
c = c1 + e	th = f + g	___om=be+z3
f1 = b1<<2	___al=z2s<<2	___ga=hs<<2
a = f1 * x	ga = z1 * z2	la = hs * g
f3 = a<<1	___la=ga<<1	___al=la<<1
d1 = c * c	i = y1 * z2c	___be=om*om
x12 = d1 - f3	___al=i-la	om = la –x3
y31 = a - x3	h = f - g	___ga=om-al
y32 = y31 * c	om = y2 * z1c	al = om * j
y12 = y32 - b	j = om - i	y3 = al - th

Fig. 1. PA-resistant code, *WR*, for point doubling(1st column)

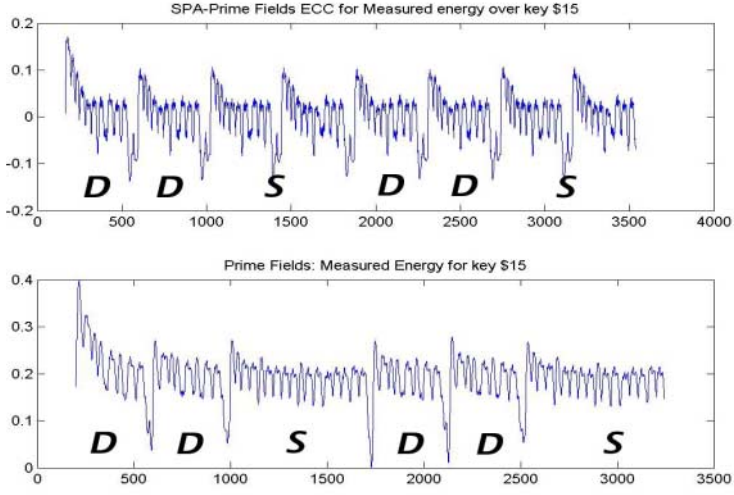


Fig. 2. Comparison of power traces of original code at bottom with power-attack resistant code WR1 for the same key at top

multiplications were variable (since the final summation may require one or two subtractions of the prime polynomial [7]). These caused minor delay differences in our codes. Additionally the signed NAF higher level algorithms were designed so that the routines in between the *sum1, sum2,* and *double* were also identical (ie. so one could not distinguish *sum* followed by *double* or *double* followed by *sum,* or *double* followed by *double*).

The direct implementation of the Weierstrass Curve with Jacobi projective coordinates has a very low value of implemented security (or low resistance to power attacks) on the DSP processor. This is illustrated in the power trace shown at the bottom (2^{nd},3^{rd} column) of figure 2, where S is the *sum* routine and D is the *double* routine. The power trace at the bottom, or original point multiplication, shows 8 bumps for the double routine, where each bump is the power dissipation of a multiplication (field) operation or a squaring (field) operation (narrower than the multiplication). This rise in power dissipation is due to the higher instruction level parallelism in the code as well as the use of higher power dissipating types of instructions, such as a 32X32 bit integer multiplication. Since the point summation is almost double the execution time of a point double, the power traces can easily be used to obtain the secret key and thus are very vulnerable to power attacks on this VLIW DSP processor. The implementation with redundant operations is shown in the power trace at the top of figure 2. The top power trace used an extra field and point summing multiplication at the beginning of each double and sum's routines to prevent the compiler from eliminating dual input operands to the double routine (however a less costly shift field operations could have been used). The two power traces in figure 2 perform point multiplication on the same key but the top is far more resistant to simple power attacks.

3 Analysis of Uncertainty in Power-Attack Resistance

The power traces for several keys were obtained for the software implementations of the Weierstrass curve(*WR*) and the Jacobi curve (*JC*) running on the SC140 DSP processor. In *WR,* the code was further optimized replacing redundant operations with more efficient implementations and detailed assembly modification to ensure good cycle-accurate timings of the *sum* routines and *double* routine. All power traces were obtained by executing the cryptographic algorithms on the SC140 at 100MHz (for illustration purposes, though power traces at 300MHz were very similar), using a pattern generator and high speed oscilloscope to capture the power traces. In the power trace plots the y-axis represents the current variation (in Amps centered by the oscilloscope at zero, and amplified by the probe by a factor of 5) and the x-axis represents the time (currently sampled data points). Matlab was used for signal analysis of the power waveforms.

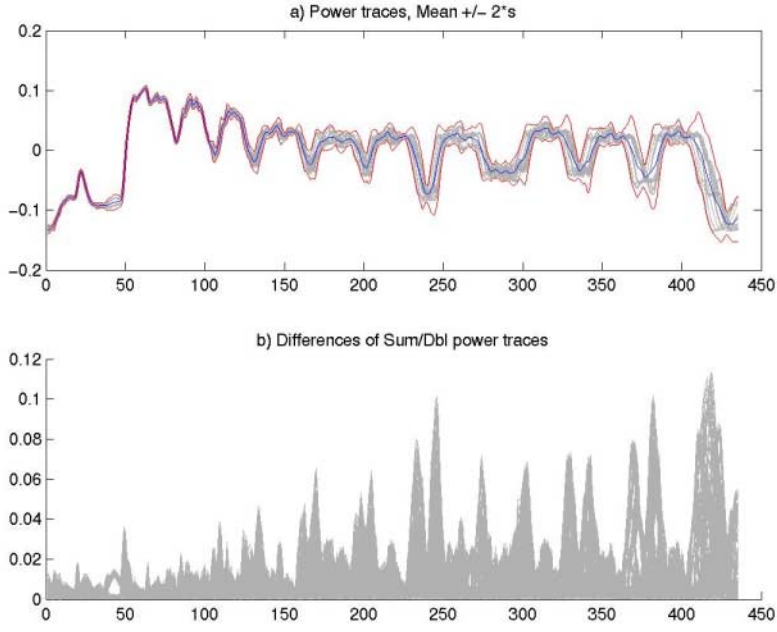


Fig. 3. Top: means, variances; Bottom: differences of *WR1* sum/dbl power traces

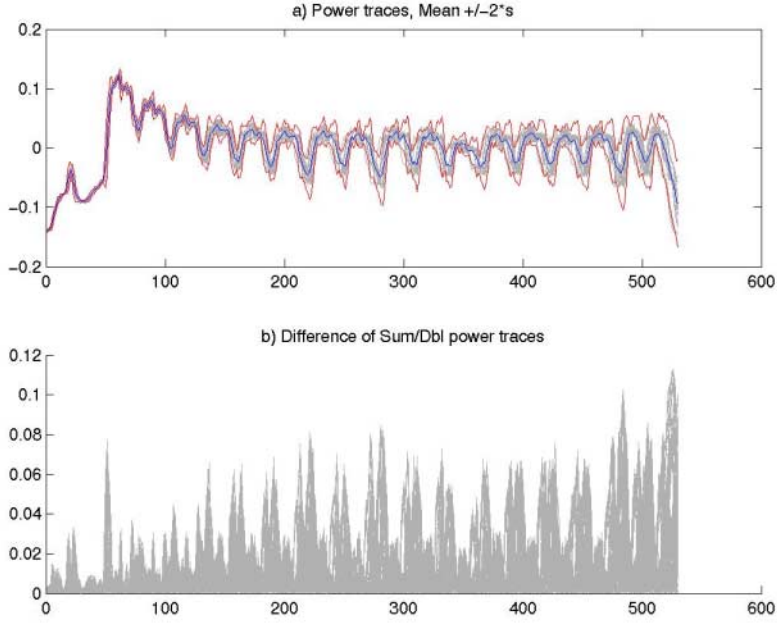


Fig. 4. Top: means, variances; Bottom: differences of *JC* sum/dbl power traces

Table 2. Field operation counts for original Weierstrass EC code and modified EC code for power-attack resistance

Sum	Original	PA-resistant Code (*WR*)
# of Multiplications	13	14
# of shifts	4	10
# of Squarings	0	4
# of Additions/subtractions	6	12
Double		
# of Multiplications	4	7
# of shifts	5	5
# of Squarings	4	2
# of Additions/subtractions	6	6
Cycles for key $0b	29,517	37,294
CodeSize	7,626	9,618

Initially code was implemented by modifying the code in figure 1, by specifically removing the first redundant shift by 3 bits (or <<3) in the sum routines and replacing b2<<3 by the first redundant multiplication (z2=y2*x2 becomes b=b2*8) in the double routine to reduce the latency. Power traces of this code, *WR1*, were extracted and the variances and the mean plus or minus two times the standard deviation were computed for *sum1, sum2, double* and plotted in figure 3. It was compared to the variance and standard deviation of power traces obtained from the Jacobi curve, *JC*, (which is mathematically power-attack resistant, since the point doubling and summing use the same routine) shown in figure 4. The superimposed Jacobian power traces in figure 4 each show 17 power bumps, all for multiplications except the last two which are squarings (see Appendix B). The average variances of both *double* and *sum* power traces were *3.12E-4* for the *Weierstrass curve, WR1,* and *1.911E-4* for the *Jacobi curve, JC.*

However it is difficult to use this average variance or absolute differences of double and sum power traces (as shown at the bottom of both figures) to determine which part of the code needs to be modified to increase security. Note also in these figures that the differences of the sum and double routines shown in the bottom plot of the figures are high where the power traces have the highest slopes (due to timing shift effects near the rise and fall of power dissipation during integer multiplication or squaring routines), thus not providing much information.

To further analyze the power traces and identify regions of software implementation which were possibly insecure, the variances and means were integrated into an implementation security index (*ISI*), shown in equation (1).

$$ISI_{1,2}(t) = \left(\frac{(\bar{x}_1(t) - \bar{x}_2(t))}{\sqrt{\frac{(s_1(t))^2}{n_1} + \frac{(s_2(t))^2}{n_2}}} \right)^{-1} \quad (1)$$

Equation (1) utilizes the means, $\bar{x}_1(t), \bar{x}_2(t)$ and standard deviations $s_1(t), s_2(t)$ for each time t and the number of power subtrace samples n_1, n_2. This formula could be used to analyze differences in the power traces of the *sum* (sample set 1) and the *double* (sample set 2) routines representing $ISI_{S,D}(t)$, or it could be used to analyze larger portions of the power trace including software implementations in between the *sum* and *double* routines. For example, a subtrace sample could be any part of the power trace such as a *double* followed by a *double*, DD, (or *sum1* followed by a *sum2*, SS) representing the $ISI_{DD,SS}(t)$ formula. Using this formula, areas of the power trace where the variance was low and the means of the *double* and *sum* routines differed were identified as a low security measure. In other cases if the differences of means were small or variances were large then a high security measure is indicated by the statistic.

Figure 5 shows four plots of *WR1*, from top to bottom the variances (Vars), the $ISI_{S,D}(t)^{-1}$ variable values (or ISI where peaks or valley's indicate a security problem), the difference of means (or DPA), and the actual sum, double means in the last bottom plot (Means). The oval identifies a peak of $ISI_{S,D}(190)^{-1}$ (in the *x* value near 190). This peak identified a problem in the security of the software implementation of *WR1*, near the second multiply. This problem was verified (see bottom mean plot) as a power difference due to multiplying a number with a large number of zeros (192-bit number = $08) in the double routine and not in the sum routine (where a normal multiplication of two more random 192-bit numbers is performed). In the DSP processor, multiplication with zero's dissipates larger power than a random or all 1's number (due to precharged busses, etc). Note that the difference of means or DPA (plot below the $ISI_{S,D}(t)^{-1}$) did not pick up any significant difference relative to the other regions of the plot. The $ISI_{S,D}(t)^{-1}$ measure shows more variation over earlier t regions than the DPA due to extremely small variances, since these subtraces were aligned at the beginning.

The modified code, called *WR*, removed the **8* from the double routine and replaced it with a redundant multiply of two full 192-bit numbers (whose result is put in a temporary variable), and introduced shifts to accomplish the **8* functionality, as detailed in figure 1. The power trace of the modified code is shown in figure 6, where the mean and variance and differences are plotted. In this figure the *sum2-double* power subtraces are shown. A horizontal line is added for comparison of field operations in both routines. The $ISI_{SD,DS}(t)^{-1}$ for the resulting code, *WR*, is shown in figure 7 and it has an average *ISI* (or mean of $|ISI_{SD,DS}(t)|$) of *0.49*. The peak circled in figure 7 of $ISI_{SD,DS}(97)^{-1}$ indicates a security leak.

It is interesting to note that the difference of means (DPA) also has peaked indicating a possible security leak. However DPA continues to peak through the rest of the power traces in figure 7, whereas the $ISI_{SD,DS}(t)^{-1}$ flattens out. Peaking through the DPA indicates differences of means exist, however they are not significant according to $ISI_{SD,DS}(t)^{-1}$ since the variances are also large in these areas. Again these large variances and difference of means occur due to timing shifts in the power traces. For example consider the DPA peak and valley centered near t=150. These result from the timing shifts in the rise and fall of power dissipation of the 192-bit multiplication routine. In this example, since the rise or fall of power is large, any timing shift will

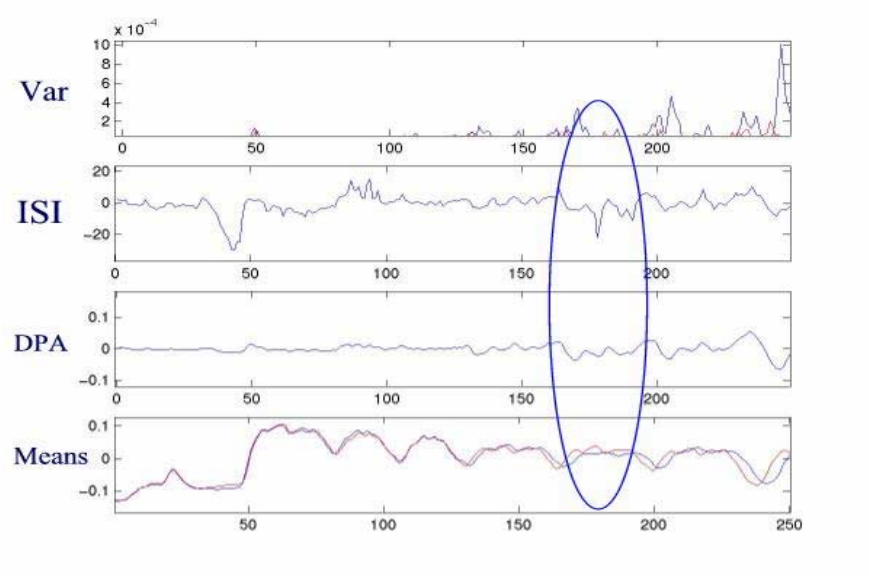


Fig. 5. Variances, $ISI_{SD}(t)^{-1}$, DPA, and means of *WR1*

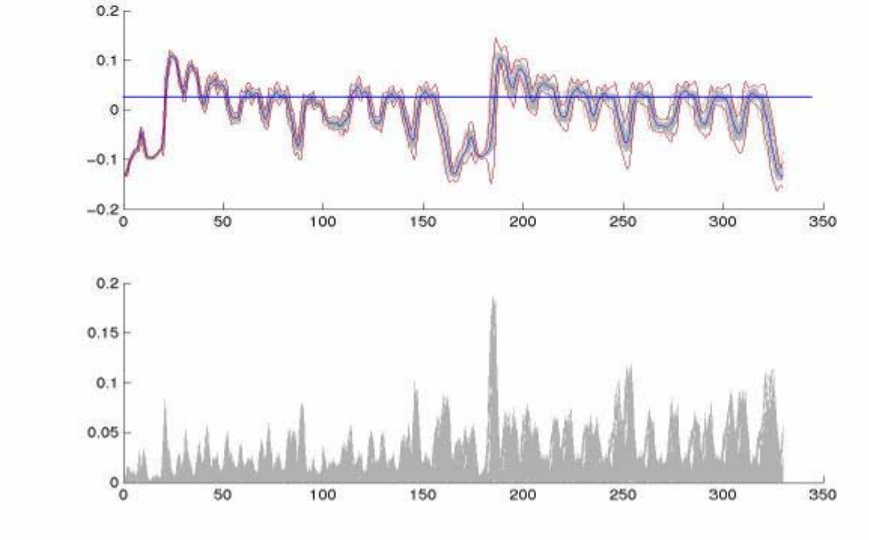


Fig. 6. Top: means, variances; Bottom: differences of sum/double subtraces of *WR*

create a large difference of means which is picked up by DPA methods. However in ISI since the timing shifts produce a large variance over the many power subtraces, no peak or security leak is correctly identified. Hence $ISI_{SD,DS}(t)^{-1}$ tends to identify which

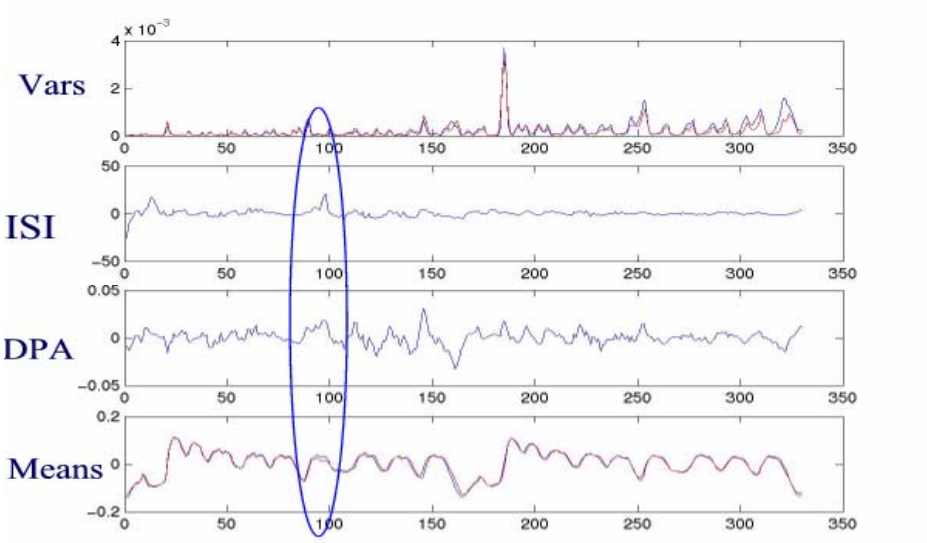


Fig. 7. Variances, $ISI_{SD,DS}(t)^{-1}$, DPA, and means of double-double/sum-sum power traces of *WR*

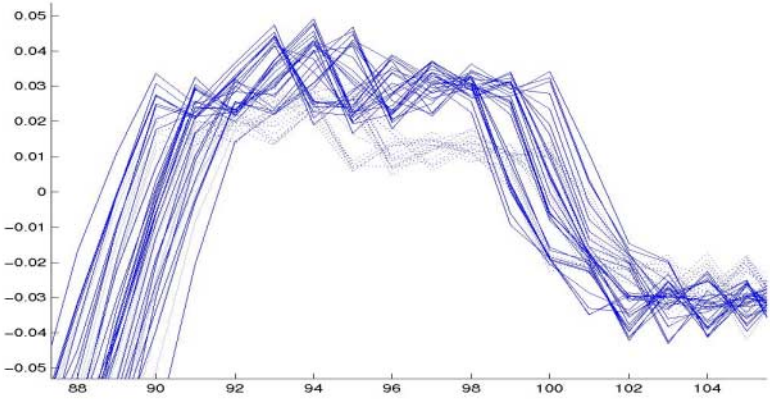


Fig. 8. Hole produced by memory stalls in highly parallel section of integer multiplication

difference of means are in fact significant (or exist with small variances). However the user must still verify that the $ISI_{SD,DS}(t)^{-1}$ peak is not a result of strictly very small variances alone. Looking closer at this $ISI_{SD,DS}(97)^{-1}$ peak, figure 8 shows superimposed power traces over this region and the *hole* or gap between the *sum2* and *double (and sum1)* routines (see x-axis between 96 and 98). The peak indicates a valid difference between the double and sum2 power traces due to memory stalls within the highly parallel integer multiplication routine.

In the SC140 a memory stall will cause an extra one cycle delay whose power characteristic is similar to a nop instruction (or no-operation instruction) which has the

lowest power dissipation. In SC140, memory stalls will occur whenever more than one request to the same memory module but a different row is made in the same processor clock cycle. Since the DSP processor uses a unified memory space (where both program and data is stored in same memory), memory stall identification is more complex since it involves the program code as well. In this case, it was determined that in the second sum routine, memory stalls due to conflicts between the program and the loading of data in the middle of a loop with high instruction level parallelism caused the power not to peak as much as it did in the other *sum1* routine, or *double* routine. The lower dotted traces in figure 8 are the *sum2* routine, while the upper line traces are the *sum1* and *double* routines respectively.

The Jacobi form of the curve was also analyzed and the variances, $ISI_{S,D}(t)^{-1}$ variables, and means are shown in figure 9. The final *JC* code had an average *ISI* of *0.44*. Figures 7 and 9 both show high absolute values of $ISI_{S,D}(t)^{-1}$ near the beginning of the power traces and in some other areas. In these cases, the absolute peak occurs due to very low variances and does not represent a security problem.

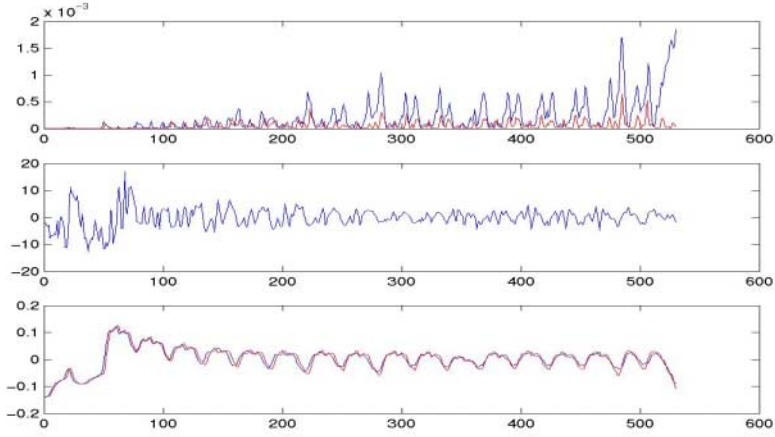


Fig. 9. Variances, $ISI_{S,D}(t)^{-1}$, means of Jacobi form of curve, *JC*

The bar chart in figure 10 from left to right shows the energy dissipation (mJ) of *WR, JC* and clock cycle counts for *WR, JC* for each key (ie. $b, $10, and a random 192-bit key). The number of clock cycles for key $b and $10 have been divided by 100 in the figure for scaling purposes. In figure 10, the cycle count of the 192-bit key is divided by 10,000 (1.81M cycles for *JC* and 1.25M cycles for *WR*, a 44.7% improvement) and the energy of the 192-bit key is divided by 100. The average *ISI* of *WR* and *JC* is respectively 0.49 and 0.44. The energy per bit (energy dissipation of 192-bit key divided by 192) of the *WR* and *JC* implementations is 10.37 and 14.96 respectively (a 44.2% improvement). The code size of *WR* is 9618 bytes compared to *JC* which only requires 7338 bytes (since the *double* and *sum* are the same routine).

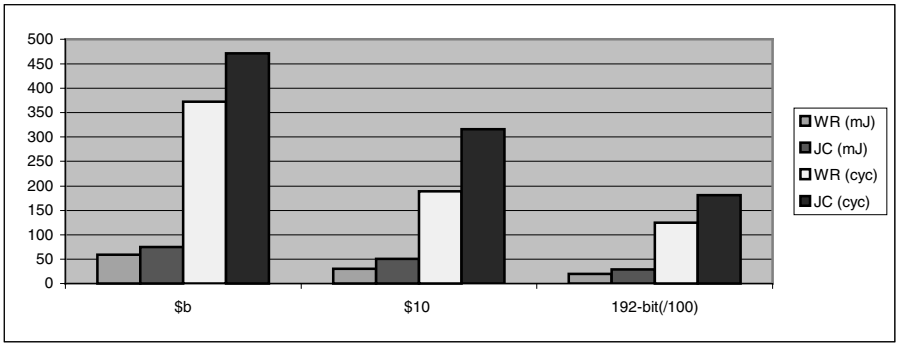


Fig. 10. Energy dissipation and cycle comparison of *WR* with *JC*

4 Discussions and Conclusions

The methodology presented in this paper, has shown that *ISI* can provide important information for cryptographic applications being implemented by embedded system designers. Previous methods suggested, such as simple power attacks, or differencing can be improved by exploring variances and *ISI*. This design exploration has been used to develop code which has improved security yet lower energy dissipation and higher performance compared to the Jacobi curve implementation. The lower energy dissipation will be important for secure implementations in portable devices.

Unlike previous research mathematical approaches to power-attack resistance, this research has examined techniques for ensuring the security of the software implementation through modification of power and energy dissipation. Design exploration of verified elliptic curve point multiplication routines running on a complex VLIW DSP processor core is presented. Previous methods suggested, such as SPA, or DPA (not easily extended for complex architectures, since there are multiple active busses each clock cycle) can be improved by exploring variances and *ISI* which handle small timing shifts. For the first time, a new metric, the implementation security index, *ISI*, has been introduced for quantizing security of implementations. Real power traces have been measured, and security from power-attacks verified with real hardware VLSI chip power measurements. This methodology for the design of secure software for the SC140 DSP processor can in general be applied to other processors.

Results show that *WR* code improves energy dissipation, performance, and implementation security index by 1.44 times, 1.44 times, and 1.11 times respectively compared to our implementation of previously research routines, *JC*, with a 31% increase in code size. This metric can be used for design exploration of security in addition to performance, code size and energy dissipation. This research is crucial for supporting a methodology for designing software that is not only optimized for performance, power and cost, but also for implementation security.

Acknowledgments. The author would like to thank NSERC, Motorola, and CITO for their support through funding of this research. The author also would like to thank R.Muresan for his acquisition of the power traces using [15] and A.Sathiananthan for coding the cryptographic algorithms and running them on the Star*Core board.

References

1. P. Kocher, "Timing attacks on implementations of Diffie-Hellman, RSA, DSS, and other systems", LNCS 1109, 1996.
2. S. Dusse, B. Kaliski, "A cryptographic library for the Motorola DSP56000", vol 473, LNCS, May 1990, pp. 230–244.
3. P. Liardet, N. Smart, "Preventing SPA/DPA in ECC systems using the Jacobi Form", LNCS 2162, May 2001, pp 391–401.
4. "Star*Core 140 DSP Core Reference Manual", Motorola/Lucent, Sept 1999.
5. T. Wollinger, M. Wang, J. Guajardo, C. Paar, "How well areHigh End DSPs Suited for the AES Algorithms?"
6. IEEE Std 1363-2000, IEEE Standard specifications for public-key cryptography, IEEE computer Soc. 2000.
7. D. Hankerson, J. Hernandez, A. Menezes, "Software Implementation of NIST Elliptic Curves over Prime Fields", White Paper, www.certicom.com, 2000
8. Chudnovsky, D.V., G.V. Chudnovsky, "Sequences of Numbers generated by addition in formal groups and new primality and factorization tests", Applied Mathematics, Vol.7, pp 385–434, 1986.
9. K. Itoh, M. Takenaka, N. Torii, S. Temma, Y. Kurihara, "Fast implementation of public-key cryptography on a DSP TMS320C6201", CHES '99, vol 1717, LNCS, 1999, pp. 61–72.
10. M. Joye, J. Quisquater, "Hessian Elliptic Curves and Side-Channel Attacks" LNCS 2162, May 2001, pp 402–410.
11. P. Kocher, J. Jaffe, B. Jun, "Differential Power Analysis" Crypto'99, LNCS 1666, 1999.
12. T. Messerges, E. Dabbish, R. Sloan, "Investigations of Power analysis attacks on Smart-cards" USENIX workshop on Smartcard Technology, 1999.
13. O. Kommerling, M. Kuhn, "Design principles for Tamper-resistant Smartcard Processors", USENIX workshop on Smartcard Technology, 1999.
14. I. Blake, G. Seroussi, N. Smart, "Elliptic Curves in Cryptography" , LMS 265, Cambridge Univ. Press, 2000
15. R. Muresan, C. Gebotys, "Current consumption dynamics at instruction and program level for a VLIW DSP Processor", ACM Proc. of ISSS, Oct 2001, pp. 130–135.

Appendix: A

The Weierstrass model [8,15] used was $\mathbf{E: y^2 \equiv x^3 - 3x + b \pmod p}$ where (hexadecimal notation is denoted by 0x)
b=0x64210519e59c80e7 0fa7e9ab 72243049 feb8deec c146b9b1

p= 0xffffffffffffffffffffffffffffffffffffffffffffffff (or

627710173538668076383578942320766641608390870039032496127 9 or

x^{192}-x^{64}-1) *using (Jacobian) projective coordinates (where* $x=x/z^2, y=y/z^3$*) and the following NIST recommended x,y starting points and:*

x = 0x188da80e,0xb03090f6,0x7cbf20eb,0x43a18800,0xf4ff0afd,0x82ff1012
y = 0x07192b95,0xffc8da78,0x631011ed,0x6b24cdd5,0x73f977a1,0x1e794811
z = 0x00000000,0x00000000,0x00000000,0x00000000,0x00000000,0x00000001
Other starting points were also investigated where z>1.
Example: the output values for key $0b were:
Value of x: 3e677863 ed84f02a 514987dd f5ec9fee 26cbc7bf 8794ca26
Value of y: 75bed8f8 327b78cd eb1d339e d6e9d58d 856922e5 6c3ca607
Value of z: 8c83fb04 a32bc227 9e07c3d0 6bfad1e1 ae9357aa 99a48ae5

Appendix: B

Jacobi form of curve, represented as an intersection of two quadratics[8,3] **E: $w^2 + x^2 - z^2 = 0$, $k^2w^2 + y^2 - z^2 = 0$** *, p =* x^{192}-x^{64}-1, *with* k^2 [2] and starting points given as:
w={0x7a73b10f, 0xd4201d0c, 0xf0a56204, 0xba70362f, 0x2471ac47, 0x067277d1 };
x={0x12712cc2, 0xcbe55812, 0x2bcb2aaa, 0x00a9e313, 0xc75c9c34, 0x15d2b44a };
y={0x47fc02ce, 0xa38ea373, 0x2eae6122, 0xb9d9f5e6, 0xab9dd76a, 0x300be399 };
z={0x01379630, 0x88fd6a29, 0x50f0f425, 0xa78b7b28, 0x98fd71c7, 0xa23f074d };
k^2={0x33148392, 0xa8a1abb4, 0xd16e45ba, 0xa2451dbb, 0x983a69d4, 0x286eca33 };
Example: the output points for key $0b were
Value of w: 98d7bb57 b34b19d3 399b7ed 2371a568 4274c9aa 38297506
Value of x: 45cfa54b ee52c9a0 ca3b06bb c2c9641f 6634e224 465267dc
Value of y: d2c1c135 e88469b2 f695c8c3 6362c15d 816fc025 dc1c8ba8
Value of z: a1b9de72 e60f5d59 e5b92102 e1937046 b28420a3 1db8b731
Implemented Code: (c0,c1,c2,c3)= (a0,a1,a2,a3)+(b0,b1,b2,b3)
a3b1=a3*b1 , a0b2=a0*b2 , a2b0=a2*b0 , a1b3=a1*b3 , c01= a3b1*a0b2 , c0=c01+ a2b0*a1b3 , c11=a3b1*a1b3 , c1=c11-a2b0*a0b2 , a3a2=a3*a2 , b3b2=b3*b2 , a3a2b3b2=a3a2*b3b2 , a0a1=a0*a1 , b0b1=b0*b1 , a0a1b0b1=a0a1*b0b1 , k^2a0a1b0b1= k^2*a0a1b0b1 , c2= a3a2b3b2 - k^2a0a1b0b1 , a3b1_s= (a3b1) ^2 , a2b0_s= (a2b0) ^2 , c3= a3b1_s + a2b0_s

Address-Bit Differential Power Analysis of Cryptographic Schemes OK-ECDH and OK-ECDSA

Kouichi Itoh[1], Tetsuya Izu[2], and Masahiko Takenaka[1]

[1] FUJITSU LABORATORIES Ltd.,
64, Nishiwaki, Okubo-cho, Akashi, 674-8555, Japan
{kito,takenaka}@flab.fujitsu.co.jp
[2] FUJITSU LABORATORIES Ltd.,
4-1-1, Kamikodanaka, Nakahara-ku, Kawasaki, 211-8588, Japan
izu@flab.fujitsu.co.jp

Abstract. The differential power analysis (DPA) is a powerful attack against the implementation of cryptographic schemes on mobile devices. This paper proposes an alternative DPA using the addresses of registers of elliptic curve based cryptosystems (ECC) implemented on smart cards. We call the analysis the *address-bit DPA* in this paper. The analysis was originally investigated by Messerges, Dabbish and Sloan, however it was thought to be of no effect if the intermediate data are randomized. We extend the analysis and show how the extended analysis works against scalar exponentiations even if the implementation is resistant against the data-based DPA. We show experimental results of our analysis of cryptographic schemes OK-ECDH and OK-ECDSA, which are candidates of the CRYPTREC project in Japan, and evidence of their weakness.

Keywords. Differential power analysis (DPA), address-bit DPA, elliptic curve cryptosystems (ECC), scalar exponentiation, OK-ECDH, OK-ECDSA

1 Introduction

The key agreement scheme OK-ECDH [17] and the digital signature scheme OK-ECDSA [18], developed by HITACHI, are candidates of the CRYPTREC project, in which a list of cryptographic schemes suitable for e-government in Japan are being made [6]. OK-ECDH (OK-ECDSA) is based on the discrete logarithm problem over the Montgomery form elliptic curves [15] but the scheme is very similar to the standard ECDH (ECDSA) [8,23]. As we will discuss the scalar exponentiation of OK-ECDH and OK-ECDSA, we refer them as OK-Schemes (OKS) in the following.

The self-evaluation reports of OKS have claimed resistance against side channel attacks [17,18]. The side channel attacks, firstly proposed by Kocher et al. [11,12], are attacks in which an attacker observes side channel information such

B.S. Kaliski Jr. et al. (Eds.): CHES 2002, LNCS 2523, pp. 129–143, 2003.

as computing time and power consumption from a cryptographic device, and attempts to reveal secret information (a secret key) hidden in the device. At the moment, the Simple Power Analysis (SPA) and the Differential Power Analysis (DPA) are typical examples of the side channel attacks. Implementers should pay the most attention to the attacks and take measures against each of them in order to avoid these attacks.

The power consumption changes in accordance with the Hamming weights of data passed in the device. The DPA is classified into two types; the data-bit DPA, in which an attacker reveals dependence of values of data on the difference of power consumption [13,4], and the address-bit DPA, in which an attacker reveals those of addresses of registers [14]. The address-bit DPA is based on the fact that if we load data from various addresses, the power consumption of the device changes in accordance with a difference of Hamming weights of addresses. Thus the address-bit DPA is as noteworthy as the data-bit DPA [14]. It is, however, thought to be of no effect if the intermediate data in the device are randomized.

This paper extends the analysis and shows how the extended address-bit DPA works against scalar exponentiations of elliptic curve based cryptosystems (ECC), even if the algorithm is resistant against the data-bit DPA. We assumed two conditions. One is that the algorithm in the device is known. The other is that the number of registers used in the algorithm is small. These conditions make our address-bit DPA easier. This paper also shows experimental results of our address-bit DPA of OKS. As the (recommended) algorithms for OKS is public and the number of registers is at most 3, the above conditions are satisfied. We show evidence of the weakness of OKS against our address-bit DPA.

The paper is organized as follows: we introduce previous results of DPA in Section 2. The validity of the address-bit DPA is shown in Section 3. Then, in Section 4, we explain how to apply our DPA to OKS together with experimental results.

2 Preliminaries

2.1 Elliptic Curve

In this paper, we discuss elliptic curves over $K = GF(p)$, a finite field with p-elements for a prime p. Let E be an elliptic curve over K and $E(K)$ be a set of points on the curve including the special point $\mathcal{O}$ (the point of infinity). The set $E(K)$ has an additive group structure. A concrete algorithm for computing an addition for given points is found in a textbook ([3], for example). For two points P_1, P_2 on $E(K)$, we denote an operation $P_1 + P_2$ as ECADD (where $P_1 \neq P_2$), and an operation $2 * P_1$ as ECDBL. For a given elliptic curve $E(K)$, a point P on $E(K)$, and an integer d, computing $d * P = P + P + ... + P$ (d times) is called a scalar exponentiation and P, d are called the base point and the exponent, respectively. A scalar exponentiation is computed by a combination of ECADD and ECDBL. An addition chain determines such combination. Let d

be an n-bit integer and consider a binary expression $d = d[n-1] * 2^{n-1} + d[n-2] * 2^{n-2} + ... + d[1] * 2^1 + d[0]$ ($d[n-1] = 1$). A standard binary chain is given in the following algorithm (Algorithm 1).

Algorithm 1. Binary chain

```
INPUT: d, P
OUTPUT: d * P
1: Q[0] = P
2: for i=n-2 down to 0 {
3:   Q[0] = ECDBL(Q[0])
4:   if d[i]==1  Q[0] = ECADD(Q[0],P)
5: }
6: return Q[0]
```

2.2 Side Channel Attack

Side Channel Attacks (SCA) are proposed by Kocher et al. [11,12], in which an attacker observes side channel information such as computing time and power consumption from a cryptographic device (smart cards), and attempts to reveal secret information (a secret key) hidden in the device without breaking it physically. The attacks are valid if there is dependence between the secret information and the power consumption. At the moment, the Simple Power Analysis (SPA) and the Differential Power Analysis (DPA) are typical examples of side channel attacks. Implementers should pay the most attention to the attacks and take measures against each of them in order to avoid these attacks.

The SPA uses observed side channel information. In Algorithm 1, ECADD is computed only when $d[i] = 1$. An attacker can guess the value of $d[i]$ by checking a pattern of the power consumption and able to reveal the secret key. Coron proposed a countermeasure so called the add-and-double-always method in which ECDBL and ECADD are always computed for all $d[i]$ [4]. Then operations make a fixed pattern of side channel information and the attacker cannot obtain any information by SPA.

The power consumption changes in accordance with a difference of Hamming weights of data. The DPA analyzes these differences from side channel information obtained through a lot of observations. As the Coron's DPA [4] is involved in the Messerges-Dabbish-Sloan's DPA [13], we only introduce the latter one. Messerges-Dabbish-Sloan classified their DPA into three cases depending on the assumption of the attacker. The following explanations are devoted to ECC, but similar attacks are applicable to the famous RSA cryptosystem.

Single-Exponent, Multiple-Data (SEMD): In SEMD, we assume that the attacker knows one exponent d_k, able to measure the traces of power consumption (power traces) for any inputs, but does not know the algorithm. The attacker, who is going to reveal a secret key d_u, first measures the power traces of

the device with d_k on input various random values and obtains an average power trace. Next, he inputs the same values into the device with d_u and obtains an averaged power trace. Then a difference of these two power traces determines the order of ECADD and ECDBL computed in the device, because the trace is 0 only when two exponents operate the same operations at the same time.

Multiple-Exponent, Single-Data (MESD): In MESD, we assume that an attacker can measure the power traces with any exponents, but does not know the algorithm. The attacker measures a power trace with d_u on input a certain value. Suppose the attacker knows $d_u[n-1], \cdots, d_u[i+1]$ of d_u. Then, he/she guesses $d_u[i]$, measures a trace with it on input the same value and obtains a difference of traces. If the guess is correct the difference corresponding $d_u[i]$ is 0 and the attacker convinces the correctness of his/her guess. Thus the secret key d_u is revealed by MESD by repeating the same procedures.

Zero-Exponent, Multiple-Data (ZEMD): In ZEMD, we assume that the attacker knows the algorithm of a scalar exponentiation, knows modules, and able to simulate computations in the device. The attacker measures power traces on input various random values and obtains power traces for each input. Next, the attacker guesses $d_u[i]$ of d_u used in the first module and obtains resulted data of the module for each input by simulations. Then he/she divides the results into two parts depending on their Hamming weights, computes average power traces for each part and obtains a difference of traces. Then there appear spikes in the difference if his guess is correct, and no spikes otherwise. Thus the secret key d_u is revealed by ZEMD by repeating the same procedures.

In the add-and-double-always countermeasure [4], ECDBL and ECADD are computed repeatedly and pattern of the power trace is fixed for any inputs. So SEMD and MESD cannot reveal the secret key. However ZEMD is valid for this countermeasure. One approach to resist ZEMD is to make the simulation impossible by randomizing intermediate values. Coron's Randomized Projective Coordinate (RPC) [4] and Joye-Tymen's randomized isomorphic curve [10] are good examples.

2.3 OK-ECDH and OK-ECDSA

The key agreement scheme OK-ECDH [17] and the digital signature scheme OK-ECDSA [18], developed by HITACHI, are candidates of the CRYPTREC project, in which cryptographic schemes suitable for e-government in Japan are being evaluated [6]. As we will discuss scalar exponentiations of OK-ECDH and OK-ECDSA, we refer them as OK-Schemes (OKS) in this paper. OKS is based on the elliptic curve discrete logarithm problem and the schemes are very similar to standard ECDH and ECDSA [8,23]. An outstanding difference is that all operations of OKS are performed on the Montgomery form elliptic curves [15], which is defined by $By^2 = x^3 + Ax^2 + x \quad A, B \in K,\ B(A^2 - 4) \neq 0$. A special addition formula, which does not use the y-coordinates of points, offers a fast

scalar exponentiation on this curve. OKS uses these techniques as well and hence fast cryptographic operations are possible [20,22]. We show an outline of OK-ECDH in Algorithm 2, for example, where P is a base point and a key pair (d_A, Q_A) satisfies $Q_A = d_A * P$. Roughly speaking, OK-ECDH is obtained by operating ECDH on Montgomery form elliptic curves. See specifications [17,18] for detailed descriptions.

Algorithm 2. Outline of OK-ECDH

INPUT: Q_V, a (one-time) public key of an entity V
OUTPUT: z, a shared key

1. Generate a one-time key pair (d_U, Q_U) (satisfying $Q_U = d_U * P$).
2. Send Q_U to the entity V
3. Compute a point $Q = d_U * Q_V$ on Montgomery-form elliptic curve.
4. (Option) Compute a point $R = cQ$ and then if $R = \mathcal{O}$ then terminate, where c is the cofactor of the Montgomery-form elliptic curve.
5. If $Q = \mathcal{O}$ then terminate.
6. Transform the x-coordinate x_Q of the point Q to an octet string z.
7. Output shared key z

As OKS uses the special addition formula, it uses an alternative addition chain (Montgomery ladder, Algorithm 3) rather than the standard chain (Algorithm 1). The Montgomery ladder computes ECDBL and ECADD repeatedly, hence the chain is resistant against SPA, SEMD, MESD [19]. Against ZEMD, OKS uses Randomized Projective Coordinates (RPC) in order to resist it [21, 20]. Moreover, developers claimed in their self-evaluation reports that an cryptographic scheme is SCA-resistant, if the secret information and the order of operations are independent and intermediate values are randomized [17,18].

Algorithm 3. Montgomery ladder

INPUT: d, P
OUTPUT: $d * P$

```
1: Q[0] = P, Q[1] = ECDBL(P)
2: for i=n-2 to downto 0 {
3:   Q[2] = ECDBL(Q[d[i]])
4:   Q[1] = ECADD(Q[0],Q[1])
5:   Q[0] = Q[2-d[i]], Q[1] = Q[1+d[i]]
6: }
7: return Q[0]
```

3 Address-Bit DPA

Messerges-Dabbish-Sloan's DPA [13] described in Section 2.2 tries to find dependence of values of data on a difference of power traces. On the other hand, they also proposed alternative DPA which tries to find those of addresses of registers [14]. The latter analysis is based on the fact that if we load same data from different addresses of registers, the power consumption changes in accordance with a difference of Hamming weights of addresses. They show experimental results of basic characteristics of the address-bit DPA and introduced an idea of the attack for DES implementation [14]. However, they did not show a concrete result of the DPA attack experiment, so that the address-bit DPA does not attract much attention and it is thought to be of no effect if the intermediate data are randomized.

In this section, we extend a concept of the address-bit DPA and show how the extended analysis works for elliptic curve based cryptosystems even if the algorithm is resistant against the data-bit DPA.

3.1 Outline

The original address-bit DPA watches a difference of addresses via loading same data from different addresses. The power consumption changes when different data are loaded from different addresses of registers. If the influence of data in a difference of power traces is erased, a secret key can be revealed by watching a difference of addresses. Indeed, the influence of data is erased by averaging the power traces and the averaged power trace only depends on a difference of Hamming weights of register addresses. This is a basic idea of our address-bit DPA. The attack is successful if there is a close dependence between a secret key and addresses of accessed registers. In general, this approach may be hard because a lot of registers are used in the implementation. But in our target of ECC, the number of registers is small and the situation makes our analysis easier than general cases.

3.2 Experimental Results

In order to show the validity of our address-bit DPA, we show experimental results in the following. We have two registers $Q[0], Q[1]$ and given unknown value d (0 or 1). We load 8-bit data $L = 500$ times from $Q[d]$ and try to guess d. Here the data in $Q[d]$ are changed into random values after every load. Such procedure is described as `A = Q[d]` in the algorithmic level. In a low-power device such as smart cards, this procedure is divided into two stages; (1) determining the address of $Q[d]$, and (2) loading data with the address.

Our strategy is as follows. First we observe L power traces for $d = 0$ (a). Next, we observe L power traces for d_b (b) and d_c (c) ($d_b, d_c = 0, 1,\ d_b \neq d_c$). Let $S_{a,i}, S_{b,i}, S_{c,i}$ be the i-th power traces and S_a, S_b, S_c be their averages. Then

the differences of power trace (differential power traces) D_{ab}, D_{ac} are defined by the following:

$$D_{ab} = \frac{1}{L}\sum_{i=1}^{L} S_{a,i} - \frac{1}{L}\sum_{i=1}^{L} S_{b,i} = S_a - S_b,$$
$$D_{ac} = \frac{1}{L}\sum_{i=1}^{L} S_{a,i} - \frac{1}{L}\sum_{i=1}^{L} S_{c,i} = S_a - S_c.$$

Then there should appear spikes in D_{aj} if $d_a \neq d_j$, no spikes in D_{aj} if $d_a = d_j$, where $j \in \{b, c\}$. Our differential power traces D_{ab}, D_{ac} are in Figure 1. Two spikes are found in D_{ac} (arrowed)[1]. From these figures, we are convinced that $d_b = 0$ and $d_c = 1$. In fact, these guesses are correct.

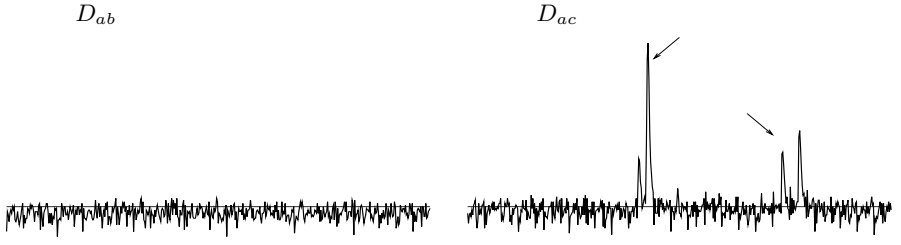


Fig. 1. Differential power traces D_{ab} and D_{ac}

In the experiment, an address $\&Q[d]$ is determined by a secret data d, and the value of the address has an influence on the power trace. Thus we can guess the value of d by using the close dependence between addresses and data. We conclude that the secret key d can be revealed by the address-bit DPA, even if the data are randomized.

Note 1. To be exact, the target of Messerges-Dabbish-Sloan's address-bit DPA [14] is only (2). The spike (1) is detected by the data-bit DPA as well because it occurs through data-loading of addresses of registers.

4 Proposed Address-Bit DPA

This section discusses the address-bit DPA of OK-ECDH and OK-ECDSA based on SEMD and ZEMD. In order to resist the data-bit DPA, OKS uses the Randomized Point Coordinates (RPC) countermeasure. Here intermediate data in the device are varied (randomized) even if the input is unchanged. So MESD

[1] The first spike in D_{ac} corresponds to the addressing (1), and the latter spike corresponds to the loading (2).

cannot be applied to OKS because a 'Single Data' is not available. Moreover there is no need to mention 'MD' for distinguishing SEMD and ZEMD. We just call SE or ZE analysis in the following. Note that in the SE analysis, we assume the algorithm is known. This is stronger condition than those of original SEMD, but there is no problem for our case because the algorithm of OKS is public.

We analyze OKS implemented by either following Implementation 1 or Implementation 2. The algorithms are described in C-like language, where d is an n-bit secret key, $d[i]$ is the i-th bit of d and $Q[0], Q[1], Q[2]$ are registers for intermediate data.

Implementation 1

```
Q[0]=P, Q[1]=ECDBL(P)
for i=n-2 to downto 0 {
  Q[2]=ECDBL(Q[d[i]]) (*11)
  Q[1]=ECADD(Q[0],Q[1])
  Q[0]=Q[2-d[i]], Q[1]=Q[1+d[i]] (*12)
}
return Q[0]
```

Implementation 2

```
Q[0]=P, Q[1]=ECDBL(P)
for i=n-2 to downto 0 {
  Q[2]=ECDBL(Q[d[i]]) (*21)
  Q[1]=ECADD(Q[0],Q[1])
  t=&Q[0], &Q[0]=&Q[2-d[i]], &Q[2-d[i]]=t (*22)
  t=&Q[1], &Q[1]=&Q[1+d[i]], &Q[1+d[i]]=t (*23)
}
return Q[0]
```

Implementation 1 uses $d[i]$ in (*11) and (*12) not to operate different procedures but to load data from different registers. Note that recommended scalar exponentiation algorithm in [17,18] and the algorithm in [21] is equivalent to Implementation 1.

Implementation 2 uses $d[i]$ in (*21), (*22) and (*23). While (*12) in Implementation 1 'copies' the values into $Q[0], Q[1]$, (*22) and (*23) in Implementation 2 'swap' the addresses of registers of $Q[0]$ with $Q[2 - d[i]]$ and $Q[1]$ with $Q[1 + d[i]]$. Note that, in Implementation 1, addresses of registers $Q[0], Q[1], Q[2]$ are unchanged. And in Implementation 2, addresses of registers $Q[0], Q[1], Q[2]$ are changed depending on the value $d[i]$, although operations are same.

Our address-bit DPA uses the differences of addresses of intermediate registers $Q[0], Q[1], Q[2]$. In a scalar exponentiation in OKS (Algorithm 3), ECDBL and ECADD are computed repeatedly and a pattern is fixed independent from

the exponentiation. But inputs are randomly varied because of the RPC. As we showed in Section 3, the influence of randomized data on power consumption can be erased by averaging traces. Then the difference of averaged power traces comes from (*) in Implementation 1 or 2 and a secret key can be revealed.

4.1 SE Attack

In the SE attack, we assume that an attacker knows one exponent d_k, able to measure the power traces for any inputs, as in SEMD. Moreover we assume that Implementation 1 is used. The attacker measures the power traces with d_k on input various values and obtains an average power trace for d_k. We denote the power trace corresponding to the i-th bit $d_k[i]$ of the j-th measurement as $S_{k,j}[i]$ and an averaged power trace corresponding to $d_k[i]$ as $S_k[i]$. Next he/she measures power traces with unknown exponent d_u on input same values and obtains an average power trace for d_u. We denote the power trace corresponding to the i-th bit $d_u[i]$ of the j-th measurement as $S_{u,j}[i]$ and an averaged power trace corresponding to $d_u[i]$ as $S_u[i]$. Then the differential power trace $D[i]$ is given by the following:

$$D[i] = \frac{1}{L}\sum_{j=1}^{L} S_{k,j}[i] - \frac{1}{L}\sum_{j=1}^{L} S_{u,j}[i] = S_k[i] - S_u[i],$$

where L is the number of measurements. On the other hand, in Implementation 1 of OKS, ECDBL and ECADD are computed in a same manner independent from d_k, d_u. So we expect that $S_{k,j}[i]$ and $S_{u,j}[i]$ are signals generated by completely the same operations. Indeed as $S_k[i], S_u[i]$ are averaged power traces for various data, the influence of data is erased as in Section 3. So we have

$$D[i] \simeq \begin{cases} 0 & \text{if } d_k[i] = d_u[i] \\ \text{nonzero} & \text{if } d_k[i] \neq d_u[i] \end{cases}.$$

Thus a difference of averaged traces determines the values of $d_u[i]$, because the difference is 0 if $d_k[i] = d_u[i]$ and the difference is nonzero if $d_k[i] \neq d_u[i]$. We show an experimental result of the SE attack against OKS in the following. The parameters we used in the experiment are as follows:

$$\begin{array}{lll} p = \texttt{0x200011}, & A = \texttt{0x14c82a}, & B = \texttt{0x11133f}, \\ h = \texttt{0x8019d}, & x = \texttt{0x1b144d}, & y = \texttt{0x1aa97d}, \end{array}$$

where the Montgomery form elliptic curve is defined by $By^2 = x^3 + Ax^2 + x$ over $GF(p)$, the order of the curve is $4h^2$ and a base point is $P = (x, y)$. We measured scalar exponentiations (implemented by Implementation 1) $L = 500$ times with $d_k = 1111\cdots$. We measured power traces corresponding to the most significant 4 bits for each exponent. The differential power trace is shown in Figure 2. There is no spike for $i = n-1, n-3$ and there are spikes for $i = n-2, n-4$. From the figure, we can determine the secret bits as $d_u = 1010\cdots$ and indeed our guess was correct.

[2] The order of the Montgomery form elliptic curve is divisible by 4.

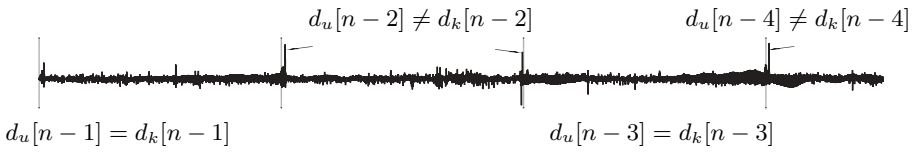


Fig. 2. Differential power traces D_{ab} and D_{ac}

Note 2. We have two spikes for $i = n-2$ in Figure 2, the one corresponds to (*11) and the other to (*12). The second spike for $i = n-4$ is omitted in the figure.

Note 3. In the experiment we used 22-bit parameters, while the bit length of the specification of OKS is 162-bit. This is only because of a simplification, not a theoretical reason. In the above attack, we measured the power traces of successive 4-bit exponents in order to draw comprehensible figures. If we know the timing when spikes appear in the trace, we only need to store data according to the timings for performing the effective analysis. By using this technique, our analysis will be successful with larger parameters.

Remark that, if the parameter is larger than 22-bit, there is no difference except that the interval between the first and the second spikes in Figure 2 becomes longer.

4.2 ZE Attack

In the ZE attack, we assume that an attacker knows the algorithm of a scalar exponentiation (namely, Implementation 1 or 2), and the module, and able to simulate the computation in the device, and able to measure the power traces for any inputs, as in ZEMD. Moreover we assume that Implementation 1 is used. The attack depends on the implementation.

Implementation 1: An attacker measures the power traces on input various random values with unknown secret key d_u and obtains an average trace. Next, the attacker decompose the average trace into modules for each bit $d_u[i]$, where a module is consists of an ECDBL and an ECADD. We note that, as a scalar exponentiation in OKS is processed by computing ECDBL and ECADD repeatedly, this decomposition is done quite easily. Then the attacker computes the differences of average traces. We denote the power trace corresponding to the i-th bit $d_u[i]$ of the j-th measurement as $S_{u,j}[i]$ and an averaged power trace corresponding to $d_u[i]$ as $S_u[i]$. Then the differential power trace $D[a,b]$ of $d_u[a]$ and $d_u[b]$ is given by the following:

$$D[a,b] = \frac{1}{L}\sum_{j=1}^{L} S_{u,j}[a] - \frac{1}{L}\sum_{j=1}^{L} S_{u,j}[b] = S_u[a] - S_u[b],$$

where L is the number of the measurements. On the other hand, in Implementation 1 of OKS, ECDBL and ECADD are computed in a same manner independent from d_u, we expect that $S_u[a]$ and $S_u[b]$ are traces generated by completely the same operations. Indeed, as $S_u[a]$ and $S_u[b]$ are averaged power traces of random data, the influence of data is erased here. Then we have

$$D[i] \simeq \begin{cases} 0 & \text{if } d_u[a] = d_u[b] \\ \text{nonzero} & \text{if } d_u[a] \neq d_u[b] \end{cases}.$$

A difference of traces determines the values of $d_u[i]$, because the difference is 0 if $d_u[a] = d_u[b]$ and the difference is nonzero if $d_u[a] \neq d_u[b]$. Thus a secret key d_u is revealed by the ZE against Implementation 1.

We show an experimental result of the ZE attack against Implementation 1 in the following. We used the same parameters as in the previous experiment. We computed scalar exponentiations (implemented by Implementation 1) $L = 500$ times with an unknown exponent d_u and measured power traces $S_{u,j}[i]$ corresponding to the most significant 4 bits for each exponent. Then we calculated an average trace $S_u[i]$, decomposed it into $S_u[0], \cdots, S_u[3]$[3], and obtained DPA bias signals which is shown in Figure 3. There are no spike in $D[0, 2]$, while there are two spikes in $D[0, 1]$ and $D[0, 3]$ in the figure. From these results, we can determine the secret bits are $d_u = 1010\cdots$, because the most significant bit of d_u is 1. Indeed our guess was correct.

Note 4. We have two spikes in $D[0, 1], D[0, 3]$, the one is corresponding to (*11) and the other is to (*12) as in the SE attack.

Implementation 2: As in the attack against Implementation 1, the attacker decomposes the average trace into modules for each bit $d_u[i]$ of a secret key d_u, where a module is consists of an ECDBL and an ECADD. Here in Implementation 2, the addresses of registers $Q[0], Q[1], Q[2]$ are varied depending on the exponent. That is, the differences of average power trace of modules do not have any sense. So we analyze a transition of addresses. Let (a,b,c) denote addresses of $Q[0], Q[1], Q[2]$. Then the transition is as in Figure 4.

Suppose the current addresses are (a,b,c) which corresponds to $d_u[i]$. First, we consider the recovery of addresses (Case 1). From the transition, the addresses (a,b,c) comes back to (a,b,c), if the exponent is one of

$$R = \{00, 111, 0101, 1010, 011011, 101101, 110110\}.$$

So if the exponent is $r \in R$, the DPA bias signal $D[i, i + |r|] = 0$ in all of (*21),(*22) and (*23), where $|r|$ is a bit length of r, and one of them is nonzero otherwise. Note that if an exponent is 000, the address of $Q[0]$ transits a $\rightarrow$ c $\rightarrow$ a and we have (c, b, a) after the procedure.

Next, let us consider the transition of addresses in (*21) (Case 2). Here the address of $Q[0]$ is used if $d_u[i] = 0$, and the address of $Q[1]$ is used if $d_u[i] = 1$.

[3] It is described $0, \cdots, 3$ for easiness to explain though these suffixes should be precisely described $n - 1, \cdots, n - 4$.

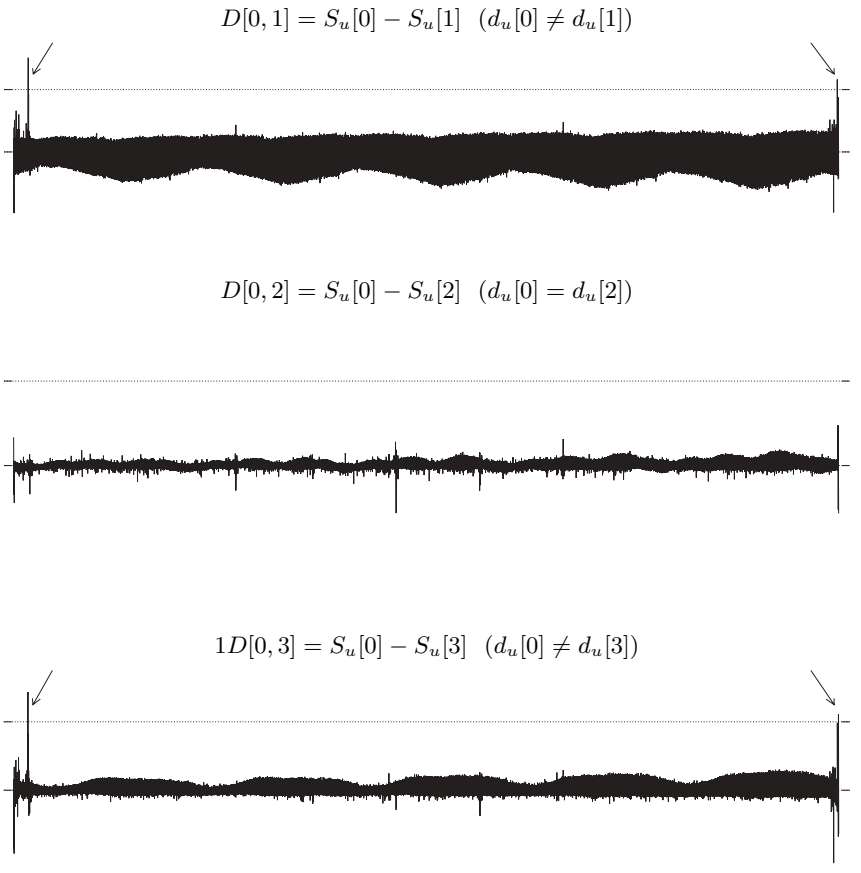


Fig. 3. Differential power traces by the ZE attack against Implementation 1

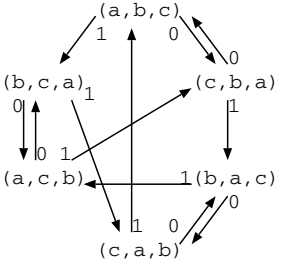


Fig. 4. Address transition of $Q[0], Q[1], Q[2]$ in Implementation 2

Suppose successive bits of d_u are same, then the same addresses are used in (*21) and we have no spikes in the DPA bias signal. Otherwise, namely successive bits are different, we have spikes in the bias signal. For example, suppose an exponent is 1000. Then we have no spikes in the differences $D[i, i+1], D[i, i+3]$, $D[i+1, i+3]$, and have spikes in $D[i, i+2], D[i+1, i+2], D[i+2, i+3]$. Note that if an exponent is 1000, the address of $Q[0]$ transits b → b → a → b and we have (a, c, b) after the procedure.

Let us reveal an secret exponent d_u from $D[i, j]$ by using the above observations. Suppose the initial addresses of $Q[0], Q[1], Q[2]$ are unknown, and we can distinguish $D[i, j]$ being 0 or nonzero in (*21),(*22),(*23). By Case 2, we obtain appeared types of addresses used in (*21), which is described as x,y,z, according to the appeared order. Here we obtain all patterns of x,y,z for 4-bit exponents, which are shown in Table 1. The patterns are sorted lexically.

Table 1. Relative patterns of addresses in (*21)

Exponent	Pattern	Exponent	Pattern	Exponent	Pattern	exponent	pattern
1, 0, 0, 0	x, x, y, x	0, 1, 1, 0	x, y, x, x	0, 1, 0, 1	x, y, y, x	1, 1, 1, 1	x, y, z, x
1, 0, 1, 1	x, x, y, x	0, 0, 0, 0	x, y, x, y	1, 1, 0, 0	x, y, y, x	0, 0, 1, 1	x, y, z, y
1, 0, 1, 0	x, x, y, y	0, 0, 0, 1	x, y, x, z	0, 1, 0, 0	x, y, y, z	0, 0, 1, 0	x, y, z, z
1, 0, 0, 1	x, x, y, z	0, 1, 1, 1	x, y, x, z	1, 1, 0, 1	x, y, y, z	1, 1, 1, 0	x, y, z, z

We have same patterns for (1000,1011), (0111,0101), (1100,0100), (0010,1110), and we cannot distinguish them. But (1000,1011), (0001,0111) can be distinguished by Case 1, because of the exponent includes 000. Thus for 4-bit exponents, 10 patterns out of 16 patterns are uniquely determined. If we make a similar table for 6-bit exponents, they are distinguished uniquely.

5 Concluding Remarks

This paper proposed the address-bit DPA against the elliptic curve based cryptosystems, together with experimental results of OK-ECDH and OK-ECDSA. We showed evidence of the weakness against our address-bit DPA. However, as the attack is targeted to the implementation, it seems to have no relation to the underlying mathematical problem, i.e. the elliptic curve discrete logarithm problem.

The address-bit DPA is based on the dependence between an secret key and address of registers used in the algorithm. That is, the countermeasures by randomizing data, such as [4,10], are not effective on our analysis. The result suggests that an exponent should be randomized as well. The exponent blinding [4,13], in which a scalar d is replaced by $d' = d + r\phi$ (r is a random number, ϕ is the order of the curve), the exponent splitting [5], in which d is split into r and $d - r$, and the overlapped window method [24] are examples. OKS will be secure

against our analysis if such countermeasures has been applied. In any case, the recommended algorithm in the current specification of OKS is vulnerable against our analysis.

Other countermeasure against the address-bit DPA is to use registers with same Hamming weights of addresses. This approach will be successful if power comsumption of the device follows the characteristic of the Hamming weight model. However, in the Linear model and the Quadratic model [1], even if the Hamming weights are same, digit positions are distinguishable and the countermeasure has no effect against our analysis.

Our analysis is expected to be applicable to other scalar exponentiation algorithms in [2,4,7,9], for example. A detailed discussion and experiments will be our future work.

Acknowledgment. The authors would like to thank our colleagues, especially Naoya Torii, for their helpful suggestions. We also thank anonymous referees of this paper for their comments.

References

1. M. Akkar, R. Bevan, P. Dischamp, and D. Moyart, "Power Analysis, What is Now Possible...", Asiacrypt 2000, LNCS 1976, pp. 489–502, Springer-Verlag, 2000.
2. E. Brier, and M. Joye, "Weierstraß Elliptic Curves and Side-Channel Attacks", PKC 2002, LNCS 2274, pp. 335–345, Springer-Verlag, 2002.
3. I. Blake, G. Seroussi, and N. Smart, *Elliptic Curves in Cryptography*, Cambridge University Press, 1999.
4. J. Coron, "Resistance against differential power analysis for elliptic curve cryptosystem", CHES'99, LNCS 1717, pp. 292–302, Springer-Verlag, 1999.
5. C. Clavier, and M. Joye, "Universal exponentiation algorithm – A first step towards provable SPA-resistance –", CHES2001, LNCS 2162, pp. 300–308, Springer-Verlag, 2001.
6. Cryptography Research & Evaluation Committees (CRYPTREC), Japan. `http://www.ipa.go.jp/security/enc/CRYPTREC/index-e.html`
7. W.Fischer, C.Giraud, E.Knudsen, and J.P.Seifert, "Parallel Scalar Multiplication on General Elliptic Curves over F_p Hedged Against Non-Differential Side-Channel Attacks", Cryptology ePrint Archiver, 2002/007, IACR. Available from `http://www.iacr.org/`
8. IEEE P1363, Standard Specifications for Public-Key Cryptography, 2000.
9. T. Izu, and T. Takagi, "A Fast Parallel Elliptic Curve Multiplication Resistant against Side Channel Attacks", PKC 2002, LNCS 2274, pp. 280–296, Springer-Verlag, 2002.
10. M. Joye, and C. Tymen, "Protections against differential analysis for elliptic curve cryptography", CHES2001, LNCS 2162, pp. 377–390, Springer-Verlag, 2001.
11. C. Kocher, "Timing attacks on Implementations of Diffie-Hellman, RSA, DSS, and other systems", Crypto'96, LNCS 1109, pp. 104–113, Springer-Verlag, 1996.
12. C. Kocher, J. Jaffe, and B. Jun, "Differential power analysis", Crypto'99, LNCS 1666, pp. 388–397, Springer-Verlag, 1999.

13. T.S. Messerges, E.A. Dabbish, and R.H. Sloan, "Power Analysis Attacks of Modular Exponentiation in Smartcards", CHES'99, LNCS1717, pp. 144–157, Springer-Verlag, 1999.
14. T.S. Messerges, E.A. Dabbish, and R.H. Sloan, "Investigations of Power Analysis Attacks on Smartcards," preprint, USENIX Workshop on Smartcard Technology, 1999.
15. P. Montgomery, "Speeding the Pollard and elliptic curve methods for factorizations", Mathematics of Computation, vol.48, pp. 243–264, 1987.
16. National Institute of Standards and Technology, Recommended Elliptic Curves for Federal Government Use, in the appendix of FIPS 186-2.
17. Key Agreement Scheme OK-ECDH, HITACHI, 2001. Documents are available from `http://www.sdl.hitachi.co.jp/crypto/ok-ecdh/index.html`
18. Digital Signature Scheme OK-ECDSA, HITACHI, 2001. Documents are available from `http://www.sdl.hitachi.co.jp/crypto/ok-ecdsa/index.html`
19. K. Okeya, H. Kurumatani, and K. Sakurai, "Elliptic curves with the Montgomery form and their cryptographic applications", PKC 2000, LNCS 1751, pp. 446–465, Springer-Verlag, 2000.
20. K. Okeya, K. Miyazaki, and K. Sakurai, "A fast scalar multiplication method with randomized projective coordinates on a Montgomery-form elliptic curve secure against side channel attacks", ICISC 2001, LNCS 2288, pp. 428-439, Springer-Verlag, 2001.
21. K. Okeya, and K. Sakurai, "Power analysis breaks elliptic curve cryptosystem even secure against the timing attack", Indocrypt 2000, LNCS 1977, pp. 178–190, Springer-Verlag, 2000.
22. K. Okeya, and K. Sakurai, "Efficient elliptic curve cryptosystem from a scalar multiplication algorithm with recovery of the y-coordinate on a Montgomery-form elliptic curve", CHES 2001, LNCS 2162, pp. 126–141, Springer-Verlag, 2001.
23. Standards for Efficient Cryptography Group (SECG), Specification of Standards for Efficient Cryptography.
24. J. Yajima, K. Itoh, M. Takenaka, and N.Torii, "DPA countermeasure by improving the window method", to appear in the proceeding of CHES 2002.

2Gbit/s Hardware Realizations of RIJNDAEL and SERPENT: A Comparative Analysis

A.K. Lutz[1], J. Treichler[1], F.K. Gürkaynak[2], H. Kaeslin[3], G. Basler[1], A. Erni[1], S. Reichmuth[1], P. Rommens[1], S. Oetiker[2], and W. Fichtner[2]

[1] Department of Information Technology and Electrical Engineering
ETH Zürich, CH-8092 Zürich Switzerland
{alutz, jtreichl, gbasler, aerni, sreichmu, prommens}@ee.ethz.ch
[2] Integrated Systems Laboratory
Department of Information Technology and Electrical Engineering
ETH Zürich, CH-8092 Zürich Switzerland
{kgf,oes,fw}@iis.ee.ethz.ch
[3] Microelectronics Design Center
ETH Zürich, CH-8092 Zürich Switzerland
kaeslin@ee.ethz.ch

Abstract. We present and evaluate efficient VLSI implementations of both Rijndael and Serpent. The two cipher algorithms have been implemented by two comparable design teams within the same timeframe using the same fabrication process and EDA tools. We are thus in a position to compare to what degree the Rijndael and Serpent ciphers are suitable for dedicated hardware architectures. Both ASICs support encryption as well as decryption in ECB mode and include on-chip subkey generation. The two designs have been fabricated in a 0.6μm 3LM CMOS technology. Measurement results verified an encryption and decryption throughput of 2.26Gbit/s and 1.96Gbit/s for Rijndael and Serpent respectively. Circuit complexity is in the order of 300k transistors in either case.

1 Introduction

While efficient hardware implementation was one of the evaluation criteria [3] of the Advanced Encryption Standard (AES), relatively few hardware designs with FPGAs have been presented [7,9,10] and even less so as ASICs [4,8]. There have been very few reports on hardware implementations [6] even after Rijndael was declared the AES standard. A detailed summary of the above mentioned implementations can be found in [11].

While FPGA based crypto-system solutions offer significant performance, especially for System-on-a-Chip (SoC) designs and large scale productions, customized ASIC modules of crypto-systems are indispensable. Our study focuses on actual ASIC implementations of the AES cipher Rijndael and the runner-up algorithm Serpent on silicon.

B.S. Kaliski Jr. et al. (Eds.): CHES 2002, LNCS 2523, pp. 144–158, 2003.

The hardware evaluation of the AES candidates by Weeks [4] and Ichikawa [8] are based on synthesis results only and are very general in nature. Rather than finding an optimum implementation for each of the implemeted algorithms, they concentrate on comparing all algorithms using a similar architectural approach. Area and speed estimations based on synthesis results may be used to compare different architectural choices, but not all architectures obtained by logic synthesis will lend themselves to physical design with the same efficiency.

In our study we have set strict limits on the maximum area, usable clock frequency range, number of parallel inputs and outputs and design time and formed two separate design teams with identical ASIC design experience to optimize the algorithms for maximum throughput. In section 2 we present an overview of the architectural options and define the solution space. Details on the hardware optimizations for the Rijndael cipher are given in section 3 while those of Serpent follow in section 4. Both circuits are then compared to each other in section 5 before section 6 presents our conclusions.

2 Common Design Issues

2.1 The Rijndael and Serpent Algorithms

Figure 1 shows the main algorithmic structure of Rijndael and Serpent for encrypting one block of data with 128bit keys. The names of the operations correspond to those described in [1,2]. Both algorithms can be seen as a succession of several transformation rounds which all data blocks have to undergo. Note that the initial and final rounds may be, depending on the algorithm, slightly modified versions of the regular transformation rounds. Each round uses at least one subkey derived from the user key. While Rijndael uses the same transformation round throughout, each Serpent round makes use of one out of eight different S-boxes.

	Rijndael	Serpent
Initial Transformation	AddRoundKey	
Round Transformation	9 x SubBytes, ShiftRows, MixColumns, AddRoundKey	31 x Key Mixing, S-Boxes, Linear Transformation
Final Transformation	SubBytes, ShiftRows, AddRoundKey	Key Mixing, S-Boxes, Key Mixing

Fig. 1. Algorithmic operations of Rijndael and Serpent

2.2 A Fair Comparison

For either of the two algorithms, a team of three was assigned the task to develop a VLSI circuit with the best throughput they could obtain from a die size of 50mm^2 in a 0.6μm 3LM CMOS technology. While such a hard bound on the circuit area limited the design space for both designs, it is typical for real-world applications. A large chip not only costs more to manufacture, but also suffers from a lower yield and higher parasitic interconnect capacitances.

Since a high throughput rate was desired, the work concentrated on designing a 128bit core that supports a simple ECB mode. It can be shown that the general architecture developed for the ECB mode (or simple derivations thereof) will lend itself to optimal implementations supporting other modes of operations such as CFB, CBC, OFB and CTR.

Both chips had to be designed to share the same pinout in a 144pin PGA package to ease testing and system integration. Also note that the number and rate of off-chip connections may account for a significant portion of the power budget of the design. Both designs feature a cryptographic core that operates on 128bit parallel data words internally. The external interface of both chips consists of three separate 32bit I/O channels for plaintext, ciphertext and the user key respectively. A separate I/O controller is used to schedule the data transfers from the 32bit external interface to the 128bit internal core.

Both design teams were given 14 weeks to complete the project. All team members were graduate students in EE and, hence, had very similar backgrounds and levels of expertise in IC design. Also, the EDA tools, cell libraries, computing resources and fabrication processes made available to them were the same. Incidentally, front-end design was carried out with tools by ModelSim and Synopsys while SiliconEnsemble by Cadence Design Systems was used for the back-end design. Cell library and chip fabrication on multi-project wafers were provided by austriamicrosystems (AMS). This arrangement has made it possible to compare the hardware realizations of the two cipher algorithms on a level ground.

2.3 Overall Architectural Choices

VLSI designers always strive to maximize hardware efficiency or, which is the same, to minimize the area-time-product (AT). Figure 2 illustrates the most prominent architectural transforms for arithmetic/logic hardware along with their impact on chip area and throughput.

Fairly small circuits are obtained from iteratively decomposing the computation such as to make it run on a single hardware round. Each data block must then be recycled through that datapath as many times as the cipher algorithm has transformation rounds. Extra control logic is required if one round computationally differs from the next such as in the occurrence of the Serpent cipher. Conversely, fast but large architectures result from mapping all transformation rounds into hardware directly followed by a generous addition of pipeline registers.

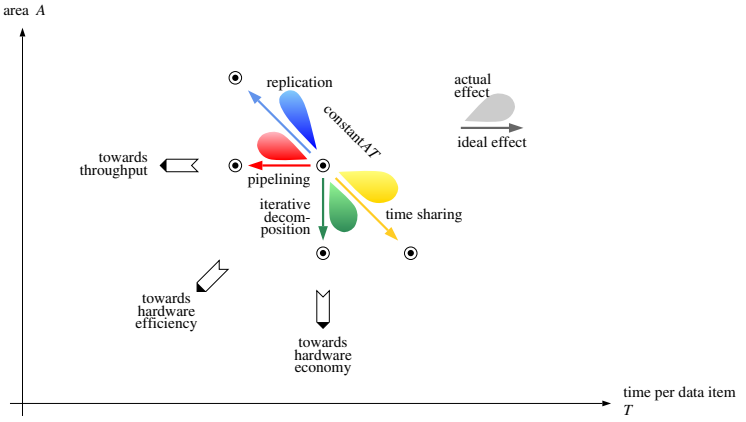


Fig. 2. Architectural transforms along with their impact (after [5]).

Many more architectures are situated somewhere in between these extremes. Their construction asks for a carefully balanced combination of pipelining, iterative decomposition, and possibly also replication of hardware units.

Key choices include:

- the number of rounds to instantiate in hardware,
- the degree of pipelining, that is number of registers per round,
- the organization of the datapath hardware, e.g. deciding on the optimum locations of [de]muxes and of pipeline registers, and
- the cycle-by-cycle schedule for the entire computation run.

The result of a simple analysis comparing the ECB throughput and datapath area for different architectural choices of both Serpent and Rijndael using the target 0.6μm technology can be seen in fig.3. The data for the graph has been compiled by synthesizing a single hardware round for each algorithm and extrapolating the performance based on the performance of this round. Note that this simplistic analysis only contains the hardware required for the rounds. Subkey generation and/or storage units, controllers and I/O circuitry are not included in this calculation. Additional effects like increased interconnection delay for larger designs, and clock distribution problems related to high clock rates have also not been considered. While the architectures selected for implementation are shown to have datapath areas of only 12mm^2 both implementations ended up being 50mm^2 in total, a fourfold increase.

In fig.3, a 14-round Rijndael cipher capable of running with a 256bit user key is considered. This architecture can be realized by instantiating 1, 2, 7 or all 14 rounds in hardware. Similarly for Serpent, realizations instantiating 1, 2, 4, 8, 16 or all 32 rounds have been considered. The lower part of the graph has been magnified in the inset. The inset roughly covers the solution space that was actually available in the context of our ASIC design project.

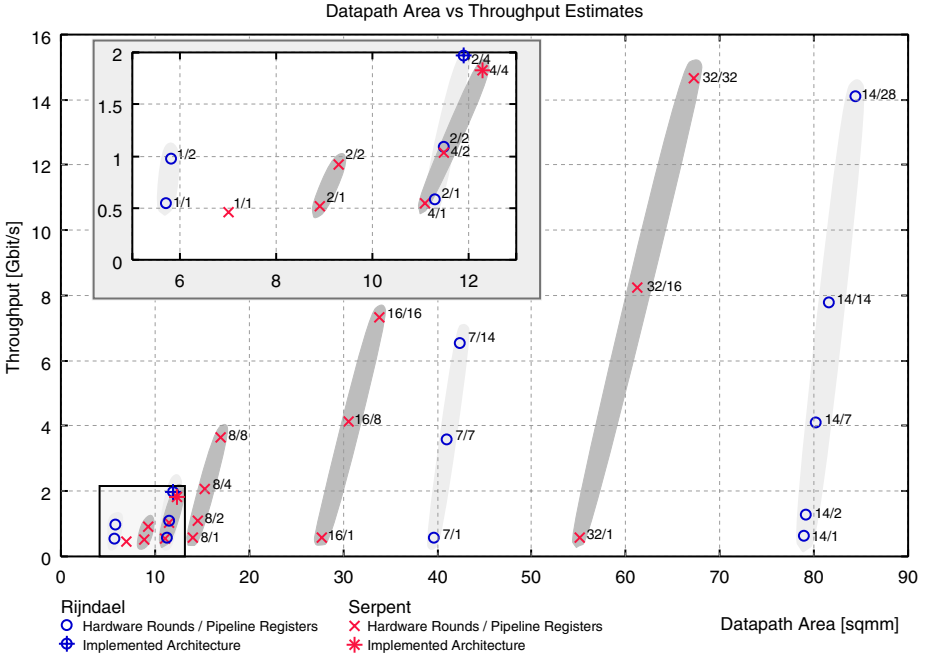


Fig. 3. Architectural choices for Rijndael and Serpent, throughput and datapath area estimations as a function of instantiated hardware rounds and pipeline stages for Rijndael and Serpent. The throughput estimations are for 128bit keys.

Subkey generation is another issue as both algorithms require several subkeys that must be derived from the user key. These subkeys are then applied to a data block while it moves through the individual rounds. One has the choice to compute the subkeys on the fly for each round or to generate all of them in advance before any data processing takes place, storing them in registers until needed. Depending on the number of subkeys, the chip area to be set aside for storage may be substantial.

The decryption operation places additional constraints on hardware. In its most basic form, both the ordering and function of the rounds must be reversed, so that the last encryption round is undone first during decryption. For a realization that does not store all the subkeys, the last subkey must thus be computed prior to the decryption operation.

Some cipher algorithms use reversible round structures where the same hardware can be used for encryption and decryption. As opposed to this, both Rijndael and Serpent rely on certain computational operations within their transformation rounds that require separate datapath elements for encryption and decryption, thereby increasing hardware complexity.

3 The Rijndael Implementation

3.1 Sharing Look-Up Tables between En- and Decryption

As illustrated by fig.1, each encryption round consists of four consecutive operations named SubBytes, ShiftRows, MixColumns and AddRoundKey. ShiftRows is a fixed permutation of the byte order and needs no extra circuitry. MixColumns can be implemented as a sequence of a few XOR gates while AddRoundKey is a simple XOR operation on all 128bits. The only operation that is onerous to implement in hardware is SubBytes.

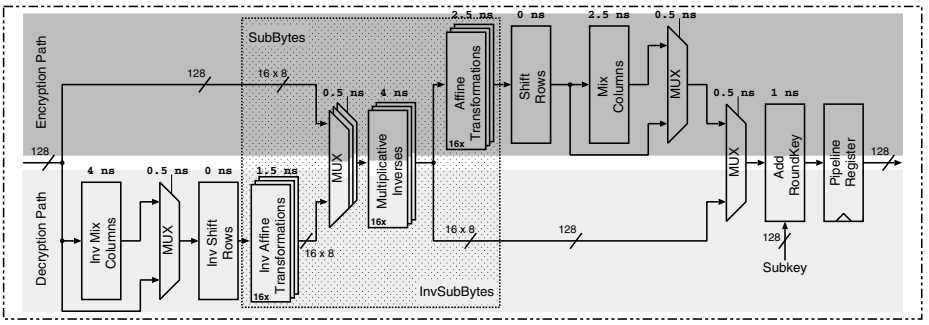


Fig. 4. A Rijndael round that shares LUTs and AddRoundKey function.

SubBytes consists of 16 concurrent S-box operations for which no more efficient solution than using 8bit×8bit look-up tables (LUT) seems to exist. The 16 LUT's account for about 85% of a round's combinational logic, so this was where to look for area reductions. Our idea was to find out whether the LUT's could somehow be shared between encryption and decryption in spite of the fact that the two operations are computationally different.

A Rijndael S-box is composed of two transformations [1]:

1. Take the multiplicative inverse in the finite field GF(2^8) with the element {00} being mapped onto itself.
2. Apply an affine transformation.

As opposed to the expensive LUT needed to implement the multiplicative inverse, the affine transformation is easily obtained from a few XOR gates.

The inverse S-box operation consists of the inverse affine transformation followed by the multiplicative inverse. Instead of using two separate LUT's, it is thus possible to compute both the S-box and the inverse S-box operation from a single LUT used in conjunction with either the affine or the inverse affine transformation, see the framed part of fig.4. The savings in the order of 30% to 50% of area so obtained eventually made it possible to instantiate two such hardware rounds on the chip, see fig.7, thereby almost doubling overall throughput.

3.2 Reorganizing a Cipher Round for Pipelining

As can be seen in fig.4, which also includes the propagation delays in the datapath, the longest path in this configuration is about 12ns. The next goal was to maximize throughput by recurring to intraround pipelining, that is by inserting pipeline registers into the datapath hardware of one round. The architecture of fig.4 is unsuitable for doing so because no location can be found for an extra register that would significantly cut down the longest path for both encryption and decryption.

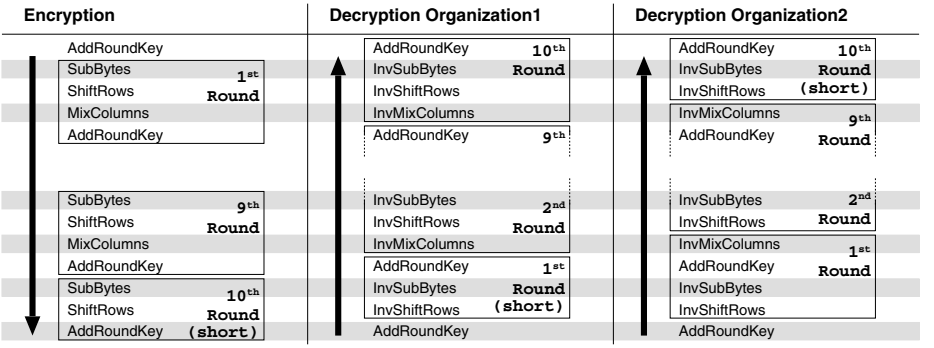


Fig. 5. The two options for delimiting one Rijndael decryption round.

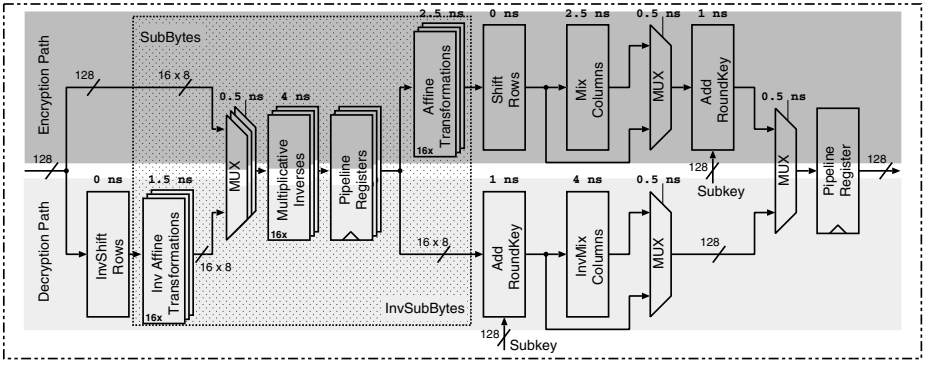


Fig. 6. A reorganized Rijndael round amenable to intraround pipelining.

Therefore, the hardware architecture of one round was reorganized without altering the circuit's functionality. Figure 5 displays two options for delimiting decryption rounds. The first option corresponds to fig.4 whereas the second option is shown in fig.6. This second option is amenable to pipelining while the

first is not. By inserting pipeline registers after the multiplicative inverse, the longest path is cut down from 12ns to 7ns thereby greatly improving throughput once more at little extra cost.

3.3 Precomputing Subkeys

The chosen architecture with two pipelined hardware rounds implies that a total of four 128bit data blocks are being processed concurrently at any given time. Computing the subkeys on the fly seemed no desirable option in this case because four subkey computation units would be required to provide the four different subkeys. Therefore all eleven subkeys are precomputed and stored in registers.

Also, working from precomputed subkeys avoids any key setup time when changing from encryption to decryption or vice versa. Each 128bit data block can thus either be encrypted or decrypted independently of the precedent block.

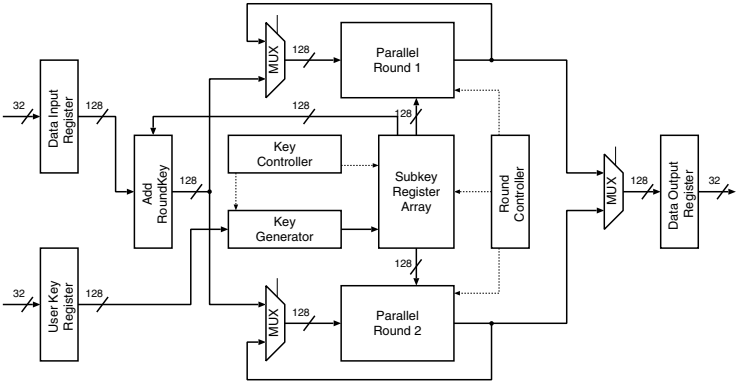


Fig. 7. Final architecture of the Rijndael chip.

4 The Serpent Implementation

4.1 Separating Decryption from Encryption

All of the 32 Serpent encryption rounds follow the same pattern, see fig.1. A key mixing operation where a 128bit subkey is XORed with the data block comes first, followed by an array of 32 parallel 4bit×4bit S-boxes. A subsequent linear transformation concludes the transformation round. Only in the final round is the linear transformation replaced by an extra key mixing operation with a 33rd subkey.

During decryption, both the S-boxes and the linear transformation need to be inverted. As opposed to the Rijndael algorithm, no method was found to reuse the same S-boxes for both encryption and decryption. This is no real

handicap because the much smaller size of the Serpent LUT's would hardly justify introducing multiplexers and other control hardware anyway. Only the relatively small key mixer might be reused. In this situation, it seemed more efficient to implement separate datapaths for encryption and decryption with no elements shared. The result is shown in fig.8.

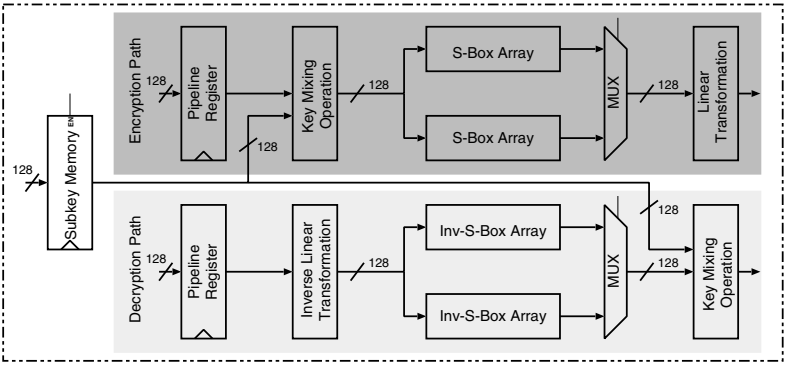


Fig. 8. A Serpent round with two separate datapaths.

4.2 Systematic Allocation of Hardware Resources

The basic solution space for the realization of the Serpent algorithm has been presented in fig.3. The computed figures suggested that a solution that includes a total of four hardware rounds followed by a pipeline register after each round was the best choice, considering that the maximum chip area of 50mm^2 was a hard limit.

As the Serpent algorithm makes use of eight different types of S-boxes, there is no way to avoid implementing all of them in hardware. Unless one can afford to instantiate eight or more hardware rounds, multiplexers and control logic must be included to switch look-up tables in and out depending on the cipher round currently being processed. This adds to the cumulative area occupied by one round and introduces extra data delay. The four hardware rounds are clearly visible in fig.9 which shows the chip's overall architecture.

4.3 Generating Subkeys on the Fly

Precomputing the full set of Serpent subkeys and storing them would require more than 4Kbit of memory which corresponds to an area of more than 3.6mm^2. This seeming impractical, it was decided to compute all subkeys on the fly concurrently with data processing.

Each round has a single associated subkey register, that stores the subkey to be applied to a data block at a certain time. Four consecutive data blocks

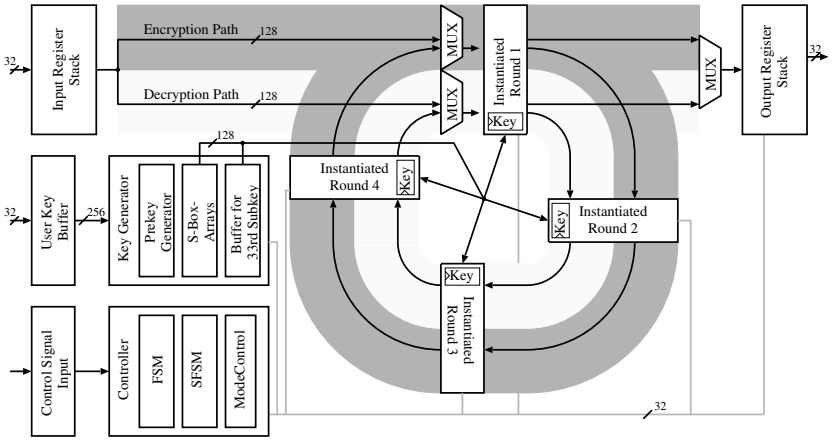


Fig. 9. Final architecture of the Serpent chip.

are processed one after the other in the same hardware round using the same subkey. Thus, the content of a subkey register can be applied four times in a row. As a consequence, three of the four existing rounds can always reuse the subkey stored within the local subkey register. One round, however, requires that a new subkey be delivered. This new subkey is supplied by a key generator that is capable of computing one new subkey during each clock cycle. After the last subkey has been prepared, the user key gets reloaded from an internal buffer to serve as starting point for computing the first round key again.

The last round of the Serpent algorithm presents a small problem as it uses an additional round key instead of the linear transformation. Since the key generator is not able to supply more than a single subkey per clock cycle, this 33rd subkey is computed ahead of time and stored in a register. Consequently, the key generator has to complete one full run to obtain the last round key prior to data processing, which results in a relatively long key setup time.

As the subkeys are not stored, the key generator also needs to be able to calculate the subkeys in reverse order. To accomplish this, once the encryption mode has generated the last two subkeys, a second, parallel, key generation unit is used to generate the subkeys in reverse order.

5 The Two Integrated Circuits Compared

Figure 10 shows the floorplans of the two VLSI chips while table 1 compares their key technical characteristics. Recurring to a multiproject wafer (MPW) service, both the Rijndael and the Serpent designs have been fabricated in prototype quantities. All measured figures in table 1 refer to the physical circuits so obtained. In either case, throughput figures in the order of 2Gbit/s have been obtained in a mature 0.6μm technology.

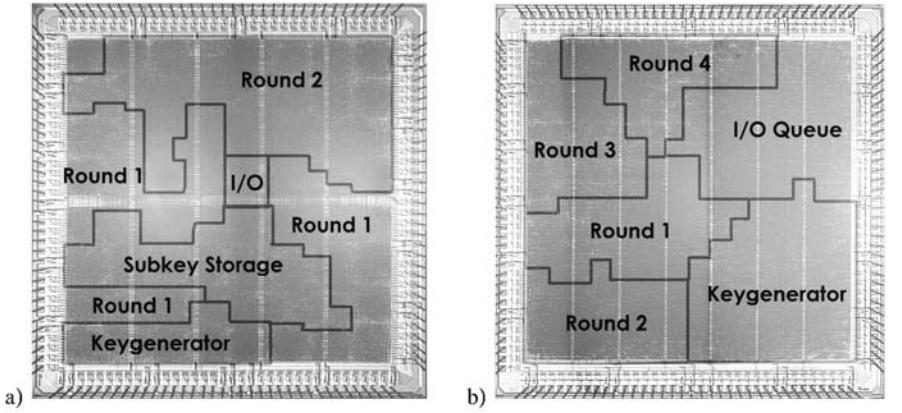


Fig. 10. The photomicrographs of a) Rijndael and b) Serpent with their main components highlighted.

While the Serpent circuits were quickly found to work correctly, the Rijndael chips exhibited a systematic malfunction. The problem has eventually been traced back to a mistake in the postprocessing of layout data for MPW assembly. A couple of lines running on the topmost metal have inadvertently been shorted together.

Coincidentally, only one of the two hardware encryption/decryption rounds of fig.6 has been afflicted. This made it possible to work around the defect and to verify the correct operation of the second round. The throughput figure presented for Rijndael is the throughput figure for the design with both parallel rounds working at the clock rate at which the second round was shown to be working.

The various functional blocks have been identified in the physical layouts of the two chips. In either design, the round controller occupies a relatively small area and is not particularly marked. Also note that approximately half of the large I/O queue region shown on the Serpent floorplan (fig.10b) includes subcircuits from other functional blocks.

The Serpent key generator can be seen to occupy a larger area than its Rijndael counterpart. This is mainly because all eight separate 4-bit S-box variants of the Serpent cipher need to be instantiated 32 times in order to calculate the subkeys on the fly.

In either floorplan, a significant proportion of the core area is lost to routing overhead. Also observe that Synopsys area estimations are off by a factor of two. In our opinion, the reasons for this poor area utilization are very wide data words in conjunction with a target technology that provides just three layers of metal, and standard cells with overly many routing blockages.

In theory, a more aggressive pipelining strategy should result in still higher throughput rates. Yet, we felt that clock frequencies much beyond 100MHz in

Table 1. Our Rijndael and Serpent implementations compared.

	Rijndael	Serpent
rounds in the algorithm	10	32
rounds instantiated in hardware	2	4
key length	128bit	256bit
subkey computation	stored	on the fly
core key agility [clock cycles (ns)]		
new encryption key	3 (34)	21 (171)
new decryption key	23 (260)	21 (171)
switch between encryption and decryption	0 (0)	21 (171)
number of flip-flops	2'607	3'274
number of transistors	300k	300k
technology	0.6μm 3LM	0.6μm 3LM
process name	AMS CUA	AMS CUA
area per hardware round	6.3mm^2	3.1mm^2
area for subkey generation	4.5mm^2	3.8mm^2
estimated chip area (after synthesis)	22.5mm^2	21.6mm^2
actual chip area (after physical layout)	49.0mm^2	49.0mm^2
data throughput in ECB mode (encr or decr)	2.26Gbit/s	1.96Gbit/s
@ clock frequency	88.5MHz	122.9MHz
latency [clock cycles (ns)]	28 (316)	56 (455)

our target technology would give rise to significant difficulties with off-chip data transfer and with clock distribution.

Both designs have been balanced for encryption and decryption and indeed achieve similar throughput rates for either operation. The Rijndael implementation sports remarkably short key setup times and, most notably, does not require any additional setup time when switching between encryption and decryption with the same user key. The Serpent circuit's key setup time, on the other hand, suffers from the necessity to calculate the last subkey in advance.

The core circuits have been designed to run with 128bit data but practical considerations have limited input and output width to 32bit. Nevertheless, no performance is lost thanks to careful scheduling of I/O operations.

The original AES specification calls for three key lengths of 128, 192 and 256bit respectively. To simplify design, only 128bit implementations have been considered in this study. The Serpent design is capable of using key lengths of up to 256bit without any modification, whereas the Rijndael circuit would need to be adapted to accommodate multiple key lengths.

The throughput figures presented refer to the ECB mode of operation. Of the various feedback modes proposed for AES use [12], only the CTR mode will achieve similar data throughput. For CFB, OFB and CBC modes of operation without interleaving the highest throughput per area can be obtained if and only if an iterative architecture with only a single hardware round is implemented without any pipelining. This solution is on the bottom left corner of the solution

space presented in fig.3 and is expected to have resulted in a throughput in the order of 500Mbit/s for both algorithms.

6 Conclusions

To begin with, note that the two ciphers have many traits in common, both lend themselves fairly well for hardware implementation. Most importantly, there are no feedback loops whatsoever in ECB and CTR mode that would present unsurmountable bottlenecks when in search of maximum throughput. A number of tricks are instrumental in turning a purely algorithmic prescription into a highly efficient architecture, but this is common practice in VLSI design.

We have made valuable contributions towards designing optimum hardware for Rijndael by relocating the boundary between two consecutive rounds and by restating table look-up operations. Both designs benefit from the evaluation of distinct subkey generation schemes and from the systematic exploration of architectural trade-offs.

Table 2 compares the two cipher algorithms from the perspective of a VLSI architect. The fact that the same S-box LUT can be reused for encryption and decryption throughout all rounds probably is the most important advantage of Rijndael. Conversely, the fact that the number of rounds and subkey generation are dependent on the width of the user key are less desirable features.

Table 2. The two ciphers compared from a VLSI architect's point of view.

Rijndael	Serpent
+Small number of rounds (10...14). +Small number of subkeys (11...15). −No. of rounds depends on key length.	−Large number of rounds (32). −Large number of subkeys (33). +Fixed number of rounds.
+No complex mathematical operations. +All rounds are identical (same S-box type throughout). −Large S-box (8bit×8bit).	+No complex mathematical operations. −Eight different S-box types. +Small S-boxes (4bit×4bit).
−Cipher is not involutory. +Look-up tables can be made to share between en- and decryption. ∘ Not all hardware components can be shared between en- and decryption.	−Cipher is not involutory. −All S-boxes and their inverses must be implemented in hardware. −Almost no hardware components can be shared between en- and decryption.
−Key generation varies with key width.	+Wider key entails no extra complexity. −No efficient way to compute the 33rd subkey from the user key directly.

To our knowledge this is the only published study where the actual ASIC implementations of two AES candidates have been compared. In this respect this study differs from previous comparisons [4,8] and realizations [6], as it also takes into account real-life problems of ASIC integration such as placement and routing, interconnection parasitics, clock distribution and I/O limitations.

In [4] two extreme cases are considered: for an iterative architecture Rijndael was shown to have three times the throughput of Serpent with an estimated area 1.5 times larger. For a fully pipelined architecture, with almost identical area requirements the throughput of Serpent was thirty percent higher than Rijndael. In another study that concentrated on finding the critical path for feedback modes [8] Rijndael was found to be more than twice as fast as Serpent with an estimated area approximately twenty percent larger.

A direct comparison of our Rijndael implementation to the one presented in [6] is difficult as that implementation uses a much more advanced 0.18μm technology, estimated to be almost 10 times smaller and faster than the 0.6μm technology used in this study. Also the implementation in [6] does not support decryption and the stated throughput of 1.82Gbit/s is for 256bit data blocks. In this respect our implementation with a measured 2.26Gbit/s throughput using 128bit data blocks compares fairly well.

Considering that the two algorithms are rather different in nature, their respective performances in hardware come remarkably close. From our experience with designing circuits for a fixed key length of 128bit and for throughputs in the order of a few Gbit/s, we consider Rijndael to be more favorable than Serpent, although only slightly.

References

1. Daemen, J., Rijmen, V.: AES Proposal: Rijndael. Submission to the First Advanced Encryption Standard Candidate Conference. Ventura CA, August 1998.
2. Anderson, R., Biham, E., Knudsen, L.: Serpent: A Proposal for the Advanced Encryption Standard. Submission to the First Advanced Encryption Standard Candidate Conference. Ventura CA, August 1998.
3. Nechvatal, J., Barker, E., Bassham, L., Burr, W., Dworkin, M., Foti, J., Roback, E.: Report on the Development of the Advanced Encryption Standard (AES). NIST, Computer Security Division, Information Technology Laboratory, October 2000.
4. Weeks, B., Bean, M., Rozylowicz, T., Ficke, C.: Hardware Performance Simulations of Round 2 Advanced Encryption Standard Algorithms. National Security Agency (NSA).
5. Kaeslin, H.: From Algorithms to Architectures. Lecture Notes in VLSI Design, Microelectronics Design Center, ETH Zürich.
6. Kuo, H., Verbauwhede, I.: Architectural Optimization for a 1.82Gbits/sec VLSI Implementation of the Rijndael Algorithm. Proceedings of the Third International Workshop of Cryptographic Hardware and Embedded Systems CHES 2001, Paris, May 2001.
7. Elbirt, A., Yip, W., Chetwynd, B., Paar, C.: An FPGA-Based Performance Evaluation of the AES Block Cipher Candidate Algorithm Finalists. IEEE Transactions on VLSI, August 2001, vol. 9, no. 4, 545–557.
8. Ichikawa, T., Kasuya, T., Matsui, M.: Hardware Evaluation of the AES Finalists. Proceedings of the Third Advanced Encryption Standard Candidate Conference, New York, April 2000, 279–285.
9. Gaj, K., Chodowiec, P.: Comparison of the hardware performance of the AES candidates using reconfigurable hardware. Proceedings of the Third Advanced Encryption Standard Candidate Conference, New York, April 2000, 40–54.

10. Weaver, N., Wawrzynek, J.: A Comparison of the AES Candidates Amenability to FPGA Implementation. Proceedings of the Third Advanced Encryption Standard Candidate Conference, New York, April 2000, 28–39.
11. Gaj, K., Chodowiec, P.: Hardware performance of the AES finalists – survey and analysis of results. Technical Report, George Mason University, September 2000.
12. Dworkin, M.: Recommendation for Block Cipher Modes of Operation – Methods and Techniques. NIST Special Publication 800-38A, 2001.

Efficient Software Implementation of AES on 32-Bit Platforms*

Guido Bertoni[1], Luca Breveglieri[1], Pasqualina Fragneto[2], Marco Macchetti[3], and Stefano Marchesin[3]

[1] Politecnico di Milano , Milano, Italy
{bertoni,breveglieri}@elet.polimi.it
[2] STMicroelectronics, Agrate B.za MI, Italy
{pasqualina.fragneto}@st.com
[3] ALaRI Università della Svizzera Italiana, Lugano, Switzerland
{macchetti,marchesin}@alari.ch

Abstract. Rijndael is the winner algorithm of the AES contest; therefore it should become the most used symmetric-key cryptographic algorithm. One important application of this new standard is cryptography on smart cards. In this paper we present an optimisation of the Rijndael algorithm to speed up execution on 32-bits processors with memory constraints, such as those used in smart cards. First a theoretical analysis of the Rijndael algorithm and of the proposed optimisation is discussed, and then simulation results of the optimised algorithm on different processors are presented and compared with other reference implementations, as known from the technical literature.

1 Introduction

Rijndael, a block cipher algorithm designed by Vincent Rijmen and Joan Daemen [1], has been selected by NIST as the winner of the Advanced Encryption Standard competition [2]. Although the initial specification of the algorithm includes 128-bits, 192-bits and 256-bits as possible lengths for both the plaintext blocks and for the key material, the standard will consider only 128-bit as legal block length. In this paper we shall deal only with 128-bits blocks.

According to [3], the basic information unit for processing in the Rijndael algorithm is a byte, i.e. a sequence of eight bits treated as a single entity. The bit sequences corresponding to the input, the output and the cipher key are processed as arrays of bytes; these arrays are formed by dividing the sequences into groups of eight contiguous bits. Internally, the operations of the algorithm are performed on a two-dimensional array of bytes called "State". The State array consists of four rows of bytes, each row containing 4 bytes. The Rijndael cipher algorithm operates in rounds, a round being a fixed set of transformations to be applied to the State array. The number of these rounds is chosen depending on the key length and ranges from 10 to 14.

* Part of this work is under patenting process.

B.S. Kaliski Jr. et al. (Eds.): CHES 2002, LNCS 2523, pp. 159–171, 2003.

The Rijndael cipher algorithm is suited for an efficient implementation on a wide range of processors. The basic operations involved in the algorithm are very simple and the structure of the algorithm is straightforward. The Rijndael algorithm can be used as encryption standard in embedded systems. One important application field of the Rijndael algorithm is cryptography on smart cards. Currently DES is used in such systems, but industry is moving to replace it with the new AES algorithm.

The present paper considers optimised software implementations of the AES algorithm for several platforms, with particular regard to smart cards. Memory requirements are a fundamental issue when coding the Rijndael algorithm for smart cards. In fact, the algorithm can be considerably sped-up by precomputing part of the internal operations and storing the results in look-up tables. In our case this means that we want to achieve the best possible performance with a little amount of look-up tables, since in a smart card environment memory and silicon space are limited resources. We also have to consider that the smart card market is looking to 32-bits microprocessors as the new leading technology [5], although a large amount of manufactured cards still features 8-bits and 16-bits processors.

In this paper we shall describe a technique to enhance the time performances of the AES algorithm when running on 32-bits processors with strict silicon space constraints. This optimisation consists in a deep restructuring of the algorithm, which makes possible to organise the inner operations (i.e. the rounds and the byte-operations involved in each round) in a different way with respect to the standard formulation [12] of the algorithm. This restructuring can be nicely described in theoretical terms as a matrix transformation. The so-called State matrix of AES is transposed and the inner operations of each round are reorganised accordingly. Some inner operations are commuted with respect to other ones, and are then grouped in such a way as to fit well in processors having 32-bits words. This optimisation allows a better exploitation of the resources of the processor, and thus achieves also better time performances with respect to the standard formulation [12] of the algorithm.

The optimised version of the algorithm was coded in C for evaluation on various platforms, covering a wide range of possible applications. Namely, ARM (a typical processor for embedded systems), ST 22 (one of the most advanced 32-bits processors for smart cards) and also Intel Pentium (typical for general purpose systems) are considered. Simulations have been carried out for both Gladman's standard AES implementation in C code and our optimised AES implementation, on all the previously mentioned platforms. Some other implementations of AES, known from the literature, are also considered. The time performances obtained by simulation are summarized in tables, compared and discussed. In several of the examined cases, such results are highly favourable to our proposed optimised version of the AES algorithm.

This paper is organized as follows. Section 2 will provide a synthetic outline of the Rijndael algorithm. Section 3 will describe and analyse theoretically our new, optimised approach to the algorithm. In Section 4 we shall discuss our

implementation in C code and we shall show simulation results and comparisons with other implementations on various platforms. Section 5 concludes the paper.

2 Description of the Rijndael Algorithm

This section provides a brief recall of the AES algorithm (Rijndael, [1][2][3]), useful for understanding the subsequent optimisations, which will be described in section 3.

The core data structure of AES is the State matrix: it is a 4 * 4 bytes matrix. As we have already said in the introduction, the Rijndael encryption algorithm operates in rounds; a round is a fixed set of transformations that are applied to the State matrix. The number of these rounds is chosen depending on the length of the key; it is necessary to perform 10, 12 or 14 rounds in the cases of 128, 192 or 256-bits keys, respectively. For each round of the AES algorithm a round key is derived from the original key; this process is called Key Scheduling.

The transformations that are applied in each round are four. According to [6], they correspond to the round key addition step, the non-linear step, the dispersion step and the diffusion step. They are described as follows.

AddRoundKey. In this transformation, a round key is added to the State matrix by a simple bitwise XOR operation, that is, a sum in the field $GF(2^8)$. Each round key is obtained from the key schedule.

SubBytes. This transformation is a non-linear byte substitution operating independently on each byte of the State matrix, using a substitution table (called S-BOX). This S-BOX, which is invertible, is constructed by composing two transformations in $GF(2^8)$, an inversion and an affine function.

ShiftRows. In this transformation, the bytes in the last three rows of the State matrix are cyclically shifted over different numbers of bytes (offsets). The first row, row 0, is not rotated. Row 1 is rotated to the left by 1 byte position; row 2 is rotated to the left by 2 byte positions; row 3 is rotated to the left by 3 byte positions.

MixColumns. This transformation operates on the State matrix in a column-by-column mode, treating each column as a four-term polynomial over $GF(2^8)$. These polynomials are multiplied modulo $(x^4 + 1)$ with a fixed polynomial a(x), given by the expression:

$$a(x) = 03 * x^3 + 01 * x^2 + 01 * x + 02$$

This polynomial is coprime to $(x^4 + 1)$, and therefore the transformation is invertible. This transformation can be written under the form of a matrix multiplication. Pose $s'_c(x) = a(x) \otimes s_c(x)$, for $0 \leq c \leq 3$, that is for all the 4 columns

in the State matrix. As a result of this multiplication, the 4 bytes in a column c are replaced by the following ones (for c = 0, 1, 2 and 3):

$$s'_{0,c} = 02 * s_{0,c} \oplus 03 * s_{1,c} \oplus s_{2,c} \oplus s_{3,c}$$
$$s'_{1,c} = s_{0,c} \oplus 02 * s_{1,c} \oplus 03 * s_{2,c} \oplus s_{3,c}$$
$$s'_{2,c} = s_{0,c} \oplus s_{1,c} \oplus 02 * s_{2,c} \oplus 03 * s_{3,c}$$
$$s'_{3,c} = 03 * s_{0,c} \oplus s_{1,c} \oplus s_{2,c} \oplus 02 * s_{3,c}$$

where the * operator stands for a multiplication in $GF(2^8)$, with:

$$m(x) = x^8 + x^4 + x^3 + x + 1$$

as irreducible generator polynomial. Performing a complete round means simply applying these 4 transformations to the State matrix, in the following order:

$$round = \{SubBytes, ShiftRows, MixColumns, AddRoundKey\}$$

Performing the final round means simply applying to the State matrix the following transformations, in the order:

$$finalround = \{SubBytes, ShiftRows, AddRoundKey\}$$

The Rijndael encryption algorithm consists in an initial application of the AddRoundKey operation, followed by (number of rounds - 1) rounds and concluded with a final round. The Rijndael decryption algorithm operates by applying the inverse of all the transformations described above in reverse order, to return to the plaintext; for specific details, see [1].

3 A New Technique for Computing the Rijndael Algorithm

This section illustrates our optimised version of the Rijndael AES algorithm. Both the encryption and the decryption algorithms have been optimised. For reasons of brevity, attention is focused on encryption, giving only the necessary hints for understanding the way to optimise decryption as well. The task is divided in two parts: optimisation of the algorithm working on the State matrix, and modification of the key scheduling. In both cases, the base line consists in the transposition of the State matrix, and the consequent rearranging of the various transformations. In fact, as a consequence of the transposition of the State matrix, also the key scheduling is rearranged in a suited way. In section 3.1 the algorithm optimisation is explained, while in section 3.2 the key scheduling is considered.

3.1 The Transposed State Matrix Primitives

It is possible to enhance the throughput of the implementation of AES by changing the way in which data are represented by the software. In particular, the internal transformations of a round could be implemented by using look-up tables. In our study, we have chosen to reserve a little amount of space for look-up tables: the choice is to tabularise only the S-BOX and the inverse S-BOX transformations. All the remaining operations are computed. In particular, this means that we must carry out several GF multiplications only by means of software techniques. All the primitives considered in our study behave in a peculiar way, operating on a transposed version of the State matrix. Of course, all the steps of the algorithm must be modified in order to preserve global functionality while operating on the transposed State matrix. In particular:

The **SubBytes** transformation is not modified, since it operates on single bytes, independently of their position in the State matrix.

The **ShiftRows** transformation does not shift the rows of the State matrix any longer; instead, it operates now in the same way on the columns.

The **MixColumns** transformation is deeply revised. Denote with x_i, for $0 \leq i \leq 3$, the 32-bits words (or columns) of the transposed State matrix before applying the MixColumns transformation, and denote with y_i, for $0 \leq i \leq 3$, the 32-bits words (or columns) of the transposed State matrix after applying the MixColumns transformation. The revised version of MixColumns is then represented by the following set of equations:

$$
\begin{aligned}
y_0 &= 02 * x_0 \oplus 03 * x_1 \oplus x_2 \oplus x_3 \\
y_1 &= x_0 \oplus 02 * x_1 \oplus 03 * x_2 \oplus x_3 \\
y_2 &= x_0 \oplus x_1 \oplus 02 * x_2 \oplus 03 * x_3 \\
y_3 &= 03 * x_0 \oplus x_1 \oplus x_2 \oplus 02 * x_3
\end{aligned}
$$

The variables y_i and x_i contain the 4 bytes at position i of the columns of the State matrix in the normal, non-transposed, version of the transformation. These variables are 32 bits long. Note that here the symbol * does not denote an ordinary GF multiplication over factors (polynomials) of 32 bits. Instead, here the operator * denotes a set of 4 ordinary multiplications in the field $GF(2^8)$, performed in parallel on the 4 bytes of each 32-bits word. The generator polynomial used for representing the field $GF(2^8)$ is the standard one of AES.
A simple way to calculate all the above operations, composing the MixColumns transformation, is to use the y_i variable as an accumulator, and to use the x_i variable for storing the product of the initial values of x_i and of the 4 partial products: x_i, 2 * x_i, 4 * x_i and 8 * x_i. In the case of encryption the products to be used in the calculation are only two: 2^0 and 2^1. Therefore the MixColumns

transformation is computed in only 3 steps: a sum step, a doubling step and a final sum step. Table 1 shows the three steps.

Table 1. The three steps necessary to compute MixColumns.

First	Second	Third
$y_0 = x_1 \oplus x_2 \oplus x_3$	$x_0 = 02 * x_0$	$y_0 \oplus = x_0 \oplus x_1$
$y_1 = x_0 \oplus x_2 \oplus x_3$	$x_1 = 02 * x_1$	$y_1 \oplus = x_1 \oplus x_2$
$y_2 = x_0 \oplus x_1 \oplus x_3$	$x_2 = 02 * x_2$	$y_2 \oplus = x_2 \oplus x_3$
$y_3 = x_0 \oplus x_1 \oplus x_2$	$x_3 = 02 * x_3$	$y_3 \oplus = x_0 \oplus x_3$

In the case of decryption this double-and-add method is used in a more substantial way, and the steps necessary to compute the InvMixColumns transformation are 7: 4 sum steps and 3 doubling steps. In the case of InvMixColumns the double-and-add method can be improved by considering the particular values of the constant coefficients. Note that there are only two coefficients containing a bit 1 in the third position, namely the coefficients 0e and 0d. These coefficients are used in combination with the operands x_0 and x_2 contained both in the first row and in the third row, and are used in combination with the operands x_1 and x_3 contained both in the second row and in the fourth row. To save a doubling operation we can add these two operand pairs and store the result in x_0 and x_1, respectively. Instead of calculating the subexpression $04 * x_0 \oplus 04 * x_2$, we can calculate the subexpression $04^*(x_0 \oplus x_2)$, since we do not need to store separately either addend. Moreover, note that every coefficient contains a bit 1 in the fourth position. The last calculation deals with this bit. Hence we can add the previously computed values x_0 and x_1, which are $04^*(x_0 \oplus x_2)$ and $04^*(x_1 \oplus x_3)$, respectively, and then double them, so that the subexpression $08^*(x_0 \oplus x_1 \oplus x_2 \oplus x_3)$ is obtained, which must be accumulated to every operand y_i.

The **AddRoundKey** transformation remains quite unchanged, since it consists in a simple bitwise XOR between the State matrix and the round keys. Of course, we have to ensure that round keys are transposed before being used, which is shown in the next section.

3.2 The Transposed State Matrix Key Scheduling

As we stated before, we need to transpose the round keys before using them. A trivial solution would be simply to apply the key scheduling and then to transpose every created round key. In this way we would introduce a huge computation overhead. One alternative, which is the implemented one, is to redesign the key scheduling directly in the "transposed manner".

For 128-bits keys the key scheduling operates intrinsically on blocks of 4 32-bits words; we can calculate one new round key from the previous one. We denote the i^{th} word of the actual round key with $K[i]$, where $0 \leq i \leq 3$, and the i^{th} word of the next round key with $K'[i]$. $K'[0]$ is computed by an XOR between $K[0]$, a constant rcon and $K[3]$, the latter being pre rotated and transformed using the S-BOX. The other three words $K'[1]$, $K'[2]$ and $K'[3]$ are calculated as $K'[i] = K[i] \oplus K'[i-1]$.

We must now rewrite this set of transformations to cope with transposed keys. We indicate with K_T the transposed round key we are working on, and with K'_T the new transposed key. Clearly we have:

$$\mathbf{K_T[0]} = \begin{bmatrix} k_0 \\ k_4 \\ k_8 \\ k_{12} \end{bmatrix} \mathbf{K_T[1]} = \begin{bmatrix} k_1 \\ k_5 \\ k_8 \\ k_{13} \end{bmatrix} \mathbf{K_T[2]} = \begin{bmatrix} k_2 \\ k_6 \\ k_9 \\ k_{14} \end{bmatrix} \mathbf{K_T[3]} = \begin{bmatrix} k_3 \\ k_7 \\ k_{10} \\ k_{15} \end{bmatrix}$$

The transposed key schedule is made up of the following transformations:

$$\begin{aligned}
K'_T[0] &= K_T[0] \oplus (pad(Sbox(k_{13})) << 24) \oplus rcon \\
K'_T[1] &= K_T[1] \oplus (pad(Sbox(k_{14})) << 24) \\
K'_T[2] &= K_T[2] \oplus (pad(Sbox(k_{15})) << 24) \\
K'_T[3] &= K_T[3] \oplus (pad(Sbox(k_{12})) << 24) \\
K'_T[0] \oplus &= (K_T[0] >> 8) \oplus (K_T[0] >> 16) \oplus (K_T[0] >> 24) \\
K'_T[1] \oplus &= (K_T[1] >> 8) \oplus (K_T[1] >> 16) \oplus (K_T[1] >> 24) \\
K'_T[2] \oplus &= (K_T[2] >> 8) \oplus (K_T[2] >> 16) \oplus (K_T[2] >> 24) \\
K'_T[3] \oplus &= (K_T[3] >> 8) \oplus (K_T[3] >> 16) \oplus (K_T[3] >> 24)
\end{aligned}$$

The symbol $>> j$ ($<< j$) indicates a right (left) shift of j bit positions, with the insertion of j bits of value 0 in the most (least) significant positions, while *pad* means zero-padding of the 24 most significant bits of the word since *Sbox* returns an 8 bits value. We can note that the computation overhead with respect to the normal, non-transposed key scheduling is just few shift operations.

The key schedule for the other key sizes, i.e. 192 and 256-bits, is similar; note however that in the 192-bits case the calculations to be performed are slightly more complex. In fact, in this case it is necessary to operate on blocks of 6 32-bits words, while the round keys to be generated in the transposed form occupy 4 words of 32-bits. To reduce the overhead our solution consists in using a full array of 8 words of 32-bits for the calculations, as in the 256-bit key case. The new round key (similarly to the old one) occupies the first 4 words and the top half of the last 4 words. Then the words are suitably shifted and stored. The key schedule for 256-bits keys is very similar to that for 128-bits keys and therefore it will not be discussed here.

4 Implementation and Time Performance Figures

We have a C/C++ implementation of our proposal of the AES optimised algorithm. A simulation campaign has been carried out, in order to evaluate time performances. We report the results of the AES optimised algorithm only in the case of a key size of 128 bits, but we have tested the proposed algorithm also with 192 and 256-bits key size. The results are quite the same as those obtained with 128-bits key size, only scaled by a constant factor due to the larger number of rounds required by these versions of the AES algorithm. The time performance gain of each round, with respect to the standard AES algorithm, remains the same in all the cases.

We have chosen to make a direct comparison with an equivalent version of AES by Dr. Brian Gladman [8], since Gladman has been involved in the definition of the AES standard and his version is well referenced. For the comparison with embedded processors (ARM7, ARM9 and ST22) few refinements have been added to Gladman original implementation in C language [8].

Our code has been compiled and evaluated on some 32-bits architectures, including the ARM7TDMI and ARM9TDMI processors [7], the ST22 smart card processor by ST Microelectronics [5], and also on a general purpose Intel PentiumIII platform. These three platforms represent rather different architectures used in various systems and environments: embedded system, smart cards and PC, respectively.

The ARM7TDMI processor is a widely used 32-bits RISC CPU. It contains sixteen 32 bits registers. No cache is available and the internal structure consists of a pipeline of 3 stages [7]. The ARM9 processor differs from ARM7 in the internal structure. It is designed accordingly to the Harvard architecture model with two different busses for data and instructions, respectively, and the core is pipelined in 5 different stages. There exist different implementations of ARM9, depending on the amount of cache memory. In our simulation the standard core ARM9TDMI has been used, without cache memory, but the time latency for the accesses to the memory is only of one clock cycle, while the code and the data are stored in two different memories.

The ST 22 processor is a 32-bits RISC processor particularly designed for smart cards. It is a dedicated processor and is not used for applications different from smart cards. The internal details are not completely known; however the processor contains some 32 bits registers and does not have cache memory [5].

PentiumIII is a typical processor for PC systems. It is a 32-bits processor with a large amount of available memory, organized in three levels, with 32 kbytes of cache memory at the first level (divided in two blocks of 16 kbytes for data and instructions, respectively), with at least 256 kbytes of cache memory at the second level, and with some megabytes of RAM as central memory [4].

In the Rijndael algorithm the encryption and decryption operations must be executed using the round keys. For each round a different round key is used. The process of deriving the round keys from the original key is called Key Scheduling. The round key can be computed either in advance (key unrolling) or the so-called "on-the-fly" approach can be used. In the on-the-fly approach the var-

ious round keys are computed exactly when they are needed, and soon after they are discarded. In a software implementation of the Rijndael algorithm the "on-the-fly" approach is not useful in terms of speed since the key schedule must be performed for each data block to encrypt. Some particular systems, i.e. smart cards, do not allow using memory to store the unrolled key, both for reasons of security and of memory shortage. In those cases the on-the-fly approach is mandatory. We have implemented our optimisation of AES in two versions: the former one using key unrolling and the latter one using the on-the-fly approach. In order to implement decryption following the on-the-fly approach, it is advisable to store the last round key, from which the previous round keys can be reconstructed. This requires 16 bytes of memory. The alternative would be to generate all the round keys before starting decryption, and then to use them in reverse order; but this approach consumes more memory.

Since Gladman public C code for AES does not permit to use the on-the-fly approach, we have extended Gladman code implementing also the on-the-fly approach. In [12] it is possible to find some time performances of on-the-fly encryption algorithm on the ARM processor for smart cards applications (the so-called cAESar implementation, written in assembler language). The core processor specified in the paper illustrating the cAESar implementation is not clearly described but should be very similar to ARM7TDMI. As above mentioned, we assume to tabularise in a look-up table only the S-BOX transformation; all the other round transformations are computed as soon as they are needed, and the result is not stored for future use. The amount of used look-up tables is thus of 512 bytes for the S-BOX and the Inv-S-BOX, plus 10 bytes for the round constants rcon.

We have used the ARM simulator (ARMulator), the ST 22 development tools (courtesy by STM) and Microsoft Visual Studio 6.0. Table 2 shows the time performances of our proposal of AES and of Gladman's in the hypothesis of adopting key unrolling. Table 3 shows the time performances of our proposal of AES, Gladman's and cAESar (only on ARM) in the hypothesis of adopting the on-the-fly approach. In Table 2 Key Scheduling is listed separately from Encryption and Decryption, since it is computed completely in advance. The total time can be obtained by adding the key scheduling time to either the encryption or the decryption time. In Table 3 the key scheduling time is already included both in the encryption and the decryption time (since key scheduling is in this case interleaved with encryption or decryption).

As explained before, the application of the Rijndael algorithm consists of 3 parts, namely key scheduling, encryption and decryption. Our proposal provided us with a speed gain in the MixColumns (during encryption) and InvMixColumns (during decryption) transformations, requiring only few changes to the key scheduling. In general these two operations work as follows. A single MixColumns is a composition of sums and doublings in the field $GF(2^8)$, plus some rotations of the elements of the column. A sum in $GF(2^8)$ is a bitwise XOR of bytes and a doubling is a composition of a masking, a shift and a conditional bitwise XOR of bytes. Since a column is composed by 4 elements of the field

$GF(2^8)$, some operations can be applied in parallel to the entire column, as the whole column can be accommodated in a single register of the CPU. The InvMixColumns works in a similar way. Gladman implementation of MixColumns and InvMixColumns are applied to each one of the 4 columns of the State matrix.

Examining the Gladman implementation of MixColums, which uses the standard representation of the State matrix, it is possible to count the number of required operations. On a 32 bits platform a single MixColums requires 4 bitwise XORs plus one doubling of the four $GF(2^8)$ elements and 3 rotations. The MixColumns must be applied to the 4 columns giving a total of 16 XORs, 4 doublings and 12 rotations. Moreover, the original Gladman implementation requires an additional intermediate variable, which could be eliminated; therefore we shall not consider it.

Our optimisation of AES allows reducing the number of elementary operations. Using the transposed State matrix the rotations can be completely avoided (see sub-section 3.1) both in MixColumns and in InvMixColumns, thus yielding a speed gain. Moreover, in the InvMixColums the transposed State matrix allows to achieve a much higher speed gain. In fact, the transposed InvMixColums requires only 7 doublings and 27 sums, while in the standard (Gladman) implementation it is necessary to compute 12 doublings, 32 sums, 12 rotations and 4 intermediate variables are required. This means that using in decryption a transposed State matrix it is possible to obtain a reduction of 5 doublings, 5 sums and 12 rotations, and to eliminate completely the intermediate variables. This yields a further speed gain.

As a general comment, our proposal implementation of AES works much better than Gladman's in decryption for all platforms, both using key unrolling and key on-the-fly. In encryption the performances are instead more or less comparable. Table 2 and Table 3 need some more explanations, in order to relate the differences of time performances with the features of the adopted processor. Here they follow.

Table 2. Clock cycles required for AES on different platforms (using key unrolling).

CPU	Implementation	Key Schedule	Encryption	Decryption
ARM7TDMI	Our Proposal	634	1675	2074
	Gladman	449	1641	2763
ARM9TDMI	Our Proposal	499	1384	1764
	Gladman	333	1374	2439
ST22	Our Proposal	0.22	0.51	0.60
	Gladman	0.13	0.61	1
Pentium III	Our Proposal	370	1119	1395
	Gladman	396	1404	2152
	Gladman with tables	202 (encrypt.) 306 (decrypt.)	362	381

Table 3. Clock cycles required for AES on different platforms (using key on-the-fly).

CPU	Implementation	Encryption	Decryption
ARM7TDMI	Our Proposal	2074	2378
	Gladman	1950	3221
ARM	cAESar	2889	N.A.
	cAESar with tables	1467	N.A.
ARM9TDMI	Our Proposal	1755	1976
	Gladman	1623	2796
ST22	Our Proposal	0.72	0.82
	Gladman	0.75	1.13

ARM. The report and the discussion of the time performances start by considering the ARM processor. On this system our optimised version of AES is slightly slower then Gladman's as for encryption. This can be theoretically justified as follows: our encryption algorithm should be advantageous with respect to Gladman's, because it saves some rotation steps. But if the processor has some dedicated machine instruction able to combine bitwise XOR and rotation (like ARM7TDMI), then the advantage of having fewer rotations tends to disappear. However, when the algorithm is written in C, our implementation frequently executes additions (XOR) involving 3 operands of 32 bits each one. This fact can lead to a non-optimal use of the pipeline of the processor, and hence to degrade the performances of our optimised version of AES with respect to Gladman's. However, in decryption our proposal is considerably more efficient than Gladman's. The core of the ARM9 system is similar to the one of ARM7, but is more powerful, thus giving better results in comparison to ARM7. However, the performance ratios between our optimised version and Gladman's version for ARM9 remain approximately the same as those for ARM7.

ST 22, by ST Microelectronics, is an advanced smart card CPU. This processor is a 32-bits RISC CPU, having some 32 bits registers but without cache memory. Simulations have been carried out by using the ST 22 C compiler (courtesy by ST Microelectronics). It must be noted that ST 22 is not able to combine addition and shift (or rotate) in a single machine instruction. This means that ST 22 lacks instructions fitting particularly to the Gladman AES implementation. This fact impacts negatively on encryption when performing the simulation of the Gladman version of AES, while encryption in our optimised version does not suffer any penalty. On the other side, Gladman key scheduling is not affected by this feature of the processor, as it does not use shifts and rotations, while our version of key scheduling is affected negatively. The time performances are still comparable as for key scheduling and encryption, while they are much in favour to our optimised version as for decryption. For reasons of privacy, the performance figures are normalised to 1, instead of reporting the absolute values.

PentiumIII. As a last comparison we report the time performances on the PentiumIII processor. Note that in such a system an implementation with a

complete replacement of the calculations with look-up tables is faster then our implementation. Therefore we report 3 versions: "Gladman with tables" (all transformations are tabularised in look-up tables), Gladman and our proposal, both of which use look-up tables only for the S-BOX and Inv-S-BOX transformations. We remember that the Gladman implementation using tables for all the transformations requires a very large amount of memory, of about 20 kbytes. This usage of memory is not affordable in systems as smart cards.

It is possible to see that our optimised version of AES works well in most cases, with few exceptions. In general the largest performance gains are obtained for decryption. However, even in those cases where our AES version behaves worse than Gladman's, the difference is limited and is by far less relevant than the considerable performance gain obtained in the case of decryption. Moreover, our proposal includes some initial and final code for transposing the State matrix (that is, the data block). The initial transposition code requires about 20 cycles for all platforms, and similarly the final transposition code. The transposition code is required for making our AES optimisation equivalent to the standard one. However, should the data block to encrypt be supplied directly in transposed form, the transposition code could be stripped off.

5 Conclusions

An optimised version of the AES standard has been presented, coded in C and evaluated by simulation on various platforms: ARM for embedded systems, ST 22 for smart card applications and also Intel Pentium for general purpose systems. We have rewritten the basic transformations of the Rijndael cipher algorithm, using a transposed version of the so-called State matrix. We have shown that this relevant structural modification leads to a considerable improvement of time performances in decryption. As for encryption, the time performances of our version of AES and Gladman's are instead more or less the same. These results hold when a limited part of the AES computation is carried out using look-up tables. Namely we have supposed that only the S-BOX operation is tabularised and stored in a look-up memory.

In the other cases, that is when a considerable part of the algorithm is executed by using look-up tables, our proposal of AES and Gladman's become approximately equivalent. It must be noted, anyway, that using a large amount of memory for the look-up tables is too expensive for small systems, like for instance smart cards, and is considered unsafe, as monitoring the accesses to the look-up tables may in some cases allow to infer the key. Therefore, at least for these applications the choice of not resorting to look-up tables should be considered reasonable. Next research directions include for instance the hardware evaluation, in terms of silicon area and time latency, of our optimisation of AES, with respect to the standard implementation, and possibly the design of a suited instruction set targeted to a fast computation of AES.

References

1. J. Daemen, V. Rijmen , "AES Proposal: Rijndael", http://csrc.nist.gov/encryption/aes/, 1999
2. NIST, "Announcing the ADVANCED ENCRYPTION STANDARD (AES)," Federal Information Processing Standards Publication, n. 197, November 26, 2001.
3. B. Gladman, "A Specification for Rijndael, the AES Algorithm" http://fp.gladman.plus.com/, 2001.
4. Intel Ltd. website, www.intel.com
5. STMicroelectronics website, www.st.com
6. J. Daemen, V. Rijmen, "Efficient Block Ciphers for Smart-Cards", *Workshop on Smartcard Technology (Smartcard '99)*, pp. 29–36, USENIX Eds., 1999
7. ARM Ltd. website, www.arm.com
8. B. Gladman, available at http://fp.gladman.plus.com/cryptography_technology/rijndael/
9. D. Whiting, B. Schneier, S. Bellovin, "AES Key Agility Issues in High-Speed IPsec Implementations," *Counterpane Internet Security*, http://www.counterpane.com/aes-agility.html, 2000.
10. G. Hachez, F. Koeune, J. J. Quisquater, "cAESar Results: Implementation of Four AES Candidates on Two Smart-Cards", http://csrc.nist.gov/encryption/aes/, 1999
11. J. Daemen, V. Rijmen, "*The Block Cipher Rijndael*," in LNCS 1820, *Smart-Card Research and Applications*, pp. 288–296, J. Quisquater and B. Schneier, Eds., Springer-Verlag, 2000.
12. J. Daemen, V. Rijmen, "*Rijndael, the Advanced Encryption Standard*," Dr. Dobb's Journal, Vol. 26, No. 3, March 2001, pp. 137–139
13. M. Akkar, C. Giraud, "An Implementation of DES and AES, Secure against some Attacks," *Proceedings of CHES '01*, pp. 315–325, 2001.
14. M. McLoone, J. McCanny, "High Performance single-Chip FPGA Rijndael Algorithm Implementations," *Proceedings of CHES '01*, pp. 68–80, 2001.
15. V. Fischer, M. Drutarovsky, "Two Methods of Rijndael Implementation in Reconfigurable Hardware," *Proceedings of CHES '01*, pp. 81–96, 2001.
16. H. Kuo, I. Verbauwhede, "Architectural Optimization for a 1.82Gbits/sec VLSI Implementation of the AES Rijndael Algorithm," *Proceedings of CHES '01*, pp. 53–67, 2001.
17. A. Rudra, P.K. Dubey, C.S. Jutla, V. Kumar, J.R. Rao, P. Rohatgi, "Efficient Rijndael Encryption Implementation with Composite Field Arithmetic," *Proceedings of CHES '01*, pp. 175–188, 2001.
18. A. Dandalis, V.K. Prasanna, J.P.D. Rolim, "An adaptive cryptographic Engine for IPSec Architecutures" *Field-Programmable Custom Computing Machines, 2000 IEEE Symposium on*, pp. 132–141, 2000.

An Optimized S-Box Circuit Architecture for Low Power AES Design

Sumio Morioka and Akashi Satoh

IBM Research, Tokyo Research Laboratory, IBM Japan Ltd.,
1623-14 Shimotsuruma, Yamato-shi, Kanagawa-ken 242-8502, Japan
{e02716,akashi}@jp.ibm.com

Abstract. Reducing the power consumption of AES circuits is a critical problem when the circuits are used in low power embedded systems. We found the S-Boxes consume much of the total AES circuit power and the power for an S-Box is mostly determined by the number of dynamic hazards. In this paper, we propose a low-power S-Box circuit architecture: a multi-stage PPRM architecture over composite fields. In this S-Box, (i) the signal arrival times of gates are as close as possible if the depths of the gates from the primary inputs are the same, and (ii) the hazard-transparent XOR gates are located after the other gates that may block the hazards. A low power consumption of 29 μW at 10 MHz using 0.13 μm 1.5V CMOS technology was achieved, while the consumptions of the BDD, SOP, and composite field S-Boxes are 275, 95, and 136 μW, respectively.

1 Introduction

DES (Data Encryption Standard) has been used as a de facto standard cipher for more than 20 years. In 2001, NIST (National Institute of Standards and Technology) made Rijndael the new standard cipher AES [1,2].

Reducing the power consumption of AES (Advanced Encryption Standard) circuits is a critical problem when the circuits are used in embedded systems. Many circuit architectures for AES have been proposed recently and their performances have been evaluated by using ASIC libraries [3,4] and FPGAs [5,6]. However, most of them are simple implementations according to the AES specification, and there is no report of low-power AES implementations, as far as the authors know.

In this paper, we investigated a design methodology for a low-power AES. Although many power reduction techniques for IP cores are known for various design abstraction levels from the algorithm level to the transistor level [7], we focused on the logic level (gate level), because the power optimization techniques at this level can be applied in many applications. The optimizations at the other levels cannot be adopted as often, because the AES circuit is usually used as an IP core in a system, and changing the algorithm, operator scheduling, data-path architecture, and/or arrangements of transistors of the AES is difficult under given requirements for throughput, clock speed, and technology library.

By investigating the power consumption of each primitive component in AES circuits, we found the S-Box in the SubBytes component consumes much of the total

B.S. Kaliski Jr. et al. (Eds.): CHES 2002, LNCS 2523, pp. 172–186, 2003.

power (for instance, 75% is consumed in a 1 round/cycle loop architecture). The power consumption of the clock drivers is not large. When the circuit structure of the S-Box is changed, the power of the S-Box circuits can vary more than several-fold, due to the changes in the situations creating and propagating dynamic hazards, even though the total circuit size has less effect on the power consumption than expected. In fact, the power consumption of SOP (Sum of Products) S-Box is less than that of a composite field S-Box [8,9,10], while this circuit size is much larger.

We have developed a low-power S-Box architecture: a multi-stage PPRM (Positive Polarity Reed-Muller form [11]) architecture for compact S-Boxes. It is an improvement of the composite field S-Box, and in this S-Box, the gates are arranged so that: (i) the signal arrival times at the gates are as close as possible if the depths of the gates from the primary inputs are the same, to avoid generating dynamic hazards, and (ii) the hazard-transparent XOR gates are located after the other gates that may block the hazards, to avoid the propagation of dynamic hazards. The multi-stage PPRM S-Box archives the lowest power consumption of 29 μW at 10 MHz using 0.13 μm 1.5 V CMOS technology, and its circuit size is still much smaller than conventional S-Box implementations whose power consumptions are around 140 μW.

This paper is organized as follows. In Section 2, a standard AES circuit implementation is shown. In Section 3, the results of power analysis of the AES are described. In Section 4, the proposed S-Box architecture and its ASIC implementation results are explained.

2 AES Algorithm and Its Circuit Implementations

2.1 AES Algorithm

In the AES algorithm [1,2], a sequence of four primitive functions, SubBytes, ShiftRows, MixColumns, and AddRoundKey, are executed Nr-1 times. Each loop is called a round and the concrete value of Nr is 10, 12, or 14 depending on the key length. Prior to this main loop, AddRoundKey is executed for initialization. After executing the main loop, SubBytes, ShiftRows, and AddRoundKey is executed are executed once more as the final round. In the decryption process, the inverse operations of each primitive function are executed.

SubBytes is a one-byte input/output nonlinear transformation that uses 16-byte (128-bit) S-Boxes. Each S-Box is a multiplicative inversion on a Galois field GF(2^8) followed by an affine transformation. The irreducible polynomial used by the field is $m(x)=x^8+x^4+x^3+x+1$. ShiftRows is a cyclic shift operation in each row of four 4-byte data blocks using 0~3-byte offsets. MixColumns treats the 4-byte data blocks in each column as coefficients of a 4-term polynomial, and multiplies the data modulo x^4+1 with a fixed polynomial. AddRoundKey is a simple bit-wise XOR operation on the 128-bit round keys and the data.

In Fig. 1, a standard circuit implementation of AES, which executes 1 round per clock, is shown. A sequence of round operations is implemented as a combinational circuit and its input and output are connected to a 128-bit data register. The 3:1 selector before the AddRoundKey (XOR) is used to skip some operations in the first and last rounds.

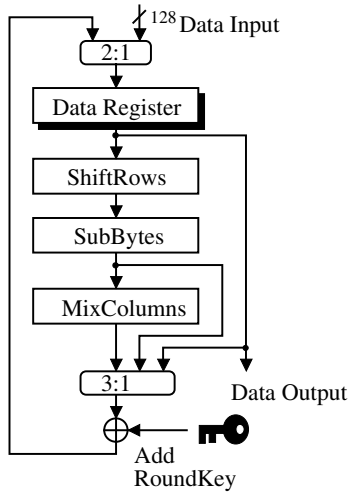


Fig. 1. A standard implementation of an AES encryption circuit (1 round/clock)

2.2 Various S-Box Circuit Implementations

There are two approaches for designing S-Box circuits:

(1) construct a multiplicative inversion circuit and an affine transformation circuit independently, and then connect these two circuits in serial,
(2) construct a single circuit directly whose input-output relation is equivalent to the S-Box.

In Method (1), circuit area reductions using mathematical theorems over Galois fields (GF) [12] are possible. Various methods for constructing compact inversion circuits over GF have been studied, based on Fermat's Little Theorem [14], the Low-latency algorithm of Itoh and Tsujii [13,14], the extended Euclid's Algorithm, and so on. In particular, the composite field (or tower field) inversion [8] is effective over $GF(2^8)$, and it can be used to create compact AES implementations [9,10]. The detail of the composite field technique will be explained in the next sub-section.

In Method (2), a fast implementation is possible. The S-Box circuit can be obtained from its truth table by using two-level logic such as SOP, POS (Product of Sums), PPRM [11], or by using decision diagrams such as BDD (Binary Decision Diagram) [15]. Our investigations' results in [16,17] show that the variable ordering of the BDD does not have much effect on the size and speed of the S-Box and GF inverter. Based on this research, we designed a very fast S-Box called twisted-BDD [17], which is 1.5 to 2 times faster than the other S-Box implementations.

Although in many AES implementations the table-lookup method is used, where the S-Box circuit is automatically synthesized using EDA tools, the performance of these synthesized circuits is usually close to that of SOP or BDD implementations.

2.3 A S-Box Implementation Based on Composite Field Technique

Fig. 2 shows the outline of an S-Box implementation using the composite field technique [9,10]. The most costly operation in the S-Box is the multiplicative inversion over a field A (the AES field), where A is extended from GF(2) with the irreducible polynomial $m(x)$ mentioned in Section 2.1. To reduce the cost of this operation, the following 3-stage method is adopted:

(Stage 1) Map all elements of the field A to a composite field B, using an isomorphism function δ

(Stage 2) Compute the multiplicative inverses over the field B.

(Stage 3) Re-map the computation results to A, using the function δ^{-1}.

The composite field B in Stage 2 is constructed not by applying a single degree-8 extension to GF(2), but by applying multiple extensions of smaller degrees. To reduce the cost of Stage 2 as much as possible, it is known to be efficient to construct the composite field B using repeated degree-2 extensions under a polynomial basis using these irreducible polynomials [10],

$$\begin{cases} \mathrm{GF}(2^2): & x^2+x+1 \\ \mathrm{GF}((2^2)^2): & x^2+x+\phi \\ \mathrm{GF}(((2^2)^2)^2): & x^2+x+\lambda \end{cases}$$

where $\phi=\{10\}_2$, $\lambda=\{1100\}_2$. The inverter over the field above has fewer GF(2) operators compared with the composite field used in [9],

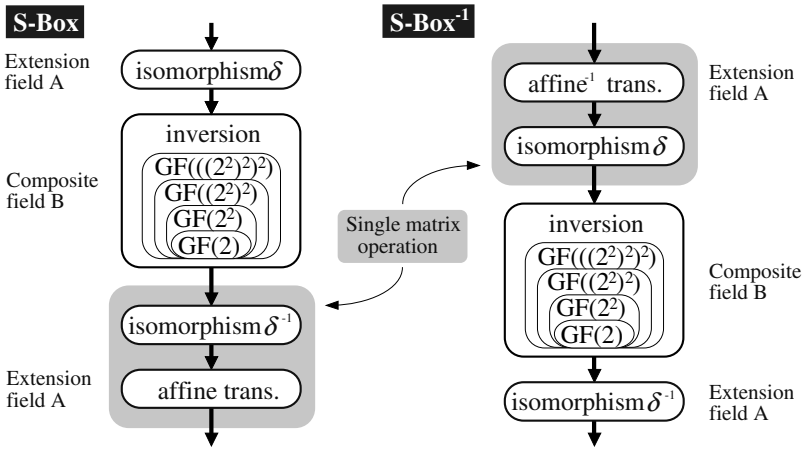


Fig. 2. Computation sequences of composite-field-based S-Box

$$\begin{cases} \mathrm{GF}(2^4): & x^4 + x + 1 \\ \mathrm{GF}((2^4)^2): & x^2 + x + \omega_{14} \end{cases}$$

where $\omega_{14}=\{1001\}_2$.

An implementation of Stage 2 is shown in Fig. 3. For any composite fields $\mathrm{GF}((2^m)^n)$, computing the multiplicative inverses can be done as a combination of operations over the sub-fields $\mathrm{GF}(2^n)$, using the following equation [8]:

$$P^{-1} = (P^r)^{-1} \cdot P^{r-1}, \quad \text{where} \quad r = (2^{nm}-1)/(2^m-1). \tag{1}$$

For AES ($n = 2$, $m = 2^2$), this equation becomes

$$P^{-1} = (P^{17})^{-1} \cdot P^{16}. \tag{2}$$

The circuit in Fig. 3 is an implementation of Equation (2), with additional optimizations. In the circuit, P^{16} is computed first and then P^{17} is obtained by multiplying P by P^{16} over $\mathrm{GF}(((2^2)^2)^2)$. Because P^{17} is always an element of $\mathrm{GF}((2^2)^2)$ (i.e., the upper 4 bits of P^{17} are always 0), computing the upper 4 bits of P^{17} is unnecessary [8]. The value of $(P^{17})^{-1}$ is computed recursively over $\mathrm{GF}((2^2)^2)$, then multiplied by P^{16} over $\mathrm{GF}(((2^2)^2)^2)$, and finally P^{-1} is obtained. This final multiplication requires fewer circuit resources than conventional multiplication over $\mathrm{GF}(2^8)$, because P^{17} is an element of $\mathrm{GF}((2^2)^2)$. Further gate reduction is possible by sharing circuit gates of the three $\mathrm{GF}((2^2)^2)$ multipliers in Fig. 3, where common inputs are used. Note that our multipliers and inverter over subfields $\mathrm{GF}((2^2)^2)$ and $\mathrm{GF}(2^2)$ are also small.

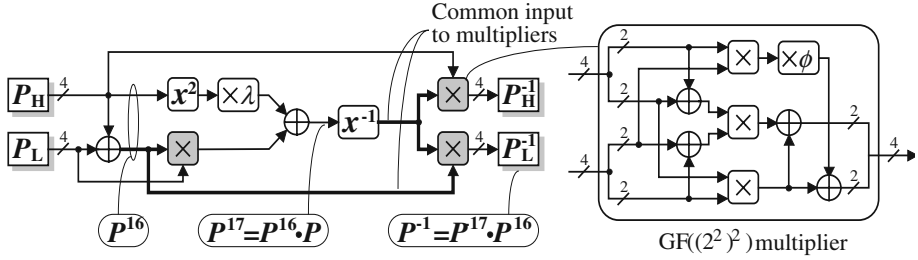


Fig. 3. Computation sequences of composite-field-based inverter

The isomorphism functions δ and δ^{-1} in Stages 1 and 3 were constructed as follows. First, search for a primitive element α in the field A and a primitive element β in the field B, where both α and β are roots of a same primitive irreducible polynomial. Any primitive irreducible polynomial can be used, and here we used $p(x)=x^8+x^4+x^3+x^2+1$. Once such elements α and β are found, the definition table of the isomorphism functions δ and δ^{-1} are immediately determined, where α^k is mapped to β^k (or β^k to α^k) for any $1 \le k \le 254$. The hardware implementation of these functions can be obtained by mapping the basis elements of A (or B) into B (or A), and these mappings are described as multiplications of constant matrices over GF(2). The concrete descriptions of δ and δ^{-1} are

$$\delta = \begin{bmatrix} 1 & 1 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 & 1 & 0 & 1 & 0 \\ 0 & 1 & 1 & 1 & 1 & 0 & 0 & 1 \\ 0 & 1 & 1 & 0 & 0 & 0 & 1 & 1 \\ 0 & 1 & 1 & 1 & 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 1 & 0 & 1 & 0 & 1 \\ 0 & 1 & 1 & 1 & 1 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 \end{bmatrix} \quad \delta^{-1} = \begin{bmatrix} 1 & 0 & 1 & 0 & 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 \\ 0 & 1 & 1 & 1 & 1 & 0 & 0 & 1 \\ 0 & 1 & 1 & 1 & 1 & 1 & 0 & 0 \\ 0 & 1 & 1 & 0 & 1 & 1 & 1 & 0 \\ 0 & 1 & 0 & 0 & 0 & 1 & 1 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 & 0 & 1 & 1 & 1 \end{bmatrix},$$

where the least significant bits are in the upper left corners. These matrices can be merged with affine transformations for circuit size reduction, as shown in Fig. 2. All of these isomorphism functions and the constant multipliers in the S-Boxes are implemented as XOR arrays, and their Boolean logic is compressed by applying a factoring technique based on a greedy algorithm [18].

3 Analysis of Power Consumption in AES Circuits

3.1 Power Analysis Method

For analyzing the power consumption of the AES circuits, a simulation-based analysis method was used. In this method, a timing simulation (or delay simulation) at the gate level is performed using a given set of test input data, and the switching activities of all internal gates are logged. Then the circuit power is computed from this simulation log and the cell information of the target ASIC library. Although this analysis method requires much more CPU time than the often used probabilistic method (a kind of static analysis method) [7], it is much more accurate because the effects of dynamic hazards can be reflected in the power estimation results.

3.2 Power Consumption of Various S-Box Architectures

The power estimation results for each AES component in a 0.13 μm ASIC standard cell library are shown in Table 1. Note that the estimation results depends on the test data, although we observed no large differences between the various test data sets we used. In this table, the SOP S-Box is used. Clearly, the SubBytes operations (16 S-Boxes) consume much more power than the other components.

In Tables 2 and 3, a performance comparison of various S-Boxes in 0.13μm and 0.18μm ASIC libraries, including the proposed method (which will be described in Section 4), is shown. We checked all patterns of primary input switching ($2^8 \times 2^8$ = 65,536 patterns) and the average power consumptions obtained are shown in the tables. Although the absolute values of circuit performance are different between the two ASIC libraries, the relative relationships within each architecture are almost the same.

Until these experimental results were obtained, we thought the power consumption of an AES with the composite field S-Box would be the lowest, because the circuit size of the composite field S-Box is very small. However, the estimated power consumption was unexpectedly large. In addition, the power consumption of the BDD S-Box was much larger than that of the SOP, although their speed and size are similar.

Table 1. Power consumption of each AES component
(0.13 μm 1.5 V CMOS standard cell, 1 gate = 2 way-NAND)

	Observed Max. Power (μW@10MHz)	Power ratio (%)
SubBytes (SOP S-Box $\times$16)	1,940	75
MixColumns	262	10
AddRoundKey	> 10	> 1
Data Selectors	> 10	> 1
FFs + Clock Drivers	400	15

Table 2. Comparison of various S-Box architectures
(0.13 μm 1.5 V CMOS standard cell, 1 gate = 2 way-NAND)

	Delay (ns)	Size (gate)	Average Power of S-Box (μW@10 MHz)
Itoh and Tsujii [14]	2.79	1,771	2,100
PPRM (1-stage)	1.14	2,241	343
BDD	0.69	1,399	275
Twisted-BDD [17]	0.43	2,818	272
Table lookup	0.68	2,623	144
Composite Field [10]	2.19	354	136
SOP (1-stage)	0.69	1,650	95
Proposed method (3-stage PPRM)	1.43	712	29

Table 3. Comparison of various S-Box architectures
(0.18 μm 1.8 V CMOS standard cell, 1 gate = 2 way-NAND)

	Delay (ns)	Size (gate)	Average Power of S-Box (μW@10 MHz)
Itoh and Tsujii [14]	4.11	1,540	3,490
PPRM (1-stage)	1.32	2,242	408
Twisted-BDD [17]	0.66	1,977	334
BDD	0.96	857	332
Table look-up	0.91	1,706	206
Composite Field [10]	3.01	305	166
SOP (1-stage)	0.97	1,142	138
Proposed method (3-stage PPRM)	1.86	701	51

3.3 Analysis

We determined that the power consumption of the S-Boxes was strongly influenced by the number of dynamic hazards. In fact, the following two characteristics A and B, which are the main reasons for creating or propagating dynamic hazards, are significantly different among the S-Boxes.

A. Differences of Signal Arrival Time at Each Gate

The dynamic hazards can occur when the signal arrival times are different between multiple inputs of a gate. If multiple gates are connected serially and some of the internal gates generate hazards, then the hazards propagate into the circuit path and some extra power is consumed.

Regarding S-Boxes, the composite field S-Box involves many crossing and/or branched signal paths. The signal arrival times of the internal gates are very different and as a result, the gates (or sub-operators) can switch many times per single transition of the primary inputs, as shown in Fig. 4. This is the main reason for its relatively large power consumption in spite of the small circuit size. The situation is the same with the BDD S-Box and S-Box based on the algorithm of Itoh and Tsujii. On the other hand, the signal arrival time of each gate is not much different in the S-Boxes based on two-level logic, such as the SOP and PPRM S-Box.

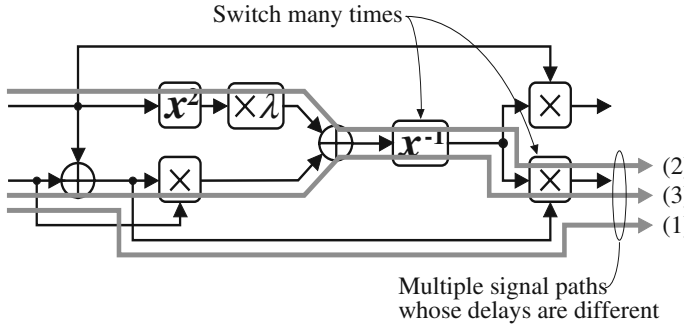


Fig. 4. Dynamic hazards caused by differences of signal arrival time

B. Propagation Probability of Signal Transitions

The kinds of the logic gates closely relates to the propagation of hazards. In particular, the use of XOR gates can increase the power consumption, because their probability of propagating a signal transition is 1, i.e. all hazards can be propagated from input ports to output port, while the probabilities are 0.5 in the other gates such as AND and OR. The use of many XORs is another main reason for the large power consumptions of the composite field S-Box and PPRM S-Box, compared to the SOP S-Box.

4 The Proposed Low Power S-Box Architecture and Its Evaluation Results

4.1 The Proposed Multi-stage PPRM Architecture

The complicated signal paths of the composite field S-Boxes, which is the main reason for their large power consumption, can be simplified by converting some parts of the S-Box logic into two-level logic. Although converting the entire circuit into a single two-level logic structure would increase the circuit size and lose an important advantage of the composite field S-Box, converting carefully selected sub-components into two-level logic can generate a much better S-Box, if an appropriate partitioning of the S-Box into sub-components is done.

In Fig. 5, the example S-Box circuit with the best performance in our trials is shown. Three sub-components of the composite field S-Box were converted into PPRM form: the pre-inversion section, the inversion section, and the post-inversion section (see Appendix for their complete description). Two signal paths not passing through the inversion section were connected to delay chains whose delay time is close to that of the inversion. In this circuit, the signal arrival times at the gates are almost equal, if the depths of the gates from primary inputs are the same. In addition, the power consumption of the pre-inversion part can be greatly reduced, because the XOR gates are located after the AND gates. The original pre-inversion part has the

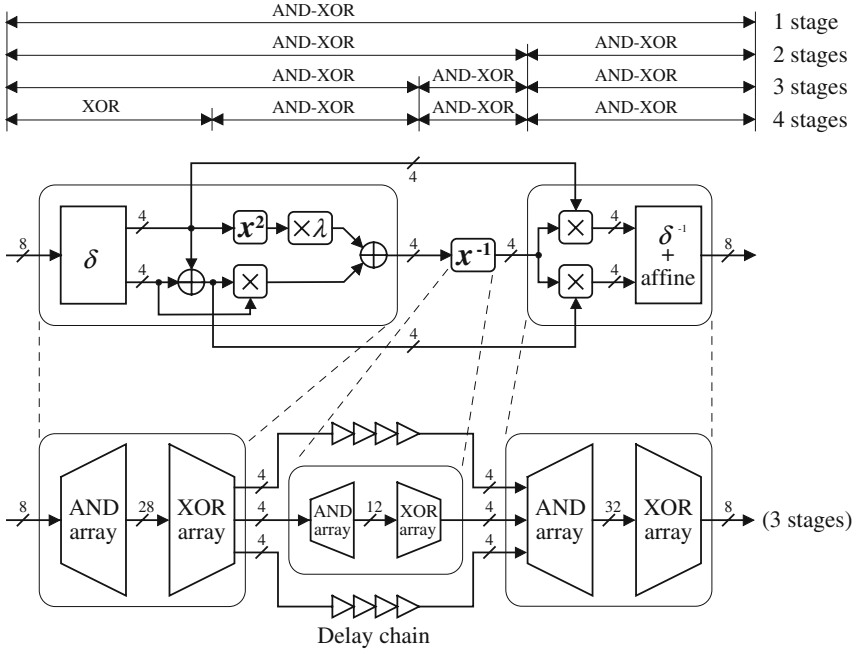


Fig. 5. A conventional composite field S-Box (top) and the proposed method called multi-stage PPRM (bottom)

opposite XOR-AND structure, and the XOR gates must always switch state on any change of the primary inputs.

As shown in Tables 2 and 3 in Section 3.2, the 3-stage PPRM S-Box achieves the lowest power consumption. Its circuit size is much smaller than conventional S-Box implementations. The circuit speed is not so fast, but it is still fast enough for embedded systems.

When other partitioning methods such as a 2-stage PPRM, a 4-stage (XOR)-AND-XOR etc. were used (Fig. 5), the power consumption was increased, as shown in Table 4. The original composite field S-Box has a 5-stage (XOR)-AND-XOR structure. Of those the authors tested, the 3-stage PPRM gives the lowest power consumption. Another 2-stage PPRM implementation besides the one shown in Fig. 5 (combined inversion and post-inversion sections) is possible, but its circuit size and power consumption are similar to those of the 1-stage PPRM implementation. In addition, when each section was implemented in a form other than PPRM (such as SOP), the circuit size and power consumption became significantly larger (see Table 4), because these sections contain many XOR gates.

Please note that computing over a composite field is highly recommended, although the proposed circuit architecture only requires the use of Equation (2) in Section 2.3 and the equation is independent of the field (i.e., the equation even holds if a composite field is not used). If the inversion is not done over a composite field, P^{17} in Equation (2) becomes an 8-bit vector and as a result, the circuit size must be much larger.

Table 4. Number of logic stages vs. S-Box performance
(0.13 μm 1.5 V CMOS standard cell, 1 gate = 2 way-NAND)

		Size (gate)	Average Power of S-Box (μW@10 MHz)
PPRM	1-stage	2,241	343
	2-stages	1,445	273
	3-stages	712	29
	4-stages	413	88
	5-stages (Composite Field [10])	354	136
SOP	1-stage	1,650	95
	2-stages	5,891	612
	3-stages	6,114	697

4.2 Circuit Sharing between S-Box and S-Box^{-1}

The proposed multi-stage PPRM architecture is suitable for implementing AES circuits that perform both encryption and decryption, because most gates of the S-Box (for encryption) and the S-Box^{-1} (for decryption) can be shared.

An example shared S-Box circuit is shown in Fig. 6. In this circuit, an inverter over the AES field is shared between the S-Box and the S-Box^{-1}. The power consumption of the shared S-Box is larger than that of the unshared 3-stage PPRM S-Box, but it is still smaller than most S-Box implementations (see Table 5). Similar circuit sharing is possible for the composite field S-Box, but the power consumption is very large. Note

that SOP and the other conventional S-Box implementations require two different circuits to support both encryption and decryption, and the total circuit size (S-Box + S-Box^{-1}) can exceed 2,000 gates.

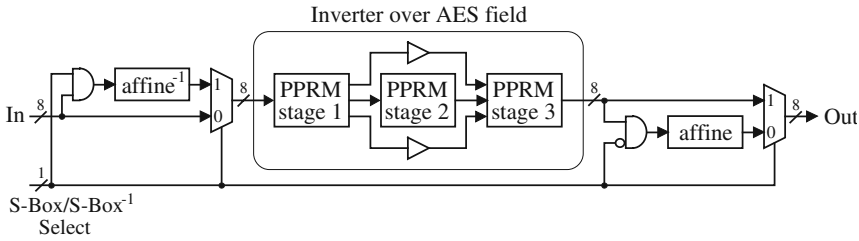


Fig. 6. Circuit sharing between S-Box and S-Box^{-1} under the multi-stage PPRM architecture

Table 5. Performance comparison of shared S-Box architectures (0.13 μm 1.5 V CMOS standard cell, 1 gate = 2 way-NAND)

	Delay (ns)	Size (gate)	Average Power (μW@10 MHz)
Composite Field [10]	2.53	381	179 (S-Box)
			189 (Inv S-Box)
Proposed Method (3-stage PPRM)	2.00	725	79 (S-Box)
			70 (Inv S-Box)

5 Conclusion

In this paper, we have developed a multi-stage PPRM architecture for low-power S-Box circuits, because the S-Boxes consume much of the total power of AES designs. The power consumption of S-Box circuits can be significantly reduced by avoiding the creation and propagation of dynamic hazards. A power consumption of only 29 μW at 10 MHz using a 0.13 μm 1.5V CMOS technology was achieved, while the power of the conventional BDD, SOP and composite field S-Boxes are 275, 95 and 136 μW, respectively.

References

1. J. Daemen and V. Rijmen, “AES Proposal: Rijndael,” http://csrc.nist.gov/ encryption/aes/rijndael/Rijndael.pdf.
2. National Institute of Standards and Technology (NIST), “Advanced Encryption Standard (AES)”, FIPS Publication 197, http://csrc.nist.gov/encryption/ aes/index.html, Nov. 2001.
3. H. Kuo et al., "Architectural Optimization for a 1.82 Gbits/sec VLSI Implementation of the AES Rijndael Algorithm," *Proc. CHES2001*, LNCS Vol. 2162, pp. 53–67, 2001.

4. B. Weeks et al., "Hardware Performance Simulation of Round 2 Advanced Encryption Standard Algorithm," http://csrc.nist.gov/encryption/aes/round2/ NSA-AESfinalreport.pdf.
5. M. McLoone et al., "High performance single-chip FPGA Rijndael algorithm implementations," *Proc. CHES2001*, LNCS Vol. 2162, pp. 68–80, 2001.
6. V. Fischer et al, "Two methods of Rijndael implementation in reconfigurable hardware," *Proc. CHES2001*, LNCS Vol. 2162, pp. 81–96, 2001.
7. A.P. Chandrakasan and R.W. Brodersen (eds.), *Low Power Digital CMOS Design*, Kluwer Academic Publishers, 1995.
8. J. Guajardo and C. Paar, "Efficient Algorithms for Elliptic Curve Cryptosystems," *CRYPTO'97*, LNCS Vol. 1294, pp. 342–356, 1997.
9. A. Rudra et al, "Efficient Rijndael encryption implementation with composite field arithmetic," *Proc. CHES2001*, LNCS Vol. 2162, pp. 175–188, 2001.
10. A. Satoh, S. Morioka, K. Takano, and S. Munetoh, "A Compact Rijndael Hardware Architecture with S-Box Optimization," Advances in Cryptology – *ASIACRYPT 2001*, LNCS Vol. 2248, pp. 239–254, 2001.
11. T. Sasao, "AND-EXOR expressions and their optimization", in Sasao, editor: *Logic Synthesis and Optimization*, Kluwer Academic Publishers, pp. 287–312, 1993.
12. I.F. Blake, X. Gao, R.C. Mullin, S.A. Vanstone and T. Yaghoobian, *Applications of Finite Fields*, Kluwer Academic Publishers. 1993.
13. T. Itoh and S. Tsujii, "A Fast Algorithm for Computing Multiplicative Inverses in GF(2^m) using Normal Bases," *Information and Computation*, Vol.78, No. 3, pp. 171–177, 1988.
14. S. Morioka and Y. Katayama, "O($\log_2 m$) Iterative Algorithm for Multiplicative Inverse in GF(2^m)," *IEEE Intl. Symp. On Info. Theory (ISIT2000)*, pp. 449 ff., 2000.
15. R.E. Bryant, "Graph-Based Algorithms for Boolean Function Manipulation," *IEEE Trans. on Computers*, Vol. C-35, No. 8, pp. 677–691, 1986.
16. S. Morioka, Y. Katayama, and T. Yamane, "Towards Efficient Verification of Arithmetic Algorithms over Galois Fields GF(2^m)," *13th Conference on Computer Aided Verification (CAV'01)*, LNCS Vol. 2102, pp. 465–477, 2001.
17. S. Morioka and A. Satoh, "A 10 Gbps Full-AES Crypto Design with a Twisted-BDD S-Box Architecture," *2002 IEEE Intl. Conf. on Computer Design (ICCD2002)*, 2002.
18. S. Morioka and Y. Katayama, "Design Methodology for one-shot Reed-Solomon Encoder and Decoder," *1999 IEEE Intl. Conf. on Computer Design (ICCD'99)*, pp. 60–67, 1999.

Appendix: PPRM Representations of Each Stage (3-Stage PPRM)

Variables x_7-x_0 denote primary inputs of an S-Box/S-Box^{-1}/Inverter and y_7-y_0 denote primary outputs (x_7 and y_7 are MSB). The other variables such as a and b denote internal wires.

A-1. Stage 1 of S-Box

$a_3 = x_7 \text{ xor } x_5$
$a_2 = x_7 \text{ xor } x_6 \text{ xor } x_4 \text{ xor } x_3 \text{ xor } x_2 \text{ xor } x_1$
$a_1 = x_7 \text{ xor } x_5 \text{ xor } x_3 \text{ xor } x_2$
$a_0 = x_7 \text{ xor } x_5 \text{ xor } x_3 \text{ xor } x_2 \text{ xor } x_1$
$b_3 = x_5 \text{ xor } x_6 \text{ xor } x_2 \text{ xor } x_1$
$b_2 = x_6$
$b_1 = x_7 \text{ xor } x_5 \text{ xor } x_3 \text{ xor } x_2 \text{ xor } x_6 \text{ xor } x_4 \text{ xor } x_1$
$b_0 = x_7 \text{ xor } x_5 \text{ xor } x_3 \text{ xor } x_2 \text{ xor } x_6 \text{ xor } x_0$

$c_3 = (x_5 \text{ and } x_1) \text{ xor } (x_7 \text{ and } x_1) \text{ xor } (x_5 \text{ and } x_2) \text{ xor } (x_5 \text{ and } x_6) \text{ xor } (x_5 \text{ and } x_7) \text{ xor } (x_5 \text{ and } x_4) \text{ xor } (x_7 \text{ and } x_4) \text{ xor } (x_5 \text{ and } x_0) \text{ xor } (x_7 \text{ and } x_0) \text{ xor } (x_3 \text{ and } x_1) \text{ xor } (x_4 \text{ and } x_1) \text{ xor } (x_3 \text{ and } x_2) \text{ xor } (x_2 \text{ and } x_4) \text{ xor } (x_4 \text{ and } x_6) \text{ xor } (x_2 \text{ and } x_1) \text{ xor } (x_2 \text{ and } x_6) \text{ xor } (x_6 \text{ and } x_1)$

$c_2 = (x_6 \text{ and } x_1) \text{ xor } (x_2 \text{ and } x_6) \text{ xor } (x_3 \text{ and } x_6) \text{ xor } (x_7 \text{ and } x_6) \text{ xor } (x_1 \text{ and } x_0) \text{ xor } (x_2 \text{ and } x_0) \text{ xor } (x_3 \text{ and } x_0) \text{ xor } (x_4 \text{ and } x_0) \text{ xor } (x_6 \text{ and } x_0) \text{ xor } (x_7 \text{ and } x_0) \text{ xor } (x_5 \text{ and } x_2) \text{ xor } (x_5 \text{ and } x_3) \text{ xor } (x_2 \text{ and } x_4) \text{ xor } (x_3 \text{ and } x_4) \text{ xor } (x_5 \text{ and } x_7) \text{ xor } (x_7 \text{ and } x_2) \text{ xor } (x_5 \text{ and } x_6) \text{ xor } (x_3 \text{ and } x_2) \text{ xor } (x_7 \text{ and } x_3)$

$c_1 = (x_2 \text{ and } x_1) \text{ xor } (x_2 \text{ and } x_4) \text{ xor } (x_5 \text{ and } x_4) \text{ xor } (x_3 \text{ and } x_6) \text{ xor } (x_5 \text{ and } x_6) \text{ xor } (x_2 \text{ and } x_0) \text{ xor } (x_3 \text{ and } x_0) \text{ xor } (x_5 \text{ and } x_0) \text{ xor } (x_7 \text{ and } x_0) \text{ xor } x_1 \text{ xor } (x_5 \text{ and } x_2) \text{ xor } (x_7 \text{ and } x_2) \text{ xor } (x_5 \text{ and } x_3) \text{ xor } (x_5 \text{ and } x_7) \text{ xor } x_7 \text{ xor } x_2 \text{ xor } (x_3 \text{ and } x_2) \text{ xor } x_4 \text{ xor } x_5$

$c_0 = (x_1 \text{ and } x_0) \text{ xor } (x_2 \text{ and } x_0) \text{ xor } (x_3 \text{ and } x_0) \text{ xor } (x_5 \text{ and } x_0) \text{ xor } (x_7 \text{ and } x_0) \text{ xor } (x_3 \text{ and } x_1) \text{ xor } (x_6 \text{ and } x_1) \text{ xor } (x_3 \text{ and } x_6) \text{ xor } (x_5 \text{ and } x_6) \text{ xor } (x_7 \text{ and } x_6) \text{ xor } (x_3 \text{ and } x_4) \text{ xor } (x_7 \text{ and } x_4) \text{ xor } (x_5 \text{ and } x_3) \text{ xor } (x_4 \text{ and } x_1) \text{ xor } x_2 \text{ xor } (x_3 \text{ and } x_2) \text{ xor } (x_4 \text{ and } x_6) \text{ xor } x_6 \text{ xor } x_5 \text{ xor } x_3 \text{ xor } x_0$

A-2. Stage 2 of S-Box

$d_3 = (c_3 \text{ and } c_2 \text{ and } c_1) \text{ xor } (c_3 \text{ and } c_0) \text{ xor } c_3 \text{ xor } c_2$

$d_2 = (c_3 \text{ and } c_2 \text{ and } c_0) \text{ xor } (c_3 \text{ and } c_0) \text{ xor } (c_3 \text{ and } c_2 \text{ and } c_1) \text{ xor } (c_2 \text{ and } c_1) \text{ xor } c_2$

$d_1 = (c_3 \text{ and } c_2 \text{ and } c_1) \text{ xor } (c_3 \text{ and } c_1 \text{ and } c_0) \text{ xor } c_3 \text{ xor } (c_2 \text{ and } c_0) \text{ xor } c_2 \text{ xor } c_1$

$d_0 = (c_3 \text{ and } c_2 \text{ and } c_0) \text{ xor } (c_3 \text{ and } c_1 \text{ and } c_0) \text{ xor } (c_3 \text{ and } c_2 \text{ and } c_1) \text{ xor } (c_3 \text{ and } c_1) \text{ xor } (c_3 \text{ and } c_0) \text{ xor } (c_2 \text{ and } c_1 \text{ and } c_0) \text{ xor } c_2 \text{ xor } (c_2 \text{ and } c_1) \text{ xor } c_1 \text{ xor } c_0$

A-3. Stage 3 of S-Box

$y_7 = (d_3 \text{ and } a_0) \text{ xor } (d_2 \text{ and } a_1) \text{ xor } (d_1 \text{ and } a_2) \text{ xor } (d_0 \text{ and } a_3) \text{ xor } (b_2 \text{ and } d_3) \text{ xor } (b_3 \text{ and } d_2) \text{ xor } (b_2 \text{ and } d_2) \text{ xor } (d_3 \text{ and } a_3) \text{ xor } (d_3 \text{ and } a_1) \text{ xor } (d_1 \text{ and } a_3) \text{ xor } (b_0 \text{ and } d_2) \text{ xor } (b_2 \text{ and } d_0) \text{ xor } (d_3 \text{ and } a_2) \text{ xor } (d_2 \text{ and } a_3) \text{ xor } (b_0 \text{ and } d_3) \text{ xor } (b_1 \text{ and } d_2) \text{ xor } (b_2 \text{ and } d_1) \text{ xor } (b_3 \text{ and } d_0)$

$y_6 = \text{'1' xor } (a_0 \text{ and } d_2) \text{ xor } (a_2 \text{ and } d_0) \text{ xor } (d_3 \text{ and } a_3) \text{ xor } (a_0 \text{ and } d_1) \text{ xor } (a_1 \text{ and } d_0) \text{ xor } (d_3 \text{ and } a_2) \text{ xor } (d_2 \text{ and } a_3) \text{ xor } (a_0 \text{ and } d_0) \text{ xor } (d_3 \text{ and } a_0) \text{ xor } (d_2 \text{ and } a_1) \text{ xor } (d_1 \text{ and } a_2) \text{ xor } (d_0 \text{ and } a_3)$

$y_5 = \text{'1' xor } (d_3 \text{ and } a_3) \text{ xor } (d_3 \text{ and } a_1) \text{ xor } (d_1 \text{ and } a_3) \text{ xor } (d_3 \text{ and } a_2) \text{ xor } (d_2 \text{ and } a_3) \text{ xor } (b_2 \text{ and } d_2) \text{ xor } (b_0 \text{ and } d_2) \text{ xor } (b_2 \text{ and } d_0) \text{ xor } (b_3 \text{ and } d_3) \text{ xor } (b_1 \text{ and } d_3) \text{ xor } (b_3 \text{ and } d_1) \text{ xor } (d_3 \text{ and } a_0) \text{ xor } (d_2 \text{ and } a_1) \text{ xor } (d_1 \text{ and } a_2) \text{ xor } (d_0 \text{ and } a_3)$

$y_4 = (d_3 \text{ and } a_1) \text{ xor } (d_1 \text{ and } a_3) \text{ xor } (a_0 \text{ and } d_0) \text{ xor } (b_3 \text{ and } d_3) \text{ xor } (b_0 \text{ and } d_1) \text{ xor } (b_1 \text{ and } d_0) \text{ xor } (d_3 \text{ and } a_0) \text{ xor } (d_2 \text{ and } a_1) \text{ xor } (d_1 \text{ and } a_2) \text{ xor } (d_0 \text{ and } a_3) \text{ xor } (a_1 \text{ and } d_1) \text{ xor } (b_2 \text{ and } d_2) \text{ xor } (b_0 \text{ and } d_0)$

$y_3 = (b_0 \text{ and } d_1) \text{ xor } (b_1 \text{ and } d_0) \text{ xor } (b_0 \text{ and } d_2) \text{ xor } (b_2 \text{ and } d_0) \text{ xor } (b_1 \text{ and } d_3) \text{ xor } (b_3 \text{ and } d_1) \text{ xor } (b_0 \text{ and } d_0)$

$y_2 = (a_0 \text{ and } d_2) \text{ xor } (a_2 \text{ and } d_0) \text{ xor } (a_0 \text{ and } d_1) \text{ xor } (a_1 \text{ and } d_0) \text{ xor } (b_1 \text{ and } d_1) \text{ xor } (b_2 \text{ and } d_2) \text{ xor } (d_3 \text{ and } a_1) \text{ xor } (d_1 \text{ and } a_3) \text{ xor } (b_0 \text{ and } d_2) \text{ xor } (b_2 \text{ and } d_0) \text{ xor } (b_3 \text{ and } d_3) \text{ xor } (a_0 \text{ and } d_0) \text{ xor } (b_0 \text{ and } d_3) \text{ xor } (b_1 \text{ and } d_2) \text{ xor } (b_2 \text{ and } d_1) \text{ xor } (b_3 \text{ and } d_0) \text{ xor } (b_0 \text{ and } d_0)$

$y_1 = \text{'1' xor } (d_3 \text{ and } a_0) \text{ xor } (d_2 \text{ and } a_1) \text{ xor } (d_1 \text{ and } a_2) \text{ xor } (d_0 \text{ and } a_3) \text{ xor } (b_1 \text{ and } d_1) \text{ xor } (b_2 \text{ and } d_3) \text{ xor } (b_3 \text{ and } d_2) \text{ xor } (d_3 \text{ and } a_3) \text{ xor } (d_3 \text{ and } a_1) \text{ xor } (d_1 \text{ and } a_3) \text{ xor } (b_3 \text{ and } d_3) \text{ xor } (d_3 \text{ and } a_2) \text{ xor } (d_2 \text{ and } a_3) \text{ xor } (b_0 \text{ and } d_0)$

y_0 = '1' xor (d_3 and a_0) xor (d_2 and a_1) xor (d_1 and a_2) xor (d_0 and a_3) xor (a_0 and d_2) xor (a_2 and d_0) xor (b_0 and d_1) xor (b_1 and d_0) xor (d_2 and a_2) xor (b_0 and d_2) xor (b_2 and d_0) xor (b_1 and d_3) xor (b_3 and d_1) xor (d_3 and a_2) xor (d_2 and a_3) xor (b_0 and d_0)

B-1. Stage 1 of S-Box^{-1}

a_3 = x_6 xor x_1 xor x_7 xor x_2
a_2 = '1' xor x_6 xor x_1 xor x_3 xor x_0 xor x_2 xor x_7
a_1 = '1' xor x_6 xor x_4 xor x_5 xor x_0
a_0 = '1' xor x_4 xor x_5 xor x_3
b_3 = '1' xor x_6 xor x_1 xor x_2 xor x_5
b_2 = x_3 xor x_0 xor x_5
b_1 = '1' xor x_6 xor x_4 xor x_0 xor x_3 xor x_1
b_0 = x_4 xor x_5 xor x_3 xor x_6 xor x_7 xor x_2
c_3 = x_7 and x_1 xor (x_2 and x_1) xor (x_3 and x_1) xor x_0 xor (x_7 and x_0) xor (x_5 and x_3) xor (x_3 and x_0) xor x_4 xor (x_7 and x_4) xor (x_5 and x_4) xor (x_2 and x_0) xor (x_4 and x_2) xor (x_6 and x_2) xor (x_4 and x_1) xor (x_6 and x_0) xor (x_6 and x_4)
c_2 = '1' xor x_0 xor x_7 xor (x_7 and x_0) xor (x_7 and x_5) xor (x_6 and x_5) xor (x_1 and x_0) xor (x_3 and x_1) xor x_1 xor x_6 xor (x_7 and x_1) xor x_4 xor (x_4 and x_2) xor (x_4 and x_1) xor (x_6 and x_4) xor (x_7 and x_2)
c_1 = x_1 xor (x_5 and x_1) xor (x_6 and x_1) xor (x_3 and x_0) xor (x_5 and x_3) xor (x_5 and x_0) xor (x_6 and x_5) xor x_2 xor (x_4 and x_2) xor (x_6 and x_2) xor x_0 xor x_5 xor x_4 xor x_7 xor (x_7 and x_0) xor (x_7 and x_4) xor (x_6 and x_4) xor x_6 xor (x_7 and x_2) xor (x_7 and x_1) xor (x_3 and x_2)
c_0 = '1' xor (x_3 and x_2) xor (x_4 and x_2) xor x_3 xor x_4 xor (x_7 and x_4) xor (x_6 and x_5) xor (x_6 and x_4) xor (x_1 and x_0) xor (x_4 and x_1) xor (x_6 and x_1) xor (x_3 and x_0) xor (x_4 and x_3) xor (x_5 and x_4) xor (x_7 and x_0)

B-2. Stage 2 of S-Box^{-1}

Same as the stage 2 of the S-Box in A-2.

B-3. Stage 3 of S-Box^{-1}

y_7 = (d_3 and a_0) xor (d_2 and a_1) xor (d_1 and a_2) xor (d_0 and a_3) xor (a_0 and d_2) xor (a_2 and d_0) xor (a_1 and d_1) xor (a_0 and d_1) xor (a_1 and d_0) xor (b_1 and d_1) xor (b_0 and d_1) xor (b_1 and d_0) xor (b_2 and d_3) xor (b_3 and d_2) xor (b_2 and d_2)
y_6 = (d_2 and a_2) xor (a_0 and d_2) xor (a_2 and d_0) xor (d_3 and a_3) xor (d_3 and a_1) xor (d_1 and a_3) xor (b_2 and d_2) xor (b_0 and d_2) xor (b_2 and d_0) xor (b_3 and d_3) xor (b_1 and d_3) xor (b_3 and d_1)
y_5 = (a_0 and d_2) xor (a_2 and d_0) xor (d_3 and a_3) xor (d_3 and a_1) xor (d_1 and a_3) xor (a_1 and d_1) xor (a_0 and d_1) xor (a_1 and d_0) xor (d_3 and a_2) xor (d_2 and a_3) xor (b_1 and d_1) xor (b_0 and d_1) xor (b_1 and d_0) xor (b_2 and d_3) xor (b_3 and d_2) xor (b_2 and d_2)
y_4 = (a_0 and d_2) xor (a_2 and d_0) xor (d_3 and a_1) xor (d_1 and a_3) xor (a_0 and d_1) xor (a_1 and d_0) xor (a_0 and d_0) xor (b_0 and d_2) xor (b_2 and d_0) xor (b_3 and d_3) xor (b_1 and d_3)

xor (b_3 and d_1) xor (b_1 and d_1) xor (b_0 and d_1) xor (b_1 and d_0) xor (b_2 and d_3) xor (b_3 and d_2)

y_3 = (a_0 and d_1) xor (a_1 and d_0) xor (d_2 and a_2) xor (a_0 and d_0) xor (d_3 and a_3) xor (b_0 and d_3) xor (b_1 and d_2) xor (b_2 and d_1) xor (b_3 and d_0) xor (b_0 and d_2) xor (b_2 and d_0) xor (b_1 and d_1) xor (b_0 and d_1) xor (b_1 and d_0)

y_2 = (d_3 and a_1) xor (d_3 and a_0) xor (d_2 and a_1) xor (d_1 and a_3) xor (d_1 and a_2) xor (d_0 and a_3) xor (a_0 and d_0) xor (a_1 and d_1) xor (b_0 and d_3) xor (b_1 and d_2) xor (b_2 and d_1) xor (b_3 and d_0) xor (b_0 and d_2) xor (b_2 and d_0) xor (b_1 and d_1) xor (b_0 and d_1) xor (b_1 and d_0)

y_1 = (a_0 and d_1) xor (a_1 and d_0) xor (d_2 and a_2) xor (a_0 and d_0) xor (d_3 and a_3)

y_0 = (a_0 and d_2) xor (a_2 and d_0) xor (d_3 and a_1) xor (d_1 and a_3) xor (a_0 and d_1) xor (a_1 and d_0) xor (a_0 and d_0) xor (b_2 and d_2) xor (b_0 and d_2) xor (b_2 and d_0) xor (b_1 and d_3) xor (b_3 and d_1) xor (b_0 and d_0) xor (b_1 and d_1) xor (b_2 and d_3) xor (b_3 and d_2)

C-1. Stage 1 of Inverter over the AES Field

Same as the stage 1 of the S-Box in A-1.

C-2. Stage 2 of Inverter over the AES Field

Same as the stage 2 of the S-Box in A-2.

C-3. Stage 3 of Inverter over the AES Field

Same as the stage 3 of the S-Box^{-1} in B-3.

Simplified Adaptive Multiplicative Masking for AES

Elena Trichina, Domenico De Seta, and Lucia Germani

Cryptographic Design Center, Gemplus Technology R & D
Via Pio Emanuelli 1, 00143 Rome, Italy
{elena.trichina,domenico.deseta,lucia.germani}@gemplus.com

Abstract. Software counter measures against side channel attacks considerably hinder performance of cryptographic algorithms in terms of memory or execution time or both. The challenge is to achieve secure implementation with as little extra cost as possible. In this paper we optimize a counter measure for the AES block cipher consisting in transforming a boolean mask to a multiplicative mask prior to a non-linear Byte Substitution operation (thus, avoiding S-box re-computations for every run or storing multiple S-box tables in RAM), while preserving a boolean mask everywhere else. We demonstrate that it is possible to achieve such transformation for a cost of two additional multiplications in the field.
However, due to an inherent vulnerability of multiplicative masking to so-called zero attack, an additional care must be taken to securize its implementation. We describe one possible, although not perfect, approach to such an implementation which combines algebraic techniques and partial re-computation of S-boxes. This adds one more multiplication operation, and either occasional S-box re-computations or extra 528 bytes of memory to the total price of the counter measure.

1 Introduction

With the increasing research endeavors in the field of *side-channel attacks* both hardware and software implementations of cryptosystems have to take into account various counter measures. The main techniques are timing attacks [10], simple (SPA) and differential power analysis (DPA) [11], and electromagnetic attacks [7] . A particularly worrying factor is that the first three attacks can be mounted using cheap resources. The last one requires more sophisticated set-up, including the design of special probes and development of advanced measurements methods.

In what follows we do not describe how the attacks work; papers [10,11,9,13] provide an excellent study of this topic; we just outline their main principles. Side-channel attacks work because there is a correlation between the physical measurements taken during computations (e.g., power consumption, computing time, EMF radiation, etc.) and the internal state of the processing device, which is itself related to the secret key. An SPA is an attack where the adversary can

B.S. Kaliski Jr. et al. (Eds.): CHES 2002, LNCS 2523, pp. 187–197, 2003.

directly use a single power consumption signal to break a cryptosystem. For example, if an implementation of a cryptographic primitive includes branches that depend on the secret data, particularly, if the bodies of the *'then'* and *'else'* branches differ, an SPA attack can be successfully mounted with very inexpensive resources.

A DPA attack uses statistical analysis to extract information from a collection of power consumption curves obtained by running an algorithm many times with different inputs. Then the analysis of the probability distribution of points on the curves is carried on. The DPA uses a correlation between power consumption and specific key-dependent bits which appear at known steps of the encryption computations. For example, a selected bit b at the output of one S-box of the first round of the Advanced Encryption Standard (AES) [6] will depend on the known input message and 8 unknown bits of the key. The correlation between power consumption and b can be computed for all the 256 values of 8 unknown bits of the key. The correlation is likely to be maximal for the correct guess of the 8 bits of the key. Then the attack can be repeated for the remaining S-boxes.

It has been claimed that all "naive" implementations can succumb to attacks by power analysis technique. The only solution is to re-implement cryptosystems taking into account a wide range of counter measures, although the cost in terms of performance and memory usage can be high. General strategies to combat side-channel attacks are [9]:

- de-correlate the output traces on individual runs (e.g., by introducing random timing shifts and wait states, inserting dummy instructions, randomization of the execution of operations, etc.);
- replace critical assembler instructions with ones whose "consumption signature" is hard to analyze, or re-engineer the critical circuitry which performs arithmetic operations or memory transfers;
- make algorithmic changes to the cryptographic primitives so that attacks are provably inefficient on the obtained implementation, e.g., masking data and key with random mask generated at each run.

It had been shown [3,9] that among these, algorithmic techniques are the most versatile, all-pervasive, and may be the most powerful. Also, in many contexts it is the cheapest to put in place.

In [1], Akkar and Giraud described a practical implementation of the AES using a new *adaptive masking method*. The idea is the following: the message is masked by means of a traditional XOR operation with some random X at the beginning of the algorithm; and thereafter everything is almost as usual. The XOR operation is compatible with the AES structure except for an inversion in the field; hence the mask must be arithmetic on $GF(2^8)$. For this, the authors devised a technique of transforming a boolean mask into a multiplicative mask, namely a *modified byte substitution*. Of course, the value of the mask at some fixed step (e.g., at the end of the round) must be known in order to re-establish the expected value at the end of the execution. Fig. 1 illustrates the difference between one round of the AES with and without masking counter measure. In what follows we review the proposed counter measure and suggest a new solution

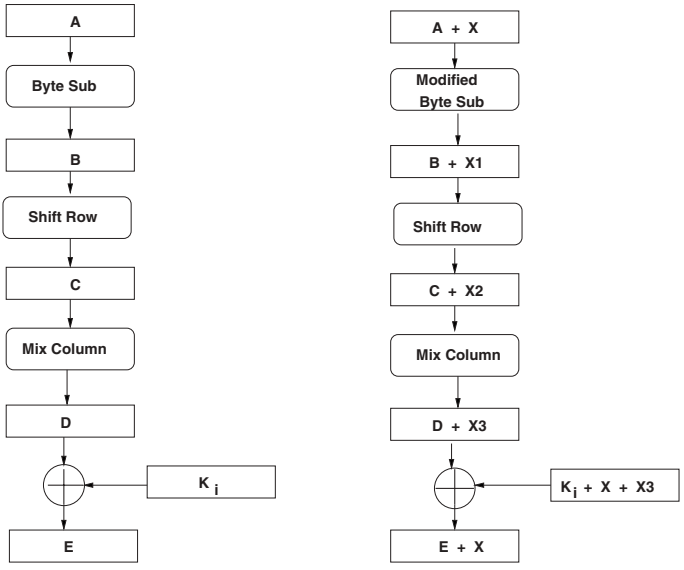


Fig. 1. One round of the AES with and without multiplicative masking counter measure.

based on the same idea; a solution that significantly simplifies the structure of the algorithm and reduces the number of expensive field operations. After which we conduct a security analysis of the simplified method and propose some techniques for its secure implementation.

2 Adaptive Masking Method for AES

The block cipher Rijndael [6] became an official new advanced encryption standard (AES) in 2001. It means that the AES will be used as the standard cryptographic algorithm for financial transactions, for telecommunication applications, and in many areas where DES is currently used. The large potential market is making it worthwhile for chip manufactures to run AES on their Smart Card micro-controllers. Since Smart Cards are easy victims to side-channel attacks, implementation of counter measures is mandatory. However, a price to pay must not be prohibitive for devices such as Smart Cards that have limited memory; and their on-line usage requires reasonable time performance.

2.1 The Rijndael Round

For simplicity, we consider the 128-bit block- and and key sizes version on the basis that the cryptanalytic study of the Rijndael during the standardization process was primarily focused on this version. For a complete mathematical

specification of the Rijndael algorithm we refer readers to [6]. An encryption module is shown in Fig. 2. The total number of rounds (counting the extra round performed at the end of enciphering) is 10, the key block length and data block length are both equal to 4.

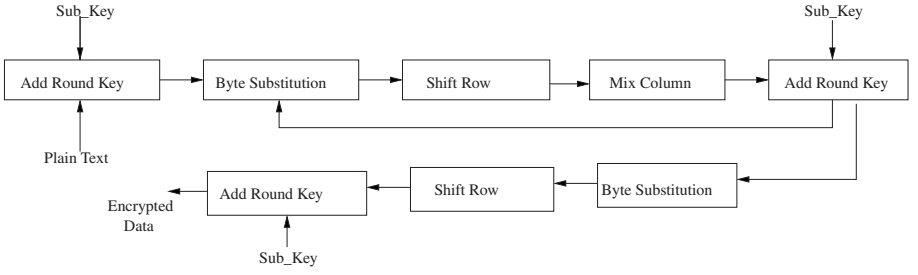


Fig. 2. The main flow of the algorithm.

In the Rijndael, the 128-bit data block is considered as a 4×4 array of bytes. The algorithm consists of an initial data/key addition, 9 full rounds (when the key length is 128 bits), and a final (modified) round. A separate key scheduling module is used to generate all the sub-keys, or *round keys*, from the initial key; a sub-key is also represented as 4×4 array of bytes. The full Rijndael round involves four steps.

The *Byte Substitution* step replaces each byte in a block by its substitute in an S-box. The S-box is an invertible substitution table which is constructed by a composition of two transformations, as Fig. 3 illustrates:

- First, each byte $A_{i,j}$ is replaced with its reciprocal in $GF(2^8)$ (except that 0, which has no reciprocal, is replaced by itself).
- Then, an affine transformation f is applied. It consists of
  - a bitwise matrix multiply with a fixed 8×8 binary matrix M,
  - after which the resultant byte is XOR-ed with the hexadecimal number $'63'$.

The S-box is usually implemented as a look-up table consisting of 256 entries; each entry is 8 bits wide; but it also can be computed "on-a-fly". Although the latter takes more time, it saves memory.

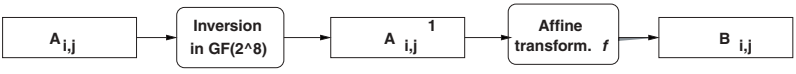


Fig. 3. Two steps of Byte Substitution transformation.

Next comes the *Shift Row* step. Each row in a 4×4 array of bytes of data is shifted 0, 1, 2 or 3 bytes to the left in a round fashion, producing a new 4×4 array of bytes.

In the *Mix Column* step, each column in the resultant 4×4 array of bytes is considered as polynomial over $GF(2^8)$ and multiplied modulo x^4+1 with a fixed polynomial $c(x) =' 03'x^3+'01'x^2+'01'x+'02'$. The operation of a multiplication with a fixed polynomial $a(x) = a_3x^3+a_2x^2+a_1x^1+a_0$ can be written as a matrix multiplication where the matrix is a circular matrix with the first row equal to a_0, a_3, a_2, a_1, each subsequent row is obtained by a circular shift of the previous one by 1 position to the left. Since multiplication is carried out in $GF(2^8)$, the product is calculated modulo irreducible polynomial $m(x) = x^8+x^4+x^3+x+1$, or $'11B'$ in hexadecimal representation.

The final step, *Add Round Key*, simply XOR-es the result with the sub-key for the current round.

In parallel to the round operation, the round key is computed in the *Key Scheduling Block*. The round key is derived from the cipher key by means of key expansion and round key selection.

Round keys are taken from the expanded key (which is a linear array of 4-byte words) in the following way: the first round key consists of the first N_b words, the second of the following N_b words, etc. The first N_k words are filled in with the cipher key. Every following word $W[i]$ is obtained by XOR-ing the words $W[i-1]$ and $W[i-N_k]$.

For words in positions that are multiples of N_k, the word is first rotated by one byte to the left; then its bytes are transformed using the S-box from the *Byte Substitution* step, after which XOR-ed with the round-dependent constant.

2.2 Adaptive Multiplicative Masking

It is easy to see that the problem of implementing a masking counter measure comes from the *Byte Substitution* transformation, which is the only non-linear part. One known solution [13] consists in masking a table look-up T which implements the S-box with two boolean masks, the input mask R_{in} and the output mask R_{out}, as follows: $T[A_{i,j}] = T'[A_{i,j} \oplus R_{in}] \oplus R_{out}$. This implies that the masked table must be computed for each pair R_{in}, R_{out}. If done "on-a-fly", it takes time. Another solution is to fix a pair R_{in}, R_{out} prior each run, and precompute table look-ups for all such pairs. If one wants to mask every byte in 128-bit data, it would require as much as 256×16 bytes, or 4K of memory, which is not desirable for memory-limited devices like Smart Cards.

[1] suggests a method that allows to obtain the scheme without S-box recomputations. The message is masked at the beginning of the algorithm by XOR-ing it with a random value, generated for every new run; and thereafter everything is nearly as usual.

Since the mask must be arithmetic on $GF(2^8)$, the transformation *"boolean mask to multiplicative mask"* is devised such that the first step of the *Byte Substitution*, namely, the inversion in $GF(2^8)$, produces a masked multiplicative inverse of the input data, as shown in Fig. 4. Here $X_{i,j}$ is an 8-bit random

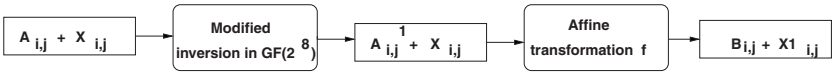


Fig. 4. Modified Byte Substitution with masking counter-measure.

value which masks data $A_{i,j}$ and $X1_{i,j} = f(X_{i,j})$ (this comes from the affine property of f). Then, the reverse transformation *"multiplicative mask to boolean mask"* is performed to restore an additive mask on the inverse data before an affine transformation f takes place. The full scheme of the modified inversion

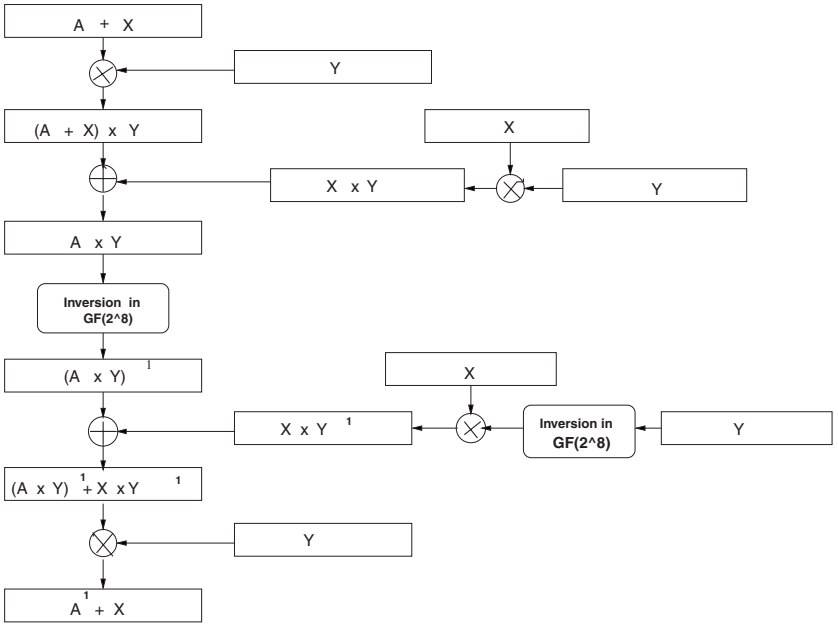


Fig. 5. Modified inversion in $GF(2^8)$ with masking counter measure.

is depicted in Fig. 5. As one can see, it requires an additional random variable, $Y_{i,j}$, one additional inversion, and 4 extra multiplications in the field. During all stages of the modified inversion, intermediary values seem to be independent from $A_{i,j}$.

While the masking methods, like in [13], must respect a masking condition at each step of the algorithm, using the transformed method one only needs to know the value of the mask at a fixed step (e.g., at the end of the round, or at the end of a non-linear part). The expected value is re-established after the computations at the end of the algorithm.

3 Simplified Multiplicative Masking

The idea of the simplified transformed masking is the same as the one described above. At the beginning of the byte substitution, the input value is $A_{i,j} \oplus X_{i,j}$, where $X_{i,j}$ is a random byte (we can safely drop indices i, j). We found a very efficient method that allows us to have $A^{-1} \oplus X$ at the end of the inversion without compromising value A. It can be described using solely algebraic laws for operations in finite fields.

Let us approach our goal from two directions simultaneously: from the input $A \oplus X$ working forwards to get $A^{-1} \oplus X^{-1}$, and from the output $A^{-1} \oplus X$ working backwards, also towards $A^{-1} \oplus X^{-1}$.

1. Suppose, we managed to obtain $A \otimes X$ from $A \oplus X$ without compromising A; then applying inversion in $GF(2^8)$, we get $A^{-1} \otimes X^{-1}$. Here how it can be done.
   a) We want to have a multiplicative masking $A \otimes X$ from an additive masking $A \oplus X$. A distributivity law $(a \oplus b) \otimes c = (a \otimes c) \oplus (b \otimes c)$ gives us the idea. Substituting A for a and X for c, we get
   $$(A \oplus X) \longrightarrow (A \oplus X) \otimes X = A \otimes X \oplus X^2.$$
   b) To obtain a pure multiplicative mask, we have to get rid of X^2. An algebraic law $a \oplus a = 0$ can be applied here
   $$A \otimes X \oplus X^2 \longrightarrow A \otimes X \oplus X^2 \oplus X^2 = A \otimes X.$$
   c) At this stage, one can safely apply inversion in $GF(2^8)$
   $$A \otimes X \longrightarrow (A \otimes X)^{-1} = A^{-1} \otimes X^{-1}.$$
2. Now we face a symmetric task of obtaining the additive mask $A^{-1} \oplus X$ from the multiplicative mask $A^{-1} \otimes X^{-1}$. The algebraic law $x^{-1} \otimes x = 1$ will help to get rid of the X^{-1}; but before doing so, an additional step must be taken.
   a) Ensure that the value A^{-1} will not be revealed in the process:
   $$A^{-1} \otimes X^{-1} \longrightarrow A^{-1} \otimes X^{-1} \oplus 1.$$
   b) Now we can get rid of X^{-1}:
   $$A^{-1} \otimes X^{-1} \oplus 1 \longrightarrow (A^{-1} \otimes X^{-1} \oplus 1) \otimes X = A^{-1} \otimes 1 \oplus X = A^{-1} \oplus X.$$

Fig. 6 depicts this method graphically. Compare it with Fig. 5. As one can see, ours is a significant simplification of the solution in [1]; it requires no extra inversion in $GF(2^8)$, and only two extra multiplications and one squaring.

One can argue that the first step in the "boolean mask to multiplicative mask" transformation, can jeopardize the security of the masking because $(a \oplus x) \otimes x$ is not fully random for a random x. Indeed, as has been pointed out in [5], the equation $(a \oplus x) \otimes x = y$ has either zero solution, or two solutions; namely,

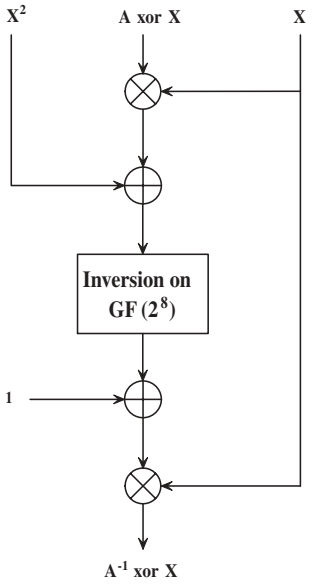


Fig. 6. Inversion in $GF(2^8)$ with simplified multiplicative masking.

substituting x with $z \otimes a$ in the equation above, we obtain $z^2 \oplus z = y/a^2$. We know that the equation $z^2 \oplus z = b$ has zero solution if $Trace(b) = 1$, and two solutions if $Trace(b) = 0$. So $(a \oplus x) \otimes x$ reaches only half of the elements in $GF(2^8)$. However, it is sufficiently random to serve the purpose.

If the original additive mask is restored at the end of the round, as illustrated in Fig. 1, there are no limitations on the choice of the random 128-bits mask X apart from the requirements that none of its bytes $X_{i,j}$ is equal to zero.

However, one can imagine the computation scheme, where, instead of restoring the original mask $X_{i,j}$ at the end of each round, one simply goes on with the computations, taking the value $X1_{i,j} = f(X_{i,j})$ as the mask for the second round, $X2_{i,j} = f(X1_{i,j})$ for the third, etc.

Only at the end of the computations the data is unmasked. The corresponding "correction" on the mask has to be carried out in parallel with the main algorithm. In this case, due to the nature of the *Mix Column* and *Shift Row* operations, some of the random bytes $XK_{i,j}$ for the K-th round, $K = 2, ...9$, can turn to zeros.

A mathematical analysis of the effect of the *Mix Column* and *Shift Row* operations on bytes of the mask indicates, and a simple computer experiment with all possible choice of random bytes for a 128-bit random confirms, that the sufficient condition which effectively prevents this from happening is that no two bytes of the initial 128-bit random X should be the same.

4 Securized Implementation of the Simplified Multiplicative Masking

Let us ask a question: how to implement the simplified multiplicative masking in a secure way? A straightforward implementation can lead to a potential security flaw, as had been pointed out in [2,4,8]. The flaw consists in the fact that multiplicative mask masks only **non-zero values**, i.e., zero input value is mapped into zero by the inversion. In other words, if an attacker can detect that the value before (i.e., $A_{i,j} \otimes X_{i,j}$) and after (i.e., $(A_{i,j} \otimes X_{i,j})^{-1}$) the inversion is 0, he/she gets an information on $A_{i,j}$. An attacker can exploit this fact and mount the first order DPA attack as if no masking has been applied.

Hence, not to reveal the weakness, nowhere in the implementation there should be a moment where both, $A_{i,j} \otimes X_{i,j}$ and $(A_{i,j} \otimes X_{i,j})^{-1}$, are read or written in clear. In other words, the counter measure shown in Fig. 6 must be implemented in a completely protected environment. Currently we are working on such an implementation. The obvious solution stems from the nature of our simplified masking.

Indeed, since the constant 1 is to be added to every entry of the inverse table (see Fig. 6), it can be done in advance, while creating the table itself. This, however, may not protect from the attacker who now, instead of looking for 0 as the resulting value of the table lookup will look for 1. The situation can be remedied as follows.

- Prior to a run of the AES algorithm, all entries of a table are XOR-ed with some random constant value K. Then the result of the table lookup for $(A_{i,j} \otimes X_{i,j})$ will be $(A_{i,j} \otimes X_{i,j})^{-1} \oplus K$.
- The subsequent multiplication with $X_{i,j}$ produces $((A_{i,j})^{-1} \oplus (K \otimes X_{i,j}))$, which either can be carried to the affine transformation f with $K \otimes X_{i,j}$ as a new random or can be replaced with $X_{i,j}$ by further XOR-ing the table lookup result with $X_{i,j} \oplus (K \otimes X_{i,j})$.

This adds 256 bytes and one more field multiplication to the implementation; however, computing a simplified multiplicative masking is still more efficient than re-computing S-boxes for every run.

Still, the problem remains: how not to reveal $A_{i,j} \otimes X_{i,j}$ during computations? We do not have a good answer to this question yet, which undoubtedly weakens our counter measure. To prevent reading $A_{i,j} \otimes X_{i,j}$ in clear from the inverse table T, the upper-most operation $\oplus$ with X^2 in Fig. 6 must never be actually performed.

One solution is that operation $\oplus$ with $(X_{i,j}^2 \oplus M)$ for some a-priori chosen M is carried out, simultaneously updating T in such a way that for a new table T': $T'[B \oplus M] = T[B]$. M could be chosen so that this re-computation amounts to simple re-shufling of the indices; for example, if $M = 1$, T' is obtained from T by simply "swapping" each even and odd entries. Obviously, from time to time, M and, respectively, the table T' must be re-newed.

Another solution brings us back to S-box re-computations: instead of recovering $A_{i,j} \otimes X_{i,j}$ from $A_{i,j} \otimes X_{i,j} \oplus X_{i,j}^2$ a new table T' such that $T'[A_{i,j} \otimes X_{i,j} \oplus$

$X^2_{i,j}] = T[A_{i,j} \otimes X_{i,j}]$ is computed. Since only the first and the last rounds are most vulnerable to the DPA, it seems enough to apply re-computations (or store two extra pre-computed S-boxes) only for these rounds.

A general algorithm to compute such T' given some table T and a random value X is described below.

Look-up table re-computation.

```
Input:  table T;
        random X = (x_7, ..., x_1, x_0)
Output: table T' such that T'[b+X] = T[b] for b = 0..255
   T' := T;
   For every x_i from (x_7, ..., x_0) in random order do:
      If x_i = 1 then
         (1) split T' into blocks, each block containing 2^(x_i)
             subsequent elements from T;
         (2) swap pairwise j-th and j+1-st blocks;
         (3) assign the result to T';
   Return T'
```

Notice, that the algorithm reads bits of X at random which provides some protection from an attacker during re-computations.

The proposed securized implementation of the simplified adaptive masking is computationally more efficient than full S-box re-computations, thus representing a compromise between cost and security.

5 Conclusion

We have shown that the *Modified Byte Substitution* can be implemented in a way that to some degree avoids a severe security flaw paying a price of additional multiplications and RAM usage.

However, many security features are a matter of trade-offs. Described in this paper implementation of the simplified multiplicative masking provides similar protection as an S-box re-computation but has lower implementation costs, in terms of both, memory and execution time; namely, the price to pay is, apart from numerous XOR operations, only three to four extra multiplications in $GF(2^8)$ per round plus an occasional re-shuffling of the inverse table stored in ROM. While may not ensuring a complete protection from a sophisticated attacker due to an inherent vulnerability of a multiplicative masking in the field, the method increases the number of power curves acquisitions and thus can be sufficient for a low-end line of Smart Cards.

Another, quite different solution would be not to work in the field $GF(2^8)$, but in the field $GF(2^8 + a)$ where a is chosen such that $2^8 + a$ is prime. The we can avoid zero by replacing it by $2^8 + 1$. This is what had been done in IDEA cipher as a solution to the problem of inverting a number modulo 2^{16} during decryption [12].

References

1. Akkar, M., Giraud, C.: An implementation of DES and AES, secure against some attacks. Proc. *Cryptographic Hardware and Embedded Systems: CHES 2001*, LNCS **2162** (2001) 309–318
2. Akkar, M., Goubin, L.: A generic protection against high-order differential power analysis. *Manuscript*
3. Chari, S., Jutla, C., Rao, J., Rohatgi, P.: Towards sound approaches to counteract power-analysis attacks. Proc. *Advances in Cryptology – Crypto'99*, LNCS **1666** (1999) 398–412
4. Courtois, N., Akkar, M.: Time and memory efficiency in protecting against higher order power attacks. *Manuscript*
5. Coron, J.-S., personal communication
6. Daemen, J., Rijmen, V.: *The design of Rijndael: AES – The Advanced Encryption Standard*. Springer-Verlag Berlin Heidelberg, 2002
7. Gandolfi, K., Mourtel, C., Oliver, F.: Electromagnetic analysis: concrete results. Proc. *Cryptographic Hardware and Embedded Systems: CHES 2001*, LNCS **2162** (2001) 251–261
8. Golic, J., Tymen, Ch.: : Multiplicative masking and power analysis of AES. Proc. *Cryptographic Hardware and Embedded Systems: CHES 2002*, LNCS **2523** (2002) *These proceedings.*
9. Goubin, L., Patarin, J.: DES and differential power analysis. Proc. *Cryptographic Hardware and Embedded Systems: CHES'99*, LNCS **1717** (1999) 158–172
10. Kocher, P.: Timing attacks on implementations of Diffie-Hellmann, RSA, DSS, and other systems. Proc. *Advances in Cryptology – Crypto'96*, LNCS **1109** (1996) 104–113
11. Kocher, P., Jaffe, J., Jun, B.: Differential power analysis. Proc. *Advances in Cryptology – Crypto'99*, LNCS **1666** (1999) 388–397
12. Massey, J. L., Lai, X.: Device for the conversion of a digital block and use of same. U.S. Patent # 5,214,703, 25 May 1993
13. Messerges, T.: Securing the AES finalists against power analysis attacks. Proc. *Fast Software Encryption Workshop 2000* LNCS **1978** (2000) 150–165

Multiplicative Masking and Power Analysis of AES

Jovan D. Golić[1⋆] and Christophe Tymen[2,3]

[1] Rome CryptoDesign Center, Gemplus
Via Pio Emanuelli 1, 00143 Rome, Italy
jovan.golic@gemplus.com
[2] Gemplus Card International
34 rue Guynemer, 92447 Issy-les-Moulineaux, France
christophe.tymen@gemplus.com
[3] École Normale Supérieure
45 rue d'Ulm, 75230 Paris Cedex 05, France

Abstract. The recently proposed multiplicative masking countermeasure against power analysis attacks on AES is interesting as it does not require the costly recomputation and RAM storage of S-boxes for every run of AES. This is important for applications where the available space is very limited such as the smart card applications. Unfortunately, it is here shown that this method is in fact inherently vulnerable to differential power analysis. However, it is also shown that the multiplicative masking method can be modified so as to provide resistance to differential power analysis of nonideal but controllable security level, at the expense of increased computational complexity. Other possible random masking methods are also discussed.

Keywords. AES, differential power analysis, countermeasures, multiplicative masking.

1 Introduction

Side-channel attacks on software or hardware implementations of various cryptosystems aim at recovering the secret key information from certain physical measurements performed on the electronic device during the computation such as the power consumption, the time, and the electromagnetic radiation. Power analysis attacks [9] are very powerful as they do not require expensive resources and as most implementations without specific countermeasures incorporated are vulnerable to such attacks. Among them, the (first-order) differential power analysis (DPA) attacks are particularly impressive, because they use relatively simple mathematical tools and techniques that are independent of the implementation of the cryptographic algorithm. Moreover, the countermeasures are typically costly in terms of speed performance and memory requirements.

⋆ A preliminary version of this work was presented at the Gemplus Quarterly meeting, La Ciotat, France, October 30–31, 2001.

B.S. Kaliski Jr. et al. (Eds.): CHES 2002, LNCS 2523, pp. 198–212, 2003.

The goal of simple power analysis (SPA) attacks is to deduce some information about the secret key, such as the Hamming weight of some parts of the key, from a single power consumption curve. This may be possible if, for example, there are branches in the computation that depend on the secret key. More generally, one can also collect a large training set of power consumption curves from different secret keys (and possibly different input data) and then use appropriate statistical hypothesis testing methods in order to identify traces or signatures of the parts of the secret key hidden in the curves. For example, key scheduling algorithms for block ciphers may especially be vulnerable in this regard, due to the absence of the randomization effect of input data. However, the statistical techniques to be used may be complicated and dependent on the particular implementation.

The DPA attack [9] requires a set of power consumption curves obtained by running the cryptographic algorithm a number of times for the same secret key and different inputs. A necessary algorithmic condition, the so-called *fundamental hypothesis*, for the DPA attack to be effective is the existence of one or more intermediate variables in the algorithm that can be expressed as or are correlated to functions depending on a small number of key bits and on known input or output data. The key bits involved may then be reconstructed by partitioning the set of curves according to the value of the chosen intermediate variable corresponding to the key bits guessed and to the input or output data known and by computing and comparing some simple statistic, such as the average, on the partitioned curves at individual points in time. The attack is successful if the correct guess about the key bits results in a significant difference between the computed average curves at one or more points in time. For other possibilities, see [4]. What makes the attack practically very interesting is that many cryptographic algorithms satisfy the fundamental hypothesis. For example, the intermediate variables in the first or the last few rounds of practical block ciphers are especially vulnerable.

A higher-order DPA attack is a generalization of the (first-order) DPA attack in which the power consumption curves are analyzed by using a joint statistic applied to collections of points in time. The general attack is more powerful, but may be more complex and considerably more complicated as the choice of these points and possibly also of the joint statistic is likely to depend on the particular implementation.

In principle, the complexity of the side-channel attacks can be increased by introducing physical or algorithmic countermeasures. A general strategy to render the SPA and DPA attacks more difficult to mount is to balance and randomize elementary computations involving the secret key, e.g., by randomly introducing dummy operations and timing shifts, as well as by randomizing the order of elementary computations and the computations themselves. A general technique to prevent the first-order or higher-order DPA attacks is random data splitting [7], [3], especially for the computation of intermediate variables satisfying the fundamental hypothesis. It is pointed out in [10] that for the (first-order) DPA, instead of splitting the data into two parts one may as well apply random masks

to data which are easier to implement. Of course, one has to be careful to mask the data completely and thus avoid weaknesses such as the one shown in [5] for a masking technique from [10]. Also, the computations involved in masking have to be performed in a secure way, which itself is not vulnerable to DPA. Random masks have to be generated for each new run of the cryptographic algorithm, but may be repeated within the algorithm. The repetitions generally increase the vulnerability to higher-order DPA. The random masks can be combined with data by using (quasi)group operations such as the bitwise addition or modular integer addition.

If an affine transformation is applied to masked data and if the masking operation is the same as the corresponding linear operation, then only the additive constant has to be recomputed for each new mask in order to maintain the equivalence of the data transformations. However, this is generally not the case with nonlinear transformations. They typically have to be recomputed and stored depending on the mask and this can be very costly for many cryptographic algorithms including AES [6]. In AES, the only nonlinear transformations are nonlinear parts of 8×8-bit S-boxes which perform the multiplicative inversion in GF(256). In [2], a masking technique is proposed which combines the usual binary additive masking with the multiplicative masking of data, using the multiplication in GF(256), thus avoiding the costly recomputation and RAM storage of S-boxes.

In this paper, it is shown that this multiplicative masking technique is vulnerable to the first-order DPA attack and in some sense even more than AES without masking. This is essentially because the all-zero input to the S-boxes is not effectively masked by the multiplicative mask and the binary additive mask is first removed in order to apply the multiplicative mask.[1] Moreover, it is argued that the weakness is inherent to the multiplicative masking and therefore cannot be remedied so as to achieve ideal security. In addition, the so-called embedded multiplicative masking technique which can achieve approximate security with a controllable security level is introduced. It is also pointed out that the masking technique [10] in which only one S-box is recomputed and stored in RAM and used repeatedly during one execution of AES is vulnerable to a relatively simple second-order DPA attack.

The main lines of the DPA attack [9] applied to AES are described in more detail in Section 2 and the multiplicative masking technique is presented in Section 3. The inherent weakness is explained in Section 4, the embedded multiplicative masking technique is proposed in Section 5, and conclusions are given in Section 6.

[1] A similar power analysis attack, although not in the DPA form, is independently given in the unpublished manuscript "Time and memory efficiency in protecting AES against higher order power attacks," by N. T. Courtois and M.-L. Akkar, from April 2002.

2 Differential Power Analysis of AES

AES is a product block cipher composed of a number of rounds each consisting of a reversible nonlinear transformation providing *confusion* and a reversible linear transformation providing *diffusion*, where the linearity is with respect to the binary field, GF(2). The expanded secret key is bitwise added to the plaintext and to the output of each round. The nonlinear transformation consists of identical 8×8-bit S-boxes each performing the byte substitution ByteSub defined as the multiplicative inversion in GF(256), leaving the all-zero input intact, followed by an affine 8×8-bit transformation. The linear transformation consists of a permutation of output bytes of S-boxes denoted as ShiftRow followed by a linear transformation denoted as MixColumn, which is removed from the last round. More details can be found in [6], but are irrelevant for our analysis.

According to [9], the DPA attack on AES consists of the following stages. The intermediate variables satisfying the fundamental hypothesis are the output bytes of the S-boxes or of just the nonlinear parts of the S-boxes in the first round. Each of them is a function of the input byte which itself is a bitwise sum of the corresponding plaintext and expanded key bytes. Accordingly, if the plaintext byte is known and the key byte is guessed correctly, then the S-box output byte can be computed correctly. The objective of the attack is to recover the expanded key in a byte-by-byte divide-and-conquer manner.

In the first stage, a sufficient number, N, of curves are obtained by measuring the power consumption during the execution of (the first round of) AES for the same secret key and N different plaintexts. The average C of these N curves is then computed. The second stage is performed for each of the S-boxes in the first round. For each of 256 possible values of the targeted expanded key byte K, a subset of M (on average, $M = N/2^m$) plaintexts resulting in a chosen fixed m-bit value of the partial output byte of the considered S-box are identified, the corresponding M curves are extracted, and their average $C(K)$ computed. For example, the chosen fixed value may have maximal or minimal Hamming weight (all-one or all-zero m-bit words). More generally, if one knows good power consumption models of the involved components, an optimal subset of M curves can be chosen according to a set of 2^{8-m} or of any other number of output byte values best distinguished from the others with respect to power consumption, as proposed in [1] for $m = 1$.

A value K is then assumed to be correct if the difference between the two average curves, $C(K)$ and C, contains one or more noticeable peaks. The peaks are due to the same value of the S-box output being computed at the same time for each of the extracted M curves, if the value K is correct, and to unbalanced power consumption associated with different S-box output values. *This is the main point of the DPA attack.* On the other hand, if the value K is incorrect, then the outputs of the S-box will vary and the peaks will not be observed. More precisely, this is the case for $m < 8$. For $m = 8$, as the S-boxes are reversible, a fixed output byte value implies that the input byte value is also fixed. Therefore, even if the guessed value K is incorrect, then for the extracted curves both the input and output bytes will have fixed values, different for different K, also

giving rise to observable peaks, possibly of different magnitudes for different K. As a consequence, the reliable statistical distinction of the correct K may be infeasible.

If m decreases, then M increases, but the impact of the repeated computation on each of the M extracted power consumption curves becomes statistically less significant. Also, it is not clear how one can simultaneously use more than just one fixed output m-bit value in order to increase the statistical distinction between the correct and incorrect key values. Nonetheless, this may be possible.

According to the key schedule in AES, if the key size is not bigger than the plaintext block size, then the recovered expanded key bytes directly specify the secret key. Otherwise, the DPA attack should also be applied to the second round of AES in order to recover the whole secret key.

3 Multiplicative Masking of AES

The starting idea of the method proposed in [2] in order to prevent the DPA attack on AES is to use the binary additive mask which is compatible with the binary linear or affine transformations in AES. Accordingly, as far as the affine transformations are concerned, only the additive constants are affected by this mask. However, if the additive mask is applied to the input of the nonlinear part of an S-box in AES, then this nonlinear part has to be recomputed for each new mask used. Recall that the nonlinear part, F, of the S-box transformation ByteSub is the multiplicative inversion in GF(256) extended by mapping the all-zero input into the all-zero output. For simplicity, F is called the inversion in GF(256). The main idea from [2] is to use the nonzero multiplicative mask, with respect to the multiplication in GF(256), for the data passing through F, without having to recompute and store F. To this end, one has to convert the additive mask into the multiplicative mask at the input of each F and to reproduce the additive mask from the multiplicative mask at the output of each F. A way, secure with respect to DPA, of converting the masks is suggested in [2]. More details are given below.

Fig. 1 shows the data flow in the i-th round of AES without and with the masking countermeasure. A general rule in all the figures presented is that the expressions for input, output, and all intermediate data are displayed within the rectangular blocks. It is assumed that the ByteSub and Modified ByteSub transformations act on all the bytes in a block. Note that the expanded key is bitwise added to the plaintext to form the input to the first round, that the MixColumn transformation is removed from the last round, and that the additive mask is not produced at the output of the last round. According to [2], the additive mask X is the same in every round. In fact, keeping the same additive mask in every round would matter if the S-boxes had to be recomputed for each new mask, because in that case the same recomputed S-boxes could be used in each round. So, restoring the same mask X in the last step of each round is not really needed. Instead, one can just add the expanded key K_i and thus effectively obtain the output mask $X^{(3)}$. Only in the last round, the output mask has to

be removed. Here, the masks are transformed by the linear transformations as $X^{(1)} = L(X)$, $X^{(2)} = \text{ShiftRow}(X^{(1)})$, and $X^{(3)} = \text{MixColumn}(X^{(2)})$, where L denotes the linear part of the affine transformation of ByteSub combined for all the bytes in a block. So, essentially only the ByteSub transformation has to be modified, because of the nonlinear part contained.

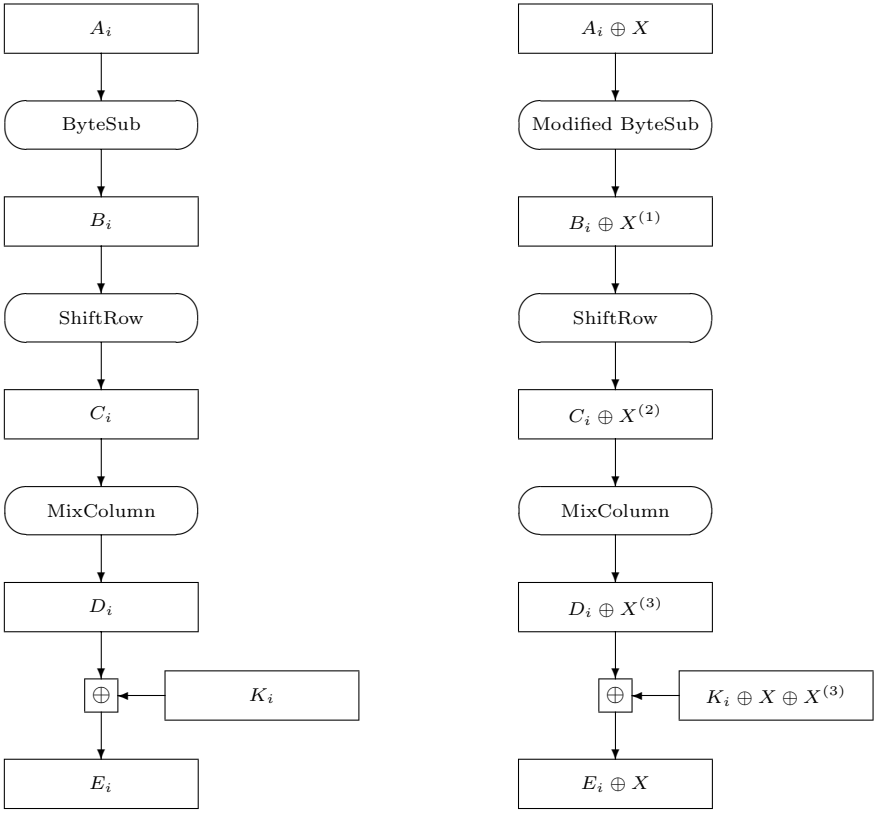


Fig. 1. The round i of AES without and with masking countermeasure.

The data flow through the original and modified ByteSub transformations, acting on bytes, is shown in Fig. 2 (the index j stands for a particular byte in a block and the index i stands for a particular round). The affine transformation is unchanged, and only the nonlinear transformation, F, has to be modified. This is achieved by using a nonzero multiplicative mask $Y_{i,j}$ in a way displayed in Fig. 3, which is self-explanatory.

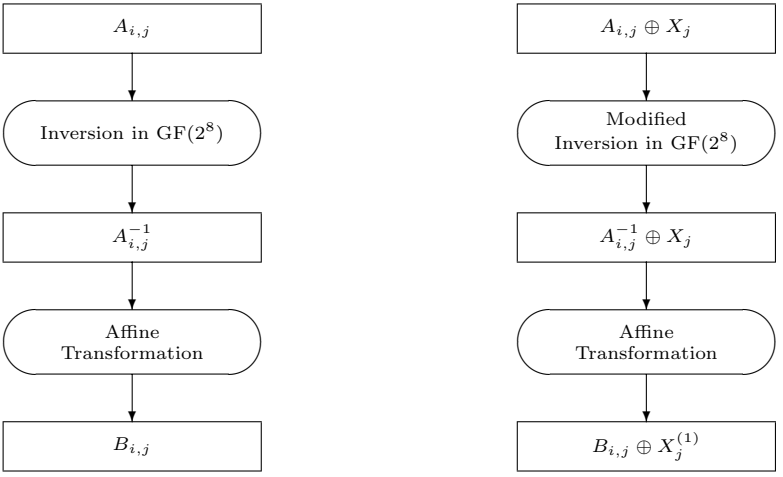


Fig. 2. The ByteSub transformation without and with masking countermeasure.

Recall that the addition in GF(256) is the same as the bitwise addition. It follows that F does not have to be recomputed and stored in a look-up table for each new mask $Y_{i,j}$. This is due to the multiplication in GF(256) being compatible with F or, more precisely, to the equality

$$F(A \otimes Y) \;=\; F(A) \otimes F(Y) \tag{1}$$

where $F(Y) \neq 0$ if $Y \neq 0$, so that $F(A)$ can be recovered from $F(A) \otimes F(Y)$. In other words, if a masked input is transformed by F itself, then the masked desired output is obtained. So, one just has to convert the multiplicative into the additive mask and vice versa, and that can be done by one more inversion, four multiplications, and two additions in GF(256).

Note that in general, if two, possibly different, group operations $*$ and $\bullet$ are used for masking the input and output data for a transformation F, respectively, then the masked data should be transformed by the modified transformation F' satisfying $F'(A * Y_1) = F(A) \bullet Y_2$. Equivalently, F' is defined by

$$F'(A) \;=\; F(A * Y_1^{-1}) \bullet Y_2, \tag{2}$$

where Y_1 and Y_2 are the input and output masks, respectively, which can be mutually related. In order to resist DPA, F' should not be directly implemented by using F and (2). For example, a secure way would be to use a look-up table for F', but it has to be recomputed and stored in RAM for every new pair of masks Y_1 and Y_2.

The multiplicative masks $Y_{i,j}$ and the additive masks X_j can be randomly chosen so as to be uniformly distributed and mutually independent. Also, $Y_{i,j}$ can be the same for each round i and possibly related to X_j, but this generally increases the vulnerability to higher-order DPA. Since all the intermediate variables in Fig. 3 are masked, it is claimed in [2] that the masked AES should be resistant to the (first-order) DPA attack. This masking method is important, because it avoids the recomputation and storage of S-boxes for each new run of AES, which would, for example, require 256×16 bytes of RAM for the 128-bit AES if all the S-boxes in a round are masked by mutually independent masks.

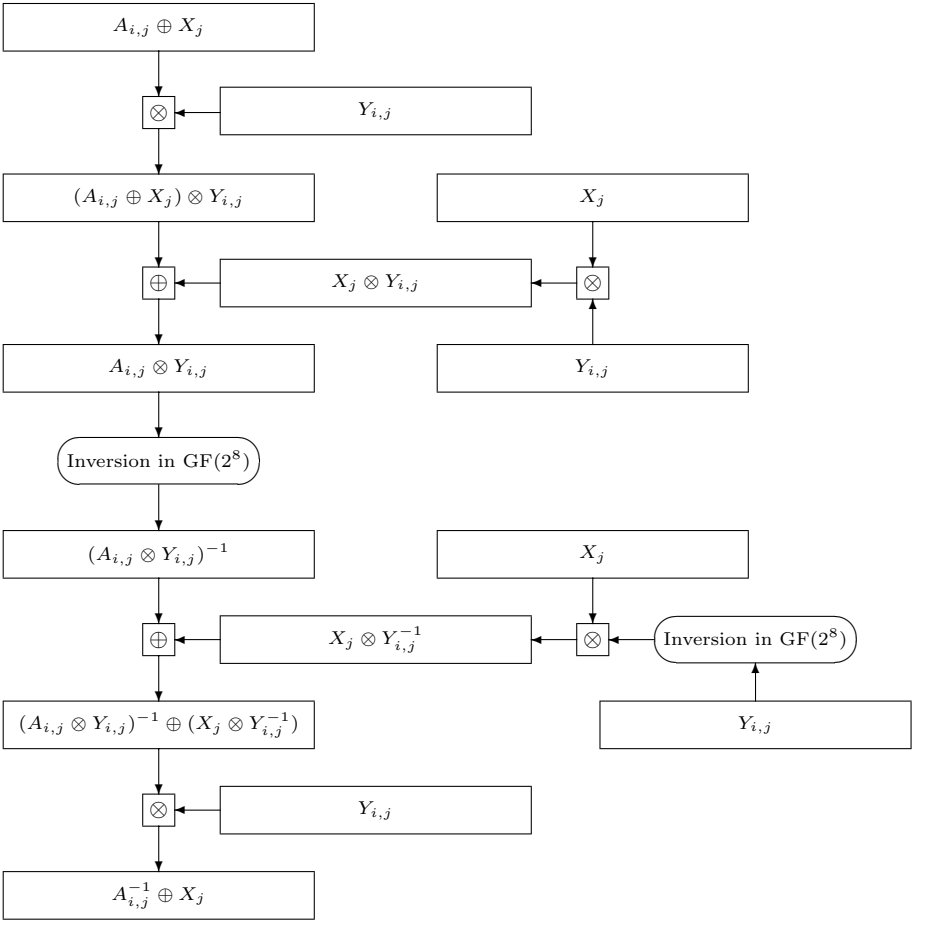


Fig. 3. Modified inversion in GF(256) with multiplicative masking countermeasure.

4 Differential Power Analysis of Masked AES

In this section, a subtle security flaw of the masking method [2] described in Section 3 is pointed out. In addition, it is argued that the multiplicative masking for AES is inherently vulnerable to the DPA attack.

The basic problem with the multiplicative mask is that it does not mask the all-zero byte value of data, that is, the all-zero byte remains unchanged after masking by a multiplicative mask. On the other hand, the all-zero (intermediate) data bytes cannot be avoided in AES. As a consequence, there are intermediate variables in the modified inversion scheme from Fig. 3 that are not masked completely and satisfy the fundamental hypothesis for DPA by being correlated to a function depending on only 8 key bits and 8 plaintext bits.

More precisely, in the first round of the masked AES, the vulnerable intermediate variables are the input byte $Z_{1,j} = A_{1,j} \otimes Y_{1,j}$ and the output byte $Z_{2,j} = F(A_{1,j} \otimes Y_{1,j})$ of the block implementing the inversion in GF(256). Note that the data byte $A_{1,j}$ is given as $A_{1,j} = P_j \oplus K_{0,j}$ where P_j and $K_{0,j}$ are the corresponding plaintext and expanded key bytes, respectively. It follows that

$$K_{0,j} = P_j \Rightarrow Z_{1,j} = 0 \Rightarrow Z_{2,j} = 0. \tag{3}$$

So, interestingly, it turns out that the (first-order) DPA attack on the masked AES can be mounted in essentially the same way as on the original AES without masking, which is described in Section 2. The difference is that one has to target the all-zero input byte or, equivalently, the all-zero output byte of F. In other words, for each of 256 possible values of the corresponding expanded key byte $K_{0,j}$, the power consumption curves for which $P_j = K_{0,j}$ are extracted and used for identifying the correct key. To this end, appropriately chosen plaintexts can reduce the required number of power curves. Since $m = 8$, the DPA attack on AES without masking would not be effective, as explained in Section 2.

However, for the masked AES, the DPA attack will be able to distinguish between correct and incorrect guesses of the expanded key byte, because of the randomization effect provided by the random multiplicative mask. Observe that if $K_{0,j}$ is guessed correctly, then the peaks will appear because of the repeated simultaneous computation of not only the all-zero output byte $Z_{2,j}$, but also the all-zero input byte $Z_{1,j}$. Altogether, the DPA attack may be more effective than the one on AES without masking, especially if one cannot find an effective way to simultaneously use more than just one fixed target m-bit value in the DPA attack on AES, where $m < 8$, or, more generally, a way to use (possibly optimal) partitions of power consumption curves into more than just two sets, provided that the power consumption models are available.

Now, the question is if the described DPA attack can be somehow prevented by using some other implementation of the multiplicative masking. For example, one may try to replace the modified inversion performed on $A_{i,j}$ by the modified inversion performed on some nonzero input byte whenever $A_{i,j} = 0$, and then to replace the computed output value by the desired one. However, detecting whether $A_{i,j} = 0$ and replacing the computed output value require

specific computations that are themselves vulnerable to the DPA attack. In conclusion, it appears that the weakness of the multiplicative masking for AES is hard to remove ideally. Nevertheless, there may exist measures for reducing the weakness.

Of course, it would be practically important, especially for applications where the space is very limited, to find another masking method that will not require the recomputation and storage of S-boxes for every new run of AES. To this end, one has to use group or, more generally, quasigroup operations for masking the whole range of possible byte values which would at least simplify the secure computation and/or storage of F' according to (2), where F is the inversion in GF(256). However, this does not appear to be very likely.

In the next section, we propose an approximate, nonideal solution to the problem which is based on a random embedding of GF(256) into a larger algebraic structure so that the zero value is mapped into a set of values and all the operations remain compatible with GF(256).

5 Embedded Multiplicative Masking

5.1 Overview of Countermeasure

We represent the field GF(256) as the ring of binary polynomials in x modulo an irreducible polynomial $P(x)$ of degree 8. Let $Q(x)$ be another binary irreducible polynomial that is coprime to $P(x)$ and has degree k. Then GF(256) is a subring of the ring $\mathbf{R} = \mathrm{GF}(2)[x]/(PQ)$, which is isomorphic to $\mathrm{GF}(256) \times \mathrm{GF}(2^k)$, with the isomorphism $U \mapsto (U_P, U_Q)$, where the two coordinates are defined as $U_P = U \mod P$ and $U_Q = U \mod Q$.

To repair the multiplicative masking described above, we suggest to use the random mapping $\rho : \mathrm{GF}(256) \to \mathbf{R}$ defined by

$$\rho(U) = U + RP \tag{4}$$

where R is a randomly chosen polynomial of degree less than k (a k-bit word). Our basic idea relies on the fact that the zero in GF(256) is mapped onto 2^k possible values in $\mathbf{R}$ and should hence be more difficult to detect when k increases. Since $\rho(U)_P = U$, ρ only randomizes the second coordinate, so that choosing R of degree k or larger and taking the result modulo PQ will not increase the randomization effect.

Let $F' : \mathbf{R} \to \mathbf{R}$ be a mapping defined by $F'(U) = U^{254}$. Then, because of $(F'(U)_P, F'(U)_Q) = (U_P^{254}, U_Q^{254})$, F' coincides with F on GF(256), and if 7 does not divide k, then U_Q^{254} is an 1-1 function of U_Q, so that F' does not deteriorate the randomization induced by ρ (for $k = 7$, U^{509} will do). The embedded multiplicative masking countermeasure then consists in modifying the data path in Fig. 3 so that the input data $A_{i,j} \oplus X_j$ is mapped through ρ into $\mathbf{R}$, the first multiplication and two additions are taken modulo PQ, and F' is substituted for F. The second multiplication along the data path and all other operations involving the additive and multiplicative masks remain the same, that is, modulo

P. Accordingly, in mathematical terms, the modified method is the same as the original method with respect to the first coordinate, and the only difference is in the introduced randomized second coordinate. Here, the k-bit word R essentially acts as an additional mask. Of course, in a secure implementation the two coordinates should not be computed explicitly.

5.2 Efficient Implementation

As k grows, it quickly becomes difficult to securely implement F' using a look-up table. For $k = 8$, for instance, 2^{20} bits of (ROM) memory space are required, which is unacceptable in many practical situations. As a software alternative, it is possible to evaluate F' using the traditional "square-and-multiply" method, with about 8 squarings and 4 multiplications in $\mathbf{R}$. This solution can be made more efficient by chosing a specific representation for $\mathbf{R}$, as it is shown now.

Recall that the AES standard specifies usage of the polynomial $P_0(x) = 1 + x + x^3 + x^4 + x^8$ to represent GF(256). The idea is to choose a different polynomial that is more suitable for performing the multiplication. In particular, since in GF(2)[x]

$$1 + x^{17} = (1 + x)(1 + x^3 + x^4 + x^5 + x^8)(1 + x + x^2 + x^4 + x^6 + x^7 + x^8), \quad (5)$$

the choice $P(x) = 1+x^3+x^4+x^5+x^8$ instead of $P_0(x)$, and $Q(x) = 1+x+x^2+x^4+x^6+x^7+x^8$, $k = 8$, yields a particularly efficient encoding. The conversion between the coordinates in the two corresponding bases is achieved by applying the linear transformations determined by the 8×8 binary matrices

$$M = \begin{pmatrix} 1&1&0&1&1&0&0&1 \\ 0&0&1&1&0&1&1&1 \\ 0&0&1&1&0&1&0&0 \\ 0&0&1&1&1&0&0&1 \\ 0&0&1&0&0&1&1&1 \\ 0&1&1&0&0&1&1&0 \\ 0&1&0&0&0&1&1&0 \\ 0&0&0&1&1&1&0&0 \end{pmatrix} \quad \text{and} \quad M^{-1} = \begin{pmatrix} 1&1&1&0&0&0&1&1 \\ 0&1&1&1&0&1&0&1 \\ 0&0&0&0&0&1&1&0 \\ 0&1&0&0&1&0&0&0 \\ 0&0&1&0&0&1&1&1 \\ 0&1&1&0&1&1&1&0 \\ 0&0&0&1&1&0&0&1 \\ 0&1&1&1&1&0&0&1 \end{pmatrix} \quad (6)$$

with respect to the LSB-first representation. More precisely, the input and output bytes in Fig. 3 should be multiplied as binary vectors (one-column matrices) by M and M^{-1}, respectively, and the additive mask used should be multiplied by M, because of $M(A_{i,j} \oplus X_j) = MA_{i,j} \oplus MX_j$. Note that the output multiplication by M^{-1} restores the same additive mask. In fact, the explicit output multiplication by M^{-1} can be avoided by incorporating M^{-1} into the affine part of the ByteSub transformation shown in Fig. 2.

Let us look at how multiplication works in $\mathbf{R}_{16} = \mathrm{GF}(2)[x]/(PQ)$ (see also [13]). As all the elements of $\mathbf{R}_{16}$ can be represented as 16-bit words, let U and V be two words representing two elements of $\mathbf{R}_{16}$, with the LSB-first convention. We compute $W = U \otimes V$ in $\mathbf{R}_{16}$ by performing

$W_1, W_2 \leftarrow \text{MULX } U, V$
$W = \ll W_2$
$W \oplus = W_1$
IF $(W_2)_0 = 1$ THEN $W \oplus =$ `FFFF` ELSE $W_1 \oplus =$ `FFFF`

where MULX denotes the polynomial multiplication, $\oplus$ denotes the 16-bit XOR operation, and $\ll$ the 16-bit leftshift operation. The last operation $W_1 \oplus =$ `FFFF` is here only to ensure that the code runs in time independent of the input. Concerning the square operation, let us consider more generally the mapping $s_i : U \mapsto U^{2^i}$ in $\mathbf{R}_{16}$. As $U(x) \mapsto U(x)^{2^i} \mod (1 + x^{17}) = U(x^{2^i \mod 17})$ is simply a permutation of the coefficients of $U(x)$, s_i can easily be implemented in hardware by first permuting the bits of U, considering that the 16th bit of U is set to zero (this operation requires no logical operations), and then XOR-ing the result with `FFFF` if the resulting 16th bit is equal to 1. Hence, computing $s_i(U)$ requires basically one 16-bit XOR operation in hardware. In software, s_i can be evaluated by using a table look-up, also with a complexity of one 16-bit XOR.

The total complexity of evaluating F' can now be estimated as follows. From the decomposition $254 = 2(1 + 2 + 2^2 + 2^3(1 + 2 + 2^2) + 2^6)$, $V = U^{254}$ can be evaluated by using the following sequence of operations :

$V \leftarrow s_1(U)$
$V \leftarrow V \otimes U$
$V \leftarrow s_1(V)$
$V \leftarrow V \otimes U$
$V \leftarrow V \otimes s_3(V)$
$V \leftarrow V \otimes s_6(U)$
$V \leftarrow s_1(V)$

with the total cost of four multiplications and five calls to some s_i. This yields a total complexity of roughly 4 MULX, 17 elementary 16-bit word operations, and between 4 and 9 branching instructions. Besides the fact that our method offers some resistance to DPA, it is much faster than GCD-based algorithms, like the binary GCD of [8] or a variant of [12], which would require at least about 100 16-bit word operations. It is especially interesting for software implementations on 16-bit or 32-bit microprocessors as well as for hardware implementations.

5.3 Security Analysis

We consider a power consumption model based on the Hamming weight, that is, we assume that an attacker has access at any time to the Hamming weight of the registers of the microprocessor through the power curves. The strategy of the attacker consists in averaging the Hamming weight of the registers in order to discriminate between the case $\rho(U) = 0 \mod P$ and the case $\rho(U) \neq 0 \mod P$. The inversion algorithm presented above involves 25 elementary 16-bit word manipulations (5 different 16-bit values per multiplication and 1 per s_i).

$U \neq 0 \mod P$	$U = 0 \mod P$	$U \neq 0 \mod P$	$U = 0 \mod P$
8.01562	8.03137	7.59191	7.78039
7.00973	7.27843	6.51737	6.71373
7.92926	8.59608	8.27146	7.8902
8.06434	8.28235	7.50376	7.67059
7.00063	7.10588	6.49996	6.58824
7.98235	7.85882	8.26128	8.33725
8.00197	8.03137	7.48426	7.56078
6.97292	7.0902	6.48111	6.52549
7.92929	8.59608	8.32587	7.95294
8.01562	8.03137	7.49336	7.70196
6.99863	7.3098	6.49944	6.81569
7.99385	7.95294	8.25166	8.20784
8.00083	8.03137		

Fig. 4. Average Hamming weight of each 16-bit register used in modified inversion.

Software simulation allows us to compute exactly the average Hamming weight of each of the 25 registers, as shown in Fig. 4.

Looking at the difference of average Hamming weights between the two cases, one observes a maximum difference of about 8.5%. This is a convincing empirical argument that the proposed randomization technique is sound with respect to DPA, and we emphasize that the security level increases with k. Furthermore, as the recomputation and storage of S-boxes are not needed, in order to reduce the vulnerability to higher-order DPA one should use as many mutually independent masks as practically feasible, especially in the first and the last few rounds. In particular, the additive masks used in different rounds can be made mutually independent by using two mutually independent additive masks in the upper and lower halves of Fig. 3.

6 Conclusions

Although the proposed embedded multiplicative masking countermeasure may suffice for many applications, a possibly more secure alternative is to use random binary additive masks and accordingly recomputed S-boxes stored in RAM, for each new run of AES. In fact, it is proposed in [10] to recompute only one S-box and use it repeatedly during one execution of AES. In general, if two intermediate variables both satisfy the fundamental hypothesis for DPA and are masked by the same mask, then their mutual correlation can be used to mount a second-order DPA attack similar to the one proposed in [11]. In order to avoid this attack, the input and output masks for an S-box should be mutually independent.

In principle, increasing the number of mutually independent random masks increases the resistance against higher-order DPA as well as against more sophis-

ticated statistical analysis of power consumption curves. If different masks are generated pseudorandomly, then the security has to be examined more carefully.

With respect to the first-order and higher-order DPA, it is critical to protect the first and the last few rounds of AES by random masks, whereas the protection of intermediate rounds may be useful with respect to more sophisticated statistical power analysis. In this regard, it is safer to repeat the masks in intermediate rounds rather than in the first or the last few rounds.

Even if the same recomputed S-box is used throughout the whole AES, the (first-order) DPA attack is still prevented as it targets the individual points of power consumption curves in time. However, such a solution is vulnerable to a relatively simple second-order DPA attack, especially for implementations in which the executions of S-box transformations are well separated in time (e.g., in software or limited-space hardware).

More precisely, one can identify the execution times of any two S-box transformations in the first and/or the last round of AES, and then compare the power consumption curves at the two points when the S-box outputs (or inputs) are computed by using some simple statistic such as the average absolute value or variance of the difference. The attack is enabled by the fact that the output (or input) values of the two S-boxes are masked by the same mask. The corresponding two expanded key bytes are guessed simultaneously in order to compute the two values and the curves are then partitioned according to the bitwise XOR of these values. To increase the security, it is then desirable to randomize the order of S-box computations within a round, with preferably mutually independent randomizations in the first and the last round.

References

1. M.-L. Akkar, R. Bevan, P. Dischamp, and D. Moyart, "Power analysis, what is now possible...," Advances in Cryptology – Asiacrypt 2000, *Lecture Notes in Computer Science*, vol. 1976, pp. 489–502, 2000.
2. M.-L. Akkar and C. Giraud, "An implementation of DES and AES, secure against some attacks," Cryptographic Hardware and Embedded Systems – CHES 2001, *Lecture Notes in Computer Science*, vol. 2162, pp. 309–318, 2001.
3. S. Chari, C. Jutla, J. Rao, and P. Rohatgi, "Towards sound approaches to counteract power-analysis attacks," Advances in Cryptology – CRYPTO '99, *Lecture Notes in Computer Science*, vol. 1666, pp. 398–412, 1999.
4. J.-S. Coron, P. Kocher, and D. Naccache, "Statistics and secret leakage," Financial Cryptography – FC 2000, *Lecture Notes in Computer Science*, vol. 1962, pp. 157–173, 2001.
5. J.-S. Coron and L. Goubin, "On Boolean and arithmetic masking against differential power analysis," Cryptographic Hardware and Embedded Systems – CHES 2000, *Lecture Notes in Computer Science*, vol. 1965, pp. 231–237, 2000.
6. J. Daemen and V. Rijmen, "AES proposal: Rijndael," 1999, available at `http://www.nist.gov/aes/`.
7. L. Goubin and J. Patarin, "DES and differential power analysis: The duplication method," Cryptographic Hardware and Embedded Systems – CHES '99, *Lecture Notes in Computer Science*, vol. 1717, pp. 158–172, 1999.

8. D. E. Knuth, *The Art of Computer Programming*, Vol. 2, Addison-Wesley, Reading, MA, 1973.
9. P. Kocher, J. Jaffe, and B. Jun, "Differential power analysis," Advances in Cryptology – CRYPTO '99, *Lecture Notes in Computer Science*, vol. 1666, pp. 388–397, 1999.
10. T. Messerges, "Securing the AES finalists against power analysis attacks," Fast Software Encryption - FSE 2000, *Lecture Notes in Computer Science*, vol. 1978, pp. 150–164, 2001.
11. T. Messerges, "Using second-order power analysis to attack DPA resistant software," Cryptographic Hardware and Embedded Systems – CHES 2000, *Lecture Notes in Computer Science*, vol. 1965, pp. 238–251, 2000.
12. R. Schroeppel, H. Orman, S. O'Malley, and O. Spatscheck, "Fast key exchange with elliptic curve systems," Advances in Cryptology - CRYPTO '95, *Lecture Notes in Computer Science*, vol. 963, pp. 43–56, 1995.
13. J. H. Silverman, "Fast multiplication in finite fields GF(2^N)," Cryptographic Hardware and Embedded Systems - CHES '99, *Lecture Notes in Computer Science*, vol. 1717, pp. 122–134, 1999.

Keeping Secrets in Hardware: The Microsoft Xbox™ Case Study

Andrew Huang

Massachusetts Institute of Technology, Cambridge MA 02139, USA,
bunnie@alum.mit.edu,
http://www.xenatera.com/bunnie/

Abstract. This paper discusses the hardware foundations of the cryptosystem employed by the Xbox™ video game console from Microsoft. A secret boot block overlay is buried within a system ASIC. This secret boot block decrypts and verifies portions of an external FLASH-type ROM. The presence of the secret boot block is camouflaged by a decoy boot block in the external ROM. The code contained within the secret boot block is transferred to the CPU in the clear over a set of high-speed busses where it can be extracted using simple custom hardware. The paper concludes with recommendations for improving the Xbox security system. One lesson of this study is that the use of a high-performance bus alone is not a sufficient security measure, given the advent of inexpensive rapid prototyping services and affordable high-performance FPGAs.

1 Introduction and Background

Every cryptosystem is based on some kind of secret, such as a key. Regardless of the cipher, the security of a cryptosystem is only as strong as the secrecy of the key. Thus, some of the most startlingly effective attacks on a cryptosystem involve no ciphertext analysis, but instead find flaws in the protocols that manage the keys. Cryptosystems based on symmetric ciphers are particularly vulnerable to protocol attacks, since both the sender and the receiver must be trusted to have a copy of the same secret key. Despite the difficulty of key management in symmetric ciphers, they remain attractive because of their algorithmic simplicity and high throughput when compared to public key ciphers.

Symmetric cipher key management becomes especially problematic when the receiving party is not trusted or is in a position that can be easily compromised. This is where tamper-resistant hardware comes into play; a summary of tamper-resistance guidelines can be found in [6]. Many systems employ tamper-resistant hardware techniques in varying degrees, including the Sandia National Labs' "Stronglink" micromechanical 24-bit lock [2], the Clipper chip [1], IBM's 4758 PCI Cryptographic Coprocessor [3], Cryptographic Smartcards [5] [4], Automatic Teller Machines (ATMs), and now, video game consoles. However, trusting inadequate physical security measures to protect important secrets is risky. [15] and [16] present examples of how some of the aforementioned tamper-resistant systems can be defeated with surprisingly simple and direct methods.

B.S. Kaliski Jr. et al. (Eds.): CHES 2002, LNCS 2523, pp. 213–227, 2003.

In the case of the Xbox™ video game console from Microsoft, the secret being protected is a key and an algorithm for decrypting and verifying a bootloader. This bootloader then decrypts and verifies a kernel image. Both the bootloader and kernel image are contained in an unsecured FLASH ROM. The kernel then verifies the authenticity and integrity of the applications it runs. Thus, a chain of trust is grown, bottom up, from a seed of trust. This seed – the secret key and an algorithm – is planted in a physically secure, secret boot block.

The Xbox architecture results in the deployment of large number of identical devices, all of which contain the same secret information. As the analysis below illustrates, the security of such a system can be readily compromised, even if the secret is protected by tamper-resistant hardware and obscured by algorithmic complexity.

2 Xbox Hardware Cryptosystem Overview

The Xbox crypto protocol presents a strong defense in the face of unsecured FLASH ROM-based modifications. Please refer to figure 1. The Xbox boots from a 512-byte secret boot block that is hard-coded into the southbridge system ASIC (the "MCPX"). This boot block performs the following functions, in order:

- Loads the "jam tables", i.e., initializes the console chipset
- Turns on the processor caches
- Decrypts the kernel bootloader, contained in FLASH ROM
- Verifies that decryption was successful
- Jumps to the decrypted kernel bootloader

From there the kernel bootloader, now decrypted and verified, performs some more system initialization, decrypts a kernel image from FLASH ROM, decompresses and verifies the decrypted image, and enters the kernel. The kernel decryption key is stored within the bootloader image. Note that the secret boot block code is structured so that the bootloader decryption key is never written to main memory, thus defeating an attack that involves eavesdropping on the main memory bus.

The bootloader is encrypted with RC-4 using a 128-bit key. The decryption algorithm and key are stored in the secret boot block and executed by the Pentium CPU; the busses between the secret boot block and the CPU are not encrypted but assumed to be secure due to their high speeds. The decryption of the bootloader image is verified by checking for a 32-bit magic number near the end of the plaintext stream. One with knowledge of the secret key and the magic number can easily create original bootloader images. It is fairly clear from the code structure of the secret boot block that such a simple, unreliable check was employed because there was not enough space for anything else. The magic number check might also confuse efforts to create original bootloader code based on a key obtained without full knowledge of the secret boot block's contents, such as through a personnel leak or brute force. However, a brute force approach to recovering the bootloader is probably out of the question, since distributed.net's

"bovine" effort, running for over 4 years and currently capable of testing over 100 gigakeys/s, is still working on a 64-bit RC-5 cipher at the time of writing [7].

Given this secure boot protocol, modifying the contents of the FLASH ROM alone will stand a very low chance of revealing anything useful about the console[1]. This is compounded by the fact that the FLASH ROM contains a decoy boot block with halfway reasonable looking decryption and initialization code. The algorithm in the decoy boot block is a bastardized RC-4, and of course applying this algorithm on the ROM contents yields nothing but white noise. Further discussion on how the secret boot block was discovered is contained in the next section.

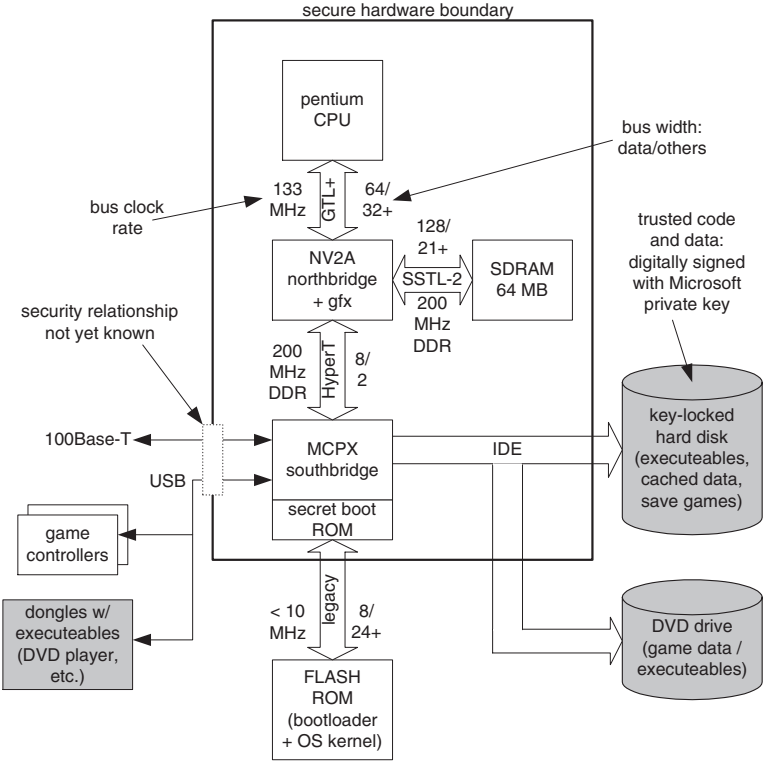


Fig. 1. Overview of the Microsoft Xbox hardware.

[1] An important exception recently discovered is described in section 6

3 Breaking the Physical Security

This section provides a chronology of how the Xbox's physical security was reverse engineered.

Reading out the FLASH ROM contents and tracing the processor's execution starting from the boot vector proved to be futile, as the contents of the boot block in the FLASH ROM were a decoy, cleverly designed to thwart such activity. The code within the FLASH ROM boot block followed the same general flow as the code within the secret boot block; however, the decryption algorithm, the keys and the ciphertext start location were incorrect but believable. The decoy boot block initially resulted in a great deal of confusion but was later explained by the discovery of the secret boot block overlay.

The realization of the existence of a secret boot block happened as a result of the observation that overwriting the processor reset vector in the FLASH ROM has no effect on the Xbox boot sequence. This led to a series of experiments that mapped out the extent of the secret boot block. The block is believed to be 512 bytes in length, situated at the highest location in processor physical memory.

The following approaches were then considered for extracting the secret boot block contents:

- Decapping the MCPX southbridge ASIC
- Using the JTAG boundary scan on the Pentium to step through the "real" boot sequence
- Probing the main SDRAM memory bus for any portions of the boot block that were written to memory
- Probing the processor-northbridge bus using a logic analyzer or custom hardware
- Probing the HyperTransport northbridge-southbridge bus using custom hardware

The direct approach of decapping the MCPX southbridge ASIC was rejected because this ASIC appears to be manufactured in a 0.15μ process with perhaps 6 or 7 metal layers (figure 2). Extracting the bootblock from this ASIC would require a delayering facility and access to an electron microscope. While there are companies, such as Chipworks, that specialize in these kinds of services, it is a difficult, expensive, and time-consuming task.

The JTAG boundary scan approach was rejected on the grounds that the TRST# pin, used to hold the JTAG chain in reset, was tied active in a manner that was difficult to modify without removing the processor. Removal and socketing of the processor was considered to be prohibitively expensive and time consuming; the cost of a BGA socket for the Pentium III is estimated to be in the hundreds to thousands of dollars. In addition, the JTAG boundary scan codes for the Pentium III are largely proprietary and would have to be reverse engineered as well.

SDRAM probing was rejected on the grounds that far too many pins (128 data pins alone) had to be simultaneously probed, and on the grounds that the decryption routine and/or key could be held entirely in processor cache and

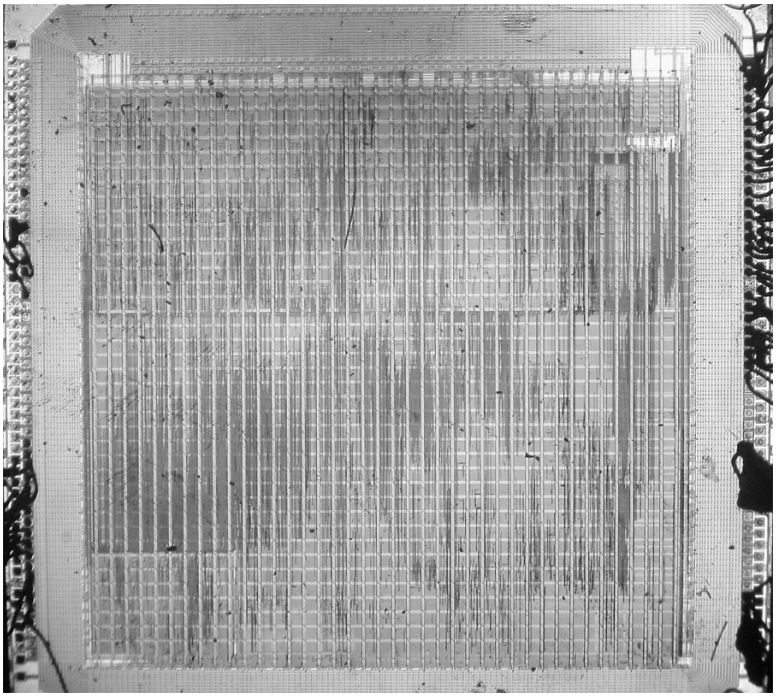


Fig. 2. Die shot of the MCPX Southbridge ASIC.

never written to SDRAM. Also, the cost of solder-on TQFP-100-to-logic-analyzer adapters is prohibitive (around $600 per adapter; four are required). Probing the processor-northbridge bus was rejected for similar reasons: at least 64 data pins had to be probed, and tapping such a large number of GTL+ signals without causing signal integrity issues was thought to be very difficult.

The northbridge-southbridge bus, however, showed promise because of its simplicity. The bus has a low signal count (10 unique) and all the signal traces are laid out on the console's motherboard in a straight flow-through fashion (12-mil center-to-center spacing within a differential pair, 13-mil spacing between differential pairs, see figure 4). In addition, the clock and strobe signals for both the transmit and receive directions are clearly labeled on the motherboard, perhaps for manufacturing debug and test reasons (figure 3). Data on the nVidia nForce chipset [9], a close relative to the Xbox chipset, indicates that the bus uses the HyperTransport (formerly known as Lightning Data Transport (LDT)) protocol. The specifications for the HyperTransport protocol are open and readily available [8].

The primary difficulties in tapping the HyperTransport bus are its high speed (200 MHz DDR) and its use of differential signaling (few logic analyzers come with support for differential signaling). It is interesting to note that Hyper-

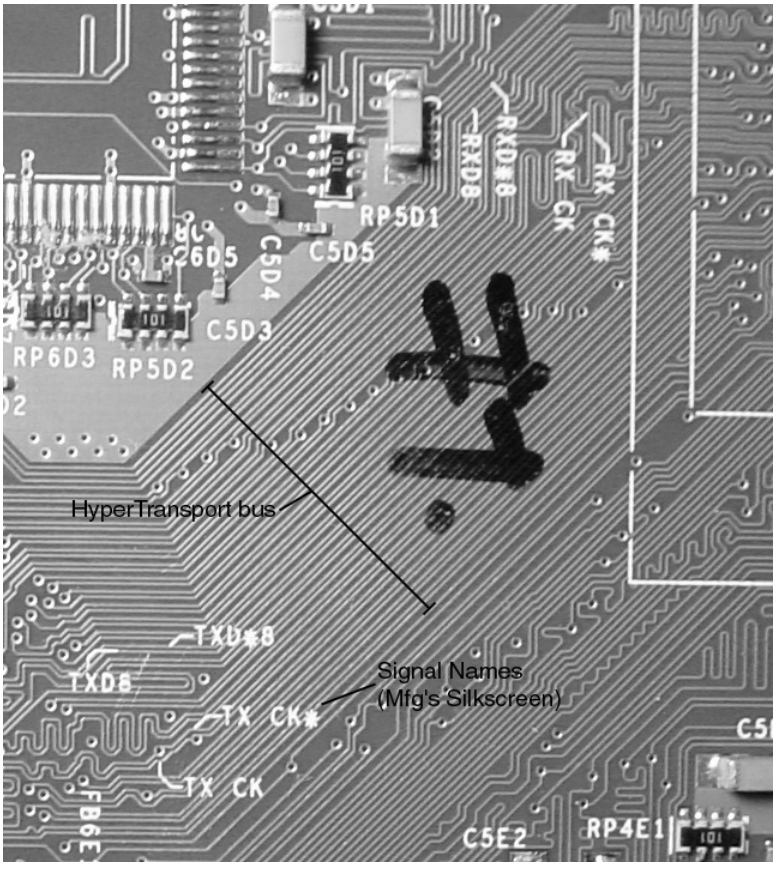


Fig. 3. HyperTransport bus layout showing silkscreen information, perhaps for manufacturing or test purposes.

Transport bus protocol analyzers are commercially available from vendors such as FuturePlus, but they cost upward of $25,000. This price does not include the high-end logic analyzer required to drive the protocol analyzer.

The alternative solution to tapping the northbridge-southbridge HyperTransport bus was to build a relatively cheap, fully custom, differential-to-single-ended "Tap Board", and to connect the output of this board to an FPGA. A Xilinx Virtex-E part was used in this study because it was readily available, as it was used as part of the author's thesis work; however, a better choice would be any of the new Xilinx Virtex-II FPGAs. A suitable Virtex-II FPGA would cost about $50 in single quantities.

The custom Tap Board can be produced with very attainable parts and processes at a reasonable cost. The Tap Board uses a two-layer, 6 mil trace/space, 15 mil hole process from Advanced Circuits, offered at a price of $33 per board in

small quantities. A Texas Instruments SN65LVDS386 LVDS-to-TTL converter was used to turn the differential HyperTransport signals into a single-ended format. It turns out that the HyperTransport physical signaling specification is similar to LVDS, but with a different common-mode offset. The output of the converter drives a cable to the FPGA board. The FPGA is configured to receive the high speed signals with the CTT (Center-Tap Terminated) "Select I/O" option. CTT is chosen because it allows the single-ended TTL drivers to be terminated with a low impedance to 1.5V and still function properly. Note that although Virtex-E FPGAs support LVDS directly, the target FPGA board was not originally designed to support the LVDS configuration.

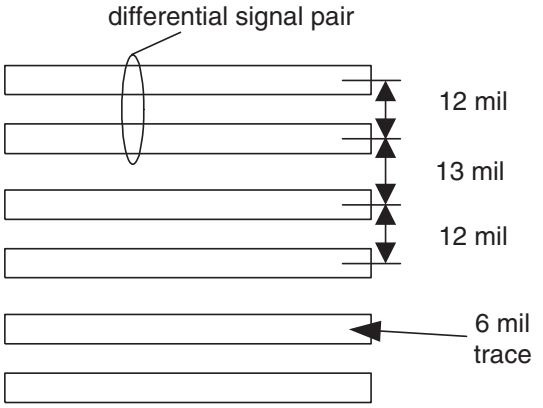


Fig. 4. Dimensions of the HyperTransport signal traces on the motherboard.

The Tap Board has on one edge a pattern of traces with no soldermask that matches the pattern of traces on the Xbox motherboard. The Tap Board was soldered directly to the Xbox's northbridge-southbridge bus. Only the receive-direction Tap Board was mounted for this study. The mating edge was shaped using a belt sander, so that the tapping traces were flush with the edge of the board, and the board could be mounted at a reclined angle to enhance solderability. The soldermask on the Xbox was removed with fine-grit sand paper, and the Tap Board was carefully aligned by hand, and then held roughly in place by soldering a coarse piece of wire between the Tap Board and the motherboard. A hard-setting adhesive, such as Miller-Stephenson Epoxy 907, was applied to fix the angle and mating distance of the Tap board to the motherboard; once the epoxy was cured, the holding wire was removed, and the traces between the Tap Board and the Xbox motherboard were easily soldered using a fine-tip iron and a microscope.

The polarity of the HyperTransport bus signals was determined by probing the idle state of the wires, assuming that their idle state had a value of 0x00.

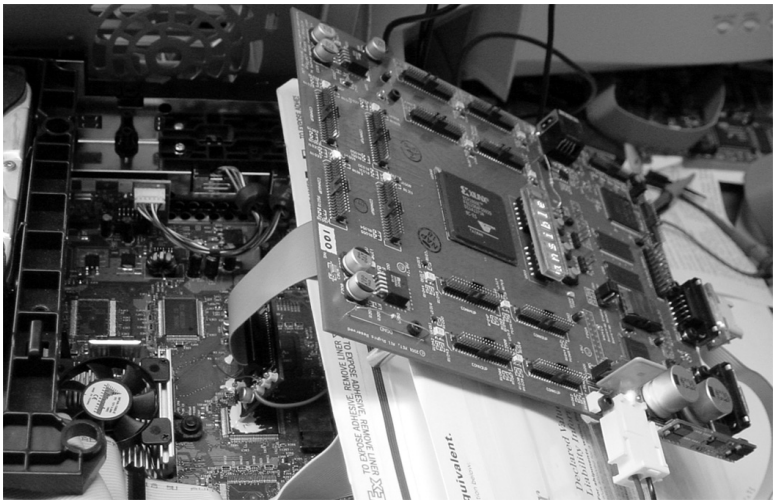


Fig. 5. Tap Board connected to the FPGA board. The FPGA board was originally developed by the author for another work.

Those signals that had the positive and negative pairs swapped relative to the Tap board layout idled to a "1". Signals with inverted polarity were restored to their true value within the trace capture FPGA.

A Xilinx Virtex-E FPGA was used to capture traces of HyperTransport bus activity. It was difficult getting the FPGA to manage the 200 MHz DDR data rates with low skew. However, careful hand-layout of the input registers, post-layout timing simulations at nominal temperature and voltage, and iterations to manually tweak delays and skews eventually centered the clock signal within the data signal on the FPGA's input registers. The retimed data was then demultiplexed to a very manageable 100 MHz single-data rate 32-bit wide bus and written into a bank of FIFOs, along with a sequence count that recorded at what cycle relative to a reset signal the data was captured. Some additional logic was incorporated into the FPGA that discarded idle values (`0x0000 0000`) from the trace FIFOs and formatted the deserialized data relative to the strobe signal, clearly identified on the Xbox motherboard as "RXD8 / RXD*8" (figure 3) in sector 5D (the Xbox motherboard has a coordinate system printed on its periphery).

The reset signal can be determined by probing traces near the HyperTransport bus that behaved like a reset signal. In reality, it is possible that some signal that was not the true reset signal was used to trigger the trace capture, but that is irrelevant as the signal chosen seemed to display a consistent timing relationship with respect to the bus. In fact, the signal used to trigger the trace capture exhibited a 350 ns runt pulse about 67 ms after power-on-reset; this runt

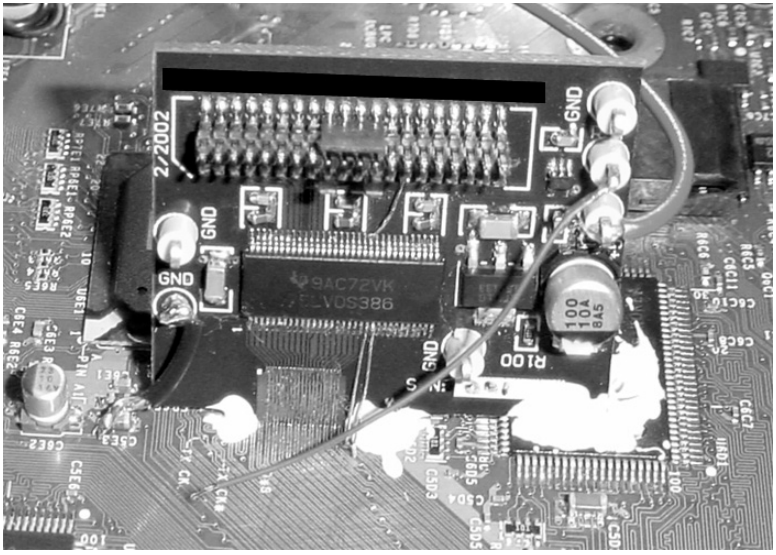


Fig. 6. Close-up of the Tap Board mounted in the Xbox.

pulse was filtered out by a state machine, as it was erroneously restarting the trace capture.

Once traces of data were captured by the FPGA, the order of the bits on the HyperTransport bus relative to the Tap Board layout could be determined. This can be done by correlating known values in the FLASH ROM with data values captured on the HyperTransport bus. A 1's count can be used to identify candidate patterns and data sequences for manual correlation. Fortunately, very early on in the trace several distinctive, sequential values are grabbed from the FLASH ROM: a few values from the lowest address in FLASH ROM, followed by a few values from the boot vector, which happens to be identical between the decoy FLASH ROM contents and the secret boot ROM contents. The order of the traces for the receive-direction bus on the motherboard are believed to be, from the outside to the inside, bit 8 (CTL strobe), 4#, 0#, 7#, 2#, 3#, CLK#, 5, 6#, and 1#. Signals with # after them are inverted with respect to the Tap Board layout.

The raw trace data captured by the FPGA was then dumped to files and manually processed. An example illustrating the format of trace data can be found in figure 7. The sequence number was critical in determining the boundaries of cache traces; blocks of 8 or 16 words are fetched by the processor, even when the caches are off. Trace data was differentiated between secret boot code and FLASH ROM data by searching for the first word of the candidate trace in a dump of the FLASH ROM; if the data could not be found in the FLASH ROM, it was guessed to be secret boot code. Because the processor boots with its caches off, the first roughly 24 million bus cycles contained repeated line fills

of the "jam table" interpreter initialization code, and were ignored as they just performed the wrote initialization of the chipsets. The caches were then turned on by the boot code, and very clear and simple to read blocks of instructions and data were found. These instruction traces were mapped into the secret boot block using the decoy FLASH ROM boot block as a template. The recovered block of code was then disassembled, and the decryption algorithm was determined to be 128-bit RC-4. Since the location of the 128-bit key within the secret boot block was ambiguous (the Tap Board only provides data traces without addresses), a brute-force search was utilized to help isolate the key. A 16-byte sliding "guess key" window over the captured data trace was used as input to an RC-4 decryption engine, and a histogram of the data output was used to determine when the key was found. This information helped resolve some ambiguities in the placement of the data within the secret boot block, and a full picture of the important code within the secret boot block was assembled.

Now that the secret boot procedure is understood, it is possible to encrypt a new ROM for the Xbox console, and to further study the structure of the Xbox bootloader and kernel. Given the RC-4 algorithm, the 128-bit key, and the magic check number at the end of the decrypted segment, one can run original code on the Xbox.

```
00000097 : 664A1D55 ::: E : 000000C6
00000D5C : 05F108F6 ::: F : 01000000
00000DE0 : 2A1A2841 ::: 1 : CC003000
00000E5D : B6FE7F68 ::: E : A0552C01
00000EDA : 5932C662 ::: 1 : 000000FD
00000F57 : F9FBA4C1 ::: E : C7C94000
00000FD4 : F7F9B6AE ::: 1 : 000000C6
00001051 : 73376133 ::: E : 9EC49400
000010CE : FD0127AD ::: 1 : 000000D6
0000114B : 34E8FD29 ::: E : C7C94000
00001245 : 1814A022 ::: 1 : 000000C6
000012C2 : 38EBD672 ::: E : C7C94000
00022526 : C6C0847E ::: 1 : 000000C6
00022527 : A26216BB ::: E : C7C94000
00022528 : 99DA5F80 ::: E : 000000C6
00022529 : 453862E3 ::: 1 : C7C94000
000226D5 : B6DF18C0 ::: E : 000000C6
000226D6 : DA562768 ::: 1 : C7C94000
000226D7 : 0F1D66E3 ::: E : 000000C6
000226D8 : DDC59B59 ::: 1 : 8D42CBCD
```

Fig. 7. An example illustrating the format of trace data captured by the FPGA. Format of the data is "sequence : data ::: aligner : unaligned data".

4 Lessons Learned

A. Kerckhoffs (1835-1903) once stated that the security of a cryptosystem must not depend on keeping the algorithm secret; this is referred to as *Kerckhoffs' Principle* [13]. Another way of stating this is that there is no security through obscurity. In particular, it is an error to assume that a secret, distributed along with the information it guards, is never revealed. For example, the Sega Dreamcast uses a proprietary GD-ROM software format; but, the drive can read CD-ROM disks. The discovery of a back door in the Dreamcast OS allowed executables to be run directly from a standard CD-ROM, thus nullifying the barrier presented by the proprietary GD-ROM format [14]. Other systems that rely on well-hidden secrets, including the Clipper chip [15] and the smartcards used widely throughout Europe to control access to services such as pay-TV, cell phones and gas, have been shown to be surprisingly vulnerable [16]. In the case of the Xbox, the Tap Board and trace capture FPGA design was developed in spare time over the duration of three weeks–including the 5-day turn time for board fabrication–for a total cost of around $50 per board. In other words, if you ship your secrets in your hardware, it is a good assumption that the users will eventually–and perhaps quickly–know your secrets.

Another lesson of this study is that the use of a high-performance bus alone is not a sufficient security measure; the advent of cheap rapid prototyping services and high performance FPGAs allows even poor students to create devices that can tap the bus. However, encrypting a bus introduces its own problems. A secure cipher on a high performance bus significantly impacts latency, power consumption, and reliability. Power consumption is increased because the activity factor for the bus approaches 100%, if the encryption scheme is any good. In this case, the power consumed driving the bus would increase by over an order of magnitude, as the observed activity factor on the northbridge-southbridge bus was well below 10%. Reliability is hurt because a single bit error, even during an idle cycle, can corrupt large blocks of data; with a stream cipher, the corruption would extend until the stream is resynchronized.

A compromise solution to the problem is to simply not embed the secret key in the hardware. In this case, the secret boot block might employ a digital signature protocol, such as Authenticode®, using public key algorithms and one-way hashes [10]. With this scheme, most of the security rests in the secrecy of the private key, and the strength of the public key algorithm and hashes. In order to prevent employee leaks from spreading a private key, a system similar to the BBN SignAssure™ could be used to manage the key so that no human ever has knowledge of the private key.

The hardware implementation of such a scheme would require a larger secret boot block (perhaps a few kilobytes in size) and a small, inexpensive serial EEPROM attached to the southbridge ASIC for storing the encrypted code signatures. In the case of the Xbox, such an serial EEPROM is already provided for the storage of non-volatile parameters such as the time zone setting, user interface language preferences, and hard drive keys. The secure boot procedure would then be implemented as follows:

- Load the jam tables and turn on the processor caches
- Compute a hash of the entire FLASH ROM, including the jam tables
- Retrieve from the serial EEPROM a public-key encrypted hash of the expected FLASH ROM contents
- Decrypt the expected hash
- Compare the expected hash versus the computed hash; if they do not match, halt the machine
- Decrypt the kernel bootloader, contained in FLASH ROM
- Verify that the decryption was successful
- Jump to the decrypted kernel bootloader

The above suggestion does not prevent someone from eavesdropping and obtaining the plaintext of the operating system code, but it does effectively defeat the replacement of FLASH memory with malicious code encrypted against a common key contained in the user's hardware. The public key scheme could be defeated, however, by either replacing the system ASIC or by using very carefully timed pulses to force values onto the HyperTransport bus or the processor's front side bus. As discussed previously, this approach is possible, but difficult; however, the tenacity of an attacker should not be underestimated. For example, a known attack on the Sony Playstation2 console was developed that is rumored[2] to work by dynamically patching its high-performance RAMBUS memory system. The difficulty of a memory patch attack could be increased by using a simple periodic hash and check of the critical code regions in memory.

To be very secure, the secret boot block should be embedded on the same silicon as the processor itself. Embedding the secret boot block in the processor silicon would be less of an issue for Sony or Nintendo, as they develop full-custom or semi-custom processors for their platforms. However, one must keep in sight that the goal on a game console is to simply make it not worth the effort of circumventing its security measures. To this end, eliminating the FLASH memory replacement attack and requiring a more sophisticated attack is possibly sufficient to deter most recreational hackers.

Buffer overrun exploits are also a point of weakness, and they work regardless of the secret boot protocol. An attacker sniffing an insecure bus could obtain the decrypted kernel code and analyze it for weaknesses. However, any machine architecture that employs guarded pointers [11] is much more difficult, if not impossible, to attack using buffer overruns. A fast, efficient guarded pointer scheme with a simple hardware implementation is described in [12]. This scheme can easily be adapted to work in a 64-bit architecture.

The failure of the Microsoft Xbox console security protocol is compounded by the fact that, as a console manufacturer, design-for-test and design-for-manufacturability is paramount. Creating a console with too much security makes it difficult to debug and manufacture. For example, the backside of the Xbox motherboard is populated with test points–including test points for every

[2] It is difficult to obtain any credible published references on the subject of the Playstation2 due to the threat of legal action against researchers and hobbyists under the Digital Millennium Copyright Act (DMCA).

pin on the FLASH ROM. These were originally installed because of the desire to quickly test for faults during manufacturing. The flip side is that one could build a custom "bed-of-nails" tester jig that uses the the FLASH-ROM test points to reprogram Xbox motherboards with any desired code. This method would be fast, inexpensive and solder-free. The lesson here is that even if a manufacturer is very confident about their trust model and security protocols, it must guard against the possibility that they may someday be broken. To this extent, a simple physical security measure, such as a spray-on conformal coating, would severely hamper the re-use of test structures for improper purposes. This of course greatly complicates the repair of hardware failures in the field, but that is a business trade-off the manufacturer must make.

A more radical alternative would be to design the gaming system using proprietary hardware and proprietary media formats, thus limiting the practical impact of any attack on the console. Game consoles are manufactured in very high volumes, so the cost of developing a simple but effective proprietary format can be amortized. The format could then be patented, providing protection against unauthorized use without the need for secrecy. This approach was taken by Nintendo with their Nintendo 64 console [17]. Although patents have a 20 year lifetime, this is an eternity in the video game console industry: the original Nintendo Entertainment System (NES) had its debut in 1985.

5 Future Work

Understanding the secret Xbox boot protocol is just the first step in understanding the Xbox, from here it will be possible to investigate the kernel and bootloader in more detail. It has been determined that the kernel is also encrypted with RC-4/128, and it is also believed to be compressed using LZX compression, a scheme employed by Microsoft's canonical distribution format, the "Cabinet" file. The structure and function of the kernel has been investigated and is now fairly well understood by the community of research enthusiasts that have grown around the Xbox.

One important issue to investigate is the privacy of users who use the Xbox for on-line tasks. It is known, through a parallel effort of the author, that information such as the serial number of the console is stored electronically and is probably accessible to the kernel. What happens to this information when the Xbox is plugged into the internet? Because of the encryption used to secure the Xbox, the nature of the information that is relayed to Microsoft's on-line game servers is unknown. Thus, important future work is to try to determine what the Xbox reveals about the user's identity and personal gaming habits.

6 Addendum

It has recently been called to the author's attention that the hardware initialization procedure of the Xbox contains a significant weakness [18]. Recall from section 2 that the first step in the Xbox boot process is to configure the console's

chipset, commonly referred to as loading the jam tables. A jam table consists of a set of opcode-argument tuples that are executed by a simple interpreter. The types of operations provided by the jam table interpreter include reads and writes to memory, PCI, and IO space, along with conditional jumps and the ability to perform register-indirect addressing. The goal of the jam table interpreter is to provide BIOS developers with a simple tool for orchestrating the complex sequence of dependencies and decisions that have to be made during a PC's initialization procedure.

In the Xbox, the jam table interpreter is located within the secret boot block, but the jam table's contents are located within an unencrypted and unverified region of the FLASH ROM. Thus, once the format of the jamtable opcodes has been deduced, a number of interesting weaknesses can be exploited. The most interesting weakness involves a bug in the Pentium processor implementation. The secure boot procedure is coded such that when the decrypted kernel image does not pass the verification step, the processor is directed to jump to a six-byte code stub in the secret boot block located at `0xFFFF FFFA`. One would expect that when the processor is finished with the stub, an exception that halts the machine should be thrown, as the program counter will overflow its segment by rolling over `0xFFFF FFFF` to `0x0000 0000`. The bug is that instead of halting, the processor happily executes whatever is located at `0x0000 0000`–which corresponds to the beginning of the SDRAM memory. Thus, by attaching a short code stub to the end of the jam table bytecode that writes a jump instruction into `0x0000 0000` and also by presenting any corrupt ciphertext image to the secret boot block, one can gain control of the Xbox without ever decrypting or having any other knowledge of the bootloader or kernel plaintexts [18].

In other words, with plaintext-only modifications in the FLASH ROM, one can entirely bypass the Xbox's security mechanism. One could easily fix this security hole, however, by verifying the jam table's contents prior to bytecode execution with a one-way hash function, or by explicitly coding all initialization functions within the secure boot block. Both of these solutions, however, would require the secure boot block to grow significantly from its current 512-byte size, and neither solution allows easy changes to the initialization procedure in case a bug is found or in case the hardware evolves as a result of cost reduction efforts.

Acknowledgments. The author would like to acknowledge the support of the on-line electronic community. The author would also like to thank the Electronic Frontier Foundation for providing legal counsel. Hal Abelson and Tom Knight also provided invaluable moral support. Finally, the author would like to thank Nikki Justis for all her love and support, and for giving him such an interesting toy for Christmas.

References

1. Federal Information Processing Standards Publication, *FIPS PUB 185: Escrowed Encryption Standard (EES)* http://www.itl.nist.gov/fipspubs/fip185.htm

2. Thomas W. Krygowski, Jeffry J. Sniegowski, M. Steven Rodgers, Stephen Montague, James J. Allen, Jerome F. Jakubczak, Samuel L. Miller, *Infrastructure, Technology and Applications Of Micro-Electro-Mechanical Systems (MEMS)*, Sandia National Laboratories, Intelligent Micromachine Department, http://www.mdl.sandia.gov/Micromachine, also appears in Sensor Expo 1999.
3. IBM, *IBM 4758 PCI Cryptographic Coprocessor*, http://www.ibm.com/security/cryptocards/
4. Gemplus (a smartcard vendor), *Gemplus Corporate Website*, http://www.gemplus.com
5. Pil Joon Lee, Eun Jeong Lee, Yong Duk Kim, *How to Implement Cost-Effective and Secure Public Key Cryptosystems* Proceedings of the First International Workshop on Cryptographic Hardware and Embedded Systems (CHES), August 1999.
6. Federal Information Processing Standards Publication, *FIPS PUB 140-2: Security Requirements for Cryptographic Modules*, http://csrc.nist.gov/publications/fips/fips140-2/fips1402.pdf
7. distributed.net, *distributed.net: Project RC5*, http://www.distributed.net/rc5/
8. HyperTransport Consortium, *HyperTransport^{TM} I/O Link Specification, Version 1.03*, http://www.hypertransport.org
9. nVidia Corporation, *nForce MCP Product Overview, 06.01v1*, http://www.nvidia.com
10. Microsoft Developer Network, *Introduction to Code Signing*, http://msdn.microsoft.com/workshop/security/authcode/intro_authenticode.asp
11. Nicholas P. Carter, Stephen W. Keckler, and William J. Dally, *Hardware support for fast capability-based addressing*, Proceedings of ASPLOS VI, October 1994, pp. 319–27.
12. Jeremy Brown, J.P. Grossman, Andrew Huang, and Thomas F. Knight, Jr., *A capability representation with embedded address and nearly-exact object bounds*, Project Aries Technical Memo 5, http://www.ai.mit.edu/projects/aries/Documents/Memos/ARIES-05.pdf
13. Auguste Kerckhoffs, *La cryptographie militaire*, Journal des sciences militaires, vol. IX, pp. 5–38, Jan. 1883, pp. 161–191, Feb. 1883.
14. Marcus Comstedt, *Dreamcast Programming – Bootable CD-Rs*, http://mc.pp.se/dc/cdr.html.
15. R. Anderson and M. Kuhn, *Tamper Resistance – a Cautionary Note*, Proceedings of the Second Usenix Workshop on Electronic Commerce, pp. 1–11, November 1996.
16. R. Anderson and M. Kuhn, *Low Cost Attacks on Tamper Resistant Devices*, IWSP: International Workshop on Security Protocols, LNCS, 1997.
17. Van Hook, et al., *High Performance Low Cost Video Game System with Coprocessor Providing High Speed Efficient 3D Graphics and Digital Audio Signal Processing*, U.S. Patent 6,239,810, May 29, 2001.
18. Private conversation with visor. visor can be reached by sending a personal message to visor on www.xboxhacker.net

A DPA Attack against the Modular Reduction within a CRT Implementation of RSA

Bert den Boer*, Kerstin Lemke, and Guntram Wicke

T-Systems ISS GmbH
Rabinstr. 8, D-53111 Bonn, Germany
BdenBoer@tpd.tno.nl, {Kerstin.Lemke,Guntram.Wicke}@t-systems.com

Abstract. Published DPA attack scenarios against the RSA implementation exploit the possibility of predicting intermediate data during a straight-forward square-multiply exponentiation algorithm. An implementation of RSA using CRT (Chinese Remainder Theorem) prevents the pre-calculation of intermediate results during the exponentiation algorithm by an attacker. In this paper, we present a DPA attack that uses byte-wise hypotheses on the remainder after the modular reduction with one of the primes. Instead of using random input data this attack uses k series of input data with an equidistant step distance of 1, 256, $(256)^2$, ..., $(256)^k$. The basic assumption of this DPA attack named MRED ("Modular Reduction on Equidistant Data") is that the distance of the input data equals the distance of the intermediate data after the modular reduction at least for a subgroup of single measurements. A function F_k that is composed of the k DPA results is used for the approximation of a multiple of the prime. Finally the gcd gives the prime. The number of DPA calculations increases linear to the number of bytes of the prime to be attacked. MRED is demonstrated using simulated measurement data. The practical efficiency is assessed. If the applicability of this attack is limited due to padding formats in RSA signature applications, the least significant bytes of the remainder after the modular reduction step can still be revealed. Multiplicative message blinding can protect the reduction modulo a secret prime against MRED.

Keywords. DPA, modular reduction, CRT, RSA, power analysis, side channel analysis, smartcard

1 Introduction

For the last years an increased research is focused on vulnerabilities of implementations of cryptographic algorithms. These vulnerabilities in the 'real world' applications are e. g. caused by the deterministic dependencies of the power consumption, electromagnetic radiation and timing characteristics on the processed data. Generally, these attacks don't leave any visible damage to the cryptographic module that can be recognised by the users. Besides that, there are

* now at: TNO TPD, PO Box 155, NL-2600 AD Delft, The Netherlands

B.S. Kaliski Jr. et al. (Eds.): CHES 2002, LNCS 2523, pp. 228–243, 2003.

fault analysis attack scenarios that aim to cause transient or permanent faults during the cryptographic calculation that can be exploited mathematically ([9], [10]). These kinds of attacks yield a new field of attacks on cryptographic algorithms. They are generally summarised as 'Side Channel Cryptanalysis' in contrast to the mathematical cryptographic analysis of the algorithm itself.

Processor smartcards that are applied in security relevant applications (e. g. digital signature) are of a special interest. These products have to guarantee that attack scenarios of 'Side Channel Cryptanalysis' that fall into a category of up to high attack potential are prevented effectively.

Power analysis attacks SPA ("Simple Power Analysis") and DPA ("Differential Power Analysis") were first published by P. Kocher et al. [1] and it turned out that the statistical attack DPA is very effective and can be applied without the knowledge of implementation details. DPA attacks were first used to compromise DES keys during the use of the DES algorithm.

The first power analysis attacks on the RSA algorithm were published by Thomas S. Messerges et al. [2]. Attack scenarios SEMD ("Single Exponent, Multiple Data"), MESD ("Multiple Exponent, Single Data") and ZEMD ("Zero Exponent, Multiple Data") were introduced. The ZEMD attack uses DPA techniques to compromise the bits of the private RSA exponent sucessively. This ZEMD attack is applied on the intermediate results during modular exponentiation. A basic precondition of this attack is that the intermediate data of the modular exponentiation can be predicted offline.

DPA attacks against RSA are classified as 'chosen ciphertext' attacks if applied at the RSA decryption. If the DPA attacks are applied against the RSA signature the attacks belong to the 'chosen plaintext' category.

Due to the effectiveness of power and timing analysis on RSA implementations algorithmic countermeasures are introduced to counteract the predictability of intermediate data. So far, a DPA attack on the CRT implementation was not published. Thus, implementations using CRT may rely on the unpredictability of intermediate data because of the modular reduction step that is carried out with one of the secret primes before the modular exponentiation starts.

2 RSA and the CRT Implementation

In this subsection we recollect the well known CRT algorithm.

The RSA cryptosystem is given by the secret RSA primes p and q, the public modulus N with $N = p\,q$, the public exponent e and the secret exponent d with $e\,d \equiv 1(\mathrm{mod}\ \mathrm{lcm}(\mathrm{p}-1, \mathrm{q}-1))$ as its parameters and the operations for decryption $y = x^d \bmod N$ and encryption $x = y^e \bmod N$.

Widely used techniques to perform the decryption operation are 'square and multiply' algorithms in conjunction with techniques using the Chinese Remainder Theorem (CRT) for known secret primes p and q.

To perform a modular exponentiation $c = a^b \bmod m$ in $\mathbb{Z}_m$, the bitwise representation $b = [b_{n-1}b_{n-2}\cdots b_1 b_0]$ is used. The 'square - multiply' algorithm

evaluates this representation either starting from the least significant bit b_0 (algorithm A1) or from the most significant bit b_{n-1} (algorithm A2).

A1:

```
t := a
c := 1
for k := 0 to n-1 do {
  if b[k]=1 then c := c*t mod m
  t := t*t mod m
}
return c
```

A2:

```
c := 1
for k := n-1 down to 0 do {
  c := c*c mod m
  if b[k]=1 then c := c*a mod m
}
return c
```

To reduce calculation time of a RSA exponentiation with the secret key one can solve a simultaneous system of modular congruencies. The existence of such a solution is ensured by the Chinese Remainder Theorem (CRT). We follow the common practice and denote also an algorithm that solves modular congruencies using the theorem as a CRT algorithm. The special case of RSA requires the representation of the exponentiation in terms of the RSA primes p and q and their recombination to the solution mod$N$. The calculation is then about four times faster than an exponentiation mod$N$.

Fermat's little theorem allows the precalculation of the reduced secret exponent values $d_p = d \bmod (p-1)$ and $d_q = d \bmod (q-1)$. Using A1 or A2 one calculates then $v_1 = x^{d_p} \bmod p$ and $v_2 = x^{d_q} \bmod q$. For the CRT algorithm according to Garner (A3) one precalculated multiplicative inverse $P_q = p^{-1} \bmod q$ is needed:

A3:

```
u := (v2-v1)*Pq mod q
y := v1+u*p
return y
```

Alternatively, the CRT algorithm according to Gauss (A4) uses the two precalculated multiplicative inverses $P_q = p^{-1} \bmod q$ and $Q_p = q^{-1} \bmod p$ and a final reduction modulo N:

A4:

```
y := (v1*q*Qp + v2*p*Pq) mod N
return y
```

Because of memory constraints implementations on smartcards generally prefer the usage of algorithm A2 and A3. Note that during exponentiation a modular reduction modulo a secret value instead of a public one takes place. This is used for the attack described below.

3 DPA against a Non-CRT Implementation

To apply DPA to RSA the attacker should have the possibility to randomly vary the input data x of the RSA implementation to be attacked. Single power consumption measurements $P(x, t)$ of the cryptographic module are typically carried out with a digital oscilloscope and stored on a file server or PC.

3.1 Key Hypotheses

If the RSA implementation uses a straigthtforward 'top-down square-multiply' algorithm (A2 of section 2) the key hypotheses are set up on the next bits of the exponent to proceed. The intermediate results of the exponentiation algorithm proceeding these bits can be determined offline. E. g. in the simplest case the attacker uses only two hypotheses, namely

1. 'the next exponent bit is 0' and
2. 'the next exponent bit is 1'.

In case that the second hypothesis is correct, correlations are present for both key hypotheses as the result of the first hypothesis is an intermediate result of the second hypothesis. The correlations for the correct key hypothesis appear last.

The number of exponent bits used can be optimised under the limitations that taking a bigger number of bits increases the computation time. Generally, it is even more useful to set up the key hypotheses on the sequence of elementary operation (squarings 'Q' and multiplications 'M'). The simplest hypotheses would be

1. 'the next 2 modular multiplication units are composed of 'QM',
2. 'the next 2 modular multiplication units are composed of 'QQ', and
   – in case that the previous correct hypothesis ends up with a 'Q' – additionally
3. 'the next 2 modular multiplication units are composed of 'MQ'.

In general, this set-up of key hypotheses is of interest if we deal with a greater number of key hypotheses. The time window in which correlations are expected can be limited to 1-2 elementary modular multiplication units.

3.2 Selection Functions

Power Analysis is based on the dependency of the power consumption, used by the hardware, on the value of intermediate data. The attacker knows or assumes a model for this dependency. A common model is that the power consumption correlates with the Hamming weight of intermediate data (see [3], [4], [5]).

The selection function has to be calculated on the intermediate result of each key hypothesis that is applied. Intermediate results of the RSA exponentiation are generally of the same bit length as the modulus used. The n-bit bus architecture of the RSA coprocessor used determines the number of bits that are taken into account for the Hamming weight. Common bus widths of cryptographic RSA coprocessors are 32 and 64 bit. It is not necessary that an attacker knows the precise internal bus width. It can be tested using DPA. DPA selection functions $d(x)$ should use the bit-width of the bus architecture to set up functions on the Hamming weight $W(x)$ of intermediate data. A simple selection function $d(x)$ assesses all intermediate data values that have a greater Hamming weight than the n-bit expectation value $E(n) = n/2$ with 1, all values with smaller Hamming weights than $E(n)$ with -1 and to ignore all values that meet the expectation value $E(n)$.

$$d(x) = \begin{cases} -1, \text{ if } W(x) < E(n) \\ 0, \text{ if } W(x) = E(n) \\ +1, \text{ if } W(x) > E(n) \end{cases} \tag{1}$$

Translating towards the values $\{-1, 0, 1\}$ looses information. The selection function $d(x)$ can be refined in the linear model to be

$$d(x) = W(x) - E(n). \tag{2}$$

The easiest selection function is the Hamming weight of intermediate data, or for example the Hamming weight of just a byte of transported data.

3.3 Correlation

DPA identifies the correct key hypothesis by assessing the absolute maximum of the correlation coefficients for each key hypothesis. The correlation is carried out between the result of the selection function $d(x, j)$ on the base of the key hypothesis j and the input data x and the power consumption $P(x, t)$ of the single measurements as a function of x and the time t. The variable t could be narrowed to a small time interval if simple power characteristics of the implementation are obvious. The number i runs through all single measurements.

$$c(t, j) = \frac{\sum_i (d(x_i, j) - \overline{d(x_i, j)})(P(x_i, t) - \overline{P(x_i, t)})}{\sqrt{\sum_i (d(x_i, j) - \overline{d(x_i, j)})^2} \sqrt{\sum_i (P(x_i, t) - \overline{P(x_i, t)})^2}} \tag{3}$$

The correlation coefficient $c(t, j)$ has to be assessed for each key hypothesis j. It will be near zero if there aren't any correlations between the selection function

$d(x, j)$ and $P(x, t)$. In case of a strong correlation $c(t, j)$ approaches 1 at some specific points in time. If there are significant correlation results at a certain key hypothesis j that do not occur at other key hypotheses this is a strong indication of the correct key values.

The formula (3) gives an insight in the notion of (cross)-correlation, but for efficient computation the formula should be reordered.

4 DPA Attack against a CRT Implementation

For performance reasons the CRT implementation is a common choice of a RSA implementation. If the Chinese Remainder Theorem is used intermediate data of the modular exponentiation algorithm are unknown, as the input value of the modular exponentiation algorithm is reduced modulo the primes p and q, respectively. The offline-prediction of intermediate data during the square-multiply algorithm is not possible anymore.

4.1 Basic Idea: Hypotheses on the Remainder

In contrast to the ZEMD that has to correlate on the intermediate results of the modular exponentiation algorithm this DPA attack on the CRT implementation attacks the modular reduction modulo one of the primes performed prior to the CRT exponentiation. It exploits power consumption signals that are caused by the processing and data bus transfers of the residue.

The DPA attack on the CRT implementation uses measurement series with input values of RSA that are equidistant. It assumes that input values can be chosen by the attacker. At the first measurement series a starting value x_0 is chosen and the following input values are generated by decrementing the previous input value by 1. We assume that each series contains m elements. Series are numbered with k. Within the second series the input values have a distance of 256: x_0, $x_0 - 1 \cdot 256$, $x_0 - 2 \cdot 256$, $x_0 - 3 \cdot 256$, ..., $x_0 - m \cdot 256$. Other series follow with stepsize 256^k until the exponent k reaches the size of the prime to be attacked.

There aren't any further restrictions on the input value x_0. We can deal with a purely random, modulus-sized number. For each series k we define the i-th value

$$x_i = x_0 - i \cdot (256)^k \,. \tag{4}$$

The DPA attack on the modular reduction sets up hypotheses on the remainder r after the reduction modulo the prime q. The aim of the first measurement series with the distance 1 of the input values is to compromise the least significant byte of the remainder r_0 that fulfills

$$r_0 \equiv x_0 \bmod q \,. \tag{5}$$

We further define

$$r_i = x_i \bmod q\,. \tag{6}$$

The further measurement series aim to compromise the k-th byte of the remainder r_0, respectively.
For demonstration purposes we focus first on the usage of the measurement series with an equal distance of 1 before we give the general approach.

We assure that all input values x_i of the first measurement series have a remainder with the prime (in the following we assume that the prime q is going to be attacked) that does not equal zero. We therefore exclude the unlikely case of crossing a multiple of q by calculating the greatest common divisor with the public modulus N and all input values x_i of the first measurement series: $gcd(x_i, N)$. In the unlikely case that a multiple of q is crossed the modulus N is directly factorised.

There are 256 hypotheses $H_{j0}(0 \le j \le 255)$ on the value of the least significant byte of r_0

$$H_{j0} \;\text{ is }\; \{r_0 \bmod 256 = j\} \tag{7}$$

that are going to be analysed with DPA.

All values of r_i are related to the value r_0. As the input values x_i are equally distant the difference between r_0 and r_i directly gives the value of the last byte of the remainder for each hypothesis

$$H_{ji} \;\text{ is }\; \{r_i \bmod 256 = (j - i) \bmod 256\}\,. \tag{8}$$

The corresponding value of H_{ji} can be read out from the Table 1.

Table 1. Table of hypotheses H_{ji}

H_{ji}	x_0	x_1	x_2	x_3	x_4	$\cdots$	x_i
H_{0i}	0	255	254	253	252	$\cdots$	$-i \bmod 256$
H_{1i}	1	0	255	254	253	$\cdots$	$(1-i) \bmod 256$
H_{2i}	2	1	0	255	254	$\cdots$	$(2-i) \bmod 256$
.....	...	...	...	...	...	..	.....
H_{255i}	255	254	253	252	251	$\cdots$	$(255-i) \bmod 256$

The correlation is carried out with the Hamming weight $W(x)$ for each hypothesis H_{ji}. The selection function $d(x, j)$ is therefore based on 8 bit (see Table 2).

The strongest results are expected for that value of j where the hypothesis corresponds to the reality. This value is called f_0 from now on. The cyclic property of H_{ji} yields secondary correlation peaks. The second strongest correlations are expected at the hypothesis $H_{j\pm 128}$. The third strongest correlations should be at the two hypotheses $H_{j\pm 64}$. Therefore there are additional indices of the correct hypothesis.

Table 2. Table of the selection functions $d(x_i, j)$ on the base of hypotheses H_{ji} using the 8-bit Hamming weight $W(x)$.

d_{ji}	x_0	x_1	x_2	x_3	x_4	$\cdots$	x_i
d_{0i}	0	8	7	7	6	$\cdots$	$W(H_{0i})$
d_{1i}	1	0	8	7	7	$\cdots$	$W(H_{1i})$
d_{2i}	1	1	0	8	7	$\cdots$	$W(H_{2i})$
....	..	..	..	..	..	..	.....
d_{255i}	8	7	7	6	7	$\cdots$	$W(H_{255i})$

As result of the first measurement series it is found that $(x_0 - f_0) \bmod q$ is divisible by 256.

4.2 The General DPA Attack: MRED

Accordingly to the first measurement series the attack scenarios on the more significant bytes of the remainder r_0 are carried out. k denotes the current byte that is attacked with DPA and the least significant byte of the remainder is referenced with $k = 0$.

The reference base of each measurement series for the successive approximation of r_0 remains x_0. All other bases are calculated by a decrement of $(256)^k$. The 256 hypotheses used for DPA are

$$H_{ji} \text{ is } \left\{ (r_i \bmod (256)^{k+1}) \operatorname{div} (256^k) = (j - i) \bmod 256 \right\}. \tag{9}$$

We define $F_k = r_0 \bmod (256)^k$. With the discovery f_{k-1} of the previous measurement series the function F_k is given by

$$F_k = \sum_{i=0}^{k-1} f_i \cdot (256)^i . \tag{10}$$

The pre-condition for the cyclic DPA attack is that

$$(x_0 - i \cdot (256)^k) \bmod q = r_0 - i \cdot (256)^k . \tag{11}$$

Whether it holds for all m elements of the measurement series depends on the fact whether $r_0 \geq m \cdot (256)^k$. Because of formula (10) this is equivalent to $(r_0 - F_k) \geq m \cdot (256)^k$. If this is not true, then there exists a $w \leq m$ such that $(r_0 - F_k) = w \cdot (256)^k$. This last equality implies that $(x_0 - F_k) = (w \cdot (256)^k) \bmod q$. To test whether such a $w \leq m$ has occurred, it is checked whether $((x_0 - F_k) - i \cdot (256)^k)$ was divisible by q for one of the candidate values $i \leq m$. This testing is done by computing the gcd of all elements of a measurement series with the modulus N and the check whether it is 1:

$$gcd(x_0 - F_k - i \cdot (256)^k, N) \stackrel{!}{=} 1. \tag{12}$$

Otherwise, the modulus N is factorised as a result of the gcd and this is the end criterion of this DPA attack. The check can be optimised by computing the product of the elements modulo N, comparing with 0 at each step and one final gcd calculation.

Then we examine the measurements on $x_0 - i \cdot (256)^k$ with DPA methods to find f_k, which means that $(x_0 \bmod q) \bmod (256)^{(k+1)}$ equals $F_{k+1} = f_0 + f_1 \cdot 256 + \cdots + f_k \cdot (256)^k$.

This procedure of alternating gcd calculation and DPA calculation finds more and more bytes from r_0, starting from the least significant byte. The function $x_0 - F_k$ continously approaches a multiple of the prime q starting from the least significant byte. Along this line the measurements can be done beforehand, while the alternate DPA search and gcd calculation in the end finds q.

The basic assumption for a successful DPA attack is formula (11). This DPA attack that is referred to "MRED" (Modular Reduction using Equidistant Data) is applicable if this equality holds at least for a certain percentage of single measurements.

MRED needs up to q_n measurement series whereas q_n is the byte length of the prime q. Each measurement series typically has to include a few hundred to a few thousand single measurements. The measurements can be taken in advance before the DPA calculations.

5 Results

The results that are expected using this DPA attack on the CRT implementation are demonstrated using simulated measurement data. The generation of these data is based on the power leakage model that the power consumption $P(x,t)$ at a certain point in time t can be split into a power contribution that varies with the Hamming weight of the data x processed, into a power consumption that represents a constant portion and a power consumption that is caused by noise [6][7]. The simulation data are generated using $P(x,t) = P'(x,t) + N(t)$, whereas $P'(x,t)$ is deterministic and depends on the Hamming weight of the data. $N(t)$ simulates a random noise level and should have zero mean. In the linear model $P'(x,t)$ is proportional to the Hamming weight $W(x)$ of the intermediate data according to the expectation value $E(n)$:

$$P(x,t) = (W(x) - E(n)) \cdot \Delta(t) + R(t) + N(t). \tag{13}$$

$\Delta(t)$ is a certain portion of power consumption for each bit transported that does not equal zero in data dependent paths. R(t) is the remaining deterministic part. Noise $N(t)$ can be ignored at statistical attacks [6].

The underlying bus-architecture is chosen to be 8 and 32 bit, respectively. The generation of simulated measurement data gives an output file for each exponentiation that contains the Hamming weight of all intermediate data processed. These output files replace the single measurement data files. The number of bits used for the calculation of the Hamming weight is given by the bus-architecture.

The starting value x_0 was chosen randomly as 128 byte value. The value of prime q was 63 byte long.

The test values used are the following.

q:

```
00 DA 2B AD CF F0 83 45 0E 4D 8F 32 EF 68 3A 57
06 DB E5 2E 15 8B 8F 9F 62 4C 15 D8 91 B9 03 56
B5 FB B8 35 88 5C E9 0B 4E 46 FF ED 68 B9 DC A8
37 5D 92 86 E5 BA B4 3B 98 A7 BE 65 90 BF 84 83
```

x_0:

```
AE 67 0D 33 82 DF 4B 8D EC DE E0 B3 7D 2B FB A2
FD F4 C3 29 1B DB 74 F7 C1 CD B4 FD 63 41 C4 DE
A5 F7 8C 79 21 C4 5A 8B 54 63 9A 41 25 D3 1F 58
4E 82 56 A2 8D E0 1A 50 C2 96 A7 89 3E 07 33 61
0A 7D 99 BC 06 28 83 A5 A6 41 53 F9 CE 14 5D 71
0B 1E D6 5A 83 3D AB 44 ED 0F E0 65 3E 32 88 AF
BD 59 EE AC 85 8B FB DD F7 B8 4C 33 DD 5D A5 FE
A9 98 A9 D9 49 01 59 5B 40 C0 CE 5A 23 78 2A 48
```

r_0:

```
00 09 47 50 DB C7 43 16 75 05 8E 99 E5 2C 92 50
96 D9 CD 3E 81 57 E3 B8 F8 15 47 BB 49 A2 8F 50
27 18 3E BD 86 A3 36 21 5A 42 E8 03 AE 1B 62 27
55 55 9A D9 B7 FF 41 FD 83 4E 33 B2 E5 A2 B5 42
```

The correct value f_0 of the least significant byte of x_0 mod q is in this example $42h$ resp. 66 in decimal representation.

5.1 First Case: 8-Bit Architecture

The DPA calculation reveals the following list of the best 17 candidates (out of 256 candidates) for the correct value of f_0 on the base of 256 single measurements.

Besides to the correct value 66 (decimal notation) secondary positive correlation signals with decreasing amplitudes appear at the relative displacements of ± 128, ± 64, ± 32, ± 16, ± 8, ± 4, ± 2 and ± 1 of the correct hypothesis 66. If the number of single measurements is not a multiple of 256 the correlation coefficients of the secondary positive correlation coefficients differ slightly.

In the Fig. 2 both positive and negative correlation coefficients are taken into account. Negative correlation coefficients occur mainly at small correlation amplitudes. Strong correlation signals are caused by positive correlation coefficients.

5.2 Second Case: 32-Bit Architecture

During generation of the simulation data the 32-bit Hamming weight is used instead of an 8 bit Hamming weight. The DPA correlation is performed on the last 8 bits of the intermediate result after modular reduction.

Hypothesis	Correlation Coefficient	Relative Displacement of f_0
66	+1.000000	0
194	+0.750000	+128
2	+0.625000	-64
130	+0.625000	+64
34	+0.562500	-32
98	+0.562500	+32
50	+0.531250	-16
82	+0.531250	+16
58	+0.515625	-8
74	+0.515625	+8
62	+0.507812	-4
70	+0.507812	+4
64	+0.503906	-2
68	+0.503906	+2
65	+0.501953	-1
67	+0.501953	+1

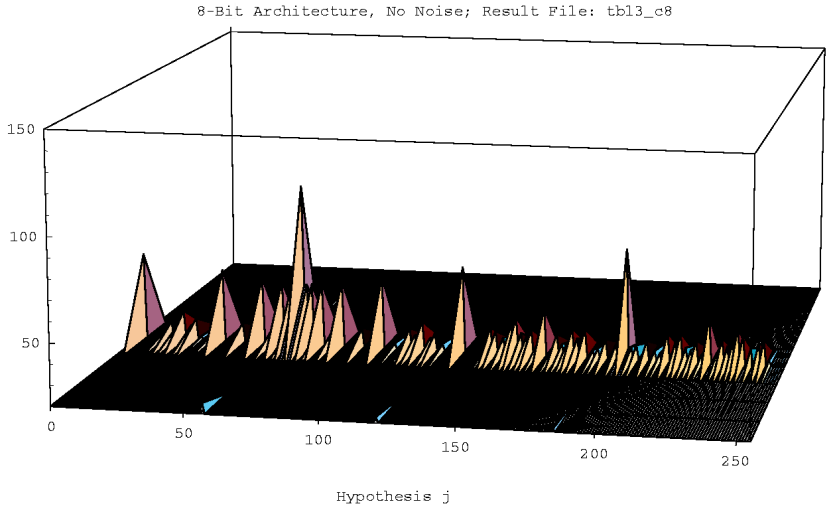


Fig. 1. Graphical representation of the absolute correlation coefficients on the base of 256 single measurements over time. Correlations coefficients $c(j,t) < |0.2|$ are neglected in this trace for clarity reasons.

The correlation coefficient at the correct value f_0 is the most significant and can be easily recognised (Fig. 3). It is more significant as in case of 32-bit random input values. Though the Hamming weight of the measurement series is based on 32 bit the correlation coefficients are nearly as significant as in case of an 8-bit architecture. Because of the equidistant step width only up to two bytes of input

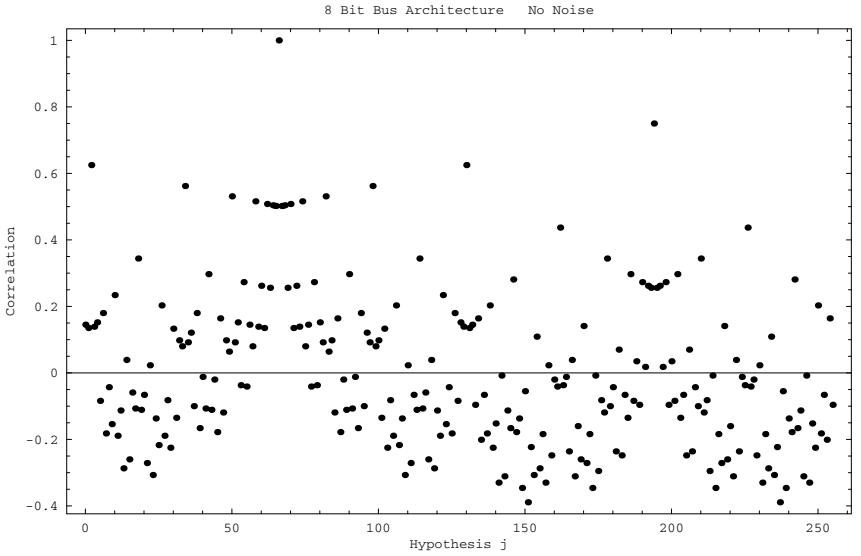


Fig. 2. Graphical representation of the correlation coefficients on the base of 256 single measurements. The smaller correlation amplitudes around $f_0 \pm 128$ of Fig. 1 turned out to be mainly of negative sign.

data and their contribution to the overall 32-bit Hamming weight are affected. A change at the more significant byte occurs at each 256th single measurement only.

It turned out that the DPA attack on the CRT is robust on different hardware architectures.

6 Efficiency of MRED

For the practical estimation of the attack potential we assume that we deal with 1024 bit RSA key size. The CRT implementation shall not include any restrictions on the input data and message blinding schemes that prevent MRED (see section 7). The smaller prime used is assumed to be about 500 bit long. After the first measurement series that serves as profiling step the further measurements series can be limited to a small time frame at the beginning of the CRT calculation that includes the modular reduction step. In general, it is assumed that an attacker needs a few hundred to a few thousand single measurements to prove DPA signals within one measurement series. The gcd check is successful after approximately 60-62 measurement series at the latest. For the overall number of single measurements we would expect $30.000 < n < 300.000$. The measurement itself consumes the most part of the time needed. For a rough estimation the overall measurement time t is expected to be 1 day $< t <$ 3 weeks. It further depends on the performance of the test set-up. The time of a DPA calculation

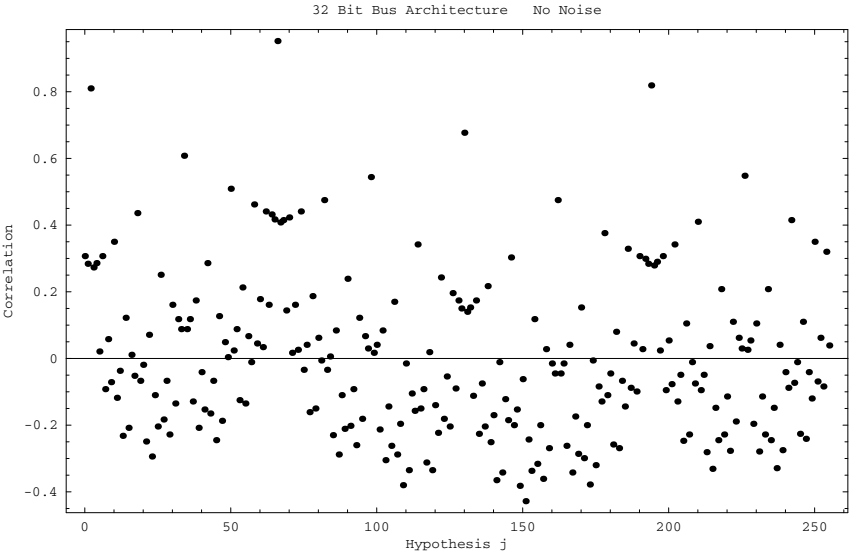


Fig. 3. Graphical representation of the correlation coefficients on the base of 256 single measurements. The correct value is $f_0 = 66$. The characteristics is shifted asymmetrically regarding to Fig. 2 (The correlation coefficient of $j = 2$ is nearly as high as of $j = 194$.)

on a standard PC should be done in the range of minutes depending on the time frame width that is used for the measurement. Additional time is generally necessary for re-synchronisation of single measurements. As it is possible to focus on a small time frame re-synchronisation should be done within minutes to at maximum 1 hour computing time for each measurement series.

MRED is linear to the bit size of the prime that is attacked. E. g. applying this attack to a 2048 bit RSA key will double the overall time.

Table 3. Summary of the Attack Efforts needed for a 1024 bit RSA key

Attack Tasks of MRED	
No. of Measurement Series:	60-62
No. of Single Measurements per Series:	500 - 5000
Single Measurement Data Size:	small
Overall Measurement Time:	1 day to 3 weeks
Overall Re-Synchronisation Time:	few hours to 2 days
No. of DPA calculations:	60-62
Overall DPA calculation time:	few hours to 1 day
Overall Time:	2 days to 1 month

Referring to the Common Criteria scheme [12] the attack potential of MRED is assessed to be in the range of 'Moderate' to 'High' for a 1024 bit RSA key. The assessment depends on the countermeasures of the implementation, the necessary adaptation work of the attacker and the further development of the attack methods.

7 Limitations and Countermeasures

There are three basic assumptions of the MRED attack, namely

1. a sufficient number of single measurements can be collected,
2. the input data x can be varied arbitrarily to construct equidistant input data, and
3. $(x_0 - i \cdot (256)^k) \bmod q$ holds $(r_0 - i \cdot (256)^k)$ at least for a subgroup of single measurements.

The first assumption deals with the number of single measurements that are needed for this DPA attack. As said before, this DPA attack against a 1024 bit RSA key demands for about $30.000 < n < 300.000$ single measurements. The upper boundary of single measurements may conflict with physical constraints of smart cards, e. g. if EEPROM write accesses are involved. Nevertheless, the authors assess that a few ten thousand measurements is a realistic number of exponentiations that can be carried out using a typical smart card. A general countermeasure to prevent these kind of statistical attacks is an usage counter for the number of RSA exponentiations. To secure the RSA decryption an additional failure counter can be implemented if a check of padding formats fails. On the other side an improvement of MRED may reduce the number of exponentiations.

The second assumption affects RSA signing ("chosen plaintext"), but not the RSA decryption ("chosen ciphertext"). The second assumption fails if the attacker has to deal with padding formats in case of digital signature applications. Typical padding formats used limit the range of variable data to the least significant 20 bytes of data that is the outcome of a hashing function. At the presence of padding formats MRED will reveal at maximum the least significant 20 bytes of the remainder of both primes p and q. Nevertheless, this information is of minor use as it doesn't give directly the least significant 20 bytes of the primes p and q, but of an unknown multiple of p and q, respectively. A possible way to proceed is a DPA attack that aims to find data correlations on the revealed bytes of the remainder during the following 'square - multiply' algorithm using the exponents d_p and d_q. If data correlations can be proven at the multiplications this leads to the disclosure of the exponents d_p and d_q. The occurence of these DPA signals during the exponentiation can be prevented by the common message blinding schemes.

The third assumption of MRED can be destroyed by message blinding. Multiplicative message blinding scheme as e. g. proposed by [11] use pairs (ν_i, ν_k) that are used for the blinding of the input data and unblinding of the result. This multiplicative blinding is applicable to prevent the likeliness of the third

assumption. Nevertheless, MRED might be useful to detect possible weaknesses within the message blinding scheme.

8 Conclusion

In this paper, we developed a new DPA attack on the remainder that can be applied at a CRT implementation of RSA to compromise one of the secret RSA primes. The basic assumption for MRED is that $(x_0 - i \cdot (256)^k) \bmod q$ holds $(r_0 - i \cdot (256)^k)$ at least for a subgroup of single measurements. The results of this MRED attack are shown on the base of simulated measurement data. Countermeasures against MRED should include the use of multiplicative blinding schemes to protect the reduction modulo a secret prime.

Acknowledgements. The authors would like to thank Robert Hammelrath and the team for the support of the work including the use of the DPA core routines and the test methods, as well as the referees for their valuable comments.

References

1. P. Kocher, J. Jaffe and B. Jun, "Differential Power Analysis", in: Proceedings of Advances in Cryptology – CRYPTO '99, Lecture Notes in Computer Science, Vol. 1666, Springer, Berlin 1999, pp. 388–397
2. T. S. Messerges, E. A. Dabbish and R. H. Sloan, "Power Analysis Attacks of Modular Exponentiation in Smartcards", in: Proceedings of Workshop on Cryptographic Hardware and Embedded Systems, Springer, Lecture Notes in Computer Science, Vol. 1717, Springer, Berlin 1999, pp. 144–157
3. J. Kelsey, B. Schneier, D. Wagner, C. Hall, "Side Channel Cryptanalysis of Product Ciphers", in: Computer Security – ESORICS 98, Lecture Notes in Computer Science, Vol. 1485, Springer, Berlin 1998, pp. 487–496
4. T. S. Messerges, E. A. Dabbish, R. H. Sloan, "Investigations of Power Analysis Attacks on Smartcards", USENIX Workshop on Smartcard Technology, USENIX Association, 1999, pp. 151–161
5. J.-S. Coron, P. Kocher, D. Naccache, "Statistics and Secret Leakage", in: Financial Cryptography 2000, Lecture Notes in Computer Science, Vol. 1962, Springer, Berlin 2001, pp. 157–173
6. T. S. Messerges, "Using Second-Order Power Analysis to Attack DPA Resistant Software", in: Proceedings of Workshop on Cryptographic Hardware and Embedded Systems, Lecture Notes in Computer Science, Vol. 1965, Springer, Berlin 2000, pp. 238–251
7. R. Mayer-Sommer, "Smartly Analyzing the Simplicity nd the Power of Simple Power Analysis on Smartcards", in: Proceedings of Workshop on Cryptographic Hardware and Embedded Systems, Lecture Notes in Computer Science, Vol. 1965, Springer, Berlin 2000, pp. 78–92
8. P. N. Fahn and P. K. Pearson, "IPA: A New Class of Power Attacks", in Proceedings of Workshop on Cryptographic Hardware and Embedded Systems, Lecture Notes in Computer Science, Vol. 1717, Springer, Berlin 1999, pp. 158–172

9. D. Boneh, R. A. DeMillo, R. J. Lipton, "On the Importance of Checking Cryptographic Protocols for Faults", in Advances in Cryptology – Eurocrypt 97, Lecture Notes in Computer Science, Vol. 1233, Springer, Berlin 1997, pp. 37–51
10. E. Biham and A. Shamir, "Differential Fault Analysis of Secret Key Cryptosystems", in: Advances in Cryptology – Crypto '97, Lecture Notes in Computer Science, Vol. 1294, Springer, Berlin 1997, pp. 513–525
11. P. Kocher, "Timing Attacks on Implementations of Diffie-Hellman, RSA, DSS, and Systems", in: Advances in Cryptology – Crypto '96, Lecture Notes in Computer Science, Vol. 1109, Springer, Berlin 1996, pp. 104–113
12. Common Criteria – Common Methodology for Information Technology Security Evaluation, CEM 99/045, Version 1.0, August 1999
13. A. J. Menezes, P. C. van Oorschot, S. C. Vanstone, Handbook of Applied Cryptography, CRC Press, Boca Raton 1997

Further Results and Considerations on Side Channel Attacks on RSA

Vlastimil Klíma[1] and Tomáš Rosa[1,2]

[1] ICZ, Prague, Czech Republic
[2] Dept. of Computer Science and Eng., FEE, Czech Technical University in Prague
{vlastimil.klima, tomas.rosa}@i.cz

Abstract. This paper contains three parts. In the first part we present a new side channel attack on a plaintext encrypted by EME-OAEP PKCS#1 v.2.1. In contrast with Manger's attack, we attack that part of the plaintext, which is shielded by the OAEP method. In the second part we show that Bleichenbacher's and Manger's attack on the RSA encryption scheme PKCS#1 v.1.5 and EME-OAEP PKCS#1 v.2.1 can be converted to an attack on the RSA signature scheme with any message encoding (not only PKCS). In the third part we deploy a general idea of fault-based attacks on the RSA-KEM scheme and present two particular attacks as the examples. The result is the private key instead of the plaintext as with attacks on PKCS#1 v.1.5 and v.2.1. These attacks should highlight the fact that the RSA-KEM scheme is not an entirely universal solution to problems of RSAES-OAEP implementation and that even here the manner of implementation is significant.

1 Introduction

In 1998, Bleichenbacher [5] described an attack on the PKCS#1 v.1.5 encoding and in 2001 Manger [15] described an attack on the improved scheme EME-OAEP PKCS#1 v.2.1, called also RSAES-OAEP. These attacks underline the significance of the theorem of RSA individual bits [13] which states that: *If RSA cannot be broken in a random polynomial time, then it is not possible to predict the value of any selected bit of the plaintext with a probability not negligibly different from 1/2*. A negligible difference for the purpose of this theorem is such $\varepsilon(n)$ that for any constant $c > 0$ it holds that $\varepsilon(n) < L(n)^{-c}$, where $L(n)$ is the length of an appropriate sufficiently large RSA modulus n. From the standpoint of side channels it is important to understand this theorem as saying: *If the value of any chosen bit of the plaintext can be predicted with a probability not negligibly different from 1/2 then RSA can be broken within a random polynomial time*. Breaking RSA [21] is understood here to mean that a value of the plaintext is obtained. Bleichenbacher's and Manger's attacks use side channels which provide the attacker with a relatively large amount of information about the plaintext (at least that the two most significant bytes are 00 02 or the first one is 00).

B.S. Kaliski Jr. et al. (Eds.): CHES 2002, LNCS 2523, pp. 244–259, 2003.

In this paper plaintext will always mean a value of m which is created immediately after an operation with a private RSA key, $m = c^d \bmod n$, not the value of M obtained after decoding m.

In Section 2 we present another possible attack on the RSAES-OAEP (PKCS#1 v.2.1) scheme. It is a chosen ciphertext based side channel attack using only the side information about Hamming weight of certain 32-bit words produced in the process of decoding m by the EME-OAEP-DECODE procedure according to PKCS#1 v.2.1. Theoretically, it is a weakening of the assumptions of Manger's and Bleichenbacher's attacks. From the practical point of view, the new attack can be used especially on smart cards. It follows from the theorem of RSA individual bits that it is necessary to prevent the leakage of any information about the individual bits of the plaintext. Our attack demonstrates that the Hamming weight of a part of the plaintext can be used to carry out a successful attack.

In Section 3 we present a very simple but efficient conversion of the Manger/Bleichenbacher breaking oracle to a universal (signature) oracle. The principle that a private RSA key should not be used simultaneously for encryption and for digital signature is well known but is very often violated in practice. Typical examples include some of the current implementations of Public Key Infrastructure (PKI), the SSL protocol etc. We show that if we can perform Bleichenbacher's or Manger's attack on the encryption scheme using PKCS#1 (v.1.5 or v.2.1) in such way that we can obtain the plaintext then we can also obtain the digital signature of any message (encoded in any way) using the same private RSA key. In the SSL protocol this means the ability to create signatures with the server-side private key and even create false servers with the identity of the original server, provided that sufficient decrypting speed can be ensured.

In Section 4 we present a new fault side channel attack on the RSA-KEM. RSA-KEM attempted to remove the structural relations in order to prevent leaking of information about the plaintext. Despite this we discovered a natural method of obtaining such information. Input plaintext for RSA-KEM consists of symmetric encryption keys, information about which can be obtained by means of an integrity check of the messages they encrypt (e.g. checking the PKCS#5 [18] padding). The result produced by the attack that uses this information is a private RSA key whilst the attacks on PKCS#1 v.1.5 and 2.1 always discovered only a plaintext.

2 Side Channel Attack on RSAES-OAEP Plaintext

In this section we will demonstrate a new method of attacking the RSAES-OAEP scheme (PKCS#1 v.2.1 [17]) at the time when decoding operation EME-OAEP-DECODE(EM, P) is performed, see Fig. 1. The attack is based on the assumption that there is a side channel carrying some information about the plaintext. In particular we assume that the attacker can obtain the Hamming weight $w(x)$ (i.e. the number of '1' bits) of a word x during the time when the plaintext m is being processed in the MGF operation (to be specified later). As it was shown in [16], this assumption is realistic

for instance in power side channels which tend to leak this information in a relatively readable way. We note that this attack is possible with some modifications even when we have access to the Hamming distance of processed data rather than the Hamming weight.

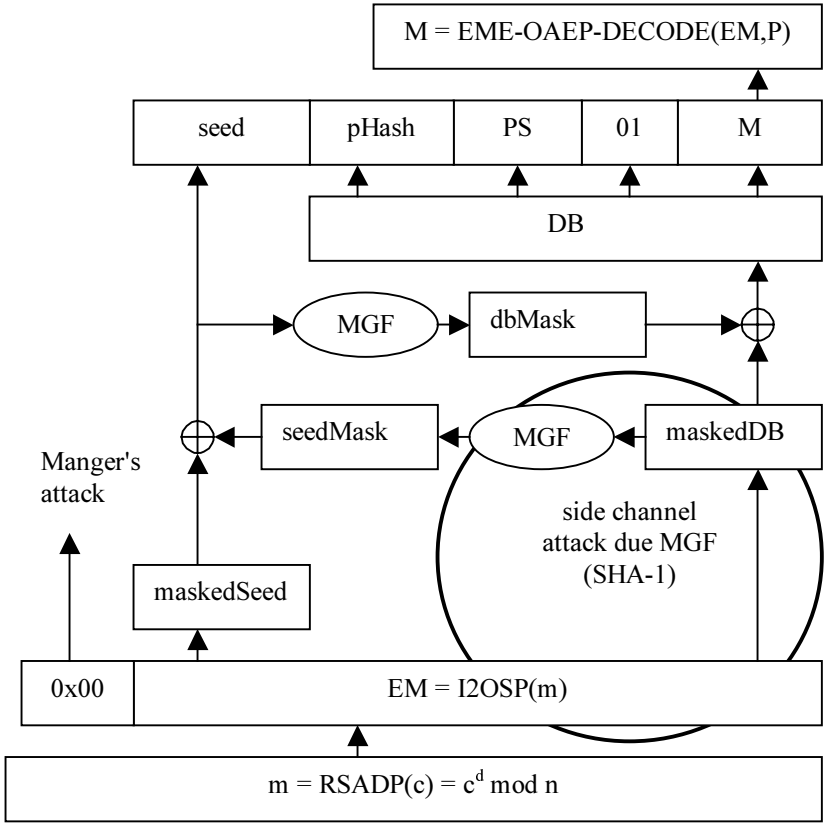


Fig. 1. New side channel attack against RSAES-OAEP

2.1 Attack Description

Consider RSA with a modulus n which has the length of L(n) bits where L(n) is the multiple of 512, i.e. L(n) = 512*k, where k is a natural number. The attack will target the RSAES-OAEP scheme during the processing of the plaintext immediately after the RSA decryption operation c^d mod n, see Fig. 1. *SeedMask* will be computed according to [17] as *seedMask* = MGF(*maskedDB*,20) = SHA-1(*maskedDB* ‖ 00 00 00 00), where the four zero bytes (we will write constants mostly in the hex. notation) are appended to the message by the MGF function. It follows from the definition of OAEP encoding that *maskedDB* always contains 64*k-1-20 bytes, so that 64*k-17

bytes (4 extra zero bytes) enter SHA-1. By the definition of SHA-1 [22] the message is divided into blocks of 64 bytes, which are processed sequentially by the compression function. Note that the least significant bit of the original message m is processed in the last block. It is followed by four zero bytes and 17 bytes of the SHA-1 padding. For various values of L(n) the particular value of the padding is different, but it is a constant known to the attacker. To present an example, we will consider n, such that L(n) = 1024. Let us denote the i-th byte of the plaintext as $m[i]$ where $m[0]$ is the least significant byte. The last block entering the SHA-1 compression function is in this case equal to m[42.....0] 00 || 00 00 00 80 || 00 00 00 00 || 00 00 00 00 || 00 00 00 00 || 00 00 03 78, where m is followed by 4 zero bytes (from MGF) and the SHA-1 padding. The padding consists of bit 1, 71 zero bits and a 64-bit representation of the message bit length. The length is 888_{10} = 0x00000000 00000378 bits in this case (64*2–17 = 111_{10} bytes). The SHA-1 compression function fills this last block into 32-bit variables W_0, ..., W_{15}, where W_8 = m[10] m[9] m[8] m[7], W_9 = m[6] m[5] m[4] m[3], W_{10} = m[2] m[1] m[0] 00, W_{11} = 00 00 00 80, W_{12} = 00 00 00 00, W_{13} = 00 00 00 00, W_{14} = 00 00 00 00, W_{15} = 00 00 03 78. And then expansion to words W_{16} , ..., W_{79} is performed according to the following relations (where S^1 denotes the left cyclic shift by one bit) $W_{16} = S^1(W_{13}$ xor W_8 xor W_2 xor $W_0)$, $W_{17} = S^1(W_{14}$ xor W_9 xor W_3 xor $W_1)$, $W_{18} = S^1(W_{15}$ xor W_{10} xor W_4 xor $W_2)$, etc. When calculating W_{16}, the first operation performed is W_{13} xor W_8, where W_{13} is a known constant. This moment is an example of a general situation when D-1 known parameters and one unknown enter a D-ary operation. Here various side channels are often applicable, especially the power side channel.

We assume that the attacker is able to gather the Hamming weight $w(W_8) \in \{0, ..., 32\}$ of word W_8 during the W_{13} xor W_8 operation (W_8 is the only unknown operand in it). The same situation arises in the following two operations as well, so we are able to gather $w(W_9)$ and $w(W_{10})$.

We number the bits of the word W_i (from the msb to the lsb) as $W_{i,31}$ $W_{i,30}$ $W_{i,29}$ ... $W_{i,0}$. We will show that now we can predict the value of $W_{10,8}$ with a probability not negligibly different from 1/2. Note that this is the value of the least significant bit (lsb) of the plaintext m. Hence, using the theorem of RSA individual bits [13] we can design an attack on the entire plaintext. It is widely known that information about the lsb of the plaintext leads to very efficient attacks [25, p.144].

2.2 Obtaining the Least Significant Bit of a Plaintext (Building an lsb-Oracle)

The procedure which leads to obtaining the value of $W_{10,8}$ is as follows. We denote the ciphertext to be attacked as c, the modulus as n and the public RSA exponent as e. First we let the attacked device decrypt and decode the original ciphertext c. During decoding we gather the values of Hamming weights $w(W_8)$, $w(W_9)$ and $w(W_{10})$. In the next step we request the equipment to decrypt and decode a value $c' = c*2^e \bmod n$. Plaintext m' is the result of this and during the calculation we will obtain Hamming weights $w(W_8')$, $w(W_9')$ and $w(W_{10}')$. If the bit $W_{10,8}$ is zero, then the decryption returns the value $m' = m >> 1$, where ">> 1" means a shift one bit to the right. Otherwise m' =

$(m + n) >> 1$. If we assume $W_{10,8} = 0$ then (W_8', W_9', W_{10}') will be created of (W_8, W_9, W_{10}) by a shift one bit to the right (with the exception of W_{10}, where the shift only affects the leftmost bits which are then independently complemented by eight zero bits). The difference between appropriate Hamming weights $w(W_8)$, $w(W_9)$, $w(W_{10})$ and $w(W_8')$, $w(W_9')$, $w(W_{10}')$ is therefore 0 or 1. More precisely $w(W_8') = w(W_8) - W_{8,0} + W_{7,0}$, $w(W_9') = w(W_9) - W_{9,0} + W_{8,0}$, $w(W_{10}') = w(W_{10}) - W_{10,8} + W_{9,0} = w(W_{10}) + W_{9,0}$ and therefore the three relations included in exactly one of the eight rows of Table 1 are valid.

Table 1. Possible relations among random variables W and W' when $W_{10,8} = 0$

$W_{9,0}$	$W_{8,0}$	$W_{7,0}$	Possible relations		
0	0	0	$w(W_{10}') = w(W_{10})$	$w(W_9') = w(W_9)$	$w(W_8') = w(W_8)$
0	0	1	$w(W_{10}') = w(W_{10})$	$w(W_9') = w(W_9)$	$w(W_8') = w(W_8) +1$
0	1	0	$w(W_{10}') = w(W_{10})$	$w(W_9') = w(W_9) +1$	$w(W_8') = w(W_8) -1$
0	1	1	$w(W_{10}') = w(W_{10})$	$w(W_9') = w(W_9) +1$	$w(W_8') = w(W_8)$
1	0	0	$w(W_{10}') = w(W_{10}) +1$	$w(W_9') = w(W_9) -1$	$w(W_8') = w(W_8)$
1	0	1	$w(W_{10}') = w(W_{10}) +1$	$w(W_9') = w(W_9) -1$	$w(W_8') = w(W_8) +1$
1	1	0	$w(W_{10}') = w(W_{10}) +1$	$w(W_9') = w(W_9)$	$w(W_8') = w(W_8) -1$
1	1	1	$w(W_{10}') = w(W_{10}) +1$	$w(W_9') = w(W_9)$	$w(W_8') = w(W_8)$

However, if $W_{10,8} = 1$, m' is not created by a shift of m, but produced as $(m + n) >> 1$. This, with a high probability, destroys the linear relations in the Table 1. By the obtained weights ($w(W_8)$, $w(W_9)$, $w(W_{10})$) and ($w(W_8')$, $w(W_9')$, $w(W_{10}')$) we determine whether they fit all relations in any single row. If so, we adopt a hypothesis that $W_{10,8} = 0$, otherwise we refuse it and assume that $W_{10,8} = 1$. The probability of establishing the bit $W_{10,8}$ correctly is close to 1 for an ideal side channel. It will be sufficient to realize that m is randomized by a hash function in MGF and n is assumed to be common, not specially constructed. Therefore, the probability of adopting the hypothesis that $W_{10,8} = 0$ if it was $W_{10,8} = 1$, can be estimated as the probability that the random variables W_8, W_9, W_{10} and W_8', W_9', W_{10}' (with the properties that lower nine bits of W_{10} are 100000000_2 and lower eight bits of W_{10}' are 00000000_2) will fit any of the relations in Table 1, which is approximately 0.008. That enables us to obtain the least significant bit of the plaintext m with a high probability and therefore, in accordance with [13] we can establish the remaining part of m.

For the demonstration purpose the procedures in [13] can be used directly, in particular we suggest the methods based on computing gcd (for details see [2]). However some improvements of these procedures are necessary when planning a real practical attack (mainly with respect to a minimization of oracle calls, because some devices may limit the total amount of RSA decryptions). First we need to compute our oracle's advantage, which we define in the following way: Let the $lsb(m)$ be the least significant bit of the plaintext m corresponding the ciphertext c and let the $O_{lsb}(c)$ be the oracle's estimate of $lsb(m)$. We assume that the oracle works according to the procedure described above. The advantage adv is defined as $adv = |P[lsb(m) = O_{lsb}(c)] - 1/2|$, where the probability of correct estimation, $P[lsb(m) = O_{lsb}(c)]$, is computed over the

probability space of all possible ciphertexts and all possible oracle internal coin tosses. From [13] we have that the *adv* must be at least non-negligible (c.f. above). The higher advantage the better oracle we have. Of course, better oracle leads to a more efficient attack. For instance, if we have an oracle with *adv* = 1/2, then we can use well known and rather quick methods, needing approximately $O(\mathrm{L}(n))$ oracle calls (c.f. for example [25, p.144]).

If *adv* < 1/2, we have to employ some methods, which are equipped with a built-in error correction. In fact, these methods must have been already employed in the proofs of theorems in [2,13]. But these proofs have rather existential form, which is not suitable for a practical attack. However there are stronger proofs developed in [9] and improved later in [10], which can be used to mount practically feasible attacks. In particular we suggest to use the *RSA inversion* algorithm ([10, p.226]), which describes a randomized algorithm for the RSA decryption, which needs approximately $O(\mathrm{L}(n)^2 adv^{-2})$ oracle calls ([10, p.223]).

Note that using the absolute value for *adv* (c.f. definition above) is possible here since there is no dependence between previous oracle responses and further oracle calls in the *RSA inversion* algorithm. Therefore we can run this algorithm (in particular parts 2. and 3. – c.f. [10, p.223]) twice, once for $\mathrm{O}_{\mathrm{lsb}}(c)$, once for $\mathrm{neg}(\mathrm{O}_{\mathrm{lsb}}(c))$, where we use simple inversion of the responses captured in the previous run. Such a method induces only a constant multiplicative slow down in the computational part of the algorithm, without an increase of the number of total oracle calls. On the other hand this method allows to exploit any correlation between oracle response and the correct value of lsb(*m*). This further relaxes requirements on the quality of particular side channel used in this attack.

There are other questions, which have to be carefully answered when developing an efficient attack – namely on how to measure Hamming weights, whether to do some error corrections during a measurement phase or whether to let it all on a majority decision used in the *RSA inversion* algorithm, etc. In this paper we strive to show that such an attack is possible and that it operates in a random polynomial time, having in mind that its concrete efficiency strongly depends on a particular implementation. From here we would like to emphasize the importance of a thorough implementation, which cannot simply be reduced to the problem of finding "the right encoding method" as was perhaps deemed earlier.

3 Note on Converting the Deciphering Oracle to a Signing Oracle

In this section we will demonstrate that if the attacker can use Bleichenbacher's or Manger's attack on the PKCS#1 v.1.5 or 2.1 encryption scheme, he/she is also able to create false signatures using the same private RSA key with any encoding of the message to be signed. This conversion is technically very simple but it has interesting practical consequences on the applications where the same key is used both for encryption and for digital signature. One example is the SSL/TLS protocol used to secure access to web servers. In its application the public key certificate at the server

sometimes permits the use of the key both for encryption and for signature. That means that a signature made by the server's private key is meaningful in the PKI system and it is not appropriate that it should be forgeable. Conversion will be demonstrated for both Bleichenbacher's attack on PKCS#1 v.1.5 and for Manger's attack on PKCS#1 v.2.1. Manger's attack uses only one element of the EME-OAEP PKCS#1 v.2.1 encoding - whether a zero occurred in the most significant byte (MSB) of the plaintext decrypted by the private key. We will denote the oracle which tells the attacker this as "*Partial information oracle*" PIO_{MSB}: $PIO_{MSB}(c)$ = "yes" iff $c = m^e \bmod n$, $MSB(m) = 0x00$. Using this oracle a decryption machine (*Whole information oracle*) WIO_{MSB} is constructed in [15]. If the plaintext has a format of $m = 00 \| ....$, then the WIO_{MSB} (using PIO_{MSB}) can extract from the ciphertext c the original plaintext $m = WIO_{MSB}(c) = c^d \bmod n$. Now, we will assume that the same private key (d) is used in another RSA scheme (with any encoding) for digital signature. The attacker can now easily forge the digital signature of any message using the same private key (d) if he/she has access to PIO_{MSB}. Let c be the message that the attacker prepares for signing. He/she then selects different random natural numbers $r = r_1, r_2, ...$ smaller than n and sends $c' = c*r^e \bmod n$ to the oracle PIO_{MSB} successively. After decryption there is calculated $m' = m*r \bmod n$ on the recipient's side. Unless the most significant byte of m' is zero, it is rejected by PIO_{MSB}: $PIO_{MSB}(c')$ = "no". Because the most significant byte of m' is random, it is zero with a probability of 1/256. After several hundreds of trials the value of c' will conform with the initial condition of Manger's attack and WIO_{MSB} then decrypts c': $m' = WIO_{MSB}(c') = (c')^d \bmod n$. The attacker then only has to calculate $m = m'* r^{-1} \bmod n$ as a valid signature of the message c. The particular type of encoding for a signature is irrelevant here. The attacker follows the same procedure when converting Bleichenbacher's attack. This attack assumes the oracle $PIO_{PKCS\text{-}CONF}$, which tells the attacker whether the plaintext produced by decryption is "PKCS#1 conforming" [5]. That means that the two most significant bytes of the plaintext must be equal to $00 \| 02$ and from the 11th byte onwards some byte must be zero (separator). On the basis of $PIO_{PKCS\text{-}CONF}$ a decryption machine $WIO_{PKCS\text{-}CONF}$ is then constructed. If the plaintext is "PKCS#1 conforming", then $WIO_{PKCS\text{-}CONF}$ can use $PIO_{PKCS\text{-}CONF}$ on the corresponding ciphertext c to obtain the original plaintext $m = WIO_{PKCS\text{-}CONF}(c) = c^d \bmod n$. Using the same procedure as above, i.e. by a randomly selected r, we test whether $PIO_{PKCS\text{-}CONF}$ on $c' = c*r^e \bmod n$ responds "yes". This time the probability of such answer is several hundred times lower than in the case of Manger's attack (depending on the number of bits of n; for 1024 it is approximately 715-times less, see [15]). As soon as such a situation occurs, the attacker can again compute $m = m'* r^{-1} \bmod n$ as a valid signature of the message c. Note that the attack described in Section 2 of this paper does not place any special requirements on the ciphertext. It is therefore suitable for forging signatures even without any changes.

In the case of the SSL/TLS protocol the concrete threat of this attack depends not on the protocol itself, but rather on the PKI, which the particular server works in. This PKI manages the server certificate and this PKI decides (via certificate attributes) whether signatures on behalf of that server are meaningful or not. In practice we have seen many server certificates, which were attributed for the purpose of document signing as well.

4 Side Channel Attack on RSA-KEM

After Bleichenbacher's attack on the scheme PKCS#1 v.1.5, the new scheme PKCS#1 v.2.1, based on the EME-OAEP encoding, was recommended for use. However, Manger's attack [15] showed that RSAES-OAEP is also vulnerable to side channel attacks. After that Shoup [23] proposed the new key encapsulation mechanism RSA-KEM. This mechanism was believed to have eliminated problems with side channels. We show that RSA-KEM is also vulnerable to some types of side channel attacks, and therefore has to be implemented carefully. Next we will describe an RSA *confirmation oracle* (CO) based on RSA-KEM and show how to use a CO to obtain a RSA private key.

4.1 Confirmation Oracle

The purpose of RSA-KEM is to transmit the symmetric key to the receiver, and so it is natural to consider the properties of the whole hybrid public-key encryption scheme $\text{H-PKE}_{\text{KEM, DEM}}$, consisting of the Data Encapsulation Mechanism (DEM) and the Key Encapsulation Mechanism (KEM) (c.f. [23]). Our attack on RSA-KEM is based on the behaviour of the entire hybrid scheme. Its requirements are sufficiently general and make it easily realizable in practical applications. We will start by reviewing some important terms from [23] in a simplified form:

The Key Encapsulation Mechanism (KEM) has this abstract interface:

KEM.Encrypt(*PubKey*) $\rightarrow$ (*K*, *C0*) - generates a symmetric encryption key *K* and by using the public key *PubKey* creates a corresponding ciphertext *C0*

KEM.Decrypt(*PrivKey*, *C0*) $\rightarrow$ (*K*) - decrypts *C0* using the private key *PrivKey* and derives the symmetric key *K* by applying the key derivation function KDF to that result

The Data Encapsulation Mechanism (DEM) has this abstract interface:

DEM.Encrypt(*K*, *M*) $\rightarrow$ (*C1*) - encrypts the message *M* with the symmetric key *K* and returns the corresponding ciphertext *C1*

DEM.Decrypt(*K*, *C1*) $\rightarrow$ (*M*) - decrypts the ciphertext *C1* with the symmetric key *K* and returns the plaintext *M*

The hybrid public-key encryption scheme $\text{H-PKE}_{\text{KEM, DEM}}$ is a combination of the KEM and DEM schemes. The algorithm for the encryption of a message *M* by the public key *PubKey* resulting in the ciphertext *C* is as follows:

1. (*K*, *C0*) = KEM.Encrypt(*PubKey*)
2. *C1* = DEM.Encrypt(*K*, *M*)
3. Ciphertext *C* = *C0* || *C1*

On the receiving end, the decryption of the ciphertext *C* with the private key *PrivKey* is carried out as follows:

1. Let *C* = *C0* || *C1*
2. *K* = KEM.Decrypt(*C0*)
3. *M* = DEM.Decrypt(*K*, *C1*)

We assume that there is no integrity check for the key K (e.g. analogous to a check used in the encoding method OAEP) however an integrity check exists for the message M in the third step. It can be based on the message padding check, as in PKCS#5 [18], on the usage of labels as described in [23], or on any other technique. We assume that the attacker will find out whether the receiver's integrity check rejects a ciphertext C. In this situation we can expect that the receiver will send an error message to the sender. Acceptance or rejection of a ciphertext C defines the *receiver oracle* (RO). On the basis of RO we can define the *confirmation oracle* (CO). This term may be defined more generally, however, we will only define the RSA confirmation oracle (RSA-CO) here.

We assume that the private key *PrivKey* is a private exponent d and n is a public modulus. Later we will show that the modulus n should be part of the private key rather than independently taken from the public key, as it is recommended in [23].

Definition. RSA confirmation oracle RSA-CO$_{d,n}(r, y)$.
Let us have a receiver oracle RO that uses RSA in the hybrid encryption H-PKE$_{KEM,DEM}$. We will construct a RSA confirmation oracle RSA-CO$_{d,n}(r, y) \rightarrow$ (ANSWER = "yes/no") as follows:

1. K = KDF(r); KDF - Key Derivation Function
2. $C0 = y$; for simplicity we omit the conversion between integers and strings
3. $C1$ = DEM.Encrypt(K, M); where M contains an integrity check
4. $C = C0 \,\|\, C1$
5. Send the ciphertext C to the receiver oracle RO$_{d,n}$. RO then continues:
    a. Compute K = KEM.Decrypt(d, $C0$) following these steps:
        i. Check if $y = C0 < n$. If not, an error has occurred.
        ii. Compute $r' = (y^d \bmod n)$
        iii. K' = KDF(r')
    b. M' = DEM.Decrypt(K', $C1$)
    c. Check the integrity of M'
    d. If it is correct, the answer of RO is "yes", otherwise it is "no"
6. The answer of RSA-CO$_{d,n}(r, y)$ is "yes", if RO returned "yes", otherwise it is "no"

We note that whenever $r = (y^d \bmod n)$, the oracle returns "yes". If $r \neq (y^d \bmod n)$ then the oracle returns "no" with a high probability close to 1 (the value depends on collisions in the function KDF and the strength of the integrity check). The key point is that an attacker may use the oracle RSA-CO$_{d,n}(r, y)$ to check the congruence $r \equiv y^d \pmod n$ without knowledge of the particular value of the private key d used in the step 5.a.ii above.

4.2 Fault Side Channel Attacks

The congruence $r \equiv y^d \ (mod\ n)$ can be confirmed with the public key as well. However, using RSA-CO$_{d,n}(r, y)$ is the natural way of exploiting the receiver's behaviour. The oracle becomes far more interesting when an error occurs in step 5.a.ii of the

algorithm above. This confirmation oracle can be used to design many attacks. Therefore we will only present a brief description of two examples to illustrate the core of this problem. We note that these attacks are targeted at the private key, rather than the plaintext. This is paradoxically caused by the absence of structural checks of the plaintext in RSA-KEM, which is really a positive quality in other contexts.

4.2.1 Faults in the Bits of the Private Exponent d

The impact of faults in the bits of the private exponent RSA was described in [3]. We will show that the confirmation oracle RSA-$CO_{d, n}$ can be used to mount these attacks on the hybrid encryption scheme based on RSA-KEM. As an example we will assume that the attacker is able to swap the i-th bit $d(i)$ of the receiver's private exponent d (in step 5.a.ii), and this change will go undetected by the receiver. Such a situation can occur with chip cards.

Let us assume that a fault occurred in the i-th bit $d(i)$ and let us denote by d' the defect value of the private exponent. Depending on the value of $d(i)$, either $d' = d + I$ or $d' = d - I$, where $I = 2^i$. Let $\alpha \equiv y^I \pmod n$ and $\alpha^* \alpha^{-1} \equiv 1 \pmod n$. For the value $r = y^{d'} \bmod n$ we have:

$r = (y^d * \alpha \bmod n)$ if $d(i) = 0$

$r = (y^d * \alpha^{-1} \bmod n)$ if $d(i) = 1$

Using the access to the confirmation oracle RSA-$CO_{d', n}$ we can find out the value of $d(i)$ in this way:

1. Randomly pick x, $0 < x < n$
2. Compute $y = x^e \bmod n$, where e is the corresponding public exponent RSA
3. Compute $r = x * \alpha \bmod n$
4. If RSA-$CO_{d', n}(r, y)$ returns "yes" then set $d(i) = 0$ else set $d(i) = 1$.

We can repeat this procedure for various bit positions (and their combinations) and thus obtain the whole private key d. In the case of irreversible changes we will gradually carry out an appropriate correction in step 3 using the previously obtained bits. In this way the corruption of d is allowed to be irreversible. Moreover, it is enough to obtain only a part of d from which the remaining bits can be computed analytically in a doable time, see overview in [6]. In [3,7] we may find other sophisticated attacks of this type. We have presented the confirmation oracle as an "interface" that allows the attacker to apply some general attacks on "unformatted RSA" to RSA-KEM.

4.2.2 The Usage of Trojan Modulus

We have mentioned that in the RSA-KEM scheme, the modulus n is not part of the private key. This would allow for a change of the modulus n without any security alarm. The following attack shows the need to change this set up.

Let us assume that we can obtain the value $r = g^d \bmod n'$ for an unknown exponent d and arbitrary values of g and n'. It is widely known that one such value r is sufficient to discover d. We can, for instance, choose a modulus n' to be a prime in the form $n' = t*2^s+1$, where t is a very small prime number and s is a very large natural number. Further we choose g to be a generator of the multiplicative group $Z_{n'}^*$.

Now we can solve the discrete logarithm problem in $Z_{n'}^*$ by a simple modification of the Pohlig-Hellman algorithm [19]. This algorithm requires the value of r, $r = g^d \bmod n'$, directly, which we cannot obtain from the confirmation oracle. We can only ask the oracle whether the pair of integers (x, g) satisfies the congruence $x \equiv g^d \bmod n'$. On a closer look at the Pohlig-Hellman algorithm we notice that it can be modified so that the value of r is not needed directly, but only in comparisons of the type $x =^? (r^\alpha \bmod n')$ for some integers x, α. It means that we only want to know whether $x =^? ((g^d)^\alpha \bmod n')$, which can be obtained by calling the confirmation oracle RSA-CO$_{d, n'}(x, g^\alpha \bmod n')$. This is the main idea of the modification. The complete algorithm A1 is presented in the next subsection.

This attack is also possible even if the modulus n is part of the private key. However in this case we can expect that it will be a little bit more difficult to plant a false value of n'. This idea can also be extended to the case when a method based on the Chinese Remainder Theorem is used for operations with the private key.

4.2.3 Algorithm A1: Computation of the Private Exponent Using the Access to a RSA Confirmation Oracle

In the following we will describe an efficient algorithm for a private exponent d computation, based on a modified Pohlig-Hellman algorithm for the discrete logarithm problem in the multiplicative group Z_p^*. This group has a special structure chosen by an attacker, because the value of p is taken to be the fraudulent modulus n'.

Proposition. Let us assume to have an access to a confirmation oracle RSA-CO$_{d, p}$, where p is a prime such that $p = t*2^s + 1$ and t is a small prime. Let g be the generator of Z_p^*. (We note that the order of Z_p^* has to be larger than the highest possible value of d.) The following procedure computes the private exponent d in the three steps.

Step 1: Computation of the value $D_s = d \bmod 2^s$

Let $d = d(b\text{-}1)*2^{b-1} + d(b\text{-}2)*2^{b-2} + ...+ d(0)$, where b is the number of bits of the binary form of d, and $d(i) \in \{0, 1\}$, for $0 \le i \le b\text{-}1$. We assume that $p\text{-}1$ is divisible by 2^i and we define $r = g^d \bmod p$ and $D(i) = d \bmod 2^i$. Let us denote $I = 2^i$ and $J = 2^j$. Then

$r^{(p-1)/I} \equiv [g^d]^{(p-1)/I} \equiv [g^{(p-1)/I}]^d \equiv [g^{(p-1)/I}]^{d \bmod I} \equiv [g^{(p-1)/I}]^{D(i)} \pmod p$, and hence

$$r^{(p-1)/I} \equiv [g^{(p-1)/I}]^{D(i)} \pmod p . \tag{1}$$

The value of $D(i)$ can be expressed as $D(i) = d(i\text{-}1)*2^{i-1} + d(i\text{-}2)*2^{i-2} + ...+ d(0)$. We will show that having access to the confirmation oracle we can easily compute the lowest s bits of the private exponent d (one bit of d per one oracle call). We will start with the lowest bit $d(0)$ and inductively go to the bit $d(s\text{-}1)$. For $i = 1$ from (1) we have $r^{(p-1)/2} \equiv [g^{(p-1)/2}]^{d(0)} \pmod p$. From the definition of r we have $r^{(p-1)/2} \equiv [g^{(p-1)/2}]^d \pmod p$, and so

$$[g^{(p-1)/2}]^d \equiv [g^{(p-1)/2}]^{d(0)} \pmod p . \tag{2}$$

We note that $g^{(p-1)/2} \equiv p\text{-}1 \pmod p$, and $[g^{(p-1)/2}]^{d(0)} \bmod p$ can achieve only two possible values, depending on the bit $d(0)$. Using the confirmation oracle, we can either confirm or refute the value of $d(0)$ in (2). Let $d(0) = 1$ and let us make the oracle call RSA-CO$_{d, p}(p\text{-}1, p\text{-}1)$, which represents the congruence (2). If the oracle returns "yes"

we set $d(0) = 1$, otherwise we set $d(0) = 0$. We note that a correctly generated private exponent RSA should induce $d(0) = 1$, therefore this step can be omitted. We determine the remaining bits of $D(s)$ inductively. We assume that we know the value $D(j)$ for some $0 < j < s$. Next we will compute the value $D(j+1)$. From (1) we have

$$r^{(p-1)/(2J)} \equiv [g^{(p-1)/(2J)}]^{D(j+1)} \pmod{p} . \tag{3}$$

Let $\alpha = d(j) * 2^j = d(j) * J$. Then $D(j+1) = d \bmod 2^{j+1} = \alpha + D(j)$. For the value on the right-hand side of (3) we have that $[g^{(p-1)/(2J)}]^{D(j+1)} \equiv [g^{(p-1)/(2J)}]^{\alpha} * [g^{(p-1)/(2J)}]^{D(j)} \equiv [g^{(p-1)/2}]^{d(j)} * [g^{(p-1)/(2J)}]^{D(j)} \equiv (p-1)^{d(j)} * [g^{(p-1)/(2J)}]^{D(j)} \pmod{p}$, so we get $r^{(p-1)/(2J)} \equiv$
$\equiv (p-1)^{d(j)} * [g^{(p-1)/(2J)}]^{D(j)} \pmod{p}$. Using the definition of r ($r = g^d \bmod p$) we obtain

$$[g^{(p-1)/(2J)}]^{d} \equiv (p-1)^{d(j)} * [g^{(p-1)/(2J)}]^{D(j)} \pmod{p} . \tag{4}$$

On the right-hand side of (4), almost entirely known values appear, with the exception of the value of $d(j)$. We will again use the confirmation oracle to decide between the two possible values of the bit $d(j)$. We guess that $d(j) = 0$ and call the oracle in the form RSA-$CO_{d,\,p}([g^{(p-1)/(2J)}]^{D(j)} \bmod p,\ g^{(p-1)/(2J)} \bmod p)$, which represents the congruence (4). If the oracle returns "yes", we set $d(j) = 0$, otherwise we do the correction $d(j) = 1$. The inductive step is finished and we have obtained $D_s = D(s)$.

Step 2: Computation of the value $D_t = d \bmod t$

It is simple to show that an integer j, under the condition $r^{(p-1)/t} \equiv [g^{(p-1)/t}]^j \pmod{p}$, satisfies that $D_t \equiv j \pmod{t}$. Whenever $j < t$, then we directly obtain that $D_t = j$. Therefore we can identify the value D_t in this step by testing every number $j = 0, \ldots, t-1$, until we find the j that satisfies the congruence $r^{(p-1)/t} \equiv [g^{(p-1)/t}]^j \pmod{p}$. This j is then the sought value of D_t. In order to determine this value we rewrite the congruence (using the definition of r) as follows:

$$[g^{(p-1)/t}]^{d} \equiv [g^{(p-1)/t}]^{j} \pmod{p} \tag{5}$$

and use the oracle in the form RSA-$CO_{d,\,p}([g^{(p-1)/t}]^j \bmod p,\ g^{(p-1)/t} \bmod p)$ gradually for $j = 0,\ldots, t-1$. The correct value of j is reached when the oracle returns "yes" and we set $D_t = j$.

Step 3: Computation of the value d

In the previous steps we have obtained two congruencies $d \equiv D_s \pmod{2^s}$ and $d \equiv D_t \pmod{t}$. It also holds that $\gcd(t, 2^s) = 1$, and so by the Chinese Remainder Theorem, there exists a single value $0 \le d < t*2^s$, satisfying both congruencies. The value of d can be computed directly as bellow:

1. Compute γ; $\gamma * 2^s \equiv 1 \pmod{t}$, a unique value exists because $\gcd(t, 2^s) = 1$
2. Compute $v = (D_t - D_s) * \gamma \bmod t$
3. $d = D_s + v*2^s$

Note that this attack requires at most $s + t$ oracle calls together with a trivially feasible number of group multiplications on Z_p^*.

4.2.4 Other Computational Faults

So far we have only considered the attacks based on modifications of the private exponent d and the modulus n. However, similar attacks may be developed, considering general permanent or transient faults that appear during RSA computations within the function KEM.Decrypt. A discussion on these attacks, however, is beyond the scope of this paper. For more details, the reader may consult papers [3, 7]. We can realistically assume that certain types of attacks described there can be used on RSA-KEM with the use of the confirmation oracle.

4.2.5 Comparison of Attacks on RSA Schemes

Manger [15] showed that the RSAES-OAEP scheme has certain problems with the most significant byte. These problems must be avoided by proper implementation. We have shown that RSA-KEM has similar problems, when fault side channel attacks can occur. Whenever we use RSA-KEM it is therefore essential to exclude fault side channels. We must carry out reliable private key integrity checks (the modulus should be a natural part of the private key) as well as using fault tolerant computations. We still need to consider the consequences of the RSA individual bit theorem and make sure that no information about any individual bit of the plaintext has leaked. Table 2 below contains a brief overview of the current state of most used RSA schemes when side channel attacks are considered.

Table 2. RSA schemes and side channel attacks

	PKCS1 v.1.5	RSAES-OAEP	RSA-KEM
Public attack	Yes	Yes	Yes
Side channel (information) used in attack	The information about whether the plaintext is PKCS#1 v.1.5 conforming	- The information about whether the most significant byte of plaintext is zero - Hamming weight of processed data	Fault side channel
Information obtained in attack	Plaintext	Plaintext	Private key

4.3. General Countermeasures

When we consider the state-of-the-art in cryptanalysis, we can specify three basic security criteria that need to be satisfied in every cryptosystem design on the RSA basis. These are:

(a) Resistance to adaptive chosen ciphertext attacks
(b) Resistance to side channel information leakage
(c) Resistance to fault side channels

Imperfect resistance to any of these types of attack can result in the ability to decrypt ciphertext (mainly (a)) or to obtain directly the value of the private key (mainly (c)). We have purposely omitted from the list resistance to purely algebraic attacks, such as problems with a low value of the private or public exponent, among other similar ones (their overview appears in [6]), since most successful attacks are based on an incorrect use of RSA and implementation faults. The problem of the correct use of RSA is rooted in the mathematics underlying the algorithm (for details see [13,2,6,7,15,5,9,10] and attacks presented there) and thus it should be examined from a mathematical perspective. It seems too risky to leave the issue in the hands of implementators. We also note that cryptanalysis has gradually accepted the assumption that an attacker has nearly unlimited access to an attacked system. We do not merely consider attacks on "data passing through" but direct attacks on autonomous cryptographic units

Furthermore, we can see that it is not possible to satisfactorily solve the defence against the types of attacks specified above by a single universal encoding of data being encrypted. This is a consequence of the fact that the encoding mechanism is only part of the whole scheme and as such can only affect part of its properties.

Now we will look at basic defence mechanisms against the above types of attacks. The first category, adaptive chosen ciphertext attacks, has not been considered in this paper. We think that a satisfactory solution is the random oracle paradigm [4], which has been successfully applied [23,24,11]. For category (b), we need to constantly bear in mind the claim in [13], and prevent any leakage of plaintext information. It is not possible to limit our attention only to the easily visible information such as the value of the most significant byte of plaintext in RSAES-OAEP. In Section 2, we showed that the leakage of information from completely other part of the scheme has also a negative effect on security. Power side channel attacks [14,16,1] and nascent theory of electromagnetic side channel attacks [20,12] is necessary to be considered a particularly high threat. However, defence measures against these channel attacks [8] are beyond the scope of this paper. It was our aim to show that these countermeasures need to be used in every single function that deals with individual parts of the plaintext. Here we focused our attention on the function SHA-1 as an example.

Finally the last category are fault attacks. The vulnerability of RSA to these attacks does not originate directly from the theorem [13]. However, it seems to be an innate quality of the RSA system [3,6,7]. As well as with the other types of attacks, certain types of encoding can more or less eliminate fault attacks. We showed that RSA-KEM, despite it seems to be well resistant to other types of attacks [23], can be easily and straightforwardly affected by fault side channel attacks. To avoid fault attacks it is recommended especially:

(i) To consistently check the integrity of the private key and of the other parameters used with it in its processing

(ii) To minimize the range of error messages

(iii) Wherever possible, to use platforms equipped with fault detection and eventually also correction facilities (fault tolerant systems)

As a rather strong countermeasure, even though not 100% sure, we can recommend to check every result $x = (y^d \bmod n)$ as $y \stackrel{?}{=} (x^e \bmod n)$, where d is the private exponent,

e is the public exponent and n is the modulus. This measure effectively prevents both attacks presented as the examples in this paper. The proof is simple: with a high probability, the relationship $e*d \equiv 1 \pmod{\mathrm{ord}(y)}$, where $\mathrm{ord}(y)$ is the order of y in the multiplicative group Z_n^*, will be violated in both examples.

5 Conclusion

The RSA individual bits theorem [13] is generally considered to be a good property of RSA. However, it also shows the way for attacks based on side channels [5,15].

We have presented another possible attack on the encryption scheme RSAES-OAEP where, in contrast with the previous work [15], we attack that part of the plaintext "shielded" by the OAEP method. In this, we use the algebraic properties of RSA, rather than some weakness of the OAEP encoding. To prevent this attack, we need to eliminate the parasitic leakage of information from individual operations in partial procedures of the entire scheme. This goes well beyond the scope of the general description of the OAEP encoding method. Next we presented a new side channel attack on the RSA-KEM. This scheme was built to prevent the parasitic leakage of information about the plaintext, especially under the consideration of chosen ciphertext attack. However, we managed to point out a side channel that allows the leakage of this information. Unlike previous attacks that returned the plaintext, this time the attacker obtains the RSA private key. The attack was again made possible by the basic multiplicative property of RSA.

Our contribution underlines the significance of the known algebraic properties of RSA in relation to rapidly evolving attacks based on side channels. Consequently, it is possible to expect similar side channel attacks in other RSA schemes that may employ different message encoding. Therefore, it is necessary to pay more attention to side channel countermeasures in implementations of these cryptographic schemes.

As a small note in our paper, we pointed out the rule of keeping RSA keys for encryption and digital signature strictly separated, which is often neglected. We assumed that the rule is not adhered to, and described an approach to convert both Manger's and Bleicherbacher's oracles for ciphertext decryption into oracles that can create valid digital signatures for arbitrarily encoded messages.

References

1. Akkar, M.-L., Bevan, R., Dischamp, P. and Moyart, D.: *Power Analysis, What Is Now Possible...*, in Proc. of ASIACRYPT 2000, pp. 489–502, 2000.
2. Alexi, W., Chor, B., Goldreich, O. and Schnorr, C.: *RSA and Rabin functions: Certain parts are as hard as the whole*, SIAM Journal on Computing, 17(2), pp. 194–209, 1988.
3. Bao, F., Deng, R.-H., Han, Y., Jeng, A., Narasimhalu, A.-D. and Ngair, T.: *Breaking Public Key Cryptosystems on Tamper Resistant Devices in the Presence of Transient Faults*, in Proc. of Security Protocols '97, pp. 115–124, 1997.

4. Bellare, M. and Rogaway, P.: *Random Oracles are Practical: A Paradigm for Designing Efficient Protocols*, October 20, 1995, originally published in Proc. of the First ACM Conference on Computer and Communications Security, ACM, November 1993.
5. Bleichenbacher, D.: *Chosen Ciphertexts Attacks Against Protocols Based on the RSA Encryption Standard PKCS#1*, in Proc. of CRYPTO '98, pp. 1–12, 1998.
6. Boneh, D.: *Twenty Years of Attacks on the RSA Cryptosystems*, Notices of the American Mathematical Society, vol. 46, no. 2, pp. 203–213, 1999.
7. Boneh, D., DeMillo, R. A. and Lipton, R. J.: *On the Importance of Checking Cryptographic Protocols for Faults*, in Proc. of EUROCRYPT '97, pp. 37–51, 1997.
8. Chari, S., Jutla, C.-S., Rao, J. and Rohatgi, P.: *Towards Sound Approaches to Counteract Power-Analysis Attacks*, in Proc. of CRYPTO '99, pp. 398–411, 1999.
9. Fischlin, R. and Schnorr, C. P.: *Stronger Security Proofs for RSA and Rabin Bits*, in Proc. of EUROCRYPT '97, pp. 267–279, 1997.
10. Fischlin, R. and Schnorr, C. P.: *Stronger Security Proofs for RSA and Rabin Bits*, Journal of Cryptology, Vol. 13, No. 2, pp. 221–244, IACR, 2000.
11. Fujisaki, E., Okamoto, T., Pointcheval, D. and Stern, J.: *RSA-OAEP Is Secure under the RSA Assumption*, in Proc. of CRYPTO 2001, pp. 260–274, 2001.
12. Gandolfi, K., Mourtel, C. and Olivier, F.: *Electromagnetic Analysis: Concrete Results*, in Proc. of CHES 2001, pp. 251–261, 2001.
13. Håstad, J. and Näslund M.: *The Security of Individual RSA Bits*, in Proc. of FOCS '98, pp. 510–521, 1998.
14. Kocher, P., Jaffe, J. and Jun, B.: *Differential Power Analysis: Leaking Secrets*, in Proc. of CRYPTO '99, pp. 388–397, 1999.
15. Manger, J.: *A Chosen Ciphertext Attack on RSA Optimal Asymmetric Encryption Padding (OAEP) as Standardized in PKCS #1*, in Proc. of CRYPTO 2001, pp. 230–238, 2001.
16. Messegers, T.-S., Dabbish, E. A. and Sloan, R. H.: *Investigations of Power Analysis Attacks on Smartcards*, in Proc. of USENIX Workshop on Smartcard Technology, pp. 151–161, 1999.
17. PKCS#1 v2.1: *RSA Cryptography Standard*, RSA Labs, DRAFT2, January 5 2001.
18. PKCS#5 v2.0: *Password-Based Cryptography Standard*, RSA Labs, March 25, 1999.
19. Pohlig S.C., Hellman M.E.: *An improved algorithm for computing logarithms over GF(p) and its cryptographic significance*, IEEE Trans. Inform. Theory, 24 (1978), 106–110.
20. Rao, J.-R and Rohatgi, P.: *EMpowering Side-Channel Attacks*, preliminary technical report, May 11 2001.
21. Rivest, R., L., Shamir, A. and Adleman L.: *A method for obtaining digital signatures and public-key cryptosystems*, Communications of the ACM, pp. 120–126, 1978.
22. Secure Hash Standard, FIPS Pub 180-1, 1995 April 17.
23. Shoup, V.: *A Proposal for an ISO Standard for Public Key Encryption (version 2.0)*, September 17, 2001.
24. Shoup, V.: *OAEP Reconsidered (Extended Abstract)*, in Proc. of CRYPTO 2001, pp. 239–259, 2001.
25. Stinson, D., R.: *Cryptography – Theory and Practice*, CRC Press, 1995.

Fault Attacks on RSA with CRT: Concrete Results and Practical Countermeasures

C. Aumüller, P. Bier, W. Fischer, P. Hofreiter, and J.-P. Seifert

Infineon Technologies
Security & ChipCard ICs
D-81609 Munich
Germany
{christian.aumueller, peter.bier, wieland.fischer, peter.hofreiter, jean-pierre.seifert}@infineon.com

Abstract. This article describes concrete results and practically validated countermeasures concerning differential fault attacks on RSA using the CRT. We investigate smartcards with an RSA coprocessor where any hardware countermeasures to defeat fault attacks have been switched off. This scenario was chosen in order to analyze the reliability of software countermeasures.
We start by describing our laboratory setting for the attacks. Hereafter, we describe the experiments and results of a straightforward implementation of a well-known countermeasure. This implementation turned out to be not sufficient. With the data obtained by these experiments we developed a practical error model. This enabled us to specify enhanced software countermeasures for which we were not able to produce any successful attacks on the investigated chips.
Nevertheless, we are convinced that only sophisticated hardware countermeasures (sensors, filters, etc.) in combination with software countermeasures will be able to provide security.

Keywords: Bellcore attack, Chinese Remainder Theorem, Fault attacks, Hardware security, RSA, Spike attacks, Software countermeasures, Transient fault model.

1 Introduction

This paper shows and proves that fault attacks on RSA with the CRT (also known as Bellcore attacks) due to [BDL] are feasible. They are indeed devastating if there are neither hardware mechanisms (sensors, filters, etc.) nor any appropriate software countermeasures implemented in the underlying smartcard ICs. However, this does not imply that modern high-security smartcard ICs are vulnerable to this kind of attacks. Instead, it shows that fault tolerance and especially sophisticated hardware countermeasures are essential for the design of secure hardware. Moreover, we stress that it is very difficult in the field to switch off these sophisticated hardware countermeasures. This has been done

B.S. Kaliski Jr. et al. (Eds.): CHES 2002, LNCS 2523, pp. 260–275, 2003.

exceptionally for our study concerning software countermeasures against the Bellcore attack.

In order to provide better security for data protection under strong encryption more and more implementations on tamper-proof devices (e.g., smartcard ICs) are proposed. The main reason is that smartcard ICs provide high reliability and security with more memory capacity and better performance characteristics than conventional magnetic stripe cards. With special characteristics of computational ability a large variety of cryptographic applications benefit from smartcard ICs. This attracted a huge amount of research on physical attacks against smartcards in 1996 due to [Koch], [BDL] and again 1999 by [KJJ], followed by [GMO,SQ]. However, most research so far focused on Timing or Power Analysis attacks. This is surprising as the frauds with smartcards by inducing faults are reality, cf., [A,AK1,AK2], whereas no frauds via Timing or Power Analysis attacks have been reported so far. Moreover, research on fault-based cryptanalysis is not very active compared to the other side-channel attacks. Furthermore, no practical investigation of the Bellcore attack is presently known. Indeed, this topic will be publicly addressed within this paper for the first time. It answers a question of Kaliski and Robshaw [KR] *of how practical these attacks might be*, answered definitely here *by physicists, designers and manufactures of secure hardware.*

The present paper is organized as follows: Section 2 briefly repeats RSA using the CRT and its fault-based cryptanalysis according to [BDL,JLQ]; it also includes and discusses the advantages and limitations of so far publicly known software countermeasures to defeat fault attacks on RSA in CRT mode. Section 3 firstly explains so-called spike attacks and their realization on smartcard ICs, their complexity from an attacker's point of view and reveals an appropriate test equipment to implement fault attacks. Secondly, we will present the resulting errors on unprotected hardware and software for RSA in CRT mode. This demonstrates the insufficiency of a straightforward implementation of a well-known countermeasure due to [Sh]. Within section 4 we basically investigate enhanced software countermeasures derived from our practical observations and our proposed model to counteract fault attacks on RSA. Eventually, section 5 adds some practical conclusions concerning software countermeasures to prevent Bellcore attacks.

2 Preliminaries

2.1 The RSA System

Let $N = p \cdot q$ be the product of two large primes of similar length. To sign a message $m \in \mathbb{Z}_N$ using RSA one computes $S := m^d \bmod N$, where d is the private exponent satisfying $e \cdot d \equiv 1 \bmod (p-1)(q-1)$ for the public exponent e. The computationally expensive part of signing is the modular exponentiation. For better efficiency most implementations exponentiate as follows: using repeated square and multiply they first compute $S_p := m^d \bmod p$ and hereafter $S_q := m^d \bmod q$. Then they construct the signature $S = m^d \bmod N$ using the

CRT. This last step takes negligible time compared to the two exponentiations. It is done efficiently by computing

$$S = S_q + \big((S_p - S_q) * (q^{-1} \bmod p) \bmod p\big) * q, \tag{1}$$

using Garner's algorithm, cf. [Kn].

The exponentiation using the CRT is much faster than the full exponentiation. To see this, observe that $S_p = m^d \bmod p = m^{d \bmod (p-1)} \bmod p$. Usually, d is of order N, while $d \bmod (p-1)$ is of order p. Consequently, computing S_p requires half as many multiplications as computing S directly. In addition, intermediate values during the computation of S_p are only half as big — they are in the range $[1, \ldots, p]$, rather than $[1, \ldots, N]$. Clearly, the same arguments are valid for the computation of S_q. When quadratic time complexity is used, multiplying two numbers in $\mathbb{Z}_p$ takes a quarter of the time as multiplying elements in $\mathbb{Z}_N$. Hence, computing S_p takes an eighth of the time of computing S directly. Thus, computing S_p and S_q this way takes a quarter of the time of computing S directly. Thus, CRT exponentiation is four times faster than direct exponentiation. This is the reason for using the CRT for RSA signature generation, cf. [CQ,MvOV].

2.2 The Fault-Based Cryptanalysis of RSA Using CRT

We briefly recall the fault-based cryptanalysis of RSA with the CRT due to [BDL,JLQ]. Assume that during the computation of an RSA signature for a message m a random error occurs during the computation of S_p. This yields a faulty signature part S'_p, whereas the computation of S_q is done correctly. The combination of S'_p and S_q via (1) will yield an incorrect signature S'. For S' it holds that $S - S' \neq 0$ but $S - S' \equiv 0 \bmod q$. Therefore, one obtains the factorization of N by computing

$$\gcd\left((m - (S')^e) \bmod N, N\right) = q.$$

2.3 Simple Software Countermeasure to Defeat the Fault Attack

Some simple ad-hoc countermeasures have been already suggested within [BDL, KR]. One approach is to perform calculations twice and the other approach suggests to verify the correctness of the signature by comparing the inverse result with the input. The first approach is very time-consuming and it cannot always provide a satisfactory solution because a permanent error may be undetectable by computing the function more than once. The second approach is to verify the correctness by comparing the inverse result with the input m. Generally, this is not a satisfactory solution since the parameter e could be a large integer and this checking procedure becomes time-consuming. Additionally, for a real life software implementation the programmer cannot rely on the fact that e is known and a small number. On the other hand, this countermeasure seems to be the safest.

An interesting countermeasure is the introduction of randomness into the RSA signature process. Here, RSA is applied to $F(m, r)$ where F is some formatting function and r is a random string which ensures that the user never signs the same message twice and the attacker does not know the signed message, cf. [BDL,BR,KR]. Other countermeasures are mentioned in [Ro].

2.4 Shamir's Software Countermeasure

Shamir's idea, cf. [Sh], is to select a random integer t and to do the following computations

$$S_{pt} := m^d \bmod p * t,$$
$$S_{qt} := m^d \bmod q * t.$$

In the case of $S_{pt} = S_{qt} \bmod t$ the computation is defined to be error free and S is computed according to the CRT recombination equation (1).

One drawback in Shamir's method, as pointed out in [JPY], is the following: Within the CRT mode of real RSA applications the value d is not known, only the values $d_p = d \bmod (p-1)$ and $d_q = d \bmod (q-1)$ are known. Although d can be efficiently computed from d_p and d_q only, as described in [FS], it will limit the acceptance of Shamir's method. Moreover, his check will be shown to be insufficient anyway. But, our enhanced software countermeasures will resolve the above critical points of his method.

2.5 General Remarks on Methods to Overcome Fault Attacks

Only very recently the field of research on fault attacks countermeasures has been emerged. For instance a series of papers [YJ,YKLM1,YKLM2,JQYY] assume that the attacker has a very precise knowledge about the implementation details and especially an absolute accurate control of the timing of his fault induction. Under this strong assumption the private exponent d can be reconstructed by abusing the implemented correctness check as an oracle for the bits of d. However, all the described fault attacks can easily be prevented by various randomization techniques for the RSA algorithm. In a side-channel secure RSA signature implementation such techniques are present.

Moreover, [YKLM1] proposed the following very interesting countermeasure: Their key idea is to influence the computation of S_q or the overall computation of S when an error occurred during the computation of S_p, or vice versa. The cryptanalysis given in section 2 shows that a successful fault attack is not possible anymore. Unfortunately it was recently shown by [BMS] that their proposal for a so-called infective RSA CRT computation is not secure.

3 Physical Fault Attacks Realization

First of all, we would like to stress again that modern high-end cryptographic devices, e.g., smartcards, are usually protected by means of various and numerous

sophisticated hardware mechanisms to detect any intrusion attempt into their system behavior, cf. [Ma,NR]. This is due to the fact that hardware manufacturers of cryptographic devices such as smartcard ICs have been aware of the importance of protecting against intrusions by, e.g., external voltage variations, external clock variations, etc. for a long time. However, it should be clear that the design of such mechanisms is a very difficult engineering task. Such mechanisms should be able to tolerate slight natural deviations from the standard values of the electrical parameter to be safeguarded. This is necessary to ensure a proper functionality of the underlying device within the specified range, as for example described in [ISO]. On the other hand they also have to detect very fast and unnatural low deviations from the specified standard range. This condition is necessary to detect any attack attempt by modifying the electrical execution conditions to alter a computation's result. For example, the standard specification [ISO] allows for the smartcard IC's contact V_{CC} under normal operating conditions a voltage supply between $4,5$V and $5,5$V.

Although there are lots of possibilities to introduce an error during the cryptographic operation of an unprotected smartcard hardware, we will only explain in detail the so-called spike attacks. The reason is that spike attacks are non invasive attacks. Thus, they require no physical opening and no chemical preparation of the smartcard IC. For further information on various methods how to enforce erroneous computations of chips we refer to [A,AK1,AK2,Gu1,Gu2, Koca,Ma].

3.1 Spikes

A smartcard of voltage class type A should be able to tolerate on the contact V_{CC} a supply voltage between $4,5$V and $5,5$V, where the standard voltage is specified at 5V. Within this range the smartcard will be able to work properly. However, a deviation of the external power supply of much more than the specified 10% tolerance could cause problems with the smartcard IC. Indeed, it could then lead to a wrong computation result, provided that the smartcard IC is still able to finish its computation completely. But most often this is not possible, as the spike causes too much trouble to the CPU of the smartcard IC. Although a spike with the explanation above seems very simple, a specific type of a power spike is determined by nine parameters. Using picture 1 we will explain them:

1. Initial value of the power supply V_2.
2. Starting point t_1 of the spike.
3. Rise time $t_2 - t_1$ of the spike.
4. Shape of the rising transition.
5. Height $V_3 - V_1$ of the power spike.
6. Length of the power spike $t_3 - t_2$.
7. Falling time $t_4 - t_3$ of the spike.
8. Shape of the falling transition.
9. Final value V_1 of the power supply.

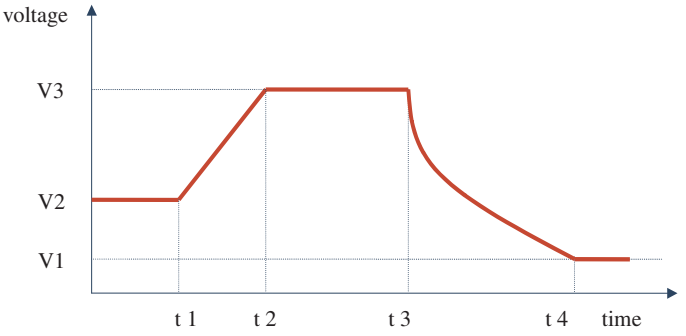


Fig. 1. Spike-parameters defining the shape of a specific spike.

This indicates the huge range of different parameters which must be scanned for penetration attacks against cryptographic devices. On the other hand, it also reveals the strong demands on the corresponding sensor and filter mechanisms. From the former discussion of spike attacks, one can envision the difficulties an attacker is confronted with, when he wants to overcome all the activated hardware countermeasures within modern high-security smartcard ICs.

3.2 Laboratory Setting

In order to systematically investigate the effects of spikes and especially our proposed countermeasures, we basically used the following spike enforcing hardware set-up, which is shown in figure 2.

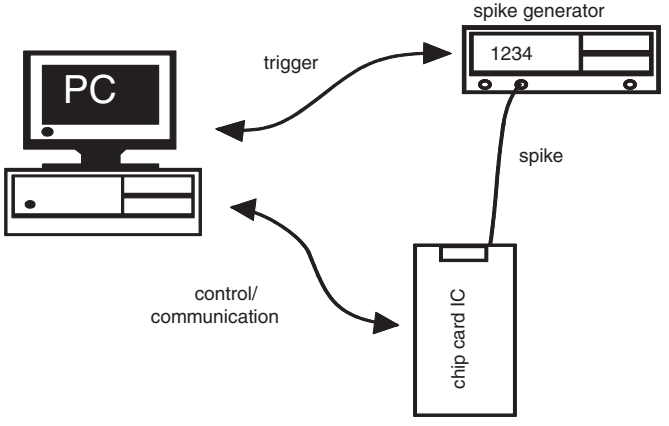


Fig. 2. Diagram of our test equipment.

With such a test set-up it is indeed possible to enforce a spike with a very high accuracy. This is necessary, if the spike shall just only enforce a tiny ran-

dom computation fault rather than a complete destruction of the smartcard's computation, which would make the smartcard's computation result unusable for a successful attack. Through the coupling of the control and communication of the smartcard with a PC, which is running a dedicated test-software, it is possible to observe and analyze the smartcard's reaction with respect to the applied spike-form as discussed above, e.g., answering with a correct/wrong answer sequence. Furthermore, the PC is responsible for the stimuli, timing and controlling of the above spike parameters. Coupled with an interface card, the spike generator is triggered by the PC which provides the time and voltage information for the specific spike to be applied to the card. The spike generator is directly connected to the power supply V_{CC} of the smartcard and provides its IC with the necessary operating voltage including the voltage drop of the spike. By means of the synchronization of the PC, the spike generator and the chipcard itself a very high attack reproducibility of more than 90% can be achieved.

Now, one has to find parameters for such a spike which enables a tiny random computation fault, but leaves the main computation untouched.

3.3 Results on Unprotected Hardware and Software

We will now discuss our results of successfully applied spike-attacks on unprotected smartcards, i.e., ICs where any hardware countermeasures against fault attacks have been switched off. Moreover, we have also switched off any (hardware and software) countermeasures against other classical side-channel attacks, like Timing Analysis [Koch], Power Analysis [KJJ], Electromagnetic Analysis [SQ,GMO], etc.

However, to introduce a spike at the right position of the RSA with the CRT, one should investigate the power profile of the critical computation first. Such a power profile of our investigated smartcard equipped with an RSA coprocessor is shown in figure 3. Let us explain this power profile a little bit more: The upper line represents the profile of the smartcard's I/O behavior. The first I/O activity is the start impulse for the smartcard and the second peak is the answer sequence given by the smartcard. Between these two peaks the smartcard is computing a 2048-bit RSA signature using the CRT. This is shown in the lower line where the main power profile of the smartcard is depicted.

The RSA-CRT computation starts at the time block 1.5 and ends at the time block 9.2. In the figure the blocks are numbered from 0 to 9. This is shown by the fact that the power consumption increases — due to the coprocessors activity. One immediately recognizes the two different exponentiations as they are the main power consumers.

In our case the first exponentiation lies in the time frame 1.6 to 5.1, and the second exponentiation lies in the time frame 5.3 to 8.8. Before the first exponentiation one recognizes the loading of the data into the crypto coprocessor for the first exponentiation, after the first exponentiation the corresponding correctness checks and as well the loading of the data into the crypto coprocessor and for the second exponentiation and after the second exponentiation again the correctness checks of the second exponentiation. Finally, one sees the CRT combination of

the two partial exponentiations followed eventually by an additional correctness check for the CRT combination.

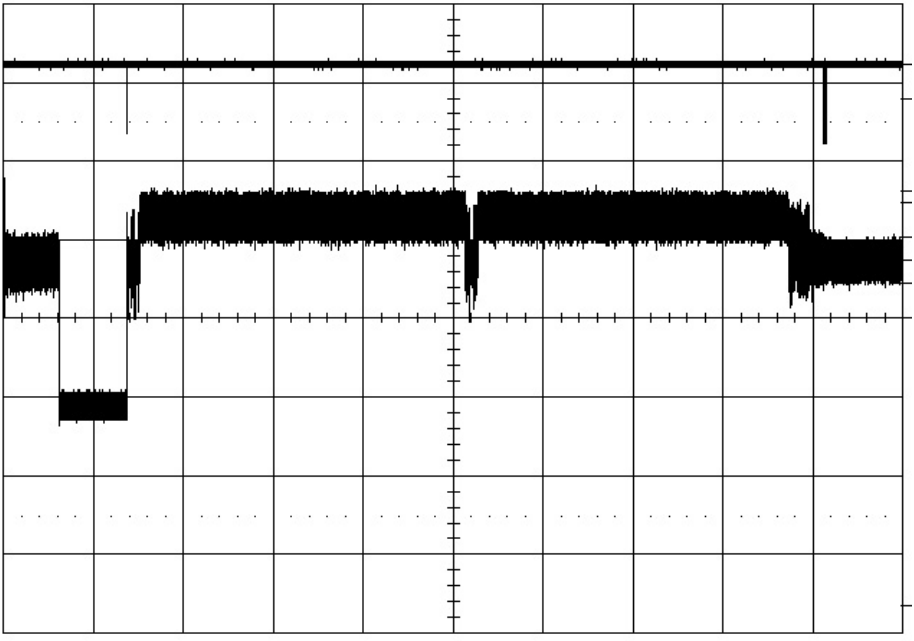


Fig. 3. Power profile of RSA with the CRT.

Results on completely unprotected RSA using the CRT. The first algorithm we attacked with our spike equipment was the pure RSA signature algorithm using the CRT:

input: $m, p, q, d_p, d_q, q^{-1} \bmod p$

$S_p := m^{d_p} \bmod p$
$S_q := m^{d_q} \bmod q$
$S := S_q + ((S_p - S_q) * q^{-1} \bmod p) * q$
return(S)

output: $m^d \bmod N$

Before discussing the results of our spike attacks on the above algorithm, we note that the inputs $p, q, d_p, d_q, q^{-1} \bmod p$ are usually stored in EEPROM, while the message m is stored in RAM. However, in order to work with the data

$p, q, d_p, d_q, q^{-1} \bmod p$ they must be moved from EEPROM into RAM or the crypto coprocessor. By varying the time when we applied the appropriate spike to the smartcard IC's power supply V_{CC}, we were able to induce the following different errors:

Observed error	Mainly due to
modification of p, q	Moving data from E^2 to coprocessor
modification of d_p, d_q	Handling data within CPU
wrong exponentiation modp, q	Error within CPU or coprocessor
modification of $q^{-1} \bmod p$	Moving data from E^2 to coprocessor
wrong combination of S_p and S_q	All listed errors
faulty signature mod$p$ *and* mod$q$	Moving data from coprocessor
wrong answer of smartcard	Fatal error within CPU

Note that the first five errors may lead to a successful attack, whereas the last two do not. Thus, we can conclude that it is absolutely necessary to have sophisticated hardware and software countermeasures to avoid such kinds of attacks. Within the remaining sections we will analyze already existing software countermeasures and also develop new and more reliable countermeasures.

Results on unprotected hardware with simple software countermeasures. Motivated by the devastating results obtained within the previous section, we hereafter tested the reliability of the naively implemented software countermeasures due to [Sh] as desribed in section 2. Thus, we applied spikes to the unprotected smartcard while computing the following RSA signature algorithm shown in figure 4.

Again, we firstly summarize some of the observed errors.

	Observed error scenarios	A	B	C
1	modification of p', q'	time dep.	time dep.	no
2	modification of d	time dep.	time dep.	yes
3	modification of d'_p, d'_q	yes	yes	yes
4	modification of r	time dep.	time dep.	yes
5	wrong exponentiation modp, q	prob. $1 - 1/r$	yes	yes
6	modification of S_p or S_q	time dep.	yes	no
7	modification of $q^{-1} \bmod p$	no	yes	no
8	error during comb. of S_p and S_q	no	yes	no
9	faulty signature mod$p$ *and* mod$q$	no	no	yes

The above table is organized as follows. The second column denotes the kind of error which might occur. Column A indicates whether the countermeasure recognizes the induced fault, column B indicates whether the corresponding faulty signature S reveals the secret key and column C says whether the countermeasure is correctly working in the corresponding case. We will briefly comment the observed errors row by row:

input: $m, p, q, d, q^{-1} \bmod p$

randomly choose a short prime r of, e.g., 32 bits
$p' := p * r$
$d'_p := d \bmod ((p-1) * (r-1))$
$q' := q * r$
$d'_q := d \bmod ((q-1) * (r-1))$

$S'_p := (m \bmod p')^{d'_p} \bmod p'$
$S'_q := (m \bmod q')^{d'_q} \bmod q'$

$S_p := S'_p \bmod p$
$S_q := S'_q \bmod q$
$S := S_q + ((S_p - S_q) * q^{-1} \bmod p) * q$

if $((S'_p \bmod r) \neq (S'_q \bmod r))$ **then**
  return(error)
else
  return(S)

output: $S = m^d \bmod (p * q)$

Fig. 4. Shamir's countermeasure.

1. During the computation of p' the value of p may be changed to some value $\tilde{p}$, such that $p' = \tilde{p}r$. Then S'_p is computed correctly modulo r, but not modulo p. If p' is destroyed later, then the check reveals the attack. If a destroyed $\tilde{p}$ will be used for the computation of d'_p then the check will not recognize this relevant fault.
2. If d is changed before the first two reductions this will not be detected but is not security relevant. If d is changed between the first two reductions, this will be recognized by the check.
3. If d'_p or d'_q is destroyed the check will detect this modification.
4. Depending on the time r is destroyed, various things can happen: either the errors will be recognized or they are not security relevant.
5. The destruction of one of the two exponentiations is the classical Bellcore attack. This will be recognized.
6. If S_p will be changed before the combination to S then the check will fail.
7. If $q^{-1} \bmod p$ will be changed then the faulty signature will reveal the key. The check will not recognize the attack.
8. Cf. last row.
9. If the correct signature is destroyed S reveals no information about the key.

4 Practical Fault Attacks Countermeasures for Unprotected Hardware

Within this section we will use the formerly discussed errors to propose a simple practical error model. Hereafter, we propose enhanced countermeasures.

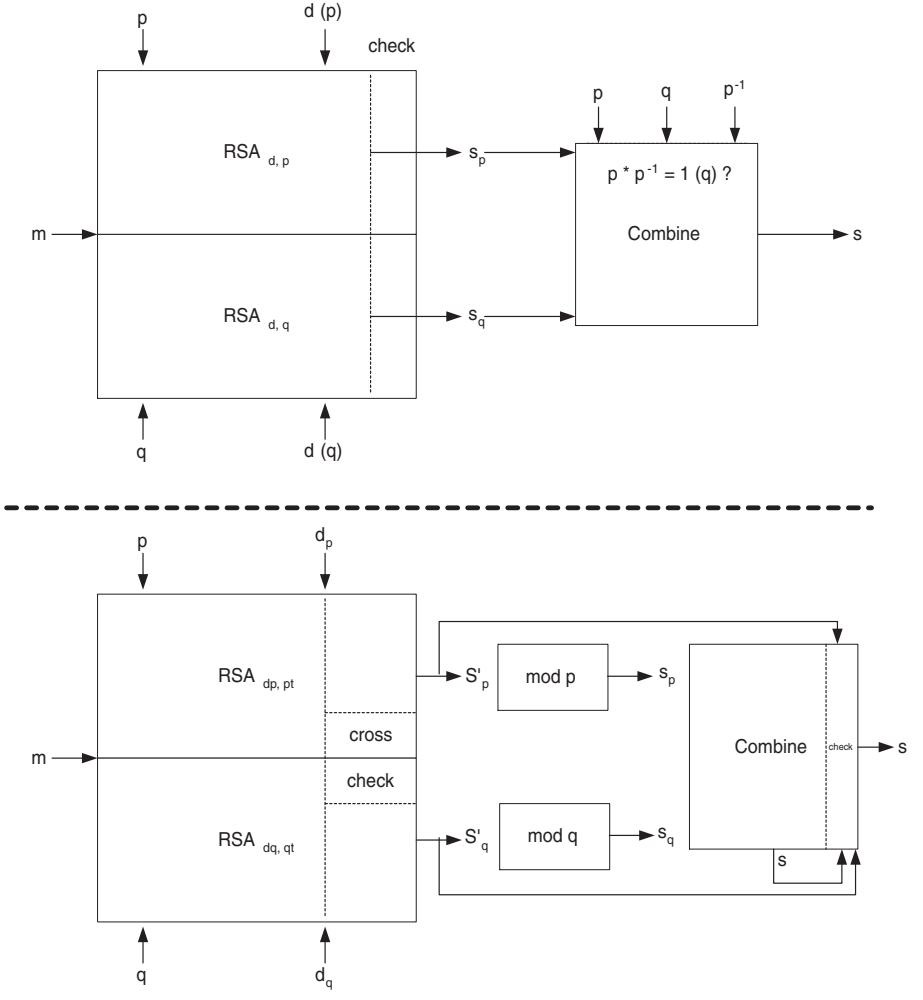


Fig. 5. Information flow during checking.

4.1 Model to Understand Resulting/Possible Faults

From the observed error scenario, we have learned by an extensive data analysis the following facts:

- During the computation, every input value to the RSA signature algorithm can be altered to a value different from the original value.
- During the computation, every variable can be changed.
- The instruction sent to the CPU or a peripheral can be changed.
- The only values to trust, are the values which are stored in ROM or EEPROM.

Armed with this knowledge, we formulated the following checking philosophy:

Check (at least in a probabilistic sense) every computed intermediate result with respect to its correctness by relying on trusted values only.

In a rough sense, this is reflected by figure 5. In this context we adapt the *transient fault model* due to [BDL] which assumes that our power spikes introduce arbitrary errors. Additionally, we assume that the attacker can induce only one spike but at a specific time chosen by himself.

4.2 Software Countermeasures Derived According to the Model

Inspired by the previous section, we developed the following countermeasures (shown in figure 6) to counteract fault attacks. It takes into account that in a practical application only d_p and d_q are given. Also, it avoids the use of the public exponent e, which in real applications is most often not known to the signature software.

We will briefly comment on this algorithm. The check after the CRT combination ensures that S is correctly computed from the data S'_p and S'_q. Therefore, it remains to guarantee that the latter ones are correct. The central check ($S_{pt}^{d_{qt}} \equiv S_{qt}^{d_{pt}} \bmod t$) proves that the two big exponentiations itself where processed in a correct way — assuming that the inputs are not compromised. Note that an erroneous pass of this check can only be due to some very subtle modifications of these input values. Such errors will be intercepted by the first two checking blocks. Finally, we would like to point out the following important advice for a careful implementation: for the two checking blocks the secret parameters d_p and d_q have to be reloaded from a secure area (EEPROM).

4.3 Measurement Results for Enhanced Software Countermeasures

By extensive penetration tests via spikes on the algorithm shown in figure 6 we obtained the following table. It proves empirically the reliability of our software countermeasures.

input: $m, p, q, d_p, d_q, q^{-1} \bmod p$

let t be a short prime number, e.g., 32 bits

$p' := p * t$
$d'_p := d_p + \text{random}_1 * (p-1)$
$S'_p := m^{d'_p} \bmod p'$
if $\neg(p' \bmod p \equiv 0 \ \wedge \ d'_p \bmod (p-1) \equiv d_p)$ **then return**(error)

$q' := q * t$
$d'_q := d_q + \text{random}_2 * (q-1)$
$S'_q := m^{d'_q} \bmod q'$
if $\neg(q' \bmod q \equiv 0 \ \wedge \ d'_q \bmod (q-1) \equiv d_q)$ **then return**(error)

$S_p := S'_p \bmod p$
$S_q := S'_q \bmod q$
$S := S_q + ((S_p - S_q) * q^{-1} \bmod p) * q$
if $\neg((S - S'_p \bmod p \equiv 0) \ \wedge \ (S - S'_q \bmod q \equiv 0))$ **then return**(error)

$S_{pt} := S'_p \bmod t$
$d_{pt} := d'_p \bmod (t-1)$
$S_{qt} := S'_q \bmod t$
$d_{qt} := d'_q \bmod (t-1)$
if $(S_{pt}^{d_{qt}} \equiv S_{qt}^{d_{pt}} \bmod t)$ **then**
  return(S)
else
  return(error)

output: $m^d \bmod (p * q)$

Fig. 6. Practically secured RSA with CRT.

Observed error scenarios	A	B	C
modification of p, p', q, q'	yes	yes	yes
modification of d'_p, d'_q	yes	yes	yes
modification of t	yes	yes	yes
wrong exp. modp, q	prob. $1 - 1/t$	yes	yes
modification of S_p or S_q	yes	yes	yes
modification of $q^{-1} \bmod p$	yes	yes	yes
error during comb. of S_p and S_q	yes	yes	yes
faulty signature mod$p$ *and* mod$q$	prob. $1 - 1/t$	no	yes

Clearly, the probability that an error is undetected is equal to $1/t$. For t a 32-bit integer, this probability is small enough; t can thus be seen as a security parameter.

5 Conclusion

We have shown that the classical Bellcore fault attack is in principal feasible when using completely unprotected microcontrollers. Moreover, it also shows that unskilled implementations of countermeasures are not always reliable. It again answers a question of Kaliski and Robshaw [KR], and shows that these attacks are indeed practical. Our investigation also reveals that one should test any conceivable countermeasures in reality against all possible attack scenarios before trusting them. This was especially done with our newly developed software countermeasures.

Although our software countermeasure seems to be very promising, we are strongly convinced that cryptographic hardware should never be used without appropriate hardware countermeasures in combination with software countermeasures. As a result, we finish with an advice given by Kaliski and Robshaw [KR] from the RSA Laboratories stating that *good engineering practices in the design of secure hardware are essential*.

References

[A] R. Anderson, *Security Engineering*, John Wiley & Sons, New York, 2001.

[AK1] R. Anderson, M. Kuhn, "Tamper Resistance – a cautionary note", *Proc. of 2nd USENIX Workshop on Electronic Commerce*, pp. 1–11, 1996.

[AK2] R. Anderson, M. Kuhn, "Low cost attacks attacks on tamper resistant devices", *Proc. of 1997 Security Protocols Workshop*, Springer LNCS vol. 1361, pp. 125–136, 1997.

[BDL] D. Boneh, R. A. DeMillo, R. Lipton, "On the Importance of Eliminating Errors in Cryptographic Computations" *Journal of Cryptology* **14**(2):101–120, 2001.

[BDHJ$^+$] F. Bao, R. H. Deng, Y. Han, A. Jeng, A. D. Narasimbalu, T. Ngair, "Breaking public key cryptosystems on tamper resistant dives in the presence of transient faults", *Proc. of 1997 Security Protocols Workshop*, Springer LNCS vol. 1361, pp. 115–124, 1997.

[BR] M. Bellare, P. Rogaway, "The exact security of digital signatures — how to sign with RSA and Rabin", *Proc. of EUROCRYPTO '96*, Springer LNCS vol. 1070, pp. 399–416, 1996.

[BS] E. Biham, A. Shamir, "Differential fault analysis of secret key cryptosystems", *Proc. of CRYPTO '97*, Springer LNCS vol. 1294, pp. 513–525, 1997.

[BMM] I. Biehl, B. Meyer, V. Müller, "Differential fault attacks on elliptic curve cryptosystems", *Proc. of CRYPTO '00*, Springer LNCS vol. 1880, pp. 131–146, 2000.

[BMS] J. Blömer, A. May, J.-P. Seifert, personal communication, April 2002.

[CQ] C. Couvreur, J.-J. Quisquater, "Fast decipherment algorithm for RSA public-key cryptosystem", *Electronics Letters* **18**(21):905–907, 1982.

[FS] W. Fischer, J.-P. Seifert, "Note on fast computation of secret RSA exponents", *Proc. of ACISP '02*, Springer LNCS vol. 2384, pp. 136–143, 2002.

[GMO] K. Gandolfi, C. Mourtel, F. Olivier, "Electromagnetic analysis: Concrete results", *Proc. of CHES '01*, Springer LNCS vol. 2162, pp. 255–265, 2001.

[Gu1] P. Gutmann, "Secure deletion of data from magnetic and solid-state memory", *Proc. of 6th USENIX Security Symposium*, pp. 77–89, 1997.

[Gu2] P. Gutmann, "Data Remanence in Semiconductor Devices", *Proc. of 7th USENIX Security Symposium*, 1998.

[HP1] H. Handschuh, P. Pailler, "Smart Card Crypto-Coprocessors for Public-Key Cryptography", *CryptoBytes* **4**(1):6–11, 1998.

[HP2] H. Handschuh, P. Pailler, "Smart Card Crypto-Coprocessors for Public-Key Cryptography", *Proc. of CARDIS '98*, Springer LNCS vol. 1820, pp. 372–379, 1998.

[ISO] International Organization for Standardization, "ISO/IEC 7816-3: Electronic signals and transmission protocols", `http://www.iso.ch`, 2002.

[JLQ] M. Joye, A. K. Lenstra, J.-J. Quisquater, "Chinese remaindering based cryptosystem in the presence of faults", *Journal of Cryptology* **12**(4):241–245, 1999.

[JPY] M. Joye, P. Pailler, S.-M. Yen, "Secure Evaluation of Modular Functions", *Proc. of 2001 International Workshop on Cryptology and Network Security*, pp. 227–229, 2001.

[JQBD] M. Joye, J.-J. Quisquater, F. Bao, R. H. Deng, "RSA-type signatures in the presence of transient faults", *Cryptography and Coding*, Springer LNCS vol. 1335, pp. 155–160, 1997.

[JQYY] M. Joye, J.-J. Quisquater, S. M. Yen, M. Yung, "Observability analysis — detecting when improved cryptosystems fail", *Proc. of CT-RSA Conference 2002*, Springer LNCS vol. 2271, pp. 17–29, 2002.

[KR] B. Kaliski, M. J. B. Robshaw, "Comments on some new attacks on cryptographic devices", *RSA Laboratories Bulletin* **5**, July 1997.

[Kn] D. E. Knuth, *The Art of Computer Programming, Vol.2: Seminumerical Algorithms*, 3rd ed., Addison-Wesley, Reading MA, 1999.

[Koca] O. Kocar, "Hardwaresicherheit von Mikrochips in Chipkarten", *Datenschutz und Datensicherheit* **20**(7):421–424, 1996.

[Koch] P. Kocher, "Timing attacks on implementations of Diffie-Hellmann, RSA, DSS and other systems", *Proc. of CYRPTO '97*, Springer LNCS vol. 1109, pp. 104–113, 1997.

[KJJ] P. Kocher, J. Jaffe, J. Jun, "Differential Power Analysis", *Proc. of CYRPTO '99*, Springer LNCS vol. 1666, pp. 388–397, 1999.

[Ma] D. P. Maher, "Fault induction attacks, tamper resistance, and hostile reverse engineering in perspective", *Proc. of Financial Cryptography*, Springer LNCS vol. 1318, pp. 109–121, 1997.

[MvOV] A. J. Menezes, P. van Oorschot, S. Vanstone, *Handbook of Applied Cryptography*, CRC Press, New York, 1997.

[NR] D. Naccache, D. M'Raihi, "Cryptographic smart cards", *IEEE Micro*, pp. 14–24, 1996.

[Pe] I. Petersen, "Chinks in digital armor — Exploiting faults to break smartcard cryptosystems", *Science News* **151**(5):78–79, 1997.

[Ro] T. Rosa, "Future Cryptography: Standards are not enough", *Proc. of Security and Protection of Information 2001*, pp. 237–245, 2001.

[RSA] R. Rivest, A. Shamir, L. Adleman, "A method for obtaining digital signatures and public-key cryptosystems", *Comm. of the ACM* **21**:120–126, 1978.

[SQ] D. Samyde, J.-J. Quisquater, "ElectroMagnetic Analysis (EMA): Measures and Countermeasures for Smart Cards", *Proc. of Int. Conf. on Research in Smart Cards, E-Smart 2001*, Springer LNCS vol. 2140, pp. 200–210, 2001.

[Sh] A. Shamir, "Method and Apparatus for protecting public key schemes from timing and fault attacks", U.S. Patent Number 5,991,415, November 1999; also presented at the rump session of EUROCRYPT'97.

[YJ] S.-M. Yen, M. Joye, "Checking before output may not be enough against fault-based cryptanalysis", *IEEE Trans. on Computers* **49**:967–970, 2000.

[YKLM1] S.-M. Yen, S.-J. Kim, S.-G. Lim, S.-J. Moon, "RSA Speedup with Residue Number System immune from Hardware fault cryptanalysis", *Proc. of the ICISC 2001*, Springer LNCS vol. 2288, pp. 397–413, 2001.

[YKLM2] S.-M. Yen, S.-J. Kim, S.-G. Lim, S.-J. Moon, "A countermeasure against one physical cryptanalysis may benefit another attack", *Proc. of the ICISC 2001*, Springer LNCS vol. 2288, pp. 414–427, 2001.

[ZM] Y. Zheng, T. Matsumoto, "Breaking real-world implementations of cryptosystems by manipulating their random number generation", *Proc. of the 1997 Symposium on Cryptography and Information Security*, 1997.

Some Security Aspects of the Mist Randomized Exponentiation Algorithm

Colin D. Walter*

Comodo Research Laboratory
10 Hey Street, Bradford, BD7 1DQ, UK
colin.walter@comodo.net

Abstract. The Mist exponentiation algorithm is intended for use in embedded crypto-systems to provide protection against power analysis and other side channel attacks. It generates randomly different addition chains for performing a particular exponentiation. This means that side channel attacks on RSA decryption or signing which require averaging over a number of exponentiation power traces become impossible. However, averaging over digit-by-digit multiplication traces may allow the detection of operand re-use. Although this provides a handle for an attacker by which the exponent search space might be considerably reduced, the number of possible exponents is shown to be still well outside the range of feasible computation in the foreseeable future.

Keywords: Randomary exponentiation, Mist exponentiation algorithm, division chains, addition chains, power analysis, DPA, DEMA, blinding, smartcard.

1 Introduction

Because smartcards have very limited scope for the inclusion of physical security measures, the prevalence of side channel leakage from embedded cryptographic systems creates the need for new algorithms which can be implemented in more secure ways than those currently in use. This is particularly true for exponentiation, which is a major process in many crypto-systems such as RSA, Diffie-Hellman and ECC. Initial power attacks required averaging over a number of exponentiations in order to reduce the effects of noise and dependence on uninteresting data [3], [4]. Although the necessary alignment of power traces can be made more difficult by the insertion of obfuscating, random, non-data-dependent operations, the data transfers between operations usually reveal the commencement of every long integer or elliptic curve operation very clearly in each individual trace. So it is usually possible to perform the averaging process and mount an attack to extract meaningful secret data. Fortunately, attacks which require such averaging can be defeated by modifying the exponent d to

* Work started while the author was at the Computation Department, UMIST, Manchester, UK

B.S. Kaliski Jr. et al. (Eds.): CHES 2002, LNCS 2523, pp. 276–290, 2003.

$d+rg$ where r is a random number and g is the order of the (multiplicative) semigroup in which the exponentiation is performed [3]. This results in a different exponentiation procedure being performed every time.

However, the author showed recently [9] that there were strong theoretical grounds for believing that, given the right monitoring equipment [6,7,1], it would be possible to break the normal m-ary exponentiation method [2] and related sliding windows techniques using a single exponentiation. This method averages over digit-by-digit products instead and relies on being able to use such averaged traces to recognise the same operands being re-used over and over. These operands are pre-computed powers of the initial text, and their use reveals the secret exponent digit. Such an attack requires no knowledge of the modulus, the input text or the output text. It would render useless the choice of $d+rg$ as a counter-measure, even for the case of $m = 2$, namely the standard binary "square-and-multiply" algorithm.

Random modifications have been proposed for the *algorithm* as well as the *arguments* of exponentiation in order to overcome these problems. The main suggestions are suitable where the multiplicative (additive) inverse is easily computed, such as in elliptic curve systems. Oswald and Aigner [5] have given one such example. For integer RSA, a novel exponentiation algorithm[1] called "MIST" was presented at RSA 2002 [11]. This seems to avoid all of the above-mentioned pitfalls and inverses are not required. It is more time efficient than the standard binary method when squares and multiplies have equal computational cost and it is comparable in space usage to 4-ary exponentiation. It can also be combined with any counter-measures which modify the arguments. The algorithm relies on the generation of random addition chains [2] which determine the operations to be performed, and it is based on previous work by the author [8] for finding efficient exponentiation schemes using division chains.

The MIST algorithm was created to defeat power analysis attacks which are able to detect the re-use of arguments. [11] considered only efficiency issues for the algorithm. The main aim here is to look at security issues, and, in particular, to establish that knowledge of operand re-use does not significantly reduce the effectiveness of the algorithm against power analysis or other similar attacks. Although information about operand re-use provides a handle which prunes a search tree for exponents considerably, it is still computationally infeasible to recover a secret RSA key in this way unless very significant secret data is obtained from other sources. A more likely scenario is that the attacker can only distinguish squares from multiplies. Then the search space is vastly larger, and so the algorithm appears to provide even more security.

2 The MIST Algorithm

For notation, let us assume that plaintext $P = C^D$ has to be computed from ciphertext C and secret key D. m will always represent a "divisor" in the sense

[1] Comodo Research Lab has filed a patent application is respect of this [10].

of [8], and d a residue modulo m, but here these are viewed as base and digit values respectively in a representation of D. A set of allowable bases m is chosen in advance (it will be $\{2, 3, 5\}$ here), and an associated table of addition chains for raising to the power m is stored in memory. Several variables are used: there are at least three for long integers which contain powers of C, namely Q, $TempC$ and P. Of these, $TempC$ is for temporary storage when Q is being raised to the power m, and so does not occur explicitly in the following code, and P contains the accumulating required output. D is updated to contain the power to which Q still has to be raised before the exponentiation is complete.

The Mist Exponentiation Algorithm [11]

{ Pre-condition: $D \geq 0$ }
Q ← C ;
P ← 1 ;
While D > 0 do
Begin
  Choose a random "base" m ;
  d ← D mod m ;
  If d ≠ 0 then P ← $Q^d \times P$;
  Q ← Q^m ;
  D ← D div m ;
  { Loop invariant: $C^{D.Init} = Q^D \times P$ }
End ;
{ Post-condition: $P = C^{D.Init}$ }

Example. For D=235, m=3 yields d = (235 mod 3) = 1 and reduces D to (235 div 3) = 78. Then m=2 would give d=0 and D=39. Next, m=5 produces d=4 and D=7; m=2 gives d=1 and D=3; m=3 generates d=0 and D=1. Then, finally, m=2 yields d=1 and so D becomes 0. The pairs (m, d) are:
(3,1), (2,0), (5,4), (2,1), (3,0) and (2,1).
The corresponding powers of C contained in the variables (Q, P) are then:
(C^1, C^0); (C^3, C^1); (C^6, C^1); $(\{C^6\}^5, \{C^6\}^4 C^1) = (C^{30}, C^{25})$;
$(\{C^{30}\}^2, \{C^{30}\}^1 C^{25}) = (C^{60}, C^{55})$; $(\{C^{60}\}^3, \{C^{60}\}^0 C^{55}) = (C^{180}, C^{55})$;
$(\{C^{180}\}^2, \{C^{180}\}^1 C^{55}) = (C^{360}, C^{235})$. □

When the base set consists of the single base 2, the method simplifies to the binary square-and-multiply algorithm in which the least significant exponent bit is processed first. In general, for fixed m, the algorithm simplifies to m-ary exponentiation but performed from right to left rather than from left to right. Since the base m is varied randomly here, the process might reasonably be called "*random-ary exponentiation*". Space and time efficiency were shown in [11] to be comparable with 4-ary exponentiation. For application to integer RSA, the multiplication is the operation in the multiplicative group of residues for the chosen modulus. Explicit mention of the modulus is not necessary. Termination is guaranteed because only base choices greater than 1 are allowed, and so D decreases on every iteration. Correctness is easily established using the loop invariant in terms of the initial value $D.Init$ of D.

The choices of base set and associated addition chains for each base/digit pair (m, d) are made with security and efficiency in mind. In particular, for efficiency the choice of addition chain for raising to the power m always includes d so that the computation of Q^m provides Q^d *en route* at little or no extra cost. Thus, these two power computations are not performed independently and consecutively, as might be implied by the code. They are to be implemented so that Q^m uses all the work done already to compute Q^d. So, in the case of RSA, the main cost of a loop iteration is only the cost of computing Q^m plus the conditional extra multiplication involving P.

The random choice of base values from a pre-chosen fixed set achieves different exponentiation schemes on successive runs and so makes impossible the usual averaging process required for differential power or electro-magnetic analysis (DPA/DEMA) [4,6].

Unlike the case for m-ary exponentiation, by reversing the direction of processing the exponent, *both* arguments in the conditional product are changed for every multiplication. In general, because the powers of C are always increasing, no power of C is repeatedly re-used during the exponentiation. So the attack described in [9] on a single exponentiation is inapplicable in its current form.

For convenience, the processing of the exponent D is presented as being performed within the main loop. For security reasons, it should probably be scheduled differently. The illustrated processing order may be less secure from the point of DPA or DEMA because it can reveal the random choice of the local base m, which should remain secret. Instead, the selection of the base and associated addition chain instructions can be performed by the CPU on-the-fly while the exponentiation is performed in parallel by a crypto co-processor, or it can be done in advance and stored when there is no co-processor. At any rate, these computations should be scheduled so as not to reveal the end points of each iteration of the main loop. Otherwise, the number and type of long integer operations during the loop iteration may leak enough information about the values of m and d, enabling D to be reconstructed. This paper shows how such data might be used to determine possible values for D.

3 The Base Choice and Addition Sub-chains

A typical safe set of allowable bases is {2,3,5}. The full list of minimal addition sub-chains for these bases is given in Table 1. For example, the third case there corresponds to computing C^5 using the three multiplications $C^1 \times C^1 = C^2$, $C^1 \times C^2 = C^3$ and $C^2 \times C^3 = C^5$. The first three addition chains provide C^d when digit d is 0, 1, 2 or 3: for $0 < d < m$ the chain already contains the value of d, while the case $d = 0$ requires no multiplication and so 0 does not need to appear. The last addition chain can be used when $d = 4$. *Minimal* here means that any other addition chains which give a power equal to the base are longer. The subchains in Table 1 are minimal. To achieve the fastest exponentiation, longer chains are usually excluded, but they might improve security.

Table 1. The Minimal Sub-chains.

1+1=2	for base 2 with any digit d
1+1=2, 1+2=3	for base 3 with any digit d
1+1=2, 1+2=3, 2+3=5	for base 5 with any digit except 4
1+1=2, 2+2=4, 1+4=5	for base 5 with any digit except 3

Table 2. A Choice for the Digit Sub-chains.

(m, d)	Multiplication Instructions
$(2, 0)$	(111)
$(2, 1)$	(112, 133)
$(3, 0)$	(112, 121)
$(3, 1)$	(112, 133, 121)
$(3, 2)$	(112, 233, 121)
$(5, 0)$	(112, 121, 121)
$(5, 1)$	(112, 133, 121, 121)
$(5, 2)$	(112, 233, 121, 121)
$(5, 3)$	(112, 121, 133, 121)
$(5, 4)$	(112, 222, 233, 121)

Table 3. ([11], Tables 6.2 and 6.3.) The limit probabilities $p_{m,d}$ of the base/digit pairs (m, d) and p_m for each base m.

(m, d)	0	1	2	3	4	p_m
2	0.3537	0.2757	-	-	-	$p_2 = 0.6294$
3	0.1826	0.0212	0.0244	-	-	$p_3 = 0.2283$
5	0.0936	0.0124	0.0110	0.0127	0.0126	$p_5 = 0.1423$

There is no instruction which updates the value of P in these addition sub-chains, but it can be represented explicitly using the following notation. Suppose the registers are numbered 1 for Q, 2 for *TempC* and 3 for P. Then the subchains can be stored as sequences of triples $(ijk) \in \{1, 2, 3\}^3$, where (ijk) means read the contents of registers i and j, multiply them together, and write the product into register k. In particular, P will always be updated using a triple of the form $(i33)$ and 3 will not appear in triples otherwise. Now, adding in the instruction for updating P yields the list of subchains given in Table 2 as one possibility. It contains one representative for each base/digit pair (m, d). Other choices are possible. Such a table requires only a few bytes of storage.

The way in which the base is chosen from the allowable set has efficiency and security implications. In [11] it was shown that the following choice provided efficiency better than the binary method and nearly as good as the 4-ary method: (Here the function *Random* returns a fresh, random real in the range [0,1].)

```
m ← 0 ;
If Random < 7/8 then
    If D mod 2 = 0 then m ← 2 else
    If D mod 5 = 0 then m ← 5 else
    If D mod 3 = 0 then m ← 3 ;
If m = 0 then
Begin
    p ← Random ;
    If p < 6/8 then m ← 2 else
    If p < 7/8 then m ← 5 else
                    m ← 3 ;
End ;
```

The resulting probabilities p_m of each base m and $p_{m,d}$ of each base/digit pair (m, d) occurring in the representation of D are given in Table 3[2]. They will be used later to assess whether certain attacks are feasible. From them it is possible to work out the average number of bases used in a MIST exponentiation scheme:

Theorem 1. ([11], Thm. 7.2) *With the choices above, the average number of digits in a* MIST *representation for D is approximately*[2] $0.7566 \times \log_2 D$.

4 The Sequence of Addition Chain Values

We now turn away from the powers of C generated during an exponentiation and concentrate on the integers contained in the corresponding addition chain. The individual addition sub-chains for each base can be formed easily into an addition chain which describes a complete exponentiation scheme for D. In terms of the triples in Table 2, the sub-chain lists just need to be concatenated. Each value is associated with one of the variables Q, $TempC$ or P according to the register in which the corresponding power of C is to be written. We will work with addition chains containing this extra detail. If S is the final value associated with Q at the end of one subchain, then, by applying the instructions listed in Table 2, the values computed in the next subchain are those listed in Table 1 multiplied by S, together with any which occurs for P.

Reconstruction of the sequence of digits and hence determination of the secret exponent is investigated using knowledge of which of these addition chain elements are equal, and which share equal summands. The following theorems will be used to show that, for the most part, we only need to look locally in the chain for such equality or sharing. With fairly minimal and reasonable restrictions on the choices of base set and associated addition sub-chains, these theorems hold much more generally.

Theorem 2. *The integers (i.e. exponents) associated with Q and P at the start of successive subchains form monotonically increasing sequences. That for Q is*

[2] The figures here are corrected after a minor bug in the software for [11].

strictly increasing and strictly dominates that for P. At the start of each subchain, Q is associated with the largest integer in the addition chain up to that point.

Proof. Initially Q is associated with 1 and the other registers with 0. So the domination property holds for the first values. Thereafter, suppose Q, *TempC* and P contain the S, T and Uth powers of C respectively at the start of a subchain for (m, d). Assume $T < S$ and $U < S$. The next values associated with Q and P are $m{\times}S$ and $U{+}d{\times}S$. Then $d < m$ means $U{+}d{\times}S < S{+}(m{-}1){\times}S = m{\times}S$. So the next value for Q is larger than the next for P. Hence the sequence for Q dominates that for P. Since $1 < m$, $S < m{\times}S$ and the next Q is larger than it was at the beginning of the subchain. So that sequence is strictly increasing. Moreover, as $U \leq U{+}d{\times}S$ the sequence for P is also increasing monotonically.

The exponents associated with *TempC* are multiples of S since the initial value T for *TempC* is unused, and the value U of P is only used to update P. So, as there are no other operations, these exponents only involve integers obtained *en route* from S to the next Q value, and are strictly smaller than it. Thus, at the start of each subchain, Q does indeed contain the largest power so far calculated. □

Theorem 3. *Except for initialisation and the calculation of the first non-trivial value for P, the addition chain contains no sum result more than once.*

Proof. There are two cases to consider. First, suppose some integer is recomputed in two different subchains. Assume the second of these subchains initially has integers S and U in Q and P respectively. Any updating operation in this subchain or any future subchain gives an integer which is a linear combination $\sigma S{+}\pi U$ for integers $\sigma > 0$ and $\pi \in \{0, 1\}$. This only creates integers $\geq S$. Pick σ and π to give the duplicated value. By the previous theorem, updating operations in previous subchains only created integers at most S. Hence S is the recomputed value and so $\sigma S{+}\pi U = S$. Since $S > 0$ and $U \geq 0$, this can only hold if $\sigma \leq 1$. Multiplications have at least two arguments, so $\sigma{+}\pi \geq 2$. Hence $\sigma = \pi = 1$, from which $U = 0$. This solution occurs only when the digit d is 1 and P is updated to its first non-trivial value. So the value recomputed in the second subchain is uniquely determined. Moreover, since by the previous theorem the values in Q are strictly increasing and represent the largest integers so far calculated, the first computation of S is only in the immediately preceding subchain. S exceeds the initial value $U = 1$ in P. So S is calculated just once in the first of these two subchains.

Now consider the case where a value is recomputed within a single chain. Such recomputation must involve an updating of P because the operations which write to Q and *TempC* generate a strictly increasing sequence. We use the same notation again. The value of the updated P is $U{+}dS$ where $d > 0$ is the digit. All other subchain additions output an integer of the form σS for some integer $\sigma > 1$. So, choosing σ from another equal value gives $\sigma S = U{+}dS$ and hence $U = (\sigma{-}d)S$ where $0 \leq U < S$. This is impossible in integers unless $U = 0$ and $\sigma = d$. So P is being given its first non-trivial value, and dS is recomputed. We

have $d > 1$ in this case since the computation of dS is done in the same subchain as P is updated. □

In fact, the re-computation of the same power of C should never occur: a good implementation should avoid the useless multiplication by 1, and should also avoid the apparent need to write the initial non-trivial value to P by using the value written to Q or *TempC* and renaming that register as P. The corresponding entries in the addition chain can then be omitted.

Example. Continuing with the example from Section 2, the sequences of exponents for Q and P are 1, 3, 6, 30, 60, 180, 360 and 0, 1, 25, 55, 235 respectively. The first non-zero value of P, namely 1, appears already in the sequence for Q, but there are no other repetitions.

5 Re-use of Summands

Our main assumption in the first threat model is that the attacker can recognise the re-use of operands. Such re-use occurs when members of the addition chain share summands. So we need to know when this can happen, i.e. when two sums in the addition chain share a common input.

Theorem 4. i) *No integer different from the first non-zero value for P is used as a summand in more than three addition chain members. Addition chain elements which share such common summands all belong to the same digit subchain. They all lie within a sequence of at most four consecutive operations, and at most one of those with the shared summand is a doubling. If three sequential operations share such a common summand, then the digit associated with the subchain to which they all belong is non-zero.*

ii) *The first non-zero value for P may be used as a summand in up to four different addition chain operations. All but the last of the sums which use this value of P belong to the same subchain, while the last (which updates P to its second non-trivial value) belongs to a different subchain and may be arbitrarily many operations after the initial case.*

Proof. First consider the summands used in the sums on either side of a digit sub-chain boundary where the Q value is S. All operands below the boundary are less than S because S is the largest value computed up to that point, and it has not been used as an operand yet. Above the boundary, all operands are S or a multiple thereof with the single exception of the previous value for P when it is next updated. So, if there are two equal summands belonging to different digit sub-chains, they must be equal to a value of P.

However, from Theorem 3 we know that, apart from the first non-zero value, values of P are distinct from each other and from values in Q or *TempC*. So, as Q and *TempC* are computed from previous values of Q and *TempC*, their arguments cannot have values equal to those acquired by P unless those arguments are equal to the first non-zero value of P. Thus, with only that possible exception, each value of P is used at most once as a summand, namely in the next sum which

updates its value. Consequently, two equal summands belonging to different digit sub-chains must actually be equal to the *first* non-zero value of P.

Hence, in case (i) for arguments different from the first non-zero value of P, equal summands appear within the same digit sub-chain. Thus they lie within a sequence equal to the longest such sub-chain, which is 4 here. Each doubling involves different operands (otherwise there would be no point in performing the subsequent doublings) and so the set of sums with a shared summand will contain at most one doubling. Checking through all the subchains given in Table 1, no operand is used in more than two operations. Moreover, such operations are adjacent and at most one of them is a square. So, when the sum which updates P is included in the middle of the sub-chain, at most three operations can occur using the same operand and, for our choice of insertion points, they are sequential. Of course, this only occurs when $d \neq 0$.

Now take case (ii) where the first non-trivial value for P is the operand under consideration. As with the other case, this operand value is used in at most two additions for updating Q and $TempC$. It is also used in at most two additions which update P. The first of these may have been optimised out, replacing the multiplication by 1 with an initialisation. This addition and those involving Q and $TempC$ all occur in the same digit sub-chain, as before. The last use, namely the second to update P, occurs for a subsequent sub-chain after arbitrarily many bases for each of which the digit is 0. □

Example. Continuing with the example from Section 2, the first three pairs $(m, d) = (3, 1)$, (2,0) and (5,4) generate the instructions 112, 133, 121, 111, 112, 222, 233, 121. These produce:

$$\begin{array}{llll} TempC & = C^1 \times C^1 & = C^2 \ ; & P & = C^1 \times C^0 & = C^1 \ ; \\ Q & = C^1 \times C^2 & = C^3 \ ; & Q & = C^3 \times C^3 & = C^6 \ ; \\ TempC & = C^6 \times C^6 & = C^{12} \ ; & TempC & = C^{12} \times C^{12} & = C^{24} \ ; \\ P & = C^{24} \times C^1 & = C^{25} \ ; & Q & = C^6 \times C^{24} & = C^{30} \ . \end{array}$$

Operand C^1 is the first non-trivial value of P and it is used 4 times: the first three lie in the subchain for $(m, d) = (3, 1)$ and the last occurs in the subchain for $(m, d) = (5, 4)$. The intermediate subchain has $d = 0$.

6 Identifying the Digit Sub-chains

In order to describe detailed operand sharing in a sequence of operations some further notation is needed. Let (123)(34) mean that in a list of exactly 4 operations, the first three share a common operand, the third and fourth share a different common operand, and no other operations in that list share a common operand. So the numbers in the cycles represent positions in the sequence of operations, starting at 1, and two operations will share a common operand if, and only if, their position numbers both belong to a common cycle in the list. Since a square or a doubling shares an operand with itself, the number of each square or doubling appears twice in its cycle, as in (112). Also, an operation which does not share operands with any other operation (or itself) will appear

in a cycle on its own, as in (2). However, there are no cases of this here. With this notation, the subchains listed in Table 2 share operands as in Table 4.

Table 4. Operand Sharing within each Digit Sub-Chain.

(m, d)	Operand Sharing
$(2, 0)$	(11)
$(2, 1)$	(112)
$(3, 0)$	(112)
$(3, 1)$	(1123)
$(3, 2)$	(113)(23)
$(5, 0)$	(112)(23)
$(5, 1)$	(1123)(34)
$(5, 2)$	(113)(234)
$(5, 3)$	(112)(24)(34)
$(5, 4)$	(114)(22)(34)

Now assume that the operand sharing pattern is known for the complete addition chain. By Theorem 3, except for the use of the first non-trivial value of P, operands which are equal in the addition chain are equal because they were explicitly selected equal in the sub-chain construction. By Theorem 4, we know that the last use of the exceptional value of P as an operand is to update P, and so it is not a square. Hence the squares in the addition chain are exactly those expected from the structure of the component sub-chains.

It is now mostly straightforward to deduce what the individual subchains are, and hence the sequence of bases and digits: each square (doubling) denotes the start of a new sub-chain with the exception of those which are the second operation in a subchain for $(m, d) = (5, 4)$. When this case occurs, there is operand sharing between the first and fourth operations of the sub-chain. This is expressed in the pattern (114). According to Theorem 4, there is normally no sharing of operands between different sub-chains. Hence, when the pattern (114) is not observed, we know that normally both squares mark the start of different sub-chains.

The only possible exception is if the shared operand in (114) is the first non-zero value taken by P. Then the sharing pattern (114)(22)(34) for (5,4) must ambiguously represent two division sub-chains which have lengths 1 and 3 respectively. The second subchain has pattern (11)(23) (when the operations are re-numbered from 1 to 3). However, that pattern does not correspond to any occurring in Table 4. So this case cannot arise. Hence:

Theorem 5. *The pattern of operand re-use in an addition chain determines the boundaries of each digit sub-chain uniquely.*

In practice, this partitioning is performed by identifying the doublings first and then writing down the patterns for operand sharing between the operations

within each partition. If the pattern (11)(23) emerges, then its partition needs to be merged with the previous one.

Example. In the same example as before, the operand sharing pattern is (11237) (44)(558)(66)(78)(9 9 10)(11 11 12)(13 13 14). Partitioning this before each square yields (11237); (44); (558); (66)(78); (9 9 10); (11 11 12); (13 13 14). Re-numbering to make each sub-chain start with instruction 1 gives (11237); (11); (114); (11)(23); (112); (112); (112). As 7 exceeds the length of the longest sub-chain, it must represent the updating of P to its second non-trivial value. So we delete it to obtain sharing only within sub-chains. Also, we must merge (114) and (11)(23) since the latter is not a pattern in Table 4. This produces (1123); (11); (114)(22) (34); (112); (112); (112) from which we can extract possible choices for the pairs (m, d): first (3,1); (2,0); (5,4) and then three occurrences of $(2, 1)$ or $(3, 0)$.

It is evident from Table 4 that every (m, d) has a distinct pattern of operand sharing except for two: $(2, 1)$ and $(3, 0)$ have identical patterns (112). Thus operand sharing almost determines the sequence of pairs (m, d):

Theorem 6. *The pattern of operand re-use in an addition chain determines the sequence of pairs* (m, d) *up to an ambiguity between* $(2, 1)$ *and* $(3, 0)$.

Theorem 7. *The average number of exponents with addition chains that have the same operand sharing pattern as one for* D *is at least* $D^{1/3}$.

Proof. By Theorem 6, base/digit pairs can be derived in most cases. The only ambiguities occur between the cases $(2, 1)$ and $(3, 0)$. Assuming successive base choices are independent, then the probability of an ambiguous case is $p_{2,1}+p_{3,0} = 0.4583$. Hence almost every other subchain has two possible choices for the base/digit pair. By Theorem 1, an average exponent contains $0.7566\times \log_2 D$ subchains. Hence the expected number of different matching exponents is about $2^{0.7566\times \log_2 D\times 0.4583} = D^{0.7566\times 0.4583} = D^{0.347}$ □

Remarks. i) The choice of base is not constrained here. A deterministic selection of, for example, a base which exactly divides the current value of D would place structural constraints which would further reduce the possible choices for D.
ii) Successive base choices are not independent, but this makes only a marginal difference to the exponent 0.347.
iii) We have assumed different choices of (m, d) lead to different values for D. This is almost always true, and makes no practical difference to the number of exponents that need to be considered in an attack.

Example. In the previous example, operand sharing gave 8 choices for the division chain. The first three (m, d)s yield $(Q, P) = (C^{30}, C^{25})$. The next (2,1) or (3,0) produces (C^{60}, C^{55}) or (C^{90}, C^{25}). The following (2,1) or (3,0) doubles the possibilities to (C^{120}, C^{115}), (C^{180}, C^{115}), (C^{180}, C^{55}) or (C^{270}, C^{25}). The last alternative theoretically doubles this number, but (3,0) is impossible because the

final (top) digit d must be non-zero. So, applying (2,1), the final output P is one of C^{235}, C^{295}, C^{235} or C^{295}. The example captures one of the few sources of repeated values, which arises from the property 2+1 = 3+0.

7 The Operand Sharing Search Space

Let us assume that the MIST algorithm is being used because re-use of operands can be detected, thereby making the standard algorithms unsuitable. So the main assumption in the threat model here is that identical operands can be detected, perhaps by averaging the power traces of digit-by-digit products using the method given in [9], or by observing different addresses being sent along the internal bus.

It is straightforward to show that, with a negligibly small number of exceptions, operand sharing in the addition chain occurs if, and only if, operand sharing occurs in the exponentiation. This is because $C^i = C^j \Rightarrow i = j$ holds almost always. Since there are only some $0.7566 \log_2 D$ sub-chains in which to check operand sharing, exceptions are unlikely to occur, even in a complete addition chain. Then Theorems 6 and 7 provide:

Theorem 8. *If an attacker can determine operand re-use from side channel leakage, then he can almost certainly deduce the sequence of pairs* (m, d) *used in the* MIST *exponentiation scheme up to an ambiguity between* $(2, 1)$ *and* $(3, 0)$. *This reduces the search space for the correct exponent to about* $D^{1/3}$.

The known ratio $p_{3,0} : p_{2,1} = 0.1826 : 0.2757$ enables the choices for D to be ranked with those closest to this ratio being selected first. Then on average fewer exponents would need to be investigated before the correct one is determined. However, for a 384-bit exponent, say, the number of choices given by the theorem is still around 2^{133} and that for a 512-bit exponent is around 2^{177}. These are reasonable minimum choices for when the Chinese Remainder Theorem is or is not used. For RSA, both cases are computationally infeasible for the foreseeable future. However, for ECC a typical 192-bit key would really be unsafe. Of course, key lengths can still be increased for both of these if necessary.

In the case of the m-ary method, knowledge of operand sharing enables the exponent to be deduced immediately without any further calculations [9]. So the MIST algorithm is much stronger against such an attack.

8 S&M Chains

A much weaker threat model is that the attacker can distinguish between squares (S) and multiplies (M). The first main task he has is to parse correctly the word created from the alphabet $\{S, M\}$ which is generated by the operations of the exponentiation scheme. We will call the word an *S&M chain*. It must correspond to a division chain [8]. The patterns of the S&M subchains corresponding to each pair (m, d) are listed in Table 5 and their probabilities in Table 6.

Table 5. The S&M Sub-chains for each Pair (m, d).

(2,0)	S	(5,0)	SMM
(2,1)	SM	(5,1)	$SMMM$
(3,0)	SM	(5,2)	$SMMM$
(3,1)	SMM	(5,3)	$SMMM$
(3,2)	SMM	(5,4)	$SSMM$

Table 6. The S&M Sub-chain Probabilities.

p_S	$= p_{2,0}$	$=$	0.3537
p_{SM}	$= p_{2,1} + p_{3,0}$	$=$	0.4583
p_{SMM}	$= p_{3,1} + p_{3,2} + p_{5,0}$	$=$	0.1393
p_{SMMM}	$= p_{5,1} + p_{5,2} + p_{5,3}$	$=$	0.0361
p_{SSMM}	$= p_{5,4}$	$=$	0.0126

Theorem 9. *Suppose squares and multiplies can be distinguished, but not individual reuse of operands. Then the average number of exponents which can generate the same sequence of squares and multiplications as a given one for D is bounded below by $D^{3/5}$.*

Proof. As before, the occurrences of S determine almost all of the subchain boundaries exactly. The exception is the case $(5, 4)$ for which $SSMM$ may split as S and SMM. Suppose we perform this split. Then the number of subchains is increased by a factor of $1+p_{SSMM}$ and the probabilities of the minimal such S&M sequences are then:

p'_S	$= (p_S+p_{SSMM})/(1+p_{SSMM})$	$=$	0.3618
p'_{SM}	$= p_{SM}/(1+p_{SSMM})$	$=$	0.4526
p'_{SMM}	$= (p_{SMM}+p_{SSMM})/(1+p_{SSMM})$	$=$	0.1500
p'_{SMMM}	$= p_{SMMM}/(1+p_{SSMM})$	$=$	0.0356

Assume, for simplicity, that the successive choices of base are independent[3]. Then the number of choices for the base/digit pair underlying $SMMM$ is 3, that for SM is 2, that for S is 1. If SMM is preceded by M, it corresponds to a complete subchain and there are 3 choices for it. However, if SMM is preceded by S, the two can be merged to form $SSMM$, giving 4 choices. Taking into account the proportion of SMMs derived from $SSMM$, p'_{SMM} breaks up into $p'_{SSMM} = p'_{SMM}(p_{SSMM}+p_S p_{SMM})/(p_{SMM}+p_{SSMM}) = 0.0611$ for the latter case and $p'_{MSMM} = p'_{SMM} - p'_{SSMM} = 0.0889$ for the former. Then the average number of ways of selecting a base/digit pair from an S&M subsequence (including repartitioning $SSMM$) is $1^{p'_S} \times 2^{p'_{SM}} \times 3^{p'_{SMMM}+p'_{MSMM}} \times 4^{p'_{SSMM}} = 1.7079$.

[3] Overall, SMM occurs with probability 0.1393, but after S its probability is 0.1410.

An average exponent contains $0.7566 \log_2 D$ subchains, and $1+p_{SSMM}$ times more minimal S&M subsequences. Hence the expected number of different matching exponents is about $1.7079^{(1+p_{SSMM}) \times 0.7566 \log_2 D} = D^k$ where $k = (1+p_{SSMM}) \times 0.7566 \log_2(1.7079) \approx 0.5916 \approx \frac{3}{5}$. □

Even more so than in the case of known operand sharing, the above demonstrates that recovery of D is well outside the reach of an attacker when only the sequence of squares and multiplies leaks for individual exponentiations.

Clearly, different choices of bases and addition sub-chains will provide different probabilities and hence may increase or decrease the strength of a given attack. Care is therefore necessary in these choices. For example, the present choices have been consciously selected to make the probabilities $p_{2,1}$ and $p_{3,0}$ the highest after $p_{2,0}$, which itself needs to be high to provide the requisite efficiency. This decreases the effectiveness of the above attacks. Moreover, the choice of sub-chains means there are no long S&M sequences with a unique base/digit interpretation which would prune the search space for matching exponents down to a computationally feasible proposition.

9 Conclusion

The "MIST" randomary exponentiation algorithm has a variety of features which make it much more resilient to attack by differential power or electro-magnetic analysis than the normal m-ary or sliding window methods. MIST uses randomly different multiplication schemes on every run in order to avoid the averaging which is normally required for such side channel attacks to succeed.

The algorithm also avoids wide re-use of multiplicands within a single exponentiation, thereby defeating some other potentially more powerful attacks. Knowledge of such operand re-use reduces the search space to about $D^{1/3}$ for an exponent D, but this leaves an infeasible quantity of computing for standard RSA applications. Furthermore, knowledge of only the sequence of squares and multiplies reduces the search space to around $D^{3/5}$. These search spaces might be reduced if an effective way can be found to share data deduced from different exponentiations. This is an open problem, but the possible danger should be adequately protected against by standard exponent blinding.

In consequence, the algorithm appears to be much safer than the standard binary, m-ary or sliding windows techniques against current state-of-the-art in DPA and DEMA side channel attacks, yet it makes no greater use of either space or time.

References

1. K. Gandolfi, C. Mourtel & F. Olivier, *Electromagnetic Analysis: Concrete Results*, Cryptographic Hardware and Embedded Systems – CHES 2001, Ç. Koç, D. Naccache & C. Paar (editors), Lecture Notes in Computer Science, **2162**, Springer-Verlag, 2001, 251–261.

2. D. E. Knuth, *The Art of Computer Programming*, vol. **2**, "Seminumerical Algorithms", 2nd Edition, Addison-Wesley, 1981, 441–466.
3. P. Kocher, *Timing Attack on Implementations of Diffie-Hellman, RSA, DSS, and other systems*, Advances in Cryptology – CRYPTO '96, N. Koblitz (editor), Lecture Notes in Computer Science, **1109**, Springer-Verlag, 1996, 104–113.
4. P. Kocher, J. Jaffe & B. Jun, *Differential Power Analysis*, Advances in Cryptology – CRYPTO '99, M. Wiener (editor), Lecture Notes in Computer Science, **1666**, Springer-Verlag, 1999, 388–397.
5. E. Oswald & M. Aigner, *Randomized Addition-Subtraction Chains as a Countermeasure against Power Attacks*, Cryptographic Hardware and Embedded Systems – CHES 2001, Ç. Koç, D. Naccache & C. Paar (editors), Lecture Notes in Computer Science, **2162**, Springer-Verlag, 2001, 39–50.
6. J.-J. Quisquater & D. Samyde, *ElectroMagnetic Analysis (EMA): Measures and Counter-Measures for Smart Cards*, Smart Card Programming and Security (E-smart 2001), Lecture Notes in Computer Science, **2140**, Springer-Verlag, 2001, 200–210.
7. J.-J. Quisquater & D. Samyde, *Eddy current for Magnetic Analysis with Active Sensor*, Smart Card Programming and Security (E-smart 2002), Lecture Notes in Computer Science, Springer-Verlag, 2002 *to appear*.
8. C. D. Walter, *Exponentiation using Division Chains*, IEEE Transactions on Computers, **47**, No. 7, July 1998, 757–765.
9. C. D. Walter, *Sliding Windows succumbs to Big Mac Attack*, Cryptographic Hardware and Embedded Systems – CHES 2001, Ç. Koç, D. Naccache & C. Paar (editors), Lecture Notes in Computer Science, **2162**, Springer-Verlag, 2001, 286–299.
10. C. D. Walter, *Improvements in, and relating to, Cryptographic Methods and Apparatus*, UK Patent Application 0126317.7, Comodo Research Laboratory, 2001.
11. C. D. Walter, *MIST: An Efficient, Randomized Exponentiation Algorithm for Resisting Power Analysis*, Topics in Cryptology – CT-RSA 2002, B. Preneel (editor), Lecture Notes in Computer Science, **2271**, Springer-Verlag, 2002, 53–66.

The Montgomery Powering Ladder

Marc Joye[1] and Sung-Ming Yen[2,⋆]

[1] Gemplus Card International, Card Security Group
Parc d'Activités de Gémenos, B.P. 100, 13881 Gémenos Cedex, France
marc.joye@gemplus.com
http://www.gemplus.com/smart/ − http://www.geocities.com/MarcJoye/
[2] Laboratory of Cryptography and Information Security (LCIS)
Dept of Computer Science and Information Engineering
National Central University, Chung-Li, Taiwan 320, R.O.C.
yensm@csie.ncu.edu.tw
http://www.csie.ncu.edu.tw/~yensm/

Abstract. This paper gives a comprehensive analysis of Montgomery powering ladder. Initially developed for fast scalar multiplication on elliptic curves, we extend the scope of Montgomery ladder to any exponentiation in an abelian group. Computationally, the Montgomery ladder has the triple advantage of presenting a Lucas chain structure, of being parallelized, and of sharing a common operand. Furthermore, contrary to the classical binary algorithms, it behaves very regularly, which makes it naturally protected against a large variety of implementation attacks.

Keywords. Exponentiation algorithms, Montgomery powering ladder, constrained environments, cryptographic implementations, fault attacks, side-channel attacks.

1 Introduction

Exponentiation or powering algorithms are of central importance in cryptography as they are the basis of (nearly) all public-key cryptosystems. Although numerous exponentiation algorithms have been devised, algorithms for constrained devices are scarcely restricted to the square-and-multiply algorithm and its right-to-left counterpart. A less-known algorithm due to Peter Montgomery is also not much greedy for memory. Developed for fast elliptic curve multiplication, this algorithm is of full generality and applies to any abelian group. Furthermore, it presents several useful features not available in the classical binary algorithms.

This paper is aimed at giving a thorough analysis of Montgomery ladder, considering both the efficiency and security issues. Among other things, we show how it reduces the memory requirements for elliptic curve computations or how it speeds up by a factor of up to 50% the evaluation of full Lucas sequences. For

⋆ Supported in part by the Ministry of Education of the Republic of China under contract EX-91-E-FA06-4-4.

B.S. Kaliski Jr. et al. (Eds.): CHES 2002, LNCS 2523, pp. 291–302, 2003.

(modular) exponentiation, we show that Montgomery ladder can be combined with the common-multiplicand technique, leading to a 33% speed-up factor. The Montgomery ladder is also prone to parallel implementations; on a bi-processor device, the running time is divided by two, compared to the non-parallel version. Last but not least, Montgomery ladder is a prime choice for a secure exponentiation as its high regularity makes it naturally resistant to various side-channel and fault attacks. A slight variant protected against the M safe-error attack is presented.

The rest of this paper is organized as follows. The next section presents the Montgomery ladder in terms of group-theoretic language. In Section 3, we analyze the efficiency of Montgomery ladder and compare it to the classical binary ladders. Section 4 analyzes the security of Montgomery ladder towards implementation attacks. Finally, we conclude in Section 5.

2 Montgomery Ladder

Originally, the so-called Montgomery ladder [16] was proposed as a means to speed up scalar multiplication in the context of elliptic curves. It has been then rediscovered several times and applied to different settings.

To ease the discussion, we give hereafter a higher description of the algorithm. We consider the general problem of computing $y = g^k$ in a (multiplicatively written) abelian group $\mathbb{G}$, on input g and k. Let $\sum_{i=0}^{t-1} k_i\, 2^i$ be the binary expansion of exponent k. The Montgomery ladder relies on the following observation. Defining $L_j = \sum_{i=j}^{t-1} k_i\, 2^{i-j}$ and $H_j = L_j + 1$, we have

$$L_j = 2L_{j+1} + k_j = L_{j+1} + H_{j+1} + k_j - 1 = 2H_{j+1} + k_j - 2$$

and so we obtain

$$(L_j, H_j) = \begin{cases} (2L_{j+1},\, L_{j+1} + H_{j+1}) & \text{if } k_j = 0\,, \\ (L_{j+1} + H_{j+1},\, 2H_{j+1}) & \text{if } k_j = 1\,. \end{cases} \tag{1}$$

Suppose that, at each iteration, a first register, say R_0, is used to contain the value of g^{L_j} and that a second register, say R_1, is used to contain the value of g^{H_j}. Equation (1) implies that

$$(g^{L_j}, g^{H_j}) = \left((g^{L_{j+1}})^2,\, g^{L_{j+1}} \cdot g^{H_{j+1}}\right) \quad \text{if } k_j = 0$$

and

$$(g^{L_j}, g^{H_j}) = \left(g^{L_{j+1}} \cdot g^{H_{j+1}},\, (g^{H_{j+1}})^2\right) \quad \text{if } k_j = 1\ .$$

Remarking that $L_0 = k$, this leads to an elegant algorithm for evaluating $y = g^k$: the Montgomery ladder.

For cryptographic applications, the group $\mathbb{G}$ may be taken as ${\mathbb{Z}_N}^*$ (e.g., for RSA or Rabin encryption/signature), ${\mathbb{F}_q}^*$ (e.g., for DH key exchange), the elements of a Lucas sequences (e.g., for LUC signature), the points of an elliptic curve (e.g., for ECDSA signature), ... Other practical applications include fast primality tests and factorization algorithms.

```
Input: g, k = (k_{t-1}, ..., k_0)_2
Output: y = g^k
R_0 ← 1; R_1 ← g
for j = t − 1 downto 0 do
  if (k_j = 0) then
    R_1 ← R_0 R_1; R_0 ← (R_0)^2
  else [if (k_j = 1)]
    R_0 ← R_0 R_1; R_1 ← (R_1)^2
return R_0
```

Fig. 1. Montgomery Ladder.

3 Efficiency Analysis

The most widely used algorithm for computing g^k are the square-and-multiply algorithm, which processes the bits of exponent k from the left to the right (Fig. 2 (a)), and its right-to-left counterpart (Fig. 2 (b)). We restrict our attention to constrained environments and do not consider more sophisticated exponentiation algorithms (see [7] for a survey).

```
Input: g, k = (k_{t-1}, ..., k_0)_2
Output: y = g^k
R_0 ← 1; R_1 ← g
for j = t − 1 downto 0 do
  R_0 ← (R_0)^2
  if (k_j = 1) then R_0 ← R_0 R_1
return R_0
```

(a) Left-to-right binary algorithm.

```
Input: g, k = (k_{t-1}, ..., k_0)_2
Output: y = g^k
R_0 ← 1; R_1 ← g
for j = 0 to t − 1 do
  if (k_j = 1) then R_0 ← R_0 R_1
  R_1 ← (R_1)^2
return R_0
```

(b) Right-to-left binary algorithm.

Fig. 2. Classical Binary Ladders.

From a computational perspective, the Montgomery ladder (Fig. 1), in its basic version, appears inferior to the classical binary algorithms as it requires $2t$ multiplications instead of $1.5t$ multiplications, on average. Nevertheless, in some cases, the Montgomery ladder may reveal itself more efficient by observing that

1. the value R_1/R_0 is invariant throughout the algorithm (and so equals g);
2. at each iteration, the two multiplications are independent;
3. at each iteration, the two multiplications share a common operand.

3.1 Lucas Chains

The key property of Montgomery ladder (Fig. 1) is that the relation $R_1/R_0 = g$ (or equivalently, $R_1 = R_0\, g$) is maintained invariant. This was noticed by

Montgomery [16,17] and applied to the ECM factorization method for a special class of elliptic curves.

```
Input: G, k = (1, k_{t-2}, ..., k_0)_2
Output: Y = kG
R_0 ← G; R_1 ← 2G
for j = t − 2 downto 0 do
  if (k_j = 0) then
    R_1 ← R_0 + R_1; R_0 ← 2R_0
  else [if (k_j = 1)]
    R_0 ← R_0 + R_1; R_1 ← 2R_1
return R_0
```

Fig. 3. Montgomery Ladder for Elliptic Curves.

Let $\boldsymbol{R_0}$ and $\boldsymbol{R_1} \in E(\mathbb{K})$ be two points on an elliptic curve E defined over a field $\mathbb{K}$. If the difference $\boldsymbol{G} := \boldsymbol{R_1} - \boldsymbol{R_0}$ is known then the x-coordinate of point $\boldsymbol{Y} = k\boldsymbol{G}$ can be computed from the x-coordinate of $\boldsymbol{R_0}$, the x-coordinate of point $\boldsymbol{R_1}$ and the x- and y-coordinates of point $\boldsymbol{G}$ [16]. Agnew *et al.* [2] (see also [13]) later observed (for binary fields $\mathbb{K}$) that the y-coordinate of $\boldsymbol{R_0}$ can easily be recovered when point $\boldsymbol{G}$ and the x-coordinates of $\boldsymbol{R_0}$ and of $\boldsymbol{R_0} + \boldsymbol{G}$ $(= \boldsymbol{R_1})$ are known. This was extended to fields $\mathbb{K}$ of characteristic $p > 3$ in [18,19] (see also [3,6,8] for general Weierstraß elliptic curves).

Because the computations can be carried out with the x-coordinates only, a lot of multiplications (in field $\mathbb{K}$) are saved, resulting in an algorithm faster than the classical binary algorithms (Fig. 2). Additionally, fewer memory is required since the y-coordinates need not to be handled (and thus stored) during the computation of $x(k\boldsymbol{G})$. The y-coordinate of $k\boldsymbol{G}$, $y(k\boldsymbol{G})$, is computed at the end of the algorithm from $\boldsymbol{G}$, $x(k\boldsymbol{G})$ and $x(k\boldsymbol{G} + \boldsymbol{G})$.

A similar technique exists for Lucas sequences. The special Lucas sequence $\{V_k(P, 1)\}$ with parameter $Q = 1$ is considered in [27] and the general case, $\{V_k(P, Q)\}$ along with the 'sister' sequence $\{U_k(P, Q)\}$ is addressed in [9] (see also [1, Section A.2.4]).

Analogously to elliptic curves, given $(V_1, U_1) = (P, 1)$, V_k and V_{k+1}, the value of U_k can be recovered as $U_k = (2V_{k+1} - P\,V_k)/\Delta$ with $\Delta = P^2 - 4Q$. Provided that division by Δ is inexpensive or that the value of Δ^{-1} is precomputed, this saves one multiplication per iteration compared to [9], resulting in a 22% improvement in the general case and a 50% improvement when $Q = 1$.

3.2 Parallel Computing

A second property of Montgomery ladder is its intrinsic disposition of being parallelized. This feature may reveal very useful in the near future as recent

```
Input: P, Q, k = (k_{t-1}, ..., k_0)_2
Output: y = V_k(P, Q)
V_0 ← 2; V_1 ← P; q_0 ← 1; q_1 ← 1
for j = t − 1 downto 0 do
  q_0 ← q_0 q_1
  if (k_j = 0) then
    q_1 ← q_0
    V_1 ← V_0 V_1 − P q_0; V_0 ← V_0^2 − 2q_0
  else [if (k_j = 1)]
    q_1 ← Q q_0
    V_0 ← V_0 V_1 − P q_0; V_1 ← V_1^2 − 2q_1
return V_0
```

Fig. 4. Montgomery Ladder for Lucas Sequences.

cryptographic tokens come equipped with several arithmetical co-processors [6, Section 5].

To exhibit the parallel nature of Montgomery ladder, we simplify the presentation of Fig. 1. Using k_j and its negation $\neg k_j$ as register indexes, the two different cases can be rewritten into a single statement as

$$R_{\neg k_j} \leftarrow R_0\, R_1; R_{k_j} \leftarrow (R_{k_j})^2 \ .$$

Hence, we clearly see that the two multiplications can be evaluated independently.

```
Input: g, k = (k_{t-1}, ..., k_0)_2
Output: y = g^k
R_0 ← 1; R_1 ← g
for j = t − 1 downto 0 do
   /* Processor 1 */ | /* Processor 2 */
   R_{¬k_j} ← R_0 R_1 | R_{k_j} ← (R_{k_j})^2
return R_0
```

Fig. 5. Parallel Montgomery Ladder.

For a modular exponentiation, if we denote by M the time for performing a multiplication, an optimized squaring takes roughly $0.8M$. So, on a bi-processor device, each iteration is performed in time M. As a result, the parallel version of the Montgomery ladder nearly attains the optimal 200% speed-up factor, over the standard Montgomery ladder (Fig. 1), for an RSA-type implementation. For elliptic curve implementations, the addition of two points or the doubling are further dissimilar [8], so that the expected gain seems sub-optimal; it is however possible to combine the operations of addition and doubling to lower the number of (field) multiplications [6] to nearly obtain the 200% speed-up factor.

3.3 Common-Multiplicand Multiplication

A third property of Montgomery ladder is that the two multiplications share a common operand: both multiplications involve R_0 when $k_j = 0$ and R_1 when $k_j = 1$. The 'common-multiplicand multiplication' method [26] is thus applicable. The method was initially developed to speed up the right-to-left binary algorithm (Fig. 2 (b)). Generalizations and improvements can be found in [22, 23].

The basic idea consists in rewriting the two involved multiplications with logical operators. Defining $R_{com} = (R_0 \,\mathsf{AND}\, R_1)$ and $R_{i,c} = (R_i \,\mathsf{XOR}\, R_{com})$, we have

$$R_i = R_{i,c} + R_{com}, \quad i \in \{0, 1\} \ . \tag{2}$$

Assume $k_j = 1$ (the case $k_j = 0$ is similar). Then the Montgomery ladder requires the computation of $R_0 \leftarrow R_0 R_1$ and $R_1 \leftarrow R_1 R_1$. From Eq. (2), this can be evaluated as $R_0 \leftarrow R_{0,c} R_1 + R_{com} R_1$ and $R_1 \leftarrow R_{1,c} R_1 + R_{com} R_1$. On average, the Hamming weight (i.e., the number of nonzero bits) of R_{com} and $R_{i,c}$ is twice smaller to that of R_i [27]. Consequently, each multiplication now requires half less binary additions, on average, that is, a 33% expected gain since the common multiplication $R_{com} R_1$ is only evaluated once.

For a modular exponentiation, the common-multiplicand method is particularly suited in certain hardware realizations (when logical operations can be processed in parallel with arithmetical operations). When the group law is more involved (as on elliptic curves), it may lead to software optimizations as well as several common (basic) operations (e.g., a field multiplication) may be saved [15].

4 Security Analysis

This section analyzes the security of Montgomery ladder towards implementation attacks. We distinguish two types of implementation attacks: side-channel attacks and fault attacks.

4.1 Side-Channel Attacks

Side-channel attacks are based on the observation that some side-channel information (e.g., timing [12] or power consumption [11]) depends on the instruction being executed and/or the data being handled.

The standard binary ladders (Fig. 2) contains conditional branchings. If the conditional branching is driven by secret data (namely, if the bits of exponent k in the computation of $y = g^k$ are secret) and if the two branches behave differently regarding some side-channel analysis (e.g., simple power analysis (SPA)) then secret data can be retrieved. To this end, dummy operations are added to the basic algorithms, so that they behave more regularly [4].

As it clearly appears in the next figure, the Montgomery ladder is already highly regular. Whatever the processed bit, there is always a multiplication followed by a squaring.

```
Input: g, k = (k_{t-1}, ..., k_0)_2
Output: y = g^k
R_0 ← 1; R_2 ← g
for j = t − 1 downto 0 do
  b ← ¬k_j
  R_0 ← (R_0)^2; R_b ← R_b R_2
return R_0
```

(a) Left-to-right binary algorithm.

```
Input: g, k = (k_{t-1}, ..., k_0)_2
Output: y = g^k
R_0 ← 1; R_2 ← g
for j = 0 to t − 1 do
  b ← ¬k_j
  R_b ← R_0 R_2; R_2 ← (R_2)^2
return R_0
```

(b) Right-to-left binary algorithm.

Fig. 6. (Simple) Side-Channel Protected Classical Binary Ladders.

```
Input: g, k = (k_{t-1}, ..., k_0)_2
Output: y = g^k
R_0 ← 1; R_1 ← g
for j = t − 1 downto 0 do
  R_{¬k_j} ← R_0 R_1; R_{k_j} ← (R_{k_j})^2
return R_0
```

Fig. 7. (Simple) Side-Channel Protected Montgomery Ladder.

Provided that the writing in registers R_0 and R_1 (resp. that the squaring of registers R_0 and R_1) cannot be distinguished from a *single* side-channel measurement, the Montgomery ladder can be implemented to prevent a given [simple] side-channel attack. It is worth noting that protections against simple side-channel attacks do not ward off the differential attacks, consisting in acquiring *several* side-channel measurements of different executions of the same algorithm and after that in performing some statistical treatment. For example, the attack of [4, § 3.2] conducted against the protected standard binary ladders (Fig. 6) readily applies the above protected Montgomery ladder. However, standard blinding techniques (e.g., [14,4]) easily prevent differential attacks.

Memory-wise, compared to the protected standard binary ladders, the protected Montgomery ladder requires one less register. Furthermore, it enjoys the useful features mentioned in Section 3.

Remark 1. The Montgomery ladder for Lucas sequences (see Fig. 4) does not behave regularly. This is however not a issue for cryptographic applications as known cryptosystems based on Lucas sequences ([20,21]) use for parameter Q the value $Q = 1$. Variables q_0 and q_1 are therefore useless.

4.2 Fault Attacks

An important lesson taught in [25] is that countermeasures must be considered globally (see also [10]). This was illustrated with the C safe-error attack in [25] and with the M safe-error in [24]. The next paragraphs analyze the security of the

Montgomery ladder regarding to the C and M safe-error attacks and highlight the interplay between different countermeasures.

Security against C safe-error attack. It was well known that a countermeasure developed against one implementation attack does not necessarily thwart another kind of implementation attack automatically. More surprisingly, in [25], it was pointed out that a countermeasure developed against a given attack, if not carefully examined, may benefit another physical attack tremendously. In that paper, a new type of computational safe-error attack (called a C safe-error attack) was mounted against the classical, side-channel protected exponentiation algorithms of Fig. 6. The C safe-error attack is developed by inducing *any* temporary random computational fault(s) inside the ALU.

It is easy to see that the protected algorithm of Fig. 6 (a) (commonly known as the square-and-multiply-always exponentiation algorithm) is susceptible to a C safe-error attack. This follows by observing that since the algorithm runs in constant time, an attacker can more easily locate the exact moment of the second multiplication "$R_b \leftarrow R_b\, R_2$" for each iteration. Moreover, when the current exponent bit, say k_j, is equal to 0, then this multiplication is a dummy operation and so has no influence on the final result. Therefore, if an attacker induces any kind of computational fault into the ALU during the operation of $R_b \leftarrow R_b\, R_2$ at jth iteration, then according to whether the final result is incorrect or not, she may deduce if $k_j = 1$ or $k_j = 0$. We note however that this attack only reveal one bit of exponent k and may be made, in some circumstances, much harder by randomizing exponent k prior to the exponentiation.

The same attack holds for the right-to-left protected algorithm of Fig. 6 (b).

For the Montgomery ladder (Fig. 7), the situation is different as there are no dummy operations. Consequently, any fault induced into the ALU will result in an incorrect exponentiation result.

Security against M safe-error attack. The M safe-error pointed out in [24] can be illustrated on the modular multiplication, $B \leftarrow A \cdot B \bmod N$, by calling the program routine listed in Fig. 8 as $B \leftarrow \texttt{MUL}(A, B, N)$. In this routine, it is assumed that multiplier B is represented in a 2^T-ary form as $B = \sum_{j=0}^{m-1} B_j\, (2^T)^j$, and both multiplicand A and multiplier B are sent to the routine `MUL` by passing their location address (i.e., the call by address technique). This call by address assumption is reasonable since it is popular for both high-level programming language (e.g., C) and all instruction-level language implementations.

The idea behind the M safe-error can be understood as follows. The value of multiplier B will be correct after the assignment operation $B \leftarrow A \cdot B \bmod N$, *even* if some blocks B_j (or Y_j with the notations of Fig. 8) of the multiplier are modified after they have been employed in the modular multiplication algorithm. As suggested in [24], this M safe-error can be avoided if B is assigned as the multiplicand, i.e., by calling the routine as $B \leftarrow \texttt{MUL}(B, A, N)$. It should be noted that the M safe-error attack needs to induce a temporary memory fault inside a register or memory location. Compared to the C safe-error attack, this

```
Input: X, Y, N
Output: R = MUL(X, Y, N)
R ← 0
for j = m − 1 downto 0 do
   R ← (R · 2^T + X · Y_j) mod N
output R
```

Fig. 8. M Safe-Error on Interleaved Modular Multiplication.

implies stronger cryptanalytic assumptions, namely a higher controllability of fault location and timing.

As presented in Fig. 7, the Montgomery ladder for modular exponentiation is vulnerable to the M safe-error attack, no matter R_0 or R_1 is passed to the routine as the multiplier in the multiplication $R_{\neg k_j} \leftarrow R_0\, R_1$. To prove above claim, we consider the two following possible implementations. Suppose first that R_1 is assigned as the multiplier (that is, exactly the algorithm of Fig. 7): $[R_{\neg k_j} \leftarrow R_0\, R_1\,;\ R_{k_j} \leftarrow (R_{k_j})^2]$. Within this design, when $k_j = 1$, the two operations at iteration j are $R_0 \leftarrow R_0\, R_1$ and $R_1 \leftarrow (R_1)^2$. Any error induced into R_1 cannot be an M safe-error. On the other hand, when $k_j = 0$, the two operations are $R_1 \leftarrow R_0\, R_1$ and $R_0 \leftarrow (R_0)^2$. An error carefully induced into the higher part of R_1 is an M safe-error (because the error in register R_1 is cleared after the assignment $R_1 \leftarrow R_0\, R_1$) and so do not influence the computation. Based on the two distinct behaviors, an attacker can recover the value of bit k_j. Likewise, if R_0 is now assigned as the multiplier, depending on whether of an error carefully induced into R_0 at iteration j is an M safe-error or not, an attacker can recover the value of bit k_j.

As mentioned in § 3.2, the Montgomery ladder can be implemented in parallel when two ALU's are available. It can be easily verified that the above M safe-error attack still applies in this parallelized implementation if these two ALU's share the source information of R_0 and R_1.

It is fairly easy to modify Montgomery ladder in order to counteract the aforementioned M safe-error attack. It suffices to perform $R_{\neg k_j} \leftarrow R_{\neg k_j}\, R_{k_j}$ at each iteration instead of $R_{\neg k_j} \leftarrow R_0\, R_1$ or $R_{\neg k_j} \leftarrow R_1\, R_0$.

```
Input: g, k = (k_{t−1}, ..., k_0)_2
Output: y = g^k
R_0 ← 1; R_1 ← g
for j = t − 1 downto 0 do
   b ← ¬k_j
   R_b ← R_b R_{k_j}; R_{k_j} ← (R_{k_j})^2
return R_0
```

Fig. 9. (Simple) Side-Channel and M Safe-Error Protected Montgomery Ladder.

When $k_j = 0$ (and $b = 1$), $R_1 \leftarrow R_1 R_0$ is executed (by calling the routine $R_1 \leftarrow \texttt{MUL}(R_1, R_0)$ with R_0 as multiplier). On the other hand, when $k_j = 1$ (and $b = 0$), $R_0 \leftarrow R_0 R_1$ is executed (by calling the routine $R_0 \leftarrow \texttt{MUL}(R_0, R_1)$ with R_1 as multiplier). It can be verified that no matter $k_j = 0$ or $k_j = 1$, any error induced into R_0 or R_1 cannot be an M safe-error. The proposed modification foils thus the above M safe-error attack.

5 Conclusion

This paper gave a generic description of Montgomery ladder in an abelian group $\mathbb{G}$. It thoroughly analyzed its main features for fast computation and secure implementation on constrained devices. We hope having convinced the reader that Montgomery ladder may be a first-class substitute of the celebrated square-and-multiply algorithm.

References

1. IEEE Std 1363-2000. *IEEE Standard Specifications for Public-Key Cryptography*. IEEE Computer Society, August 29, 2000.
2. G.B. Agnew, R.C. Mullin, and S.A. Vanstone. An implementation of elliptic curve cryptosystems over $F_{2^{155}}$. *IEEE Journal on Selected Areas in Communications*, 11(5):804–813, June 1993.
3. Éric Brier and Marc Joye. Weierstraß elliptic curves and side-channel attacks. In D. Naccache and P. Paillier, editors, *Public Key Cryptography*, volume 2274 of *Lecture Notes in Computer Science*, pages 335–345. Springer-Verlag, 2002.
4. Jean-Sébastien Coron. Resistance against differential power analysis for elliptic curve cryptosystems. In Ç.K. Koç and C. Paar, editors, *Cryptographic Hardware and Embedded Systems (CHES '99)*, volume 1717 of *Lecture Notes in Computer Science*, pages 292–302. Springer-Verlag, 1999.
5. Richard Crandall and Carl Pomerance. *Prime Numbers: A Computational Perspective*. Springer-Verlag, 2001.
6. Wieland Fischer, Christophe Giraud, Erik Woodward Knudsen, and Jean-Pierre Seifert. Parallel scalar multiplication on general elliptic curves over $\mathbb{F}_p$ hedged against non-differential side-channel attacks. Report 2002/007, Cryptology ePrint Archive, January 2002.
7. Daniel M. Gordon. A survey of fast exponentiation methods. *Journal of Algorithms*, 27:129–146, 1998.
8. Tetsuya Izu and Tsuyoshi Takagi. A fast parallel elliptic curve multiplication resistant against side channel attacks. In D. Naccache and P. Paillier, editors, *Public Key Cryptography*, volume 2274 of *Lecture Notes in Computer Science*, pages 280–296. Springer-Verlag, 2002.
9. Marc Joye and Jean-Jacques Quisquater. Efficient computation of full Lucas sequences. *Electronics Letters*, 32(6):537–538, March 1996.
10. Marc Joye, Jean-Jacques Quisquater, Sung-Ming Yen, and Moti Yung. Observability analysis: Detecting when improved cryptosystems fail. In B. Preneel, editor, *Topics in Cryptology – CT-RSA 2002*, volume 2271 of *Lecture Notes in Computer Science*, pages 17–29. Springer-Verlag, 2002.

11. Paul Kocher, Joshua Jaffe, and Benjamin Jun. Differential power analysis. In M. Wiener, editor, *Advances in Cryptology – CRYPTO '99*, volume 1666 of *Lecture Notes in Computer Science*, pages 388–397. Springer-Verlag, 1999.
12. Paul C. Kocher. Timing attacks on implementations of Diffie-Hellman, RSA, DSS, and other systems. In N. Koblitz, editor, *Advances in Cryptology – CRYPTO '96*, volume 1109 of *Lecture Notes in Computer Science*, pages 104–113. Springer-Verlag, 1996.
13. Julio López and Ricardo Dahab. Fast multiplication on elliptic curves over $GF(2^m)$ without precomputation. In Ç.K. Koç and C. Paar, editors, *Cryptographic Hardware and Embedded Systems*, volume 1717 of *Lecture Notes in Computer Science*, pages 316–327. Springer-Verlag, 1999.
14. Thomas S. Messerges, Ezzy A. Dabbish, and Robert H. Sloan. Power analysis attacks of modular exponentiation in smartcards. In Ç.K. Koç and C. Paar, editors, *Cryptographic Hardware and Embedded Systems (CHES '99)*, volume 1717 of *Lecture Notes in Computer Science*, pages 144–157. Springer-Verlag, 1999.
15. Atsuko Miyaji, Takatoshi Ono, and Henri Cohen. Efficient elliptic curve exponentiation. In Y. Han, T. Okamoto, and S. Qing, editors, *Information and Communications Security (ICICS '97)*, volume 1334 of *Lecture Notes in Computer Science*, pages 282–290. Springer-Verlag, 1997.
16. Peter L. Montgomery. Speeding the Pollard and elliptic curve methods of factorization. *Mathematics of Computation*, 48(177):243–264, January 1987.
17. Peter L. Montgomery. Evaluating recurrences of form $X_{m+n} = f(X_m, X_n, X_{m-n})$ via Lucas chains. Unpublished manuscript, January 1992.
18. Katsuyuki Okeya, Hiroyuki Kurumatani, and Kouichi Sakurai. Elliptic curves with the Montgomery form and their cryptographic applications. In H. Imai and Y. Zheng, editors, *Public Key Cryptography*, volume 1751 of *Lecture Notes in Computer Science*, pages 238–257. Springer-Verlag, 2000.
19. Katsuyuki Okeya and Kouichi Sakurai. Efficient elliptic curve cryptosystems from a scalar multiplication algorithm with recovery of the y-coordinate on a Montgomery-form elliptic curve. In Ç.K. Koç, D. Naccache, and C. Paar, editors, *Cryptographic Hardware and Embedded Systems – CHES 2001*, volume 2162 of *Lecture Notes in Computer Science*, pages 126–141. Springer-Verlag, 2001.
20. P. Smith. Cryptography without exponentiation. *Dr. Dobb's Journal*, (4):26–30, April 1994.
21. Peter J. Smith and Michael J.J. Lennon. LUC: A new public key system. In E.G. Douglas, editor, *Ninth IFIP Symposium on Computer Security*, pages 103–117. Elsevier Science Publishers, 1993.
22. Tzong-Chen Wu and Yuh-Shihng Chang. Improved generalisation of common-multiplicand algorithm of Yen and Laih. *Electronics Letters*, 31(20):1738–1739, September 1995.
23. Sung-Ming Yen. Improved common-multiplicand multiplication and fast exponentiation by exponent decomposition. *IEICE Trans. Fundamentals*, E80-A(6):1160–1163, June 1997.
24. Sung-Ming Yen and Marc Joye. Checking before output may not be enough against fault-based cryptanalysis. *IEEE Trans. on Computers*, 49(9):967–970, September 2000.
25. Sung-Ming Yen, Seung-Joo Kim, Seon-Gan Lim, and Sang-Jae Moon. A countermeasure against one physical cryptanalysis may benefit another attack. In K. Kim, editor, *Information Security and Cryptology – ICISC 2001*, volume 2288 of *Lecture Notes in Computer Science*, pages 414–427. Springer-Verlag, 2002.

26. Sung-Ming Yen and Chi-Sung Laih. Common-multiplicand multiplication and its application to public-key cryptography. *Electronics Letters*, 29(17):1583–1584, August 1993.
27. Sung-Ming Yen and Chi-Sung Laih. Fast algorithms for LUC digital signature computation. *IEE Proc.-Comput. Digit Tech.*, 142(2):165–169, March 1995.

DPA Countermeasures by Improving the Window Method

Kouichi Itoh, Jun Yajima, Masahiko Takenaka, and Naoya Torii

Fujitsu Laboratories Ltd., 64 Nishiwaki, Ohkubo-cho, Akashi 674-8555 Japan.
{kito, jyajima, takenaka}@flab.fujitsu.co.jp, torii.naoya@jp.fujitsu.com

Abstract. We propose three differential power analysis (DPA) countermeasures for securing the public key cryptosystems. All countermeasures are based on the window method, and can be used in both RSA and elliptic curve cryptosystems (ECC). By using the optimal countermeasure, performance penalty is small. In comparison with k-ary method, computation time of our countermeasure is only 105% in 1024-bit RSA and 119% in 160-bit ECC.

1 Introduction

Differential power analysis (DPA, [4]), proposed by Kocher et al., is an attack that enables extraction of a secret key stored in a cryptographic device, such as smartcard. In this attack, an attacker monitors the power consumption of the cryptographic devices, then statistically analyzes the collected power signal data to extract the secret key. This attack can be used against both secret and public key cryptosystems.

Currently, DPA is known as a big threat against the smartcard security, and the necessity of countermeasure for protecting the cryptographic device from the DPA attack is described in many standards. For example, the countermeasure against this attack is commented in FIPS 140-2, which is the US standard of the cryptographic module. Moreover, the requirement for DPA protection is included in the protection profile (PP), which is a list of smartcard security requirements based on the ISO 15408.

Various DPA countermeasures have been already proposed. Data randomizing is a well-known DPA countermeasure, in which the intermediate data is randomly transformed inside the cryptographic device. By using this technique, statistical analysis method on DPA attack is disabled, because the intermediate data on the encryption is unpredictable to the attacker. DPA countermeasure using data randomizing technique can be used both in secret and public key cryptosystems. We focus on the countermeasure for public key cryptosystems in the rest of this paper.

Previous DPA countermeasures for public key cryptosystems are described in [1] [2] [3] [5] [7] [8] [9] and [10]. Some of these countermeasures have demerits in comparison with the straight-forward implementation. Roughly speaking, these demerits are divided into 3 types. The first demerit is that, the countermeasures involve a performance penalty. Especially, an exponent splitting technique

B.S. Kaliski Jr. et al. (Eds.): CHES 2002, LNCS 2523, pp. 303–317, 2003.

described in [1] requires two times of the computation as that of the straightforward implementation. The second demerit is that, the countermeasures are not always available in both RSA and elliptic curve cryptosystems (ECC). That is, countermeasures described in [2] [3] [9] uses the technique by randomizing the representation of projective coordinates, which are available only in ECC, not in RSA. The third demerit is that, some countermeasures require the additional parameters. For example, a countermeasure described in [2] requires the parameter $\phi(n)$ in RSA where n is a public modulo, or order of the base point in ECC. This makes it hard to match the I/F of the cryptographic engine with and without countermeasure, that is, a vulnerable engine can not easily be replaced to a secure one.

In this paper, we propose three novel DPA countermeasures for securing the public key cryptosystems. All countermeasures are based on the window method, which is an efficient algorithm for computing the public key cryptosystems. By using our countermeasures, all of the above demerits are avoided. In the first countermeasure, we use the novel idea of 'overlapping window method'. In the second countermeasure, we use the window method in which the pre-computed table data is randomized. In the third countermeasure, we use a hybrid technique of the first and second countermeasure. We call these countermeasures overlapping window method (O-WM), randomized table window method (RT-WM) and hybrid randomizing window method (HR-WM) respectively. These have different characteristics, but have common three merits: (i) encryption operation is fast, (ii) available in both RSA and ECC, and (iii) additional parameter is unnecessary. Merit (i) is that, the performance penalty of our countermeasure is small. In comparison with the k-ary method, computation time of our countermeasure is 119% in ECC and 105% in RSA. Merit (ii) dues to that our countermeasures are base on the window method, and we show that two of our countermeasures have high security against DPA attacks in both RSA and ECC. By merit (iii), I/F of a cryptographic engine with and without DPA countermeasures can be the same. That is, a vulnerable engine is easily replaced to a secure one.

We describe the previous data randomizing technique in section 2, our countermeasures in section 3, security evaluation in section 4, DPA experiment result in section 5 and performance comparison in section 6.

2 Data Randomizing Techniques

Data randomizing (blinding) is a well-known DPA countermeasure described in [1] [2] [3] [5] [7] [8] [9] and [10]. These techniques make the intermediate data on the encryption operation unpredictable to the attacker by randomly transforming the data. Hence, an intermediate encryption data at a moment is one of the possible values of the randomized data. We call the number of the possible values of randomized data at a moment 'NRD'. It is easy to see that as NRD is larger, DPA attack is harder, so that the security of the data randomizing technique can be evaluated by NRD.

Previous data randomizing techniques that can be used for both ECC and RSA are described in [1] [2] [5] [7] [8] and [10]. Among these countermeasures, we

take up three typical methods in which NRD is clearly evaluated. These three countermeasures are shown in following (A), (B) and (C).

(A) exponent blinding ([2]) : Instead of a secret key d, $d' = d + r \times \phi$ is used, where r is randomly given and ϕ is an order. NRD is the number of possible values of r.
(B) calculation randomizing ([7]) : Randomly choose the bit position of a secret key d, then first perform a binary-method from the chosen bit to MSB, and second, perform a binary-method from the chosen bit to LSB by using the fist calculation result. NRD is $log_2 d$.
(C) exponent splitting ([1]) : From a secret key d, generate random numbers d_1 and d_2 that satisfy $d = d_1 + d_2$, then calculate $a^{d_1} \times a^{d_2} \pmod{n} = a^d \pmod{n}$. NRD is d.

We suppose that all above countermeasures provide enough security. That is, NRD of (A), (B) and (C) ranges from $log_2 d$ to d widely, but any countermeasures listed above can attain enough security at current technology, because the size of the spike is reduced to less than 1/100 of the device using a straight-forward implementation.

On remarking the performance, two times modular exponentiation are required in (C), which makes the performance worse. So we compare our countermeasure with (A) and (B).

3 Our Method

3.1 Overview

We propose three DPA countermeasures by improving the window method. First one is overlapping window method (O-WM), second one is randomized table window method (RT-WM), and third one is hybrid randomizing window method (HR-WM). Each countermeasure has unique characteristics from the viewpoint of operation, speed and security. Appropriate countermeasure can be chosen according to the usage of the cryptographic devices (ex. encryption algorithm is RSA or ECC). Details of our countermeasures are described in later sections. In the rest of this paper, we use following notations.

- Our countermeasures can be used in both RSA and ECC, but we unified the description for RSA for simplicity.
- We refer the ECC computation using the affine coordinates as 'ECC-2D', and that using the projective or Jacobian coordinates as 'ECC-3D'.
- d is the secret key, u is represented with $u = log_2 d$, w_i is an index value for the pre-computed table (i.e., w_i is a window) and q is the number of w_i. (i.e., $i = 0, 1, ..., q - 1$.)
- $EXP_{WM}(w_0, \ldots, w_s)$ represents the intermediate exponent (or scalar) value when the table look-up operation proceeds from w_0 to w_s, where WM represents the type of window method, that is, k-ary, O-WM, RT-WM, or HR-WM. For example, $EXP_{k-ary}(w_0, \ldots, w_s) = (\ldots(((2^k \times w_0) + w_1) \times 2^k)\ldots) \times 2^k + w_s$. Representations of $EXP_{O-WM}()$, $EXP_{RT-WM}()$ and $EXP_{HR-WM}()$ are described in the later section.

- $DAT_{WM}(a, w_0, \ldots, w_s)$ represents the intermediate data value when the table look-up operation proceeds from w_0 to w_s for an input value a, where WM represents the type of window method. When O-WM or HR-WM is used in ECC-3D, the number of possible values of $DAT_{WM}(a, w_0, \ldots, w_s)$ is much greater than that of $EXP_{WM}(w_0, \ldots, w_s)$, because of the redundant data representation in the projective coordinates. The reason for this is given in section 4-2.
- NRD means the number of possible randomized data values at any given moment when using the data randomizing techniques, as described in section 2.
- AR is attenuation ratio that represents the ratio of the size of the spikes that appears in the differential power trace with and without DPA countermeasure. Detail of AR is described in section 4-1.
- $bit(a, x, \ldots, y)$ represents the concatenation of the bit values of a, which is represented in binary, from the x-th to the y-th bit. ($x = 0, 1, \ldots; y = 0, 1, \ldots; x \geq y$.) The bit values upper than the MSB are regarded as 0. e.g., if $a = 6 = (110)_2$, $bit(a, 0) = 0$, $bit(a, 1) = 1$, and $bit(a, 4, 3, 2, 1) = (0011)_2 = 3$.

3.2 Overlapping Window Method (O-WM)

The characteristic of the O-WM is that, two continuous windows w_i and w_{i+1} 'overlap' each other at the same bit position of d. We show the steps of O-WM in algorithm 1 and an overview in figure 1. In O-WM, w_i and w_{i+1} are allowed to 'overlap' at the same bit position of d (figure 1). By overlapping the w_i, plural possible values of $\{w_0, \ldots, w_{q-1}\}$ are generated for a fixed d. Hence, the intermediate data on the encryption operation will be unpredictable to the attacker by randomly choosing one of these values. We denote h_i as the overlapping bit length between w_i and w_{i+1}. If the table look-up operation in step 18 is finished for $i = s$, $EXP_{O-WM}()$ is represented as $EXP_{O-WM}(w_0, \ldots, w_s) = (\ldots(w_0 \times 2^{k-h_0} + w_1)\ldots)2^{k-h_{s-1}} + w_s$, whose bit length is $s \times k - (h_0 + \ldots + h_{s-1})$ and lowest h_s-bit randomly value (figure 1).

In comparison with the k-ary method, the overhead for table making is the same, but the number of repeating the table look-up operations is larger.

Note 1. For securing against SPA attacks, we recommend to set h_i a fixed value h that satisfy $h \geq k/2$. This tweak provides a protection against the SPA by observing only one time execution of the cryptographic device, because multiplication and square are repeated in constant pattern. h must satisfy $h \geq k/2$ to prevent the bias distribution of w_i.

3.3 Randomized Table Window Method (RT-WM)

The characteristic of the RT-WM is that, pre-computed table data is randomized. We show the steps of RT-WM in algorithm 2 and an overview in figure 2. In step 5, pre-computed table data is generated by $tab[i] = a^{i \times 2^b + r} \pmod{n}$, where i is an index value and r is a b-bit random value. If step 19 is finished

Algorithm 1. Overlapping window method (O-WM)

1: /* pre-computed table data making */
2: for $(i = 0; i < 2^k; i++)$ $tab[i] = a^i \pmod n$;
3: /* window w_i and overlapping length h_i making */
4: /* generate random number q and $0 < h_0, h_1, \ldots, h_{q-2} < k$
5: which satisfy $q \times k + (h_0 + h_1 + \ldots h_s) = u$.
6: For securing against SPA, h_i are recommended to be
7: the fixed value $h \geq k/2$. */
8: $(h_i, q) = \text{GenRandom}()$; $u' = u - k$; $dt_{q-1} = bit(d, u-1, \ldots u')$;
9: for $(i = 0; i < q-1; i++) = \{$
10: $w_i = (\text{Random number}, \max(0,\ dt_i - 2^{h_i} + 1) \leq w_i \leq dt_i)$;
11: $dt_{i+1} = (dt_i - w_i) \times 2^{k-h_i} + bit(d, u'-1, \ldots, u'-(k-h_i))$;
12: $u' = u' - (k - h_i)$;
13: }
14: $w_{q-1} = dt_{q-1}$;
15: /* modular exponent process */
16: $v = tab[w_0]; i = 1$;
17: while $(i < q)$ {
18: $v = v^{2^{k-h_i}} \pmod n$; $v = v \times tab[w_i] \pmod n$; $i = i + 1$;
19: }
20: Return(v);

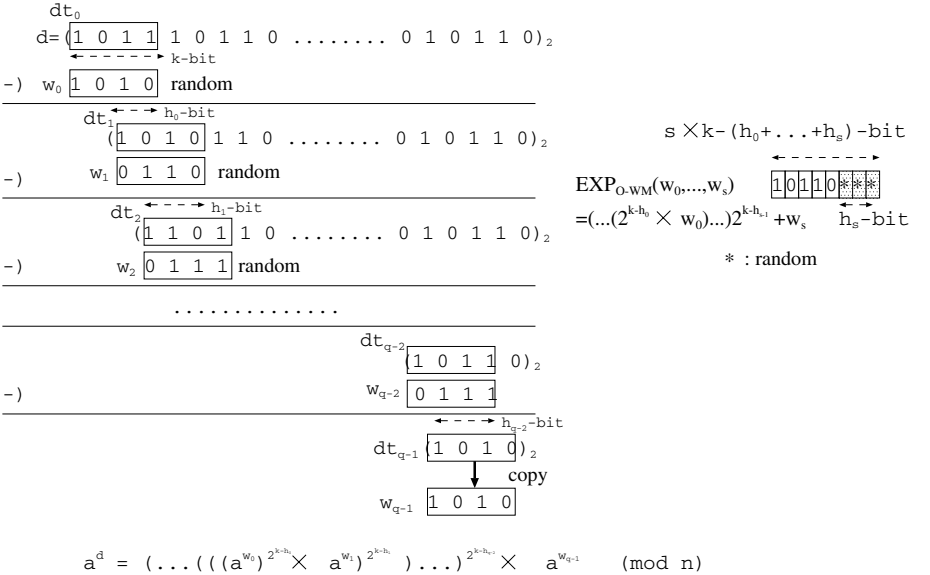


Fig. 1. Overview of O-WM and $EXP_{O-WM}(w_0, \ldots, w_s)$

Algorithm 2. RT-WM (Randomized Table Window Method)

1: /* Generate random number */
2: r = (b-bit random number);
3: /* pre-computed table data making */
4: $t = a^r \pmod n$
5: for $(i = 0; i < 2; i++)$ tab[i] $= a^{i \times 2^b} \times t \pmod n$
6: /* window making phase */
7: dw = d; q=0;
8: while $(dw \geq r \times 2^{k \times q})$ {
9: $dw = dw - r \times 2^{k \times q}; q = q + 1$
10: }
11: for $(i = 0; i < q; i++)$ {
12: $w_i = bit(dw, b + (q - i) \times k - 1, \ldots, b + (q - i - 1) \times k);$
13: }
14: $d_m = dw \pmod{2^b}$
15: /* modular exponent process */
16: if $(q == 0)$ Return$(a^{d_m} \pmod n)$;
17: $v = w_0$; $i = 1$;
18: while $(i < q)$ {
19: $v = v^{2^k} \pmod n; v = v \times tab[w_i] \pmod n; i = i + 1;$
20: }
21: /* normalization */
22: $v = v \times a^{d_m} \pmod n$;
23: Return(v);

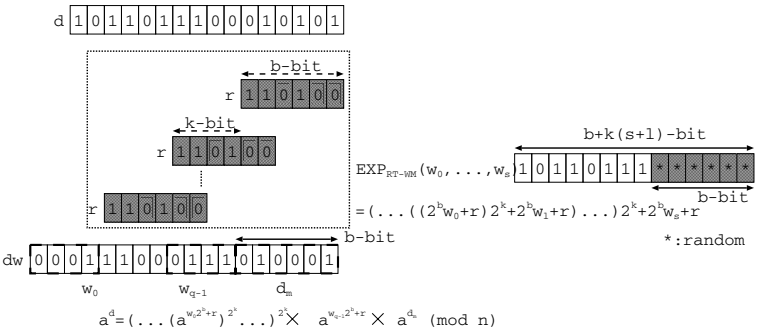


Fig. 2. Overview of RT-WM and $EXP_{RT-WM}(w_0, \ldots, w_s)$

for $i = s$, the $EXP_{RT-WM}()$ is represented as $EXP_{RT-WM}(w_0, \ldots, w_s) = (\ldots((w_0 \times 2^b + r) \times 2^k + w_1 \times 2^b + r) \times 2^k \ldots) \times 2^k + w_s \times 2^b + r$, whose lowest b-bit data is random value (figure 2). To obtain the final result $a^d \pmod n$, the randomized data must be 'normalized' at the end of the operation. This normalization step is $v = v \times a^{d_m} \pmod n$ in step 22, where d_m is b-bit value generated in step 14.

In comparison with the k-ary method, the number of repeating table look-up operations are the same, but the overhead for the computation of table making and normalization are larger.

3.4 Hybrid Randomizing Window Method (HR-WM)

HR-WM is a combination technique of O-WM and RT-WM. We show the steps in algorithm 3 and an overview in figure 3. Pre-computed table data generation

Algorithm 3. HR-WM(Hybrid Randomizing Window Method)

1: /* Generate random number */
2: r = (b-bit random number);
3: /* pre-computed table data making */
4: $t = a^r \pmod n$
5: for $(i = 0; i < 2^k; i++)$ $tab[i] = a^r \pmod n$;
6: /* window making phase */
7: $dw = d; q = 0$;
8: while $(dw \geq r \times 2^{k \times q})$ {
9: $dw = dw - r \times 2^{k \times q}; q = q + 1$;
10 }
11: $d_m = dw \pmod{2^b}$; $dw = dw/2^b$;
12: $u' = (k-h)^{(q-1)}$; $dt_0 = bit(dw, u' + k - 1, .., u')$;
13: for $(i = 0; i < q - 1; i++)$ {
14: $w_i = (Randomnumber, max(0, dt_i - 2^h + 1) \leq w_i \leq dt_i)$;
15: $dt_{i+1} = (dt_i - w_i)2^{(k-h)} + bit(dw, u' - 1, \ldots, u' - (k-h))$;
16: $u' = u' - (k-h)$;
17: }
18: $w_{q-1} = dt_{q-1}$;
19: /* modular exponent process */
20: if $(q == 0)$ Return$(a^{d_m} \pmod n)$;
21: $v = w_0$; $i = 1$;
22: while $(i < q)$ {
23: $v = v^{(2^{(k-h)})} \pmod n$; $v = v \times tab[w_i] \pmod n$; $i = i + 1$;
10 }
24: /* normalization */
25: $v = v \times a^{d_m} \pmod n$;
26: Return (v);

is the same as that in RT-WM, and w_i is generated by combination operation of RT-WM and O-WM (steps.2-18). In HR-WM, the overlapping length h is a fixed value. If step 23 is finished for $i = s$, the $EXP_{HR-WM}()$ is represented as $EXP_{HR-WM}(w_0, \ldots, w_s) = (\ldots(w_0 \times 2^b + r) \times 2^{k-h} \ldots) \times 2^{k-h} + w_s \times 2^b + r$, whose lowest $(b+h)$-bit data is random value (figure 3).

Similar to RT-WM, overhead the computation of table making and normalization are larger in proportion to b and h. But when HR-WM is used, security

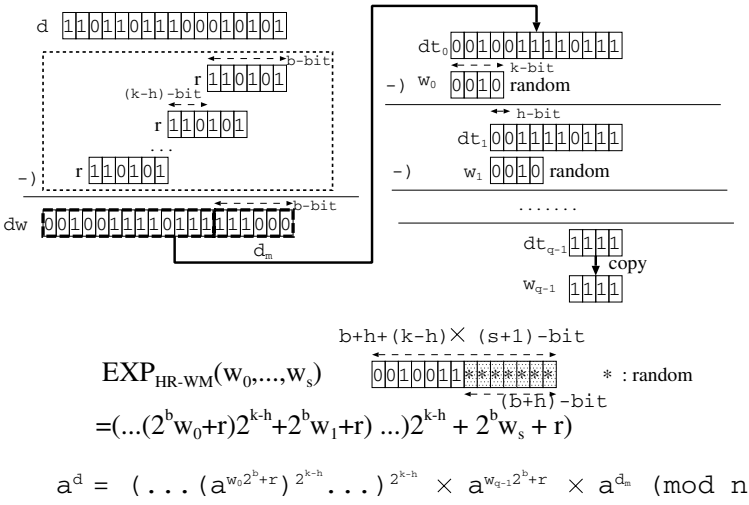


Fig. 3. Overview of HR-WM and $EXP_{HR-WM}(w_0, \ldots, w_s)$

is also strengthened by both effect of O-WM and RT-WM. So, these parameters can be smaller than that in O-WM and RT-WM for attaining the same security level. By setting parameters b and h, an optimal balance between performance and security can be chosen. In HR-WM, h can be small value such as $h = 1$, because h is not limited to $h \geq k/2$.

4 Security Evaluation against DPA

In this section, we evaluate the security of our countermeasures. At first, we describe a basic idea of the security evaluation, then discuss the security of each countermeasure. We finally show that all of our countermeasures have high security in ECC-3D, and RT-WM and HR-WM have high security in RSA and ECC-2D.

4.1 Basic Idea

Before explaining the basic idea of the security evaluation, we explain the DPA attack against the k-ary method. In this attack, the attacker repeats monitoring the power consumption of a cryptographic device N times when inputting plaintext $a_0, \ldots, a_{N-1}$. We denote these monitored data $V(a_i, t)$ where t is time.

The attacker analyzes d by guessing each $w_0, \ldots, w_s$ according to this order. If he already has guessed the correct $w'_0, \ldots, w'_{s-1}$ that satisfy $\{w'_0, \ldots, w'_{s-1}\} = \{w_0, \ldots, w_{s-1}\}$, he guesses $w'_s = w_s$, then calculates the difference power trace $\Delta(t)$ as shown (1), where e $(0 \leq e < (plaintext\ length))$ is a bit position for calculating the differential. If a spike appears in $\Delta(t)$, w'_s turns out to be correct, otherwise it turns to be incorrect.

$$\Delta(t) = \frac{2}{N}\Big(\sum_{bit(DAT_{WM}(a_j,w'_0,\ldots w'_s),e)=1} V(a_j,t) - \sum_{bit(DAT_{WM}(a_j,w'_0,\ldots w'_s),e)=0} V(a_j,t)\Big) \quad (1)$$

The attacker's possibility for succeeding in the above analysis depends whether the spike will appear or not. Hence, as the size of the spike is smaller, the analysis is harder. If the size of the spike is almost zero, he can't distinguish the correctness of the guessed w'_s. Therefore, the security against DPA can be evaluated by the size of the spike.

Here we describe the basic idea for the evaluation of the size of the spike when our countermeasure is used. In our method, w_s is randomly chosen value, so equation (1) can be transformed to the following equation (2), where $Prob[X]$ represents the probability that equation X holds. (Note that the differential power trace for $Prob[DAT_{WM}(a_j, w_0, \ldots, w_s) \neq DAT_{WM}(a_j, w'_0, .., w'_s)]$ is 0.)

$$\begin{aligned}
\Delta(t) &= \frac{2}{N}\Big(\sum_{bit(DAT_{WM}(a_j,w'_0,\ldots w'_s),e)=1} V(a_j,t) - \sum_{bit(DAT_{WM}(a_j,w'_0,\ldots w'_s),e)=0} V(a_j,t)\Big) \\
&= Prob[DAT_{WM}(a_j, w'_0, \ldots w'_s) = DAT_{WM}(a_j, w_0, \ldots w_s)] \\
&\quad \times \frac{2}{N}\Big(\sum_{bit(DAT_{WM}(a_j,w'_0,\ldots w'_s),e)=1} V(a_j,t) - \sum_{bit(DAT_{WM}(a_j,w'_0,\ldots w'_s),e)=0} V(a_j,t)\Big) \\
&\quad + Prob[DAT_{WM}(a_j, w'_0, \ldots w'_s) \neq DAT_{WM}(a_j, w_0, \ldots w_s)] \\
&\quad \times \frac{2}{N}\Big(\sum_{bit(DAT_{WM}(a_j,w'_0,\ldots w'_s),e)=1} V(a_j,t) - \sum_{bit(DAT_{WM}(a_j,w'_0,\ldots w'_s),e)=0} V(a_j,t)\Big) \\
&= Prob[DAT_{WM}(a_j, w'_0, \ldots w'_s) = DAT_{WM}(a_j, w_0, \ldots w_s)] \\
&\quad \times \frac{2}{N}\Big(\sum_{bit(DAT_{WM}(a_j,w'_0,\ldots w'_s),e)=1} V(a_j,t) - \sum_{bit(DAT_{WM}(a_j,w'_0,\ldots w'_s),e)=0} V(a_j,t)\Big) \quad (2)
\end{aligned}$$

From (2), the size of the spike will be smaller in proportion to $Prob[DAT_{WM}(a_j, w_0, \ldots, w_s) = DAT_{WM}(a_j, w'_0, \ldots, w'_s)]$. Therefore, we evaluate the security by maximum value of the probability that $DAT_{WM}(a_j, w_0, \ldots, w_s)$ is computed in the device. We call maximum value of the probability 'attenuation ratio' (AR) in the rest of this paper. (Note that the probability represents the ratio of the size of

spike of (1).) For evaluating AR, we discuss the NRD of $DAT_{WM}(a_j, w_0, \ldots, w_s)$, then evaluate the $DAT_{WM}(a_j, w_0, \ldots, w_s)$ that appears with highest probability.

4.2 O-WM

In O-WM, evaluation of AR differs among RSA, ECC-2D, and ECC-3D. This means that the NRD of $DAT_{O-WM}(a_j, w_0, \ldots, w_s)$ is evaluated using the NRD of $EXP_{O-WM}(w_0, \ldots, w_s)$ for RSA and ECC-2D, and is roughly evaluated using the NRD of the sequence $\{w_0, \ldots, w_s\}$ for ECC-3D. Therefore, we discuss RSA/ECC-2D and ECC-3D separately.

Note 2. The difference between ECC-2D and ECC-3D is due to the difference of the data representation in the projective coordinates. For example, when calculating $7A$ for $A = (X, Y, Z)$ by $(X_1, Y_1, Z_1) = 2((11)_2(X, Y, Z)) + (01)_2(X, Y, Z)$ or $(X_2, Y_2, Z_2) = 2((10)_2(X, Y, Z)) + (11)_2(X, Y, Z)$, these two points represent the same point in affine coordinates, but $X_1 \neq X_2, Y_1 \neq Y_2, Z_1 \neq Z_2$ will hold with high probability.

To see this fact, let us assume that the device computes $B_1 = f_1A + g_1A$ or $B_2 = f_2A + g_2A$ in projective coordinates for point A and scalar values f_1, f_2, g_1 and g_2, when $f_1 + g_1 = f_2 + g_2, f_1, f_2 > g_1, g_2$, and the data representation of f_1A, f_2A are different. Under this assumption, data representation of B_1 and B_2 will be different with probability $1 - 1/p$ where p is the size of the finite field.

In general, NRD of the data representation of $B_z = f_xA + g_yA$ is approximated to (NRD *of data representation of* f_xA) $\times$ (NRD *of data representation of* g_yA) when the data representation of f_xA are different from each other and $f_x > g_y$ for any x, y. So, (NRD *of data representation of* $EXP_{O-WM}(w_0, \ldots, w_s)A$) is approximated to ($NRD$ *of data representation of* $2^{k-h_{s-1}}$ $(EXP_{O-WM}$ $(w_0, \ldots, w_{s-1}))A$) $\times$ (NRD *of data representation of* w_sA), which is equal to the NRD of the sequence $\{w_0, \ldots, w_s\}$.

RSA/ECC-2D. From figure 1, $EXP_{O-WM}(w_0, \ldots, w_s)$ is $s \times k - (h_0 + \ldots + h_{s-1})$-bit and lowest h_s-bit is randomized. Therefore, the probability (or AR) that some $EXP_{O-WM}(w_0, ..., w_s)$ is used in the device, is represented as (3) for some W_{len} and W_{val}.

$$(Prob[s \times k - (h_0 + \ldots + h_{s-1}) = W_{len}]) \times (Prob[lowest\ h_s bits = W_{val}]) = \alpha(s, W_{len}) \times \beta(s, W_{val}) \quad (3)$$

$\alpha(s, W_{len})$ depends on $h_0, \ldots, h_{s-1}$ and $\beta(s, W_{val})$ depends on h_s and w_{s+1}, which are independent each other. Therefore, the maximum value of (3) is a product of each maximum value.

$\alpha(s, W_{len})$ equals to the maximum value when $h_0 + h_1 + \ldots + h_{s-1} = s \times k/2$ (note that $0 < h_i < k$). It can be calculated directly, or is approximated as a normal distribution by the central limit theorem, if s is large enough. When h_i is fixed value h, $\alpha(s, W_{len}) = 1$.

$\beta(s, W_{val})$ is a probability that the lowest h_s-bit is equal to W_{val}. Taking into account that h_s varies 1 to $k-1$, it is easy to see that varying only LSB can occur for all h_i (Note that 1-bit varying is included in h_i-bit varying.) The probability is represented as $(2^{-1} + ... + 2^{-(k-1)})/(k-1)$, where $(k-1)$ is a number of possible value h_i.

We show the graph of maximum value of AR in figure 4 when $k = 4$ and $1 \leq h_i \leq 3$. This graph can be approximated to $0.15 \times s^{-1/2}$, decreases slowly for s. So this is thought as 'weak' DPA countermeasure, but it can be used to the device whose SNR (signal-to-noise ratio) is small. Detail of the SNR is described in [6].

Here we note that the probability is a mean value. It depends on the partial value of the secret key d that decides the variable range of w_s. (See step 10 in algorithm 1.) As the partial k-bit of d corresponding to w_s is smaller, $\beta(s, W_{val})$ will be larger. (ex. In figure 1, partial k-bit of d corresponding to w_0 is $(1011)_2$, that to w_1 is $(1110)_2$ and that to w_2 is $(1101)_2$.) When h_i is fixed value h, $\beta(s, W_{val}) = 2^{-h}$.

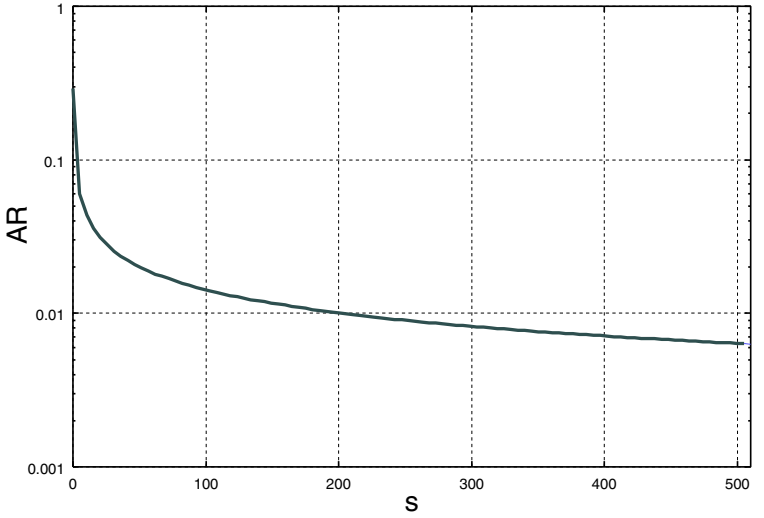


Fig. 4. AR when $k = 4$ and $1 \leq h_i \leq 3$(RSA, ECC-2D)

ECC-3D. Following (4) represents the probability that some sequence $\{w_0, \ldots, w_s\}$ is used in the device.

$$\begin{aligned}(Prob[s \times k - (h_0 + \ldots + h_{s-1}) = W_{len}]) \times 2^{-(h_0+h_1+\ldots+h_s)} \\ = \alpha(s, W_{len}) \times 2^{-(h_0+\ldots+h_s)} \end{aligned} \tag{4}$$

If h_i ranges $0 < h_i < k$, the upper bound of (4) is $2^{-(s+1)}$, and if h_i is a fixed value h, (4) is equal to $2^{-h\times(s+1)}$.

4.3 RT-WM

When RT-WM is used, intermediate encryption data is randomized by the pre-computed table data, which is given by the b-bit random number. So, AR is always equal to 2^{-b}.

4.4 HR-WM

When HR-WM is used, NRD is represented as $(NRD\ when\ using\ O-WM)\times (NRD\ when\ using\ RT-WM)$. If the encryption algorithm is RSA/ECC-2D, NRD is $2^h \times 2^b$. In HR-RM, intermediate data at any given moment is randomly chosen from one of the possible values. Therefore, AR is equal to $2^{-(h+b)}$. In ECC-3D, NRD is $2^{h(s+1)} \times 2^b$, so that AR is equal to $2^{-(h(s+1)+b)}$.

Table 1. Measured AR from DPA experiment (O-WM in RSA $k = 4, 1 \leq h_i \leq 3$)

w_s		w_0	w_3	w_6	w_9	w_{12}
AR (expected)		0.0400	0.104	0.0775	0.064	0.0556
AR (experiment)	spike1	0.0411	0.0863	0.0910	0.0672	0.113
	spike2	0.0414	0.0915	0.109	0.0825	0.120
	spike3	0.0373	0.0887	0.101	0.0653	0.0760
	spike4	0.0398	0.115	0.140	0.0815	0.128
partial 4-bit of d		$(1101)_2$	$(0111)_2$	$(0110)_2$	$(1111)_2$	$(0000)_2$

5 DPA Attack Experiment

For O-WM in RSA case, we have verified effect of the protection against DPA through the experiment. We monitored the power consumption for the RSA encryption by using 4-ary and O-WM in which $k = 4, 1 \leq h_i \leq 3$ and analyzed the secret key. When monitoring the power consumption, we have input 20000 plaintexts and set the sampling ratio 100 MHz. We have analyzed the key by making the difference power trace when guessing w_0, w_3, w_6, w_9 and w_{12}, and confirmed the spike to measure AR. In the analysis, we guessed the sequence $\{w_0, \ldots, w_s\}$ when the size of the spike that appears in (2) is maximum value.

Figure 5 shows the example of the differential power trace, and table 1 shows the expected and measured AR for 4 spikes appeared in the differential power trace. In table 1, partial 4-bit of d corresponding to w_s is also shown. The expected AR is well approximated to the measured AR, and when the partial 4-bit value of d is small, the measured AR is larger than the expected value.

6 Performance Comparison

In table 2, we show the comparison of the performance and security of our countermeasures. The input bit-length of the pre-computed table data is fixed

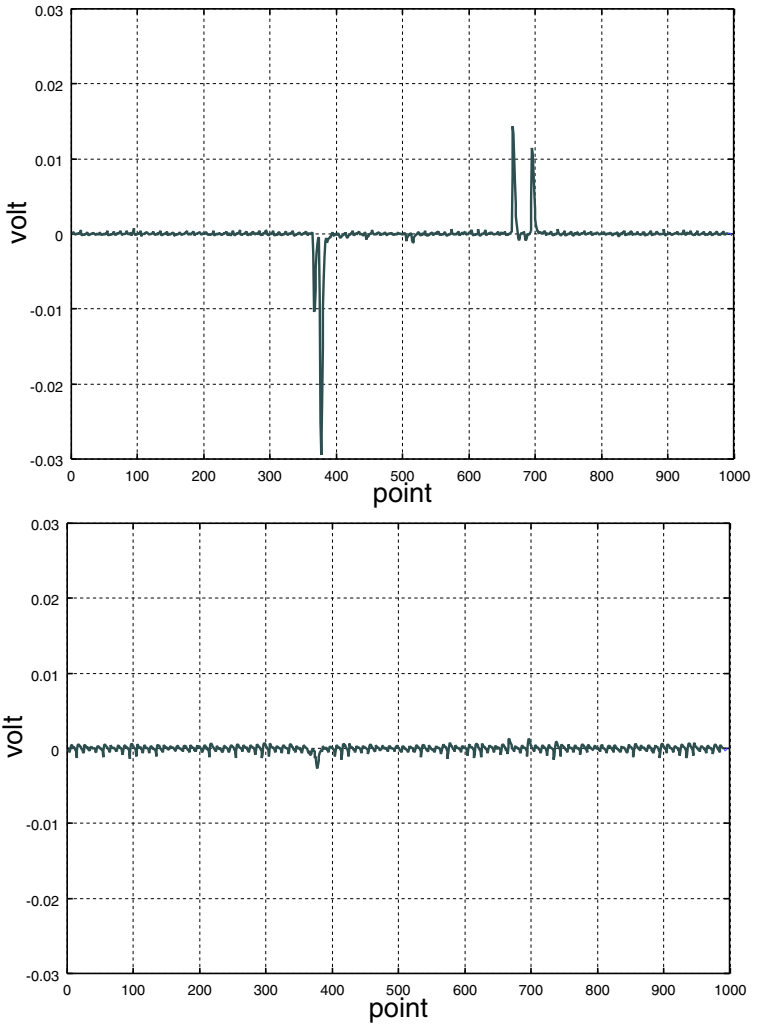


Fig. 5. Differential power traces when guessing w_3 (4-ary:upper, O-WM:lower)

Table 2. Performance comparison among proposed countermeasures

		O-WM	RT-WM	HR-WM
Time	table making	2^k	$b(2+\frac{1}{k'})+2^{k'}+2^k$	$b(2+\frac{1}{k'})+2^{k'}+2^k$
	exponent	$u(S+\frac{M}{k-h})$	$(u-b)(S+\frac{M}{k})$	$(u-b)(S+\frac{M}{k-h})$
	normalization	-	$b(S+\frac{M}{k'})+2^{k'}$	$b(S+\frac{M}{k'})+2^{k'}$
Security (AR)	RSA,ECC-2D	$Cs^{-1/2}$(h_i:random) 2^h(h_i :fixed)	2^{-b}	$2^{-(h+b)}$
	ECC-3D	$\leq 2^{s+1}$ (h_i:random) $2^{h(s+1)}$ (h_i:fixed)	2^{-b}	$2^{-(h(s+1)+b)}$

Table 3. Performance Comparison with other countermeasures

		O-WM	RT-WM	HR-WM	Coron	Messerges
ECC (160-bit)	Addition	256	279	264	241	214
	AR	$\sim 2^{-6}$(2D) $\sim 2^{-80}$(3D)	2^{-20}	2^{-11}(2D) 2^{-61}(3D)	2^{-20}	$2^{-7.32}$
RSA (1024-bit)	Multiplication	1552	1359	1416	1321	1536
	AR	$\sim 2^{-7.23}$	2^{-20}	2^{-11}	2^{-20}	2^{-10}
Additional parameter		No	No	No	Yes	No

to k-bit among these countermeasures, so that the RAM size of the table data are the same. S represents a computation time of squaring (or doubling), and M represents that of multiplication (or addition). In the 'table making' row, we assumed $S = M$ and showed the performance by the times of the multiplication (or addition). (Note that S and M are omitted in this row.) In the 'O-WM' column, h represents the average value of h_i, and AR is represented when each window method is processed from w_0 to w_s.

In table 3, comparison of our countermeasures with the Coron's (countermeasure (A) in section 2) and Messerges-Dabbish-Sloan's (countermeasure (B) in section 2) countermeasure are shown. In our countermeasures, the parameter are set to $k = 4$ (O-WM, RT-WM, HR-WM), $h = 2$ (O-WM) /1 (HR-WM), $b = 20$(RT-WM) /10 (HR-WM). In Coron's countermeasure, we suppose that the length of the random value is 20-bit, and 4-ary method is used. In Messerges' countermeasure, we supposed that binary-method is used for RSA, and signed-binary method is used for ECC.

We have evaluated the performance of these countermeasures in 1024-bit RSA and 160-bit ECC case, assuming the computation time for squaring and multiplication (doubling and addition) are the same.

6.1 Countermeasure Choice for an Encryption Algorithm

Suitable choice of our countermeasures depends on the encryption scheme and the environment of the device. We categorize them by encryption algorithm as followings.

- RSA/ECC-2D : RT-WM or HR-WM is suitable, O-WM is not recommended. In table 2, HR-WM looks like to be most suitable, but it is because parameters b are different between RT-WM and HR-WM. When parameters b are the same in these two methods, RT-WM is most suitable.
- ECC-3D : All countermeasures are suitable, but the countermeasure can be chosen according to the requirement. When the code size is required to be small, O-WM is suitable, because its computation steps are simple, similar to the k-ary method. Moreover, we recommend to fix h_i for securing against SPA attack. When the encryption speed is significant, suitable countermeasure depends on the bit length of the key. When using the short length key, O-WM is suitable. When using longer length key, HR-WM and RT-WM will be suitable in this order.

7 Conclusion

We proposed three DPA countermeasures based on the window method, O-WM, RT-WM and HR-WM. For O-WM, we assured the effect of the countermeasure through the DPA experiment. When choosing the optimal countermeasure according to the encryption scheme, the computation time ratio to k-ary method is only 105% in RSA and 119% in ECC.. In comparison with the Coron's countermeasure, our countermeasure has the merit that additional parameter is unnecessary. In comparison with the Messerges' countermeasure, encryption speed of our countermeasure is 13% faster in RSA. Except O-WM in which overlapping length is fixed, our countermeasure can protect against SPA by observing only one time execution of the cryptographic device, because square and multiplication are repeated by the constant pattern.

References

1. Christophe Clavier and Marc Joye, "Universal Exponentiation Algorithm – A First Step Towards Provable SPA Resistance", CHES 2001, LNCS 2162, pp. 300–308, Springer-Verlag, 2001.
2. Jean-Sébastien Coron, "Resistance against Differential Power Analysis for Elliptic Curve Cryptosystems", CHES'99, LNCS 1717, pp. 292–302, Springer-Verlag, 99.
3. Marc Joye and Christophe Tymen, "Protections against Differential Analysis for Elliptic Curve Cryptography", CHES 2001, LNCS 2162, pp. 377–390, Springer-Verlag, 2001.
4. Paul Kocher, Joshua Jaffe, and Benjamin Jun, "Differential Power Analysis", Advances in Cryptography-CRYPTO'99, pp. 388–397.
5. P.-Y. Liardet and N.P. Smart, "Preventing SPA/DPA in ECC Systems Using the Jacobi Form", Cryptographic Hardware and Embedded Systems, CHES 2001, pp. 391–401.
6. Thomas S. Messerges, Ezzy A. Dabbish and Robert H. Sloan, "Investigations of Power Analysis Attacks on Smartcards, ", Proceedings of USENIX Workshop on Smartcard Technology, May 1999, pp. 151–161.
7. Thomas S. Messerges, Ezzy A. Dabbish and Robert H. Sloan, "Power Analysis Attacks of Modular Exponentiation in Smartcards.", Cryptographic Hardware and Embedded Systems, CHES'99, LNCS 1717, pp. 144–157.
8. Bodo Moller, "Securing Elliptic Curve Point Multiplication against Side-Channel Attacks", Information Security-ISC 2001, pp. 324–334.
9. K. Okeya, K. Miyazaki and K. Sakurai, "A fast scalar multiplication method with randomized coordinates on a Montgomery-form elliptic curve secure against side channel attacks", ICISC 2001, LNCS 2288, pp. 428–439, Springer-Verlag, 2001.
10. Elisabeth Oswald and Manfred Aigner, "Randomized Addition-Subtraction Chains as a Countermeasure against Power Attacks", Cryptographic Hardware and Embedded Systems, CHES 2001, pp. 39–50.

Efficient Subgroup Exponentiation in Quadratic and Sixth Degree Extensions

Martijn Stam[1,⋆] and Arjen K. Lenstra[2]

[1] Technische Universiteit Eindhoven
P.O.Box 513, 5600 MB Eindhoven, The Netherlands
stam@win.tue.nl
[2] Citibank, N.A. and Technische Universiteit Eindhoven
1 North Gate Road, Mendham, NJ 07945-3104, U.S.A.
arjen.lenstra@citigroup.com

Abstract. This paper describes several speedups for computation in the order $p+1$ subgroup of $\mathbf{F}_{p^2}^*$ and the order p^2-p+1 subgroup of $\mathbf{F}_{p^6}^*$. These results are in a way complementary to LUC and XTR, where computations in these groups are sped up using trace maps. As a side result, we present an efficient method for XTR with $p \equiv 3 \bmod 4$.

Keywords: XTR, LUC, finite field, cyclotomic polynomial.

1 Introduction

Many cryptographic protocols rely on the assumed hardness of the discrete logarithm problem in certain groups. Well known examples are prime order subgroups of $\mathbf{Z}_p^*$ or of elliptic curves. Let G_x, for a positive integer x, denote a cyclic (sub)group of order x. In this paper we focus on subgroups G_q of $\mathbf{F}_{p^d}^*$ with q a prime dividing the d-th cyclotomic polynomial Φ_d evaluated at p. The cryptographic relevance of these groups was already pointed out in [11]: other subgroups of $\mathbf{F}_{p^d}^*$ can be embedded in a true subfield of $\mathbf{F}_{p^d}^*$, thereby making the discrete logarithm computation substantially easier.

Computation in finite fields is a well studied problem. However, research tends to emphasize on bilinear complexity [10], asymptotic complexity [24], or binary characteristic [1]. The case of large prime characteristic with small extension degree has been studied less extensively [5,11,2]. Moreover, usually the entire field is discussed, while hardly any attempt is made to look closely at the cryptographically interesting cyclotomic subgroup. An exception is the aforementioned article [11], but there the problem is not addressed in full detail. In this paper, we consider the groups $\mathsf{G}_{p+1} \subset \mathbf{F}_{p^2}^*$ and $\mathsf{G}_{p^2-p+1} \subset \mathbf{F}_{p^6}^*$.

Currently the fastest exponentiation methods in the subgroups G_{p+1} and G_{p^2-p+1} use trace maps, resulting in respectively LUC [20] and XTR [13]. They have the additional benefit of reducing the size of the representation of subgroup elements to a half respectively a third of the traditional representation. Application of LUC and XTR is advantageous in protocols where the subgroup operations are restricted to additions, and single or double exponentiations. But for

⋆ The first author is sponsored by STW project EWI.4536

B.S. Kaliski Jr. et al. (Eds.): CHES 2002, LNCS 2523, pp. 318–332, 2003.

more involved protocols that also require ordinary multiplications of subgroup elements or triple (or larger) exponentiations, they may lead to cumbersome manipulations that outweigh the computational advantages. As a consequence, using trace based representations in more complicated protocols may be inconvenient (unless of course the small representation size is crucial).

For that reason, we consider in this paper how exponentiation speedups in G_{p+1} and G_{p^2-p+1} can be achieved in such a way that other operations are not affected, i.e., while avoiding trace based compression methods. For quadratic extensions we show that for both $p \equiv 2 \bmod 3$ and $p \equiv 3 \bmod 4$ inversions in $\mathsf{G}_{p+1} \subset \mathbf{F}_{p^2}^*$ come for free, and that squaring in G_{p+1} is cheaper than in the field $\mathbf{F}_{p^2}$. This results in single and double exponentiations that cost about 60% and 75%, respectively, of traditional methods. Both methods are still considerably slower than LUC (see also [22]).

Our main result concerns sixth degree extensions, i.e., the case $\mathsf{G}_{p^2-p+1} \subset \mathbf{F}_{p^6}^*$. We show that for both $p \equiv 2 \bmod 9$ and $p \equiv 5 \bmod 9$ inversions in G_{p^2-p+1} are very cheap, while squaring in G_{p^2-p+1} is substantially faster than in $\mathbf{F}_{p^6}$. Moreover, the methods from [8,22] can be used to transform a k-bit single exponentiation into a $k/2$-bit double exponentiation (i.e., the product of two $k/2$-bit exponentiations). Using appropriate addition chains this results in a vastly improved single exponentiation routine, that takes approximately 26% of the time cited in [13, Lemma 2.1.2.*iii*]. The improvement for double exponentiation is less spectacular, requiring an estimated 33% compared to [13, Lemma 2.1.2.*iv*]. Our methods are slightly slower than the improved version of XTR [22], but faster than the original XTR [13].

Our proposed methods do not have the compressed-representation benefits or disadvantages of LUC or XTR. Protocols where our methods compare well to LUC and XTR are especially those based on homomorphic ElGamal encryption [7] such as Brands' protocols [3] and Schoenmakers' verifiable secret sharing scheme [19]. Another example is the Cramer-Shoup protocol [6].

Another consideration is the cost of subgroup membership checking, since the security of several cryptographic protocols stands or falls with the correctness of the generators and proper subgroup membership of other elements. For LUC the cost of the subgroup membership test is negligible. For XTR it is small but not really negligible. Testing membership of G_{p+1} and G_{p^2-p+1} as proposed in this paper only costs a small constant number of operations in the underlying field and is thus negligible, as in LUC.

The proposed methods can also be used in conjunction with LUC and XTR. Given an element in G_{p+1} or G_{p^2-p+1} the cost of computing the LUC respectively XTR representation is negligible. Going from LUC to G_{p+1} requires a square root computation in $\mathbf{F}_p$, going from XTR to G_{p^2-p+1} can be done by computing the roots of a third degree polynomial over $\mathbf{F}_{p^2}$. In both cases extra information is needed to resolve root ambiguities.

Unless indicated otherwise, all logarithms in this paper are natural.

2 Preliminaries

2.1 Computational Model

Throughout this paper we use the following conventions to measure the costs of operations. Let l be a positive integer that will be clear from the context. We use M for the cost of multiplying two l-bit numbers (without modular reduction), S for the cost of squaring an l-bit number (idem), D for reducing a $2l$-bit number modulo an l-bit number, A_1 for adding two l-bit numbers (including a reduction if needed), and A_2 for adding two $2l$-bit numbers (no reduction). A modular addition (of cost A_1) typically boils down to two or three plain l-bit additions (which makes it hard to determine whether $A_1 > A_2$ or vice versa). Consequently, the stated numbers of additions should be taken with a grain of salt. As another example, in Lemma 3.24.*iv* the cost of subgroup squaring is approximated as $2S + 2D + A_1$, assuming that the cost of subtracting one or multiplying by two is negligible compared to A_1 and A_2. Furthermore, the reduction (of cost D) is sometimes fed numbers slightly larger than $2l$-bits.

Anyway, for exponentiations we always switch back to the simplified case $A_1 = A_2 = 0$, $M = D = 0.5$, and $S = 0.3$, assuming some fixed value for l. This corresponds to the model where an l-bit modular multiplication is the unit of measurement, a squaring costs 80% of a modular multiplication, and additions are considered negligible. This simplification facilitates comparisons with other results given in the literature.

2.2 Discrete Logarithm Problem

In this paper it is assumed that the discrete logarithm problem in the order q subgroup G_q of $\mathbf{F}_{p^d}^*$ is sufficiently difficult. Here we briefly review the well known implications of this assumption for the choice of q given p^d.

It follows from the Pohlig-Hellman algorithm [15] that q is best chosen as a prime number. Furthermore, it follows from the Pollard-ρ method [16,23] that $\sqrt{q}$ should be sufficiently large, say at least 2^{80} or 2^{100} depending on the security requirements. Finally, it was shown in [11] that q divides $\Phi_d(p)$ if and only if G_q cannot be embedded in a true subfield of $\mathbf{F}_{p^d}$, under the assumption that $q > d$. This implies that, if a sufficiently large q divides $\Phi_d(p)$, then index calculus method attacks on the discrete logarithm problem in G_q cannot be mounted in any true subfield of $\mathbf{F}_{p^d}$ but must take place in the field $\mathbf{F}_{p^d}$. Thus, such attacks can be expected to take time

$$\exp((1.923 + o(1))(\log p^d)^{1/3}(\log\log p^d)^{2/3}),$$

for $p \to \infty$ and d fixed [9,17,18].

Summarizing, we find that the order q must be a prime of at least, say, 160 bits, irrespective of the value of d. For $d = 2$ we have the additional requirement that q divides $\Phi_2(p) = p + 1$ and that the bit length of the prime p is at least, say, 512. For $d = 6$ the order q divides $\Phi_6(p) = p^2 - p + 1$ and p must be a prime of bit length at least, say, 170.

2.3 Finite Field Representation

In cryptography, d-th degree extensions of finite fields are most commonly represented using either polynomial or normal bases (see [14] for definitions and details). With a proper choice of minimal polynomial (such as a trinomial with small coefficients), polynomial bases allow relatively efficient multiplication and squaring in the sense that the usual reduction stage from a degree $2d-2$ product to the degree $d-1$ result can be performed at the cost of cd additions in the underlying prime field, for a very small constant c. In general, this is not the case for normal bases, but they have the advantage that the Frobenius automorphism can be computed for free. For polynomial bases the Frobenius automorphism can be computed at a small but non-negligible cost. A class of polynomial bases combining the best of both worlds is featured in [5]. They are based on cyclotomic fields. The following theorem, a slight adaptation of [14, Theorem 2.47(ii)], says something about the extension degrees one obtains using cyclotomic fields.

Theorem 2.31 *Given a field $\mathbf{F}_{p^e}$ with p prime and some n coprime to p. Then the n-th cyclotomic field over $\mathbf{F}_{p^e}$ is isomorphic to $\mathbf{F}_{p^{ed}}$ where d is the least positive integer such that $p^{ed} \equiv 1 \bmod n$.*

This theorem implies $d|\phi(n)$. We fix $e = 1$. Furthermore, we concentrate on $d = \phi(n)$, i.e., the case that $p \bmod n$ generates $\mathbf{Z}_n^*$. This requires $\mathbf{Z}_n^*$ to be cyclic, so that n is either 2, 4, the power of an odd prime, or twice the power of an odd prime. We ignore $n = 2$, since it does not lead to a proper extension.

Actually, [5] is concerned with rings $\mathbf{Z}[\gamma]/p\mathbf{Z}[\gamma]$ where n is a prime power, γ is a primitive n-th root of unity, and p is an integer of which primality is to be determined. If p is indeed a prime generating $\mathbf{Z}_n^*$ then $\mathbf{Z}[\gamma]/p\mathbf{Z}[\gamma]$ is isomorphic to $\mathbf{F}_p[\gamma]$ supporting identical representations.

Let $\Gamma = (\gamma, \gamma^2, \ldots, \gamma^d)$ with γ as above, then Γ is a basis of $\mathbf{F}_{p^d}$ over $\mathbf{F}_p$. It is understood that an element $a \in \mathbf{F}_{p^d}$ is represented as $\bar{a} = (a_0, \ldots, a_{d-1}) \in (\mathbf{F}_p)^d$, where $a = \Gamma \cdot \bar{a}^T$. We abuse notation by identifying a and $\bar{a}$.

We are interested in finding fast single and double exponentiations for G_q, where $q|\Phi_d(p)$ (cf. Section 2.2). For that purpose we formulate fast multiplication and squaring methods for $\mathbf{F}_{p^d}$, show that squaring in $\mathsf{G}_{\Phi_d(p)}$ can be done even faster, and that the cost of p-th powering in $\mathbf{F}_{p^d}$ (and thus of inversion in $\mathsf{G}_{\Phi_d(p)}$) is virtually negligible. Of independent interest is membership testing for $\mathsf{G}_{\Phi_d(p)}$.

If $d < 105$, then $\Phi_d(p) = \sum_{i\in\mathcal{P}} p^i - \sum_{i\in\mathcal{N}} p^i$ for appropriate index sets $\mathcal{P}$ and $\mathcal{N}$. Let $a \in \mathbf{F}_{p^d}$. Since $\mathbf{F}_{p^d}^*$ is cyclic, $a \in \mathsf{G}_{\Phi_d(p)}$ if and only if $a^{\Phi_d(p)} = 1$, which is equivalent to $\prod_{i\in\mathcal{P}} a^{p^i} = \prod_{i\in\mathcal{N}} a^{p^i}$. Testing this condition requires at most d applications of the Frobenius automorphism and $|\mathcal{P}| + |\mathcal{N}| - 1$ multiplications in $\mathbf{F}_{p^d}$. Thus, for fixed d testing $\mathsf{G}_{\Phi_d(p)}$-membership costs at most $\phi(d)$ multiplications in $\mathbf{F}_{p^d}$. Membership $x \in \mathsf{G}_q$ can be established by verifying that $x^q = 1$.

The relation $\prod_{i\in\mathcal{P}} a^{p^i} = \prod_{i\in\mathcal{N}} a^{p^i}$ gives rise to d possibly dependent relations of degree $|\mathcal{P}|$. In some cases these relations can be used to speed up $|\mathcal{P}|$-th powering in $\mathsf{G}_{\Phi_d(p)}$. This is exploited to get fast squaring in $\mathsf{G}_{\Phi_d(p)}$ for $d = 2, 6$.

A major ingredient when calculating modulo Γ is writing powers $> d$ of γ as linear combinations in Γ. This reduction is performed in two stages. First, all powers higher than n are reduced using $\gamma^n = 1$; next the relation $\Phi_n(\gamma) = 0$ is used to map everything to powers of γ between 1 and $\phi(n)$. Since $d = \phi(n)$, we are done. Note that only additions and subtractions are needed for the reduction.

In [11] only pairs (p, n) are considered for which n is prime and for which p generates $\mathbf{Z}_n^*$, because they lead to so-called optimal normal bases. The relevance of such bases for characteristics > 2 is limited, and the 'cheap' reduction they achieve (just $2d-1$ additions in $\mathbf{F}_p$) is almost met by the somewhat wider class considered above.

2.4 Key Generation

Given n and $d = \phi(n)$ and a desired level of security, key generation consists of two phases: sufficiently large primes p and q have to be found with p generating $\mathbf{Z}_n^*$ and q dividing $\Phi_d(p)$, after which a generator of G_q has to be found.

2.41 Finding p and q. For small d, as in this paper, standard security requirements lead to $\log p > \log q$, cf. Section 2.2. In this case the obvious generalization of the method from [13] can be used. First, an appropriately sized prime q is selected, where $q|\Phi_d(p)$ may impose a priori restrictions on q (e.g., $q \equiv 1 \bmod 3$ for $d = 6$). Next, a root r of $\Phi_d[x] \in \mathbf{F}_q[x]$ is found and p is determined as $r + \ell q$ for $\ell \in \mathbf{Z}_{\geq 0}$ such that p is a large enough prime that generates $\mathbf{Z}_n^*$.

With larger d (or $e > 1$, cf. Theorem 2.31) one may aim for primes p that fit in a computer word (i.e., $\log_2(p) = 32$ or 64). Although this may be advantageous, $\log p$ becomes substantially smaller than $\log q$. We are not aware of an efficient method to find such p and q. If q is selected first, the probability is negligible that an appropriate p exists such that $q|\Phi_{de}(p)$. If p is selected first, there is only a very slim probability that $\Phi_{de}(p)$ has an appropriate prime factor, and finding it leads to an unattractive integer factorization problem. In this paper the possibility $\log p < \log q$ is not further discussed.

2.42 Finding a generator of G_q. This problem is easily solved by selecting $h \in \mathbf{F}_{p^d}$ at random until $g = h^{(p^d-1)/q} \neq 1$, at which point g is the desired generator. A faster method is described in [12, credited to H.W. Lenstra, Jr.]. First an element $h \in \mathsf{G}_{\Phi_d(p)}$ is constructed directly and next $g = h^{\Phi_d(p)/q}$ is computed. If $g = 1$ another h has to be generated. The specifics follow.

Let $f \in \mathbf{F}_p$ and let γ be a primitive n-th root of unity as in Section 2.3. Consider $h_f = (\gamma+f)^{(p^d-1)/\Phi_d(p)} \in \mathsf{G}_{\Phi_d(p)}$. Since $\Phi_d(p)$ divides p^d-1 irrespective of p, we can write $(p^d-1)/\Phi_d(p)$ as $r_+(p) - r_-(p)$ where r_+ and r_- are both polynomials with positive coefficients. The equation $(\gamma+f)^{r_+(p)} = h_f(\gamma+f)^{r_-(p)}$ gives rise to a system of d equations in the coefficients of h_f. Since the system only depends on p's congruency class modulo n (and not on p itself), solving the system can be done before actually picking p. The resulting h_f corresponding to several different choices for f can be hardcoded in the program. In Section 4.4 the details for G_{p^2-p+1} with $p \equiv 2 \bmod 9$ are presented.

2.5 LUC and XTR

For completeness, we give a very brief description of LUC and XTR. LUC [20] is based on the subgroup $\mathsf{G}_{p+1} \subseteq \mathbf{F}_{p^2}^*$ and the trace map $\mathrm{Tr} : \mathbf{F}_{p^2} \to \mathbf{F}_p$ defined by $\mathrm{Tr}(g) = g + g^p$. Since $g \in \mathsf{G}_{p+1}$ implies that $(X-g)(X-g^p) = X^2 - (g+g^p)X + g^p g = X^2 - \mathrm{Tr}(g)X + 1$, the roots of the polynomial $X^2 - \mathrm{Tr}(g)X + 1$ are g and its conjugate g^p. Define $V_n = \mathrm{Tr}(g^n)$, then it can easily be verified that $V_{n+m} = V_n V_m - V_{n-m}$ using $g^p = g^{-1}$ for $g \in \mathsf{G}_{p+1}$. Thus, computation of V_{n+m} from V_n, V_m, and V_{n-m} costs a multiplication (a squaring if $n = m$) in $\mathbf{F}_p$. The V_n coincide with a special instance of the Lucas-function.

XTR [13] is based on the subgroup $\mathsf{G}_{p^2-p+1} \subseteq \mathbf{F}_{p^6}^*$ and the trace map $\mathrm{Tr} : \mathbf{F}_{p^6} \to \mathbf{F}_{p^2}$ defined by $\mathrm{Tr}(g) = g + g^{p^2} + g^{p^4}$. In this case, $g \in \mathsf{G}_{p^2-p+1}$ and its conjugates g^{p^2} and g^{p^4} are the roots of the polynomial $X^3 - \mathrm{Tr}(g)X^2 + \mathrm{Tr}(g)^p X - 1$. Define $c_n = \mathrm{Tr}(g^n)$, then it can be verified that $c_{n+m} = c_n c_m - c_m^p c_{n-m} + c_{n-2m}$. Since the c_n are elements of $\mathbf{F}_{p^2}$, efficient computation of c_{n+m} requires a suitable representation for $\mathbf{F}_{p^2}$ (in particular one that supports cheap Frobenius).

Both LUC and XTR compute $\mathrm{Tr}(g^n)$ instead of g^n and in case of a double exponentiation this would be $\mathrm{Tr}(g^n h^m)$ instead of $g^n h^m$. The necessity of knowing V_{n-m} respectively c_{n-m} and c_{n-2m} makes ordinary exponentiation routines unapplicable. Nevertheless, in either case efficient exponentiation methods exist. However, the shortest addition chain is typically considerably longer than the shortest one. For further details, see [22] and the references contained therein.

3 Quadratic Extensions

In this section we discuss computing in $\mathbf{F}_{p^2}$ and $\mathsf{G}_{p+1} \subset \mathbf{F}_{p^2}^*$. Fast computations in the full field $\mathbf{F}_{p^2}$ with $p \equiv 2 \bmod 3$ are important for XTR and have been discussed in [13]. We show that the field arithmetic for $p \equiv 3 \bmod 4$ from [5, Case $p^k = 4$] can be used for XTR without significant loss of efficiency compared to $p \equiv 2 \bmod 3$. The subgroup G_{p+1} is not relevant for XTR, but it is the subgroup on which LUC is based. We show that it yields some extra computational benefits that are, however, still not competitive with LUC.

We first discuss the field arithmetic for $p \equiv 2 \bmod 3$ in general and then focus on the subgroup. The case $p \equiv 3 \bmod 4$ is dealt with similarly, first the field arithmetic and then the subgroup arithmetic. Suitable exponentiation routines that apply to either case conclude this section.

3.1 Field Representation for $p \equiv 2 \bmod 3$

3.11 Field arithmetic. Let p and q be primes with $p \equiv 2 \bmod 3$ and $q | p+1$. Then p generates $\mathbf{Z}_3^*$ and $\Phi_3(x) = x^2 + x + 1 | x^3 - 1$ is irreducible in $\mathbf{F}_p$. Let γ denote a root of $\Phi_3(x)$, then $\gamma^n = \gamma^{(n \bmod 3)}$ and in particular $\gamma^p = \gamma^2$. Hence $\Gamma = (\gamma, \gamma^2)$ is an optimal normal basis of $\mathbf{F}_{p^2}$ over $\mathbf{F}_p$. Using Γ instead of $(1, \gamma)$ leads to slightly fewer additions than the basis $(1, \gamma)$ discussed in [5, Case

$p = 3$]. The following lemma is easily implied by the formulas from [13, Section 2.1] (cf. [13, Lemma 2.1.1], [22, Lemma 2.2], and [5, Case $p = 3$]).

Lemma 3.12 *Let $a, b, c \in \mathbf{F}_{p^2}$ with $p \equiv 2 \bmod 3$.*

i. Computing a^p is free.
ii. Computing a^2 costs $2M + 2D + 3A_1$.
iii. Computing ab costs $3M + 2D + 2A_1 + 2A_2$.
iv. Computing $ac - bc^p$ costs $4M + 2D + 6A_1 + 2A_2$.

3.13 Subgroup arithmetic. Because $x^{p+1} = 1$ for $x \in \mathsf{G}_{p+1}$, we find that inversion in G_{p+1} is equivalent to p-th powering and thus for free. Let $a = a_0\gamma + a_1\gamma^2$ with $a_0, a_1 \in \mathbf{F}_p$, so $a \in \mathbf{F}_{p^2}$. Then $a \in \mathsf{G}_{p+1}$ if and only if $a^{p+1} = a^p \cdot a = 1$, i.e., $(a_1\gamma + a_0\gamma^2)(a_0\gamma + a_1\gamma^2) = 1$. This is equivalent to $a_0^2 - a_0a_1 + a_1^2 = 1$, so that G_{p+1}-membership testing costs $M + S + D + A_1 + A_2$ plus a comparison with one. This relation can also be exploited to speed up squaring in G_{p+1}, since the value of a_0a_1 follows from a_0^2 and a_1^2 using only a handful of additions. More specifically, $a^2 = (2-2a_0^2-a_1^2)\gamma+(2-a_0^2-2a_1^2)\gamma^2$, which costs $2S+2D+2A_1+3A_2$.

Free inversion in G_{p+1} also results in an advantage for simultaneous computation of ab and ab^{-1} for $a \in \mathbf{F}_{p^2}$ and $b \in \mathsf{G}_{p+1}$: since there are only four possible combinations a_ib_j, four multiplications suffice.

Lemma 3.14 *Let G_{p+1} be the order $p + 1$ subgroup of $\mathbf{F}_{p^2}^*$ with $p \equiv 2 \bmod 3$ and let $a = a_0\gamma + a_1\gamma^2 \in \mathbf{F}_{p^2}$ with $\Phi_3(\gamma) = 0$.*

i. The element a is in $\mathbf{F}_p$ if and only if $a_0 = a_1$.
ii. The element a is in G_{p+1} if and only if $a_0^2 - a_0a_1 + a_1^2 = 1$. Testing this costs $M + S + D + A_1 + A_2$.
iii. Computing a^{-1} for $a \in \mathsf{G}_{p+1}$ is free.
iv. Computing a^2 for $a \in \mathsf{G}_{p+1}$ costs $2S + 2D + 2A_1 + 3A_2$.
v. Computing ab and ab^{-1} for $b \in \mathsf{G}_{p+1}$ costs $4M + 4D + 6\min(A_1, A_2)$.

3.2 Field Representation for $p \equiv 3 \bmod 4$

3.21 Field arithmetic. Let p and q be primes with $p \equiv 3 \bmod 4$ and $q|p + 1$. Then p generates $\mathbf{Z}_4^*$ and $\Phi_4(x) = x^2 + 1$ is irreducible in $\mathbf{F}_p$. Let γ denote a root of $\Phi_4(x)$, then $\Gamma = (1, \gamma)$ is a basis of $\mathbf{F}_{p^2}$ over $\mathbf{F}_p$. (Since $\gamma^2 = -1$ the basis (γ, γ^2) looks contrived and leads to slightly more complicated reductions.) This field representation is identical to [5, Case $p^k = 4$], although the number of additions in our cost functions is slightly different.

Let $a \in \mathbf{F}_{p^2}$ be represented by $(a_0, a_1) \in (\mathbf{F}_p)^2$, i.e., $a = \Gamma \cdot (a_0, a_1)^T = a_0 + a_1\gamma$. From $\gamma^n = \gamma^{(n \bmod 4)}$ and thus $\gamma^p = \gamma^3 = -\gamma$ it follows that $a^p = a_0^p + a_1^p\gamma^p = a_0 - a_1\gamma$ so that p-th powering costs a modular negation. The cost of multiplication is $3M + 2D + 2A_1 + 3A_2$ since $ab = a_0b_0 - a_1b_1 + ((a_0 + a_1)(b_0 + b_1) - a_0b_0 - a_1b_1)\gamma$. The cost of squaring is $2M + 2D + 2A_1$ since $a^2 = (a_0+a_1)(a_0-a_1)+2a_0a_1\gamma$. The cost of computing $ac-bc^p$ for $a, b, c \in \mathbf{F}_{p^2}$ is $4M+2D+2A_1+2A_2$ since $ac-bc^p = (b_0+a_0)c_0+(b_1-a_1)c_1+((a_1-b_1)c_0+(a_0+b_0)c_1)\gamma$. By analogy with Lemma 3.12 we get the following.

Lemma 3.22 *Let $a, b, c \in \mathbf{F}_{p^2}$ with $p \equiv 3 \bmod 4$.*

i. Computing a^p costs A_1.
ii. Computing a^2 costs $2M + 2D + 2A_1$.
iii. Computing ab costs $3M + 2D + 2A_1 + 3A_2$.
iv. Computing $ac - bc^p$ costs $4M + 2D + 2A_1 + 2A_2$.

It follows from Lemmas 3.12 and 3.22 and [13] that XTR can be generalized to $p \equiv 3 \bmod 4$ without loss of efficiency compared to $p \equiv 2 \bmod 3$ as in [13].

3.23 Subgroup arithmetic. As in 3.13, inversion in G_{p+1} is equivalent to p-th powering; it costs A_1. Let $a = a_0 + a_1\gamma$ with $a_0, a_1 \in \mathbf{F}_p$, so $a \in \mathbf{F}_{p^2}$. Then $a \in \mathsf{G}_{p+1}$ if and only if $a^{p+1} = a^p \cdot a = 1$, i.e., $(a_0 - a_1\gamma)(a_0 + a_1\gamma) = 1$ which is equivalent to $a_0^2 + a_1^2 = 1$. So, G_{p+1}-membership testing costs $2S + D + A_2$. It also follows that $a^2 = 2a_0^2 - 1 + ((a_0 + a_1)^2 - 1)\gamma$ for $a \in \mathsf{G}_{p+1}$, which implies that squaring in G_{p+1} can be done faster than in $\mathbf{F}_{p^2}$.

Lemma 3.24 *Let G_{p+1} be the order $p+1$ subgroup of $\mathbf{F}_{p^2}^*$ with $p \equiv 3 \bmod 4$ and let $a = a_0 + a_1\gamma \in \mathbf{F}_{p^2}$ with $\Phi_4(\gamma) = 0$.*

i. The element a is in $\mathbf{F}_p$ if and only if $a_1 = 0$.
ii. The element a is in G_{p+1} if and only if $a_0^2 + a_1^2 = 1$. Testing this costs $2S + D + A_2$.
iii. If $a \in \mathsf{G}_{p+1}$, then computing a^{-1} costs A_1.
iv. If $a \in \mathsf{G}_{p+1}$, then computing a^2 costs $2S + 2D + A_1$.
v. Computing ab and ab^{-1} for $b \in \mathsf{G}_{p+1}$ costs $4M + 4D + 6\min(A_1, A_2)$.

3.3 Subgroup Exponentiation

For a single exponentiation we have to compute a^m, where m has roughly the same bitlength k as q. With signed flexible windows [4] of size 5, this requires about $k+1$ squarings and $7 + k/6$ multiplications in G_q. With Lemmas 3.14.*iv* and 3.24.*iv* the squaring cost is $\approx (3S + 2D)(k + 1)$ and with Lemmas 3.12.*iii* and 3.22.*iii* the multiplication cost is $\approx (3M + 2D)(7 + k/6)$. Under the assumption that $M \approx D$ and $S \approx 0.3M$ the resulting number of $\mathbf{F}_p$-multiplications is 19.1 for the precomputation plus 2.0 per exponent bit.

For double exponentiation we have to compute $a^m b^n$ for m and n of roughly equal size and with m as above. This can be computed using Solinas' trick [21, See also Appendix A], resulting in k squarings and $k/2$ multiplications in G_q. With G_q-arithmetic as above, this becomes $(2S+2D)k + (3M+2D)k/2 \approx 2.85k$ multiplications in $\mathbf{F}_p$. The precomputation of ab and ab^{-1} uses Lemmas 3.14.*v* and 3.24.*v*. Combination of these observations leads to the following theorem.

Theorem 3.31 *Let p and q be primes with $q|p+1$, $p \equiv 2 \bmod 3$ or $p \equiv 3 \bmod 4$, and $\lceil \log_2 q \rceil = k$. Let a, b be in the order q subgroup G_q of $\mathbf{F}_{p^2}^*$ and $m, n \in (0, q)$. Assuming that $M \approx D$ and $S \approx 0.3M$,*

i. computing a^m costs on average $19.1 + 2k$ multiplications in $\mathbf{F}_p$, and
ii. computing $a^m b^n$ costs on average $4 + 2.85k$ multiplications in $\mathbf{F}_p$.

These results improve previously reported ones, but the resulting exponentiations are less efficient than the LUC exponentiations. So, even though we have several related results concerning improved key selection and other choices of p, we leave the subject of quadratic extensions and move on to sixth degree extensions because there our methods appear to have a more substantial impact.

4 Sixth Degree Extension

In this section fast exponentiation routines for the group $\mathsf{G}_{p^2-p+1} \subset \mathbf{F}_{p^6}^*$ with $p \equiv 2 \bmod 9$ are described. Let f be a sixth degree irreducible polynomial over some ground field, with root γ. Consider the extension induced by γ and represented by a polynomial basis consisting of six consecutive powers of γ, such as $(1, \gamma, \ldots, \gamma^5)$ or $(\gamma, \gamma^2, \ldots, \gamma^6)$. The cost of computation in this representation depends on the general question of how many ground field multiplications are needed to multiply two degree five polynomials, and on the specific question of what f looks like. Therefore, a short word on the multiplication of fifth degree polynomials in general, before going into details about the field representation and the benefits the group offers. These results are then used in the subsequent exponentiation routines. We conclude this section with an improved key selection method.

4.1 Multiplication of Fifth Degree Polynomials

Multiplication of two polynomials of degree five can be done in 18 multiplications plus a handful of additions [2,5]. Indeed, let $G(x) = \sum_{i=0}^{5} g_i x^i$ and $H(x) = \sum_{i=0}^{5} h_i x^i$ be two fifth degree polynomials. Write $G = G_0 + G_1 x^3$ and $H = H_0 + H_1 x^3$ where G_0, G_1, H_0, and H_1 are second degree polynomials. Then

$$GH = G_0 H_0 + (G_0 H_1 + G_1 H_0) x^3 + G_1 H_1 x^6,$$

so that, with $C_0 = G_0 H_0$, $C_1 = G_1 H_1$, and $C_2 = (G_0 - G_1)(H_0 - H_1)$, it follows that

$$GH = C_0 + (C_0 + C_1 - C_2) x^3 + C_1 x^6. \tag{1}$$

Each of the C_i can be computed using 6 multiplications in the ground field. For example, because $G_0 = g_0 + g_1 x + g_2 x^2$ and $H_0 = h_0 + h_1 x + h_2 x^2$,

$$C_0 = g_0 h_0 + (g_1 h_0 + g_0 h_1) x + (g_2 h_0 + g_1 h_1 + g_0 h_2) x^2 + (g_2 h_1 + g_1 h_2) x^3 + (g_2 h_2) x^4,$$

so that, with $c_0 = g_0 h_0$, $c_1 = g_1 h_1$, $c_2 = g_2 h_2$, $c_3 = (g_0 - g_1)(h_0 - h_1)$, $c_4 = (g_0 - g_2)(h_0 - h_2)$, and $c_5 = (g_1 - g_2)(h_1 - h_2)$, we have that

$$C_0 = c_0 + (c_0 + c_1 - c_3) x + (c_0 + c_1 + c_2 - c_4) x^2 + (c_1 + c_2 - c_5) x^3 + c_2 x^4.$$

With similar expressions for C_1 and C_2 it follows that 18 ground field multiplications (or squarings) suffice to compute the product GH (or the square G^2).

If the g_i and h_i are l-bit numbers, and one is interested in an (unreduced) product with $2l$-bit or sligthly larger coefficients, then computing C_0 costs $6M + 6A_1 + 7A_2$ and the cost of computing GH as in (1) is $18M + 24A_1 + 21A_2$.

It remains to reduce GH modulo f, at a cost depending on f. This is discussed in the remainder of this section for several ground fields $\mathbf{F}_p$. In that case the resulting coefficients must be reduced modulo p at a cost of $6D$ for l-bit p.

4.2 Field Representation for $p \equiv 2 \bmod 9$

4.21 Field arithmetic. Let p be prime with $p \equiv 2 \bmod 9$. Then p generates $\mathbf{Z}_9^*$ and $\Phi_9(x) = x^6 + x^3 + 1$ is irreducible in $\mathbf{F}_p$. Let γ denote a root of $\Phi_9(x)$, then $\Gamma = (\gamma, \gamma^2, \ldots, \gamma^6)$ is a basis for $\mathbf{F}_{p^6}$ over $\mathbf{F}_p$ (in [5, Case $p^k = 9$] the similar basis $(1, \gamma, \ldots, \gamma^5)$ is used).

Let $a = \sum_{i=0}^{5} a_i \gamma^{i+1} \in \mathbf{F}_{p^6}$. From $\gamma^n = \gamma^{n \bmod 9}$ and thus $\gamma^p = \gamma^2$ it follows with $\Phi_9(\gamma) = 0$ that $a^p = a_4\gamma + (a_0 - a_3)\gamma^2 + a_5\gamma^3 + a_1\gamma^4 - a_3\gamma^5 + a_2\gamma^6$. Thus, p-th powering costs A_1. In a similar way it follows that p^3-th powering costs $2A_1$. For multiplication in $\mathbf{F}_{p^6}$ the method from Section 4.1 is used, with proper adjustment of the powers of x, e.g., $G = G_0 x + G_1 x^4$. It follows with straightforward bookkeeping that collecting corresponding powers of x in Relation (1) combined with the modular reductions costs $12A_2 + 6D$. (For the basis $(1, \gamma, \ldots, \gamma^5)$ we find that the collecting phase costs $14A_2$, which slightly improves the $18A_2$ reported in [5].) With Section 4.1 it follows that multiplication can be done for $18M + 6D + 24A_1 + 33A_2$. Doing more elaborate collecting reduces the $33A_2$ to $29A_2$. Squaring follows by replacing $18M$ by $18S$, but it can be done substantially faster by observing that

$$G^2 = (G_0\gamma + G_1\gamma^4)^2 = (G_0 - G_1)(G_0 + G_1)\gamma^2 + (2G_0 - G_1)G_1\gamma^5,$$

with $G_0, G_1 \in \mathbf{F}_p[\gamma]$ of degree two. Computing this costs $9A_1$ for the preparation of the multiplicands, two polynomial multiplications costing $6M + 6A_1 + 7A_2$ each, $7A_2$ for the collection, and $6D$ for the final reductions. It follows that squaring can be done for $12M + 6D + 21A_1 + 21A_2$. (This is A_2 more than reported in [5] for $(1, \gamma, \ldots, \gamma^5)$.)

Lemma 4.22 *Let $a, b \in \mathbf{F}_{p^6}$ with $p \equiv 2 \bmod 9$.*

i. Computing a^p or a^{p^5} costs A_1.
ii. Computing a^{p^2}, a^{p^3}, or a^{p^4} costs $2A_1$.
iii. Computing a^2 costs $12M + 6D + 21A_1 + 21A_2$.
iv. Computing ab costs $18M + 6D + 24A_1 + 29A_2$.

4.23 Subgroup arithmetic. Let $a = \sum_{i=0}^{5} a_i \gamma^{i+1} \in \mathbf{F}_{p^6}$. Membership of one of the three proper subfields of $\mathbf{F}_{p^6}$ is characterized by one of the equations $a^{p^i} = a$ for $i = 1, 2, 3$. Specifically, $a \in \mathbf{F}_p$ if and only if $a^p = a$ which is equivalent to the system of linear equations $(a_0, a_1, a_2, a_3, a_4, a_5) = (a_4, a_0 - a_3, a_5, a_1, -a_3, a_2)$. The solution $a_0 = a_1 = a_3 = a_4 = 0$ and $a_2 = a_5$ is not surprising since

$1 + \gamma^3 + \gamma^6 = 0$, so an element $c \in \mathbf{F}_p$ takes the form $-c\gamma^3 - c\gamma^6$. Similarly, $a \in \mathbf{F}_{p^2}$ if and only if $a^{p^2} = a$, which is equivalent to $a = a_2\gamma^3 + a_5\gamma^6$, and $a \in \mathbf{F}_{p^3}$ if and only if $a^{p^3} = a$ or $a = (a_3 - a_4)\gamma + (-a_3 + a_4)\gamma^2 + a_5\gamma^3 + a_3\gamma^4 + a_4\gamma^5 + a_5\gamma^6$.

More interesting for cryptographic purposes is the order $p^2 - p + 1$ subgroup G_{p^2-p+1} of $\mathbf{F}_{p^6}^*$, because that subgroup cannot be embedded in a true subfield of $\mathbf{F}_{p^6}$. The G_{p^2-p+1}-membership condition $a^{p^2-p+1} = 1$ is equivalent to $a^{p^2}a = a^p$, which can be verified at a cost of, essentially, a single $\mathbf{F}_{p^6}$-multiplication. From $a^{p^3} = a^{-1}$ it follows that inversion in G_{p^2-p+1} costs $2A_1$.

Computing $a^{p^2}a - a^p = \sum_{i=0}^{5} v_i\gamma^{i+1}$ symbolically produces

$$\begin{aligned}
v_0 &= a_1^2 - a_0a_2 - a_4 - a_4^2 + a_3a_5,\\
v_1 &= -a_0 + a_1a_2 + a_3 - 2a_0a_3 + a_3^2 - a_2a_4 - a_1a_5,\\
v_2 &= -a_0a_1 + a_3a_4 - a_5 - 2a_2a_5 + a_5^2,\\
v_3 &= -a_1 - a_2a_3 + 2a_1a_4 - a_4^2 - a_0a_5 + a_3a_5,\\
v_4 &= a_0^2 + a_1a_2 + a_3 - 2a_0a_3 - a_4a_5,\\
v_5 &= -a_2 + a_2^2 - a_1a_3 - a_0a_4 + a_3a_4 - 2a_2a_5.
\end{aligned} \tag{2}$$

If $a \in \mathsf{G}_{p^2-p+1}$, then $v_i = 0$ for $0 \le i < 6$ and the resulting six relations can be used to significantly reduce the cost of squaring in G_{p^2-p+1}. Let $V = (v_0, v_1, \ldots, v_5)$ be the vector consisting of the v_i's. Then for any 6×6-matrix M, we have that $a^2 + \Gamma \cdot (M \cdot V^T) = a^2$ if $a \in \mathsf{G}_{p^2-p+1}$, because in that case V is the all-zero vector. Carrying out this computation symbolically, involving the expressions for the v_i's for a particular choice of M yields the following:

$$a^2 = a^2 + 2\Gamma \cdot \begin{pmatrix} 0 & 0 & 0 & -1 & 0 & 0\\ 0 & -1 & 0 & 0 & 1 & 0\\ 0 & 0 & 1 & 0 & 0 & 0\\ 1 & 0 & 0 & -1 & 0 & 0\\ 0 & -1 & 0 & 0 & 0 & 0\\ 0 & 0 & 0 & 0 & 0 & 1 \end{pmatrix} \cdot V^T = \Gamma \cdot \begin{pmatrix} 2a_1 + 3a_4(a_4 - 2a_1)\\ 2a_0 + 3(a_0 + a_3)(a_0 - a_3)\\ -2a_5 + 3a_5(a_5 - 2a_2)\\ 2(a_1 - a_4) + 3a_1(a_1 - 2a_4)\\ 2(a_0 - a_3) + 3a_3(2a_0 - a_3)\\ -2a_2 + 3a_2(a_2 - 2a_5) \end{pmatrix}. \tag{3}$$

Given that we are working over a sixth degree extension, the six multiplications and reductions required for (3) seem optimal. The additions can be taken care of in several ways; a reasonable solution results in $6M + 6D + 9A_1 + 12A_2$.

Lemma 4.24 *Let G_{p^2-p+1} be the order $p^2 - p + 1$ subgroup of $\mathbf{F}_{p^6}^*$ with $p \equiv 2 \bmod 9$ and let $a = a_0\gamma + a_1\gamma^2 + \cdots + a_5\gamma^6 \in \mathbf{F}_{p^6}$ with $\Phi_9(\gamma) = 0$.*

i. The element a is in $\mathbf{F}_p$ if and only if $a = a_2\gamma^3 + a_2\gamma^6$.
ii. The element a is in $\mathbf{F}_{p^2}$ if and only if $a = a_2\gamma^3 + a_5\gamma^6$.
iii. The element a is in $\mathbf{F}_{p^3}$ if and only if $a = (a_3 - a_4)\gamma + (-a_3 + a_4)\gamma^2 + a_5\gamma^3 + a_3\gamma^4 + a_4\gamma^5 + a_5\gamma^6$.
iv. The element a is in G_{p^2-p+1} if and only if in relations (2) $v_i = 0$ for $0 \le i < 6$. This can be checked at a cost of essentially $18M + 6D$.
v. Computing a^{-1} for $a \in \mathsf{G}_{p^2-p+1}$ costs $2A_1$.
vi. Computing a^2 for $a \in \mathsf{G}_{p^2-p+1}$ costs $6M + 6D + 9A_1 + 12A_2$.

4.3 Subgroup Exponentiation

For a single exponentiation we have to compute a^m, where m has roughly the same bitlength k as q. For the case $q|p^2-p+1$ it is shown in [22, Section 4.4] that m can quickly be written as $m \equiv m_1 + m_2 p \bmod q$ with m_1 and m_2 of bitlength $k/2$. Hence a^m can be rewritten as $a^{m_1}(a^p)^{m_2}$. This can be computed using Solinas' trick [21] at the cost of $k/2$ squarings and $k/4$ multiplications in G_q. Tanja Lange pointed out to us that the precomputation only requires one group multiplication, since $a^p a^{-1} = a^{p^2}$. With Lemmas 4.24.*vi* and 4.22.*iv* the squaring and multiplication costs become $\approx (6M + 6D)k/2$ and $\approx (18M + 6D)k/4$, respectively. Assuming that $M \approx D$ this results in six $\mathbf{F}_p$-multiplications per exponent bit.

A double exponentiation $a^m b^n$, with $\log m \approx \log n$ and m as above, can be rewritten as $a^{m_1}(a^p)^{m_2} b^{n_1}(b^p)^{n_2}$ with $\approx k/2$-bit m_1, m_2, n_1, and n_2. This quadruple exponentiation can be computed using Solinas' trick simultaneously on two pairs of two exponents (paired in any way), resulting in a total of $k/2$ squarings and twice $k/4$ multiplications. With Lemmas 4.24.*vi* and 4.22.*iv* this becomes $(6M + 6D)k/2 + 2(18M + 6D)k/4 \approx 9k$ multiplications in $\mathbf{F}_p$. Combination of these obeservations leads to the following theorem.

Theorem 4.31 *Let p and q be primes with $q|p^2 - p + 1$, $p \equiv 2 \bmod 9$, and $\lceil \log_2 q \rceil = k$. Let a, b be in the order q subgroup G_q of $\mathbf{F}_{p^6}^*$ and $m, n \in (0, q)$. Assuming that $M \approx D$,*

i. *computing a^m costs on average $6 + 6k$ multiplications in $\mathbf{F}_p$, and*
ii. *computing $a^m b^n$ costs on average $12 + 9k$ multiplications in $\mathbf{F}_p$.*

The cost of this $\mathbf{F}_{p^6}$-exponentiation is comparable to XTR, cf. Section 5.

4.4 Key Selection

We elaborate on the improved key selection mentioned in Section 2.42 and similar to [12, Algorithm 4.5]. With $f \in \mathbf{F}_p$ and $h_f = (\gamma + f)^{(p^6-1)/\Phi_6(p)}$ it follows that $h_f(\gamma + f)(\gamma + f)^p = (\gamma + f)^{p^3}(\gamma + f)^{p^4}$. Solving this equation for the coefficients of h_f gives

$$h_f = \frac{\Gamma}{f^6 - f^3 + 1} \cdot \begin{pmatrix} -f + f^2 + 3f^3 - f^4 - 2f^5 \\ -f - 2f^2 + 3f^3 + 2f^4 - 2f^5 \\ (1 - f^2)^3 \\ f - f^2 + f^4 - f^5 \\ f - f^2 + f^4 - f^5 \\ -f^3(1 - 3f + f^3) \end{pmatrix}. \tag{4}$$

This gives $h_{1/2} = \frac{1}{19}(0, -12, 9, 6, 6, 1)$ and $h_2 = -\frac{3}{19}(6, 2, 3, 2, 2, 8/3)$.

Given either $h \in \mathsf{G}_{\Phi_d(p)}$, compute $g = h^{(p^2-p+1)/q}$ using [4] and Lemmas 4.24.*vi* and 4.22.*iv*. Assuming that p is only slightly larger in size than q this takes $8 \log p + 90$ ground field multiplications (note that Theorem 4.31 does not apply). The resulting g generates G_q unless $g = 1$. The probability of failure may be expected to be q^{-1}, independently for each h. This is negligible.

Remark 4.41 Our methods work, and result in identical runtimes, as long as $p \bmod 9$ generates $\mathbf{Z}_9^*$. Since $\phi(\phi(9)) = 2$, the only other case is $p \equiv 5 \bmod 9$. Several other choices of p can be handled in a similar fashion.

5 Timings

All methods were implemented to verify their correctness and runtime characteristics. The table below summarizes runtimes for $\mathsf{G}_q \subset \mathsf{G}_{p^2-p+1} \subset \mathbf{F}_{p^6}^*$ for 170-bit p and q, and compares them to the XTR timings from [13,22]. They are in milliseconds on a 600 MHz Pentium III NT laptop, averaged over 100 random p, q pairs and 100 exponentiations per pair. The timings confirm that our new methods for $\mathbf{F}_{p^6}$-subgroup exponentiation are superior to the original XTR and almost competitive with the faster version of XTR from [22]. This shows that the main reason to use XTR would no longer be its speed, but mostly its compact — and sometimes inconvenient — representation.

Table 1. XTR and $\mathsf{G}_q \subset \mathsf{G}_{p^2-p+1}$ runtimes.

	XTR in [13]	XTR in [22]	G_q
key generation	64 ms	62 ms	85 ms
single exponentiation	10 ms	7.4 ms	8.9 ms
double exponentiation	21 ms	8.6 ms	13 ms

Acknowledgements. We would like to thank Jeroen Doumen, Jan Draisma and Tanja Lange for fruitful discussions, and the CHES 2002 reviewers for their useful comments. Special thanks go to Peter Beelen, whose remarks stimulated this research and improved the group arithmetic for quadratic extensions.

References

1. G. Agnew, R. Mullin, and S. Vanstone. Fast exponentiation in $GF(2^n)$. In C. G. Günther, editor, *Advances in Cryptography—Eurocrypt'88*, volume 330 of *Lecture Notes in Computer Science*, pages 251–255. Springer-Verlag, 1988.
2. D. Bailey and C. Paar. Efficient arithmetic in finite field extensions with application in elliptic curve cryptography. *Journal of Cryptology*, 2000.
3. S. A. Brands. *Rethinking Public Key Infrastructures and Digital Certificates Building in Privacy.* PhD thesis, Technische Universiteit Eindhoven, 1999.
4. H. Cohen. Analysis of the flexible window powering algorithm. Submitted for publication, available from `http://www.math.u-bordeaux.fr/~cohen`, 2001.
5. H. Cohen and A. K. Lenstra. Supplement to implementation of a new primality test. *Mathematics of Computation*, 48(177): S1–S4, 1987.

6. R. Cramer and V. Shoup. A practical public key cryptosystem provably secure against adaptive chosen ciphertext attack. In H. Krawczyk, editor, *Advances in Cryptography—Crypto'98*, volume 1462 of *Lecture Notes in Computer Science*, pages 13–25. Springer-Verlag, 1998.
7. T. ElGamal. A public key cryptosystem and a signature scheme based on discrete logarithms. *IEEE Transactions on Information Theory*, 31(4):469–472, 1985.
8. R. Gallant, R. Lambert, and S. Vanstone. Faster point multiplication on elliptic curves with efficient endomorphisms. In J. Kilian, editor, *Advances in Cryptography—Crypto'01*, volume 2139 of *Lecture Notes in Computer Science*, pages 190–200. Springer-Verlag, 2001.
9. D. M. Gordon. Discrete logarithms in GF(p) using the number field sieve. *SIAM J. Discrete Math.*, 6(1):124–138, 1993.
10. A. Lempel, G. Seroussi, and S. Winograd. On the complexity of multiplication in finite fields. *Theoretical Computer Science*, 22:285–296, 1983.
11. A. K. Lenstra. Using cyclotomic polynomials to construct efficient discrete logarithm cryptosystems over finite fields. In V. Varadharajan, J. Pieprzyk, and Y. Mu, editors, *ACISP'97*, volume 1270 of *Lecture Notes in Computer Science*, pages 127–138. Springer-Verlag, 1997.
12. A. K. Lenstra and E. R. Verheul. Key improvements to XTR. In T. Okamoto, editor, *Advances in Cryptography—Asiacrypt'00*, volume 1976 of *Lecture Notes in Computer Science*, pages 220–233. Springer-Verlag, 2000.
13. A. K. Lenstra and E. R. Verheul. The XTR public key system. In M. Bellare, editor, *Advances in Cryptography—Crypto'00*, volume 1880 of *Lecture Notes in Computer Science*, pages 1–19. Springer-Verlag, 2000.
14. R. Lidl and H. Niederreiter. *Introduction to finite fields and their applications.* Cambridge University Press, 1994.
15. S. C. Pohlig and M. E. Hellman. An improved algorithm for computing logarithms over $gf(p)$ and its cryptographic significance. *IEEE Transactions on Information Theory*, 24:106–110, 1978.
16. J. Pollard. Monte carlo methods for index computation ($\mathrm{mod}\, p$). *Mathematics of Computation*, 32(143):918–924, 1978.
17. O. Schirokauer, Mar. 2000. Personal communication.
18. O. Schirokauer, D. Weber, and T. F. Denny. Discrete logarithms: the effectiveness of the index calculus method. In H. Cohen, editor, *ANTS II*, volume 1122 of *Lecture Notes in Computer Science*, pages 337–361. Springer-Verlag, 1996.
19. B. Schoenmakers. A simple publicly verifiable secret sharing scheme and its application to electronic voting. In M. Wiener, editor, *Advances in Cryptography—Crypto'99*, volume 1666 of *Lecture Notes in Computer Science*, pages 148–164. Springer-Verlag, 1999.
20. P. Smith and C. Skinner. A public-key cryptosystem and a digital signature system based on the Lucas function analogue to discrete logarithms. In J. Pieprzyk and R. Safavi-Naini, editors, *Advances in Cryptography—Asiacrypt'94*, volume 917 of *Lecture Notes in Computer Science*, pages 357–364. Springer-Verlag, 1995.
21. J. A. Solinas. Low-weight binary representations for pairs of integers. Technical report, CACR (University of Waterloo) preprint series, 2001.
22. M. Stam and A. K. Lenstra. Speeding up XTR. In C. Boyd, editor, *Advances in Cryptography—Asiacrypt'01*, volume 2248 of *Lecture Notes in Computer Science*, pages 125–143. Springer-Verlag, 2001.
23. E. Teske. On random walks for Pollard's rho method. *Mathematics of Computation*, 70:809–825, 2001.

24. J. von zur Gathen and M. Nöcker. Exponentiation in finite fields: theory and practice. In T. Mora and H. Mattson, editors, *AAECC-12*, volume 1255 of *Lecture Notes in Computer Science*, pages 88–133. Springer-Verlag, 1997.

A Solinas' Trick

We briefly discuss Solinas' trick for performing a double exponentiation $g^a h^b$ in a group where inversion is cheap. This naturally occurs in the context of elliptic curve cryptography, and also applies to the groups G_q as discussed in this paper.

Let g be a group element and a some exponent. Let the binary expansion of a be $\sum_{i=0}^{k} a_i 2^i$ where all $a_i \in \{0,1\}$. On average, half of the a_i's will be nonzero, hence the square-and-multiply method requires k squarings and $k/2$ multiplications.

If inversion, i.e., the computation of g^{-1} is cheap, single exponentiation can be sped up by using a signed digit representation for the exponent. Once again, write $a = \sum_{i=0}^{k} a_i 2^i$, but relax the condition on the a_i to $a_i \in \{-1,0,1\}$. The representation is no longer unique, but the non-adjacent form (NAF) is. On average, only a third of the a_i's of the NAF will be nonzero. This improves the square-and-multiply method to k squarings and $k/3$ multiplications.

A double exponentiation, i.e., the computation of $g^a h^b$ for given group elements g and h and exponents a and b, can be performed faster than two separate exponentiations using Shamir's trick. If $a = \sum_{i=0}^{k} a_i 2^i$ and $b = \sum_{i=0}^{k} b_i 2^i$ with all $a_i, b_i \in \{0,1\}$, switching to a vector notation $\underline{c} = (ab)^T$ and $\underline{c}_i = (a_i b_i)^T$ leads to $\underline{c} = \sum_{i=0}^{k} \underline{c}_i 2^i$, where $\underline{c}_i \in \{\binom{0}{0}, \binom{0}{1}, \binom{1}{0}, \binom{1}{1}\}$. Assuming a and b to be independent, about three quarter of the columns $\underline{c}_i$ will be nonzero. By precomputing the value gh corresponding to $\underline{c}_i = \binom{1}{1}$ this yields a runtime of k squarings and $3k/4$ multiplications for the square-and-multiply method.

If inversion is for free, one could consider combining the NAF with Shamir's trick. Given two random exponents, each having a NAF of length about k and an expected number of $2k/3$ zeroes, on average in $\frac{4}{9}$ of the positions both a_i and b_i be zero. This leaves $\frac{5}{9}k$ nonzero $\underline{c}_i$'s, resulting in an improvement of the square-and-multiply method to k squarings and $5k/9$ multiplications.

In [21], Solinas noted that computing the NAF's independent of each other might not be optimal to minimize the number of nonzero $\underline{c}_i$'s. As an alternative, the joint sparse form is proposed, that satisfies the following properties:

1. There are at most two consecutive nonzero columns.
2. Adjacent terms do not have opposite sign, i.e., $a_i a_{i+1} \neq -1$ and $b_i b_{i+1} \neq -1$ for all i.
3. If $a_i a_{i+1} = 1$, then $b_i = 0$ and $b_{i+1} = \pm 1$. Similarly for $b_i b_{i+1} = 1$.

An efficient algorithm is given in [21] that computes the Joint Sparse Form and it is proven that on average, half of the resulting columns $\underline{c}_i$ will be nonzero. The running time of the square-and-multiply method thus becomes k squarings and $k/2$ multiplications.

On the Efficient Generation of Elliptic Curves over Prime Fields*

Elisavet Konstantinou[1,2], Yiannis C. Stamatiou[1,2], and Christos Zaroliagis[1,2]

[1] Computer Technology Institute, P.O. Box 1122, 26110 Patras, Greece
[2] Department of Computer Engineering and Informatics, University of Patras, 26500 Patras, Greece
{konstane,stamatiu,zaro}@ceid.upatras.gr

Abstract. We present a variant of the complex multiplication method that generates elliptic curves of cryptographically strong order. Our variant is based on the computation of Weber polynomials that require significantly less time and space resources than their Hilbert counterparts. We investigate the time efficiency and precision requirements for generating off-line Weber polynomials and its comparison to another variant based on the off-line generation of Hilbert polynomials. We also investigate the efficiency of our variant when the computation of Weber polynomials should be made on-line due to limitations in resources (e.g., hardware devices of limited space). We present trade-offs that could be useful to potential implementors of elliptic curve cryptosystems on resource-limited hardware devices.

1 Introduction

Elliptic curve cryptography constitutes a fundamental and efficient technology for public key cryptosystems. One of the most important problems in elliptic curve cryptography is the generation of cryptographically secure elliptic curves over prime fields. One method to achieve this is by repeated applications of point counting [4]: select an elliptic curve (EC) at random, count its order (number of rational points on the curve), and check whether the order is *suitable*, that is, it satisfies certain conditions that guarantee cryptographic strength (i.e., resistance to known attacks). Unfortunately, this method can be extremely slow.

An alternative method which generates ECs of a suitable order is the *Complex Multiplication* (CM) method [1]. This method first determines a suitable order and then constructs an EC of that order. The input to the method is a prime p (representing the order of the prime field) from which the so-called *CM discriminant* D is computed. The EC is generated by constructing certain polynomials based on D and finding their roots.

* This work was partially supported by the IST Programme of EU under contracts no. IST-1999-14186 (ALCOM-FT) and no. IST-1999-12554 (ASPIS), and by the Human Potential Programme of EU under contract no. HPRN-CT-1999-00104 (AMORE).

B.S. Kaliski Jr. et al. (Eds.): CHES 2002, LNCS 2523, pp. 333–348, 2003.

There are two variations of Complex Multiplication, depending on whether Hilbert or Weber polynomials are used (see Section 4), that have two main differences: (i) Hilbert polynomials can have huge coefficients as the discriminant D increases, while for the same discriminant the Weber polynomials have much smaller coefficients and thus are easier to construct; (ii) the roots of the Hilbert polynomial construct directly the EC, while the roots of the Weber polynomial have to be transformed to the roots of its corresponding Hilbert polynomial to construct the EC.

When one considers hardware implementations of the CM method on embedded systems, one problem that immediately arises is that the Hilbert polynomials require high-precision floating point and complex arithmetic (i.e., large registers and floating point units) for their construction and storage. Thus, the Hilbert polynomials do not seem appropriate for hardware implementation that generates them on-line.

To alleviate this shortcoming of Hilbert polynomials, a variant of the CM method was recently proposed in [16] that turns out to be rather efficient. This variant takes as input the discriminant D and then computes the prime field's order p and the order m of the EC. The only condition for cryptographic strength posed on m is that it should be prime. Since Hilbert polynomials depend only on D, they can be precomputed off-line (for various values of D) and stored for subsequent use. Thus, if one wants to build an EC of a specific order (ensuring cryptographic strength) for a certain value of D, then one could simply index the stored Hilbert polynomial using D and, if succeeds in finding the desired curve's order, proceed with the next steps of the CM method.

Although the above variant tackles adequately the efficient construction of ECs, there may still be problems with storing and handling several Hilbert polynomials with huge coefficients on hardware devices (e.g., microcontroller chips) with limited resources. Since in such devices the size of floating-point units and the available memory for data and code are limited, it is desirable to keep their sizes as low as possible. It is perhaps because of this reason that (to the best of our knowledge) the vast majority of language tools developed for such hardware devices are based on ANSI C.

In this paper, we further investigate the space and time efficiency of the CM method by shifting our attention to the Weber polynomials. We present another variant of the CM method, similar to the one given in [16], that uses Weber polynomials. Our variant takes also as input the discriminant D, but selects the field's order p at random (or selects it from a set of prescribed primes) and subsequently computes the curve order m using a different method, requesting that m (is not necessarily prime but it) should satisfy the suitability conditions given in [4, Sec. V.7] for cryptographic strength. We have implemented our variant in ANSI C using the (ANSI C) library GNUMP [7]. Based on this implementation, we have conducted an experimental study over a large number of ECs investigating the precision requirements for the off-line generation of Weber polynomials in comparison with the generation of the corresponding Hilbert polynomials. We were also interested in investigating the efficiency of our variant that uses

precomputed Weber polynomials in comparison to the variant in [16] that uses precomputed Hilbert polynomials, and the efficiency of our variant when constructing Weber polynomials on-line. The latter is of particular importance in space-limited hardware systems.

Our experiments showed that for a wide range of discriminants and polynomial degrees the construction of Weber polynomials requires significantly less time and precision than that required for the construction of the Hilbert polynomials. The experiments revealed a trade-off between the two CM variants depending on the values of D, the polynomial degree h, and the space availability of the hardware environment on which the CM method will be implemented. In particular, our experiments showed that, for several values of D and relatively small values of h, our CM implementation requires many fewer iterations in order to find a suitable curve order m and its time efficiency compares favorably with that reported in [16]. When both h and D are getting relatively large, however, our variant becomes less time efficient than the CM variant in [16] (mainly because of the different method for computing m and finding the roots of polynomials). Hence, if there is sufficient space availability for storing either type of precomputed polynomials, the CM variant in [16] seems beneficial for large values of D and h, while ours is better for smaller values of h. On the other hand, if there are space constraints, the storage of Hilbert polynomials for large values of D and h may not be possible. Our experiments showed that, for several values of D and relatively small values of h, the time of our CM implementation for generating an EC of a suitable order by computing *on-line* the Weber polynomials compares favorably with the time the CM variant in [16] takes to generate ECs of prime order using precomputed Hilbert polynomials. Since a small value of h does not necessarily imply a compromise in security, the on-line construction of Weber polynomials could be used in such cases as an alternative to the off-line construction of Hilbert polynomials. Even in the case where a larger value of h is required, it would be more space-efficient to precompute and store the Weber polynomials for the requested large values of h and compute on-line the Weber polynomials for the smaller values of h.

The rest of the paper is organized as follows. In Section 2, we state briefly some basic definitions and results from elliptic curve theory. In Section 3, we present the basic CM method and our variant, while in Section 4 we describe the construction of the Hilbert and the Weber polynomials, along with some examples aiming at the explanation of their computational requirements. In Section 5 we discuss some implementation related issues, while in Section 6 we discuss our experimental results. We conclude in Section 7.

2 Preliminaries of Elliptic Curve Theory

In this section we review some basic concepts regarding elliptic curves and their definition over finite fields. The interested reader may find additional information in e.g., [4,21]. We also assume familiarity with elementary number theory (see e.g., [5]).

An *elliptic curve* $E(F_p)$ over a finite field F_p, where $p > 3$ and prime, is the set of points $(x, y) \in F_p$ (represented by affine coordinates) which satisfy the equation

$$y^2 = x^3 + ax + b \tag{1}$$

and $a, b \in F_p$ are such that $4a^3 + 27b^2 \neq 0$. The set of solutions (x, y) of Eq. (1) together with a point $\mathcal{O}$, called the *point at infinity*, and a special addition operation define an Abelian group, called the *Elliptic Curve group*. The point $\mathcal{O}$ acts as the identity element (details on how the addition is defined can be found in e.g., [4,21]).

The *order* m of an elliptic curve is the number of the points in $E(F_p)$. The expression $t = p+1-m$ (which measures the difference between m and p) is called the *Frobenius trace* t. Hasse's theorem (see e.g., [4,21]) states that $|t| \leq 2\sqrt{p}$ which gives upper and lower bounds for m based on p:

$$p + 1 - 2\sqrt{p} \leq m \leq p + 1 + 2\sqrt{p}. \tag{2}$$

The *order of a point* P is the smallest positive integer n for which $nP = \mathcal{O}$. Application of Langrange's theorem (see e.g., [5]) on $E(F_p)$, gives that the order of a point $P \in E(F_p)$ always divides the order of the elliptic curve group, so $mP = \mathcal{O}$ for any point $P \in E(F_p)$, which in turn implies that the order of a point cannot exceed the order of the elliptic curve.

Two important quantities associated with $E(F_p)$ are the *curve discriminant* Δ and the *j-invariant*, defined by

$$\Delta = -16(4a^3 + 27b^2) \tag{3}$$

and

$$j = \frac{-1728(4a)^3}{\Delta} \tag{4}$$

Given a j-invariant $j_0 \in F_p$ ($j_0 \neq 0, 1728$), two elliptic curves can be easily constructed. The first EC is of the form defined by Eq. (1) and can be constructed by setting $a = 3k \bmod p$, $b = 2k \bmod p$, where $k = \frac{j_0}{1728 - j_0} \bmod p$. The second EC, called the *twist* of the first, is defined as

$$y^2 = x^3 + ac^2x + bc^3 \tag{5}$$

where c is any quadratic non-residue in F_p. If m_1 is the order of an EC and m_2 is the order of its twist, then $m_1 + m_2 = 2p + 2$, i.e., if one curve has order $p + 1 - t$, then its twist has order $p + 1 + t$, or vice versa [4, Lemma VIII.3].

The security of elliptic curve cryptosystems is based on the difficulty of solving the discrete logarithm problem (DLP) on the EC group. To ensure intractability of solving this problem by all known attacks, the group order m should obey the following conditions:

1. m must have a sufficiently large prime factor (greater than 2^{160}).

2. m must not be equal to p.
3. For all $1 \leq k \leq 20$, it should hold that $p^k \not\equiv 1 \pmod{m}$.

The first condition excludes the application of type of methods like the Pohlig-Hellman [14] one to solve DLP, the second condition excludes the application of the anomalous attack [15,20,22], while the third condition excludes the MOV attack [12]. If the order of an EC group satisfies the above conditions, we shall call it *suitable*.

3 The Complex Multiplication Method and Our Variant

The theory of complex multiplication (CM) of elliptic curves over the rationals can be used to generate elliptic curves of a suitable order m, resulting in the so-called *CM method*. The CM method computes j-invariants from which is then easy to construct the EC. The method is based on the following idea (for more details see [4,8]).

Hasse's theorem implies that $Z = 4p - (p + 1 - m)^2$ is positive. This in turn implies that there is a unique factorization $Z = Dv^2$, where D is a square free positive integer. Consequently,

$$4p = u^2 + Dv^2 \tag{6}$$

for some integer u satisfying

$$m = p + 1 \pm u \tag{7}$$

D is called a *CM discriminant for the prime* p and the elliptic curve has a *CM by* D. The CM method uses D in order to determine the j-invariant and constructs an EC of order $p + 1 - u$ or $p + 1 + u$.

The method starts with a prime p and then chooses the smallest D along with an integer u to satisfy Eq. (6). Then, checks whether $p+1-u$ and/or $p+1+u$ is suitable. If neither is suitable, the process is repeated. Otherwise, the so-called Hilbert polynomials (see Section 4) have to be constructed (based on D) and their roots have to be found. A root of the Hilbert polynomial is the j-invariant we are seeking. The EC and its twist are then constructed as explained in Section 2. Since only one of the ECs has the required suitable order, the particular one can be found using Langrange's theorem by picking random points P in each EC until a point is found in some curve for which $mP \neq \mathcal{O}$. Then, the other curve is the one we are seeking.

A major problem of the CM method is the construction of the Hilbert polynomials which require high precision floating point and complex arithmetic that makes their computation very expensive.

To overcome this problem, a variant of the CM method was proposed in [16]. It takes as input a CM discriminant D ($D \equiv 3 \pmod 4$), and subsequently calculates p and m, where the only condition posed on m is that it should be a prime. The prime p is found by first picking randomly u and v of appropriate

sizes, and then checking if $(u^2 + Dv^2)/4$ is prime. An important aspect of the variant concerns the computation of the Hilbert polynomials: since they depend only on D (and not on p), they can be constructed in a preprocessing phase and stored for later use. Hence, the burden of their construction is excluded from the generation of the EC.

In the rest of the section, we shall describe an alternative to the variant of [16] with which some similarities are shared: our variant takes also as input a CM discriminant D, and then computes p and m. The differences are that it uses Weber instead of Hilbert polynomials, selects p at random (or selects it from a set of prescribed primes), computes u and v in a different way (using Cornacchia's algorithm [6]), and requires m to be suitable (cf. Section 2). Actually, the order m of the elliptic curves that we generate is of the form $m = nq$, where n is a small integer and q is a large prime (larger than 2^{160}). Weber polynomials is the default choice of our variant, since they require much less precision and, as our experiments show, result in much more efficient computation of ECs. (We would like to mention that Hilbert polynomials can be equally used as well.) The polynomials, like in [16], are also constructed in a preprocessing phase.

In the following, we shall give the main steps of our variant. In order to facilitate the discussion of the experiments in Section 6, we will include also the choice of Hilbert polynomials in the description.

Preprocessing Phase.

1. Choose a discriminant $D \equiv 0$ or $3 \pmod 4$ and $D \not\equiv 3 \pmod 8$. In the following section we will explain why this limitation is necessary.
2. Construct the Weber (or the Hilbert) polynomial using the discriminant D.

Main Phase.

3. Produce randomly (or select) a prime p and check whether Eq. (6) has a solution (u, v), where u, v are integers, using Cornacchia's algorithm [6]. This algorithm solves a slightly different form, namely the equation $p = x^2 + dy^2$, but it is easy to convert Eq. (6) into this form. If a solution (u, v) is found, then proceed with the next step. Otherwise, another prime p is chosen and the step is repeated. The prime number p is going to be the order of the underlying finite field F_p.
4. Having found a solution (u, v), the possible orders of the elliptic curve are $m = p + 1 - u$ and $m = p + 1 + u$. Check if (at least) one of them is suitable. If none is suitable, then return to Step 3. Otherwise, m is the order of the elliptic curve that we will generate and proceed to the next step.
5. Compute the roots (modulo p) of the Weber (or Hilbert) polynomial. This is accomplished by using a slight modification of Berlekamp's algorithm [2]. Transform the roots of the Weber polynomial (if it has been chosen) to the roots of the corresponding Hilbert polynomial (constructed using the same D).
6. Each (Hilbert) root computed in Step 5 represents a j-invariant. Construct the two ECs as described in Section 2 (cf. Eq. (1) and (5)).
7. Determine which one of the two ECs is of a suitable order: repeatedly pick random points P on each elliptic curve, until a point is found for which $mP \neq \mathcal{O}$. Then, we are certain that the other curve is the one we seek.

The most complicated part of the CM method is the construction of the polynomials (Weber or Hilbert). This construction is presented in the following section.

4 Construction of Hilbert and Weber Polynomials

In this section we shall elaborate more on the Hilbert and Weber polynomials and discuss their strengths and limitations.

The CM discriminant D is the only input in the construction of Hilbert and Weber polynomials, denoted by $H_D(x)$ and $W_D(x)$ respectively. They both require complex and floating point arithmetic. The drawback of Hilbert polynomials is that their coefficients can be huge and their construction demands high precision. This implies that their construction can be very time consuming and possibly impossible to be implemented in systems of limited memory or with time constraints. Weber polynomials on the other hand, have much smaller coefficients and therefore the precision that is needed for their construction is not very high. In our code, we have implemented both polynomials and in the following sections we present a comparison between them.

The Hilbert polynomial $H_D(x)$, for a given positive value of D, is defined as

$$H_D(x) = \prod_{\tau}(x - j(\tau)) \tag{8}$$

for a set of values of τ given by the expression $\tau = (-\beta + \sqrt{-D})/2\alpha$, for all integers α, β, and γ that satisfy the conditions: (i) $\beta^2 - 4\alpha\gamma = -D$, (ii) $|\beta| \leq \alpha \leq \sqrt{D/3}$, and (iii) $\alpha \leq \gamma$, (iv) $\gcd(\alpha, \beta, \gamma) = 1$, and (v) if $|\beta| = \alpha$ or $\alpha = \gamma$, then $\beta \geq 0$. We shall write $H_D[j](x)$ for $H_D(x)$ when we want to emphasize the class invariant $j(\tau)$ in the construction of the polynomial.

Note that the pairs (α, β) that satisfy the above conditions are finite, which in turn implies that the values of τ are finite and consequently the factors in the Hilbert polynomial in Eq. (8). Let

$$z = e^{2\pi\sqrt{-1}\tau} \quad \text{and} \quad h(\tau) = \frac{\Delta(2\tau)}{\Delta(\tau)} \tag{9}$$

where

$$\Delta(\tau) = z\left(1 + \sum_{n\geq 1}(-1)^n\left(z^{n(3n-1)/2} + z^{n(3n+1)/2}\right)\right)^{24}. \tag{10}$$

Then, the term $j(\tau)$ (the class invariant) is defined as

$$j(\tau) = \frac{(256h(\tau)+1)^3}{h(\tau)}. \tag{11}$$

The method we followed for the construction of the Hilbert polynomials is the one described in [4]. Let h be the *degree* or *class number* of $H_D(x)$. The bit-precision required for the generation of the Hilbert polynomials (see [1,4]) is

$$\text{H-Prec}(D) = v_0 + \binom{h}{\lfloor h/2 \rfloor} \frac{\pi\sqrt{D}}{\ln 2} \sum_{\tau} \frac{1}{\alpha} \tag{12}$$

where the sum runs over the same values of τ as the product in Eq. (8) and v_0 is a positive constant that takes care of rounding errors (typically $v_0 = 33$). Clearly, H-Prec(D) can be rather high.

The above prohibitively large precision required for the computation of the coefficients of Hilbert polynomials (even for moderate values of D) can be circumvented by using the Weber polynomials which required much smaller precision for the computation of their coefficients.

To define the Weber polynomial $W_D(x)$ we follow the approaches in [1,8]. Let α, β, γ be integers that satisfy the conditions: (i) $\beta^2 - \alpha\gamma = -D$, (ii) $|2\beta| \leq \alpha \leq \gamma$, (iii) $\gcd(\alpha, 2\beta, \gamma) = 1$, and (iv) if $2|\beta| = \alpha$ or $\alpha = \gamma$, then $\beta \geq 0$. Additionally, let $\theta = z^{-1}$ and $F(z) = (\Delta(\tau)/z)^{1/24}$ (where the complex number z and the function $\Delta(\tau)$ are those defined in Eq. (9) and (10)).

The construction of the polynomials is based on the so-called Weber functions which are defined as follows:

$$\begin{aligned} f_0(\alpha,\beta,\gamma) &= \theta^{-1/24} F(-\theta)/F(\theta^2) \\ f_1(\alpha,\beta,\gamma) &= \theta^{-1/24} F(\theta)/F(\theta^2) \\ f_2(\alpha,\beta,\gamma) &= \sqrt{2}\ \ \theta^{1/12} F(\theta^4)/F(\theta^2) \end{aligned}$$

For ease of notation, we shall occasionally drop in the following the arguments α, β, γ from the Weber functions. Let

$$\gamma_3 = (f_0^{24} + 8)(f_1^8 - f_2^8)/f_0^8$$

Then, given D, the Weber polynomials are defined as follows:

1. If $D \not\equiv 0 \pmod 3$ and $D \not\equiv 3 \pmod 8$, then
   a) If $D \equiv 7 \pmod 8$, then $W_D(x) = H_{4D}[f_0/\sqrt{2}](x)$
   b) If $D/4 \equiv 2$ or $6 \pmod 8$, then $W_D(x) = H_D[f_1/\sqrt{2}](x)$
   c) If $D/4 \equiv 5 \pmod 8$, then $W_D(x) = H_D[f_0^4](x)$
   d) If $D/4 \equiv 1 \pmod 8$, then $W_D(x) = H_D[f_0^2/\sqrt{2}](x)$
2. If $D \equiv 3 \pmod 6$, then $W_D(x) = H_D[\sqrt{-D}\gamma_3](x)$
3. Otherwise, $W_D(x) = H_D(x)$.

The above mathematical definition can be alternatively summarized by the following equation given in [8], which actually helped us in the implementation to easily construct the polynomials:

$$W_D(x) = \prod_i (x - \mathcal{C}(\alpha_i, \beta_i, \gamma_i)) \tag{13}$$

where $\alpha_i, \beta_i, \gamma_i$ satisfy the above mentioned conditions for α, β, γ, the values of i run over all possible reduced symmetric matrices $\left(\begin{smallmatrix}\alpha_i & \beta_i\\ \beta_i & \gamma_i\end{smallmatrix}\right)$ which have $D = \alpha_i\gamma_i - \beta_i^2$ as a positive square-free determinant, and the function $\mathcal{C}$ is defined as

$$\mathcal{C}(\alpha_i, \beta_i, \gamma_i) = \left[N \exp\left(\frac{-\pi\sqrt{-1}KBL}{24}\right) 2^{-I/6} \left(f_J(\alpha_i, \beta_i, \gamma_i)\right)^K\right]^G$$

where $J \in \{0, 1, 2\}$, $G = \gcd(D, 3)$, $I, K \in [0, 6]$, and L, N are positive integers. The precise values of these parameters depend on certain, rather tedious, conditions among α, γ and D that encompass the various cases of the mathematical definition of the Weber polynomials; the interested reader can find all the details in [8].

The bit-precision required for the construction of the Weber polynomials (see e.g., [23]) is

$$\text{W-Prec}(D) = v_0 + \frac{\pi\sqrt{D}}{\ln 2} \sum_i \frac{1}{\alpha_i} \tag{14}$$

where the sum runs over the same values of i as the product in Eq. (13). Hence, the precision for the construction of the two polynomials differ by a multiplicative factor of $\binom{h}{\lfloor h/2 \rfloor}$. This factor increases as the degree of the polynomials increases. Our experimental results confirm this fact and demonstrate the difference in precision and time efficiency between the construction of Hilbert and Weber polynomials.

To get an idea on the size of coefficients of Hilbert and Weber polynomials as well as on their space requirements for storing them off-line, we next give three examples for different values of D.

$$\begin{aligned} W_{40}(x) &= x^2 - x - 1\\ H_{40}(x) &= x^2 - 425692800x + 910314547200 \end{aligned}$$

$$\begin{aligned} W_{292}(x) &= x^4 - 5x^3 - 10x^2 - 5x + 1\\ H_{292}(x) &= x^4\\ &\quad -2062877098604283046080 0x^3\\ &\quad -936936225119290387594970661120000000x^2\\ &\quad +455215513863793853696299683840000000000x\\ &\quad -380259461042512404779990642688000000000000 \end{aligned}$$

$$
\begin{aligned}
W_{472}(x) &= x^6 - 12x^5 - 22x^3 - 12x - 1 \\
H_{472}(x) &= x^6 - 438370860938320369278668592000x^5 \\
&\quad +2902435100381599559257269068222097663360000000x^4 \\
&\quad -662197893286495898646518596497687462912000000000x^3 \\
&\quad +896632690216502725937652246573453867048960000000000000x^2 \\
&\quad +778276284755579240866437172085664074956800000000000000000x \\
&\quad +(8476837240896000000)^3
\end{aligned}
$$

It is clear that the memory required for the storage of Hilbert polynomials is considerably larger than that required by the Weber polynomials.

Table 1. Transforming a root R_W of a Weber polynomial to a root R_H of the corresponding Hilbert polynomial.

$d \bmod 8$	R_H
1	$\frac{(64R_W^{12}-16)^3}{64R_W^{12}} \bmod p$
2 or 6	$\frac{(64R_W^{12}+16)^3}{64R_W^{12}} \bmod p$
5	$\frac{(64R_W^{6}-16)^3}{64R_W^{6}} \bmod p$
7	$\frac{(R_W^{-24}-16)^3}{R_W^{-24}} \bmod p$

We argued that the use of the Weber polynomials has many advantages compared to the Hilbert polynomials. However, if we choose to use them in the CM method, we must transform their roots to the roots of the corresponding Hilbert polynomials. To accomplish this, the two polynomials must have the same degree in order to associate one root of the Weber polynomial to one of the Hilbert polynomial. For discriminant $D \equiv 3 \pmod 8$ the Weber polynomial's degree is two times the degree of the corresponding Hilbert polynomial and this is the reason why we discard those values of D. A detailed analysis of transforming a root R_W of a Weber polynomial to its corresponding root R_H of a Hilbert polynomial is presented in [23]. The analysis results in a table that summarizes the transformation, a modified version of which (due to a different polynomial representation we use) is presented in Table 1. The value of d is determined as follows. If $D \equiv 0 \pmod 4$, then $d = D/4$, otherwise, $d = D$.

5 Implementation

In this section, we will discuss some issues regarding the implementation of our variant of the Complex Multiplication method. The implementation has been entirely written in ANSI C using the GNU Multiple Precision [7] library for high precision floating point arithmetic and also for the generating and manipulating integers of unlimited precision. Our implementation is also part of a software

library for EC cryptography that we build [11]. The library is available from `http://www.ceid.upatras.gr/faculty/zaro/software/ecc-lib/`.

The GNUMP library, uses as a basic precision unit the *limb*, which is composed of 32 bits. Every floating point number in this library is represented by an integral number of limbs. One may modify the precision with which the floating operations are carried out using a special function that changes the number of limbs. Note, however, that 2 limbs are the minimum precision required by GNUMP for any computation.

As a first step, we implemented the basic algebraic operations for elliptic curve arithmetic. We then turned our attention to the most demanding step of the CM method, which was the construction of the Hilbert and Weber polynomials. They both require high-precision complex and floating point arithmetic with the greater demands placed, of course, by Hilbert polynomials. Also, the operations involved required the implementation of functions such as $\cos(x)$, $\sin(x)$, $\exp(x)$, $\ln(x)$, $\arctan(x)$ and $\sqrt{x}$. Since the basic complex number algebraic operations (addition, multiplication, exponentiation, and squaring) as well as a high precision floating point implementation of the above functions did not exist in GNUMP, we had to implement them from scratch. For the implementation of the particular functions we used their Taylor series expansion. As a starting point for the construction of the Hilbert polynomials, we used the code given in [24] which we considerably modified in order to support high precision floating point arithmetic. For the construction of the Weber polynomials we implemented the functions described in the IEEE Standard P1363 [8], adopting a slightly different way for producing the coefficients α, β, γ described in the standard. For the computation of the roots of polynomials modulo a prime, we used the code given in [24], which we had to modify in order to handle correctly prime numbers of any precision. Finally, the test for the suitability of the order m was done as follows. The order must be of the form $m = nq$, where n is an integer and q is a large prime (greater than 2^{160}). The test proceeds by factoring m and demanding that there are at most four small factors (smaller than 20), while one factor should be prime. If this fails, then the particular m is rejected and the process is repeated. It is easy to see that in this way, q is greater than 2^{160} for sizes of 192 or 224 bits for the field's order, since n is at most 20^4.

6 Experimental Results

Our experiments were carried out on a Pentium III (933 MHz) with 256 MB of main memory, running SuSE-Linux 7.1, and using the ANSI C gcc-2.95.2 compiler (along with the GNUMP library). All reported times are averages over 200 ECs per value of the discriminant D. For the size of the field's order, we considered two values, namely 192 and 224 bits. Our code has size 69KB, including the code for the generation of the polynomials (exclusion of the latter reduces the code size to 56KB).

We first considered experiments regarding the construction of Hilbert and Weber polynomials. Table 2 illustrates, for various values of D and h (degree

of polynomial), the required limb-precision, the number of Taylor terms, and the total time for the construction. As it turns out, the construction of Weber polynomials can be done incredibly faster, and requires a much smaller number of Taylor expansion terms. In addition, it requires only 2 limbs of precision (i.e., the minimum in terms of GNUMP) for all cases considered, while the construction of the Hilbert polynomials requires from 2 to 7 limbs depending on the values of D and h (we noticed that 2 limbs were sufficient for Weber polynomials even for larger values of D and h, e.g., $D = 9640$ and $h = 16$). Note also that the precision required by the Hilbert polynomials increases with D even if h remains the same. An interesting observation concerns the cases marked with an asterisk in Table 2.

Table 2. Construction of Weber and Hilbert polynomials. (*) Coefficients of Hilbert polynomials do not have trailing decimal zeros.

		Weber polynomial			Hilbert polynomial		
D	h	limb-precision	Taylor terms	Time	limb-precision	Taylor terms	Time
20	2	2	6	0.07	2	16	0.59
40	2	2	7	0.09	2	19	0.75
52	2	2	12	0.16	2	24	1.11
88	2	2	13	0.19	3	26	1.38
148	2	2	21	0.40	3	36	2.48
232	2	2	24	0.49	4	39	3.37
39*	4	2	20	0.54	3	32	3.29
56*	4	2	10	0.20	3	63	13.61
68	4	2	11	0.23	3	72	10.84
84*	4	2	18	0.61	3	175	237.82
120	4	2	20	0.70	4	39	7.06
132	4	2	21	0.83	4	38	6.50
136*	4	2	18	0.45	4	130	72.90
168	4	2	23	0.94	4	44	8.82
184*	4	2	22	0.64	4	197	263.55
228	4	2	27	1.26	5	51	13.01
292	4	2	28	1.00	5	90	37.97
116*	6	2	19	0.67	4	190	346.57
152	6	2	20	0.73	5	149	182.24
244*	6	2	27	1.26	5	329	1493.32
472	6	2	36	2.20	7	485	4980.67

The coefficients of the corresponding Hilbert polynomials in these cases do not have trailing decimal zeros and this seems to require a higher number of Taylor terms in order for the computations to converge. We observed that this situation does not occur when D is even and ends in 0, 2 and 8. Since the trailing zeros can be stored in a compact way, this observation would suggest which Hilbert polynomials to consider for off-line computation and storage (if one wishes to use them).

A final remark concerns the comparison of the theoretically required precision, according to Eq. (12) and (14), with that measured experimentally. Our experiments have shown that a smaller precision is required in practice. For example, for $D \in \{232, 292, 472\}$, the equations give for the Hilbert polynomials bit-precisions of H-Prec(232) = 364, H-Prec(292) = 1166, H-Prec(472) = 4983, and for the Weber polynomials bit-precisions W-Prec(232) = 215, W-Prec(292) = 249, W-Prec(472) = 312. Clearly, the precisions given in Table 2 (as multiples of 32 bits) are much smaller than these numbers.

We next turn to the efficiency of our CM implementation using only Weber polynomials. Let $\#p$ denote the number of primes that we had to try in order to find a solution (u, v) using Cornacchia's algorithm (Step 3), and let $\#m$ be the number of orders m that we tried until a suitable one was found. We shall denote by $T(p, m)$ the time required to find a prime p and a suitable order m (Steps 3 and 4), by T_5 the time required for the computation of roots of the polynomial modulo p (Step 5), by T_{67} the time required for the construction of the elliptic curve (Steps 6 and 7), and by T_{main} the total time of the main phase (Steps 3-7) of our variant. The Weber polynomials have been constructed off-line during the preprocessing phase.

Table 3. Timing estimations (in secs) of our CM variant in the 192-bit finite field.

D	h	$\#p$	$\#m$	$T(p,m)$	T_5	T_{67}	T_{main}
232	2	4	5	0.63	0.01	0.32	0.96
568	4	7	6	1.02	0.04	0.33	1.39
1432	6	12	5	1.27	0.09	0.33	1.69
3448	8	15	5	1.34	0.14	0.35	1.83
5272	10	21	5	2.04	0.21	0.38	2.63
8248	12	24	5	2.39	0.32	0.31	3.02
9172	14	28	5	2.80	0.41	0.33	3.54
9640	16	33	6	3.69	0.51	0.39	4.59
9832	18	37	7	4.55	0.76	0.35	5.66
19492	20	42	5	4.78	1.22	0.30	6.30
29908	30	59	5	6.51	1.77	0.40	8.68
39796	50	102	6	11.73	6.11	0.39	18.23
39608	100	195	8	27.42	23.45	0.35	51.22

Table 3 reports the values of the above parameters for various values of D and h and shows where exactly the time is spent throughout the steps of our CM variant. According to [4], we have to try roughly $2h$ primes before a solution can be found by Cornacchia's algorithm. This fact was verified by our experiments with surprising accuracy (cf. the third column of Table 3). The number of trials for order m are approximately the same regardless of the degree of the polynomial, which is reasonable as m is directly associated with the prime p which we choose at random. Therefore, we do not expect that the number of trials required will increase as the discriminant D increases. As expected, all

Table 4. Timing estimations (in secs) of our CM variant.

			192 bits				224 bits			
D	h	$T[W]$	$\#p$	$\#m$	$T(p,m)$	T_{main}	$\#p$	$\#m$	$T(p,m)$	T_{main}
20	2	0.07	4	10	0.82	1.23	4	10	1.30	1.72
40	2	0.09	3	9	0.73	1.12	4	9	1.21	1.60
52	2	0.16	4	8	0.75	1.02	4	8	1.12	1.53
88	2	0.19	4	7	0.61	0.94	4	8	0.98	1.41
232	2	0.49	4	5	0.63	0.96	4	6	0.83	1.30
56	4	0.20	7	10	1.45	1.88	7	11	2.63	3.20
84	4	0.61	7	9	1.40	1.77	8	9	2.29	2.83
136	4	0.45	8	8	1.43	1.80	7	9	2.27	2.77
292	4	1.00	7	6	1.13	1.45	8	8	2.20	2.68
568	4	2.00	7	6	1.02	1.39	8	7	1.98	2.43
116	6	0.67	11	9	1.89	2.33	11	13	3.90	4.39
244	6	1.26	11	9	1.86	2.30	12	11	3.58	4.17
472	6	2.20	11	8	1.52	1.96	12	10	3.33	3.96
1048	6	4.80	13	6	1.45	1.90	11	8	2.92	3.55
1432	6	6.64	12	5	1.27	1.69	11	6	2.03	2.57
376	8	2.26	16	8	2.34	2.79	16	10	4.41	4.95
952	8	6.34	16	7	2.22	2.75	15	9	3.74	4.38
1528	8	9.44	16	7	2.19	2.64	16	6	3.49	4.04
2212	8	16.99	16	6	1.71	2.18	16	6	2.94	3.20
3448	8	23.66	15	5	1.34	1.83	17	5	2.65	3.01
296	10	2.00	20	10	3.70	4.23	19	13	6.58	7.23
724	10	5.33	20	9	3.44	4.07	20	12	6.51	7.19
1268	10	9.80	20	6	2.29	2.85	19	9	4.28	5.01
3412	10	29.17	20	6	2.20	2.76	20	7	3.68	4.37
5272	10	46.49	21	5	2.04	2.63	20	5	2.84	3.54

times (except for T_{67}) increase as the degree h of the polynomial increases. The most time consuming step, as D and h increase, is the computation of the roots of the polynomials.

Table 4 elaborates further by reporting values for the most important parameters regarding various values of D for the same value of h. In the table, $T[W]$ denotes the time for constructing the Weber polynomial. A first observation is that both $T(p,m)$ and T_{main} decrease as D increases, while h remains the same. Another interesting observation is that for reasonably small values of h (which do not necessarily compromise security), our variant remains efficient even in the case where it is required that the computation of Weber polynomials should be made on-line (e.g., due to limited resources posed by hardware devices).

Comparison with related work. The implementation of the CM variant in [16] was done in `C++` using the NTL library [19], which is a high-performance `C++` library for number theory and polynomial arithmetic. Also, their implementation was equipped with clever heuristics to find quickly p and m. Their experiments were

done on a Pentium PC (450 MHz) running Windows NT, considering the same sizes of p (192 and 224 bits) and roughly similar values of D and h as we use. The size of their code was 164KB, excluding the code for precomputing the Hilbert polynomials which was done with MAPLE. In our implementation we didn't use any kind of heuristics. On the positive side, our variant uses considerably fewer iterations to find a suitable m, and is faster[1] compared to the times reported in [16] for (at least) all $h \leq 30$. On the negative side, the construction time of our variant degrades when h increases above 30 and D is sufficiently large. This is due to two reasons: (a) The efficient heuristics used in [16] to find p result in a number of iterations proportional to $ch/\sqrt{D}$ (for some constant $c \approx 300$), while in our variant the number of iterations is roughly $2h$. Hence, the larger the D, the less iterations are made by the variant of [16]. Moreover, our checking of the suitability conditions for m take clearly more time than simply checking on whether m is prime. (b) Our implementation takes more time to find the roots of the Weber polynomial than the time required by the corresponding function of the NTL library. We plan to further investigate the latter issue, as it is clear from Table 3 that it will considerably improve the total time.

There are two other efficient `C++` implementations of the CM method [3,18]. The latter uses the MIRACL [13] library and requires more code space (204KB) than that in [16]. The former uses the advanced `C++` library LiDIA [10] whose adaptation to embedded systems seems very difficult (if at all possible).

7 Conclusions

We have presented an implementation of a variant of the Complex Multiplication method for generating secure ECs. The variant uses Weber polynomials which can be either precomputed off-line and stored as their storage requirements are very low, or (if there are space limitations) can be constructed on-line without sacrificing efficiency (at least for small values of h).

Acknowledgments. We would like to thank the referees for their helpful comments.

References

1. A. O. L. Atkin, F. Morain, Elliptic curves and primality proving, *Mathematics of Computation* 61(1993), pp. 29–67.
2. E. R. Berlekamp, Factoring polynomials over large finite fields, *Mathematics of Computation* 24(1970), pp. 713–735.
3. H. Baier, and J. Buchmann, Efficient construction of cryptographically strong elliptic curves, in *Progress in Cryptology* – INDOCRYPT 2000, Lecture Notes in Computer Science Vol. 1977 (Springer-Verlag, 2000), pp. 191–202.

[1] The times given in our tables should be roughly doubled in order to be compared with the times reported in [16].

4. Ian Blake, Gadiel Seroussi, and Nigel Smart, *Elliptic curves in cryptography* , London Mathematical Society Lecture Note Series 265, Cambridge University Press, 1999.
5. D. Burton, *Elementary Number Theory*, McGraw Hill, 4th edition, 1998.
6. G. Cornacchia, Su di un metodo per la risoluzione in numeri interi dell' equazione $\sum_{h=0}^{n} C_h x^{n-h} y^h = P$, *Giornale di Matematiche di Battaglini* 46 (1908), pp. 33–90.
7. GNU multiple precision library, edition 3.1.1, September 2000. Available at: `http://www.swox.com/gmp`.
8. IEEE P1363/D13, *Standard Specifications for Public-Key Cryptography*, ballot draft, 1999. `http://grouper.ieee.org/groups/1363/tradPK/draft.html`.
9. Implementations of Portions of the P1363 Draft. `http://grouper.ieee.org/groups/1363/P1363/implementations.html`.
10. LiDIA. *A library for computational number theory*, Technical University of Darmstadt. Available from `http://www.informatik.tu-darmstadt.de/TI/LiDIA/Welcome.html`.
11. E. Konstantinou, Y. Stamatiou, and C. Zaroliagis, A Software Library for Elliptic Curve Cryptography, in *Proc. 10th European Symposium on Algorithms* – ESA 2002 (Engineering and Applications Track), Lecture Notes in Computer Science (Springer-Verlag, 2002), to appear.
12. A. J. Menezes, T. Okamoto and S. A. Vanstone, Reducing elliptic curve logarithms to a finite field, *IEEE Trans. Info. Theory*, 39(1993), pp. 1639–1646.
13. Multiprecision Integer and Rational Arithmetic C/C++ Library, `http://indigo.ie/~mscott/`.
14. G. C. Pohlig and M. E. Hellman, An improved algorithm for computing logarithms over $GF(p)$ and its cryptographic significance, *IEEE Trans. Info. Theory*, 24 (1978), pp. 106–110.
15. T. Satoh and K. Araki, Fermat quotients and the polynomial time discrete log algorithm for anomalous elliptic curves, *Comm. Math. Univ. Sancti Pauli*, 47(1998), pp. 81–91.
16. Erkay Savas, Thomas A. Schmidt, and Cetin K. Koc, Generating Elliptic Curves of Prime Order, in *Cryptographic Hardware and Embedded Systems* – CHES 2001, Lecture Notes in Computer Science Vol. 2162 (Springer-Verlag, 2001), pp. 145–161.
17. R. Schoof, Counting points on elliptic curves over finite fields, *J. Theorie des Nombres de Bordeaux*, 7(1995), pp. 219–254.
18. M. Scott, A C++ Implementation of the Complex Multiplication (CM) Elliptic Curve Generation Algorithm from Annex A, in *Implementations of Portions of the P1363 Draft.* `http://grouper.ieee.org/groups/1363/P1363/implementations.html`.
19. V. Shoup, NTL: A Library for doing Number Theory, URL: `http://shoup.net/ntl/`.
20. I. A. Semaev, Evaluation of discrete logarithms on some elliptic curves, *Mathematics of Computation*, 67(1998), pp. 353–356.
21. J. H. Silverman, *The Arithmetic of Elliptic Curves*, Springer-Verlag, GTM 106, 1986.
22. N. P. Smart, The discrete logarithm problem on elliptic curves of trace one, *Journal of Cryptography*, 12(1999), pp. 193–196.
23. Thomas Valente, *A distributed approach to proving large numbers prime*, Rensselaer Polytechnic Institute Troy, New York, Thesis, August 1992.
24. Pate Williams. Available at: `http://www.mindspring.com/~pate`.

An End-to-End Systems Approach to Elliptic Curve Cryptography

Nils Gura, Sheueling Chang Shantz, Hans Eberle, Sumit Gupta, Vipul Gupta, Daniel Finchelstein, Edouard Goupy, and Douglas Stebila

Sun Microsystems Laboratories
{Nils.Gura, Sheueling.Chang, Hans.Eberle, Gupta.Sumit, Vipul.Gupta, Daniel.F.Finchelstein, Edouard.Goupy, Douglas.Stebila}@sun.com
http://www.research.sun.com

Abstract. Since its proposal by Victor Miller [17] and Neal Koblitz [15] in the mid 1980s, Elliptic Curve Cryptography (ECC) has evolved into a mature public-key cryptosystem. Offering the smallest key size and the highest strength per bit, its computational efficiency can benefit both client devices and server machines. We have designed a programmable hardware accelerator to speed up point multiplication for elliptic curves over binary polynomial fields $GF(2^m)$. The accelerator is based on a scalable architecture capable of handling curves of arbitrary field degrees up to $m = 255$. In addition, it delivers optimized performance for a set of commonly used curves through hard-wired reduction logic. A prototype implementation running in a Xilinx XCV2000E FPGA at 66.4 MHz shows a performance of 6987 point multiplications per second for $GF(2^{163})$. We have integrated ECC into OpenSSL, today's dominant implementation of the secure Internet protocol SSL, and tested it with the Apache web server and open-source web browsers.

1 Introduction

Since its proposal by Victor Miller [17] and Neal Koblitz [15] in the mid 1980s, Elliptic Curve Cryptography (ECC) has evolved into a mature public-key cryptosystem. Extensive research has been done on the underlying math, its security strength, and efficient implementations.

ECC offers the smallest key size and the highest strength per bit of any known public-key cryptosystem. This stems from the discrete logarithm problem in the group of points over an elliptic curve. Among the different fields that can underlie elliptic curves, integer fields $F(p)$ and binary polynomial fields $GF(2^m)$ have shown to be best suited for cryptographical applications. In particular, binary polynomial fields allow for fast computation in both software and hardware implementations.

Small key sizes and computational efficiency of both public- and private-key operations make ECC not only applicable to hosts executing secure protocols over wired networks, but also to small wireless devices such as cell phones, PDAs and SmartCards. To make ECC commercially viable, its integration into secure

B.S. Kaliski Jr. et al. (Eds.): CHES 2002, LNCS 2523, pp. 349–365, 2003.
Springer-Verlag Berlin Heidelberg 2003

protocols needs to be standardized. As an emerging alternative to RSA, the US government has adopted ECC for the Elliptic Curve Digital Signature Algorithm (ECDSA) and specified *named curves* for key sizes of 163, 233, 283, 409 and 571 bit [18]. Additional curves for commercial use were recommended by the Standards for Efficient Cryptography Group (SECG) [7]. However, only few ECC-enabled protocols have been deployed in commercial applications to date. Today's dominant secure Internet protocols such as SSL and IPsec rely on RSA and the Diffie-Hellman key exchange. Although standards for the integration of ECC have been proposed [4], they have not yet been finalized.

Our approach towards an end-to-end solution is driven by a scenario of a wireless and web-based environment where millions of client devices connect to a secure web server.

The aggregation of client-initiated connections/transactions leads to high computational demand on the server side, which is best handled by a hardware solution. While support for a limited number of curves is acceptable for client devices, server-side hardware needs to be able to operate on numerous curves. The reason is that clients may choose different key sizes and curves depending on vendor preferences, individual security requirements and processor capabilities. In addition, different types of transactions may require different security levels and thus, different key sizes.

We have developed a cryptographic hardware accelerator for elliptic curves over arbitrary binary polynomial fields $GF(2^m)$, $m \leq 255$. To support secure web transactions, we have fully integrated ECC into OpenSSL and tested it with the Apache web server and open source web browsers.

The paper is structured as follows: Section 2 summarizes related work and implementations of ECC. In Section 3, we outline the components of an ECC-enabled secure system. Section 4 describes the integration of ECC into OpenSSL. The architecture of the hardware accelerator and the implemented algorithms are presented in Section 5. We give implementation cost and performance numbers in Section 6. The conclusions and future directions are contained in Section 7.

2 Related Work

Hardware implementations of ECC have been reported in [20], [2], [1], [11], [10] and [9]. Orlando and Paar describe a programmable elliptic curve processor for reconfigurable logic in [20]. The prototype performs point multiplication based on Montgomery Scalar Multiplication in projective space [16] for $GF(2^{167})$. Their design uses polynomial basis coordinate representation. Multiplication is performed by a digit-serial multiplier proposed by Song and Parhi [22]. Field inversion is computed through Fermat's theorem as suggested by Itoh and Tsujii [13]. With a performance of 0.21 ms per point multiplication this is the fastest reported hardware implementation of ECC. Bednara et al. [2] designed an FPGA-based ECC processor architecture that allows for using multiple squarers, adders and multipliers in the data path. They researched hybrid coordinate representions in affine, projective, Jacobian and López-Dahab form.

Two prototypes were synthesized for $GF(2^{191})$ using an LFSR polynomial basis multiplier and a Massey-Omura normal basis multiplier, respectively. Agnew et al. [1] built an ECC ASIC for $GF(2^{155})$. The chip uses an optimal normal basis multiplier exploiting the composite field property of $GF(2^{155})$. Goodman and Chandrakasan [11] designed a generic public-key processor optimized for low power consumption that executes modular operations on different integer and binary polynomial fields. To our knowledge, this is the only implementation that supports $GF(2^m)$ for variable field degrees m. However, the architecture is based on bit-serial processing and its performance cannot be scaled to levels required by server-type applications.

3 System Overview

Figure 1 shows the implementation of a client/server system using a secure ECC-enhanced protocol. We integrated new cipher suites based on ECC into OpenSSL [19], the most widely used open-source implementation of the Secure Sockets Layer (SSL). More specifically, we added the Elliptic Curve Digital Signature Algorithm (ECDSA), the Elliptic Curve Diffie-Hellman key exchange (ECDH), and means to generate and process X.509 certificates containing ECC keys. We validated our implementation by integrating it with the Apache web server and open-source web browsers Dillo and Lynx running on a handheld client device under Linux. To accelerate public-key operations on the server side, we designed and built a hardware accelerator connected to the host machine through a PCI interface. The accelerator is accessed by a character device driver running under the Solaris Operating Environment.

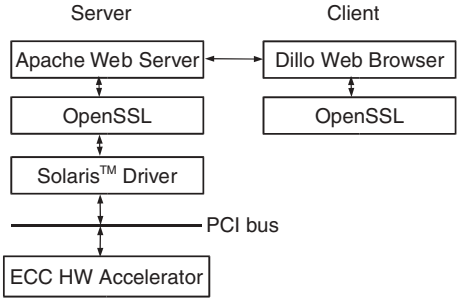


Fig. 1. Secure Client/Server System.

4 Secure Sockets Layer

Secure Sockets Layer (SSL aka TLS) [8] is the most widely deployed and used security protocol on the Internet today. The protocol has withstood years of scrutiny by the security community and, in the form of HTTPS[1], is now trusted to secure virtually all sensitive web-based applications ranging from banking to online trading to e-commerce.

SSL offers encryption, source authentication and integrity protection for data exchanged over insecure, public networks. It operates above a reliable transport service such as TCP and has the flexibility to accommodate different cryptographic algorithms for key agreement, encryption and hashing. However, the

[1] HTTPS is HTTP over an SSL-secured connection.

specification does recommend particular combinations of these algorithms, called *cipher suites*, which have well-understood security properties.

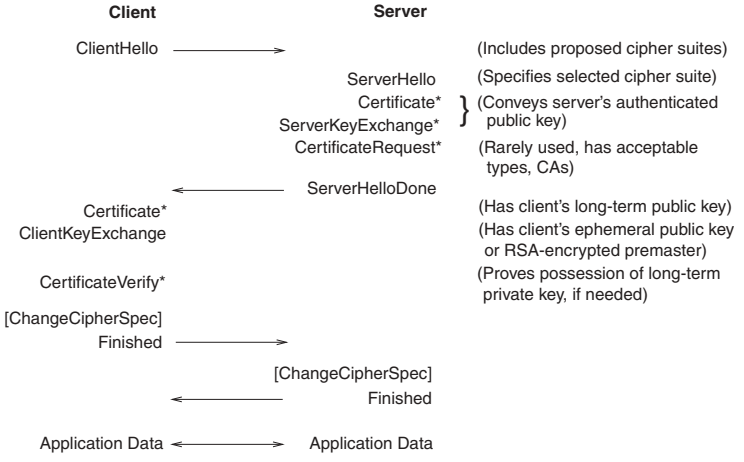


Fig. 2. SSL Handshake for an RSA-based Cipher Suite.

The two main components of SSL are the Handshake protocol and the Record Layer protocol. The Handshake protocol allows an SSL client and server to negotiate a common cipher suite, authenticate each other[2], and establish a shared *master secret* using public-key algorithms. The Record Layer derives symmetric keys from the master secret and uses them with faster symmetric-key algorithms for bulk encryption and authentication of application data. Public-key cryptographic operations are the most computationally expensive portion of SSL processing, and speeding them up remains an active area for research and development.

4.1 Public-Key Cryptography in SSL

Figure 2 shows the general structure of a full SSL handshake. Today, the most commonly used public-key cryptosystem for master-key establishment is RSA but the IETF is considering an equivalent mechanism based on ECC [4].

RSA-based Handshake. The client and server exchange random nonces (used for replay protection) and negotiate a cipher suite with ClientHello and ServerHello messages. The server then sends its signed RSA public-key either in the Certificate message or the ServerKeyExchange message. The client verifies the

[2] Client authentication is optional. Only the server is typically authenticated at the SSL layer and client authentication is achieved at the application layer, e.g. through the use of passwords sent over an SSL-protected channel. However, some deployment scenarios do require stronger client authentication through certificates.

RSA signature, generates a 48-byte random number (the *pre-master secret*) and sends it encrypted with the server's public-key in the ClientKeyExchange. The server uses its RSA private key to decrypt the pre-master secret. Both endpoints then use the pre-master secret to create a master secret, which, along with previously exchanged nonces, is used to derive the cipher keys, initialization vectors and MAC (Message Authentication Code) keys for bulk encryption by the Record Layer.

The server can optionally request client authentication by sending a CertificateRequest message listing acceptable certificate types and certificate authorities. In response, the client sends its private key in the Certificate and proves possession of the corresponding private key by including a digital signature in the CertificateVerify message.

ECC-based Handshake. The processing of the first two messages is the same as for RSA but the Certificate message contains the server's Elliptic Curve Diffie-Hellman (ECDH) public key signed with the Elliptic Curve Digital Signature Algorithm (ECDSA). After validating the ECDSA signature, the client conveys its ECDH public key in the ClientKeyExchange message. Next, each entity uses its own ECDH private key and the other's public key to perform an ECDH operation and arrive at a shared pre-master secret. The derivation of the master secret and symmetric keys is unchanged compared to RSA. Client authentication is still optional and the actual message exchange depends on the type of certificate a client possesses.

5 ECC Hardware Acceleration

Point multiplication on elliptic curves is the fundamental and most expensive operation underlying both ECDH and ECDSA. For a point P in the group $(\{(x,y)|\, y^2 + xy = x^3 + ax^2 + b; x, y \in GF(2^m)\} \cup 0, +_P)$ defined by a non-supersingular elliptic curve with parameters $a, b \in GF(2^m)$ and for a positive integer k, the point multiplication kP is defined by adding P k-1 times to itself using $+_P$[3]. Computing kP is based on a sequence of modular additions, multiplications and divisions. To efficiently support ECC, these operations need to be implemented for large operands.

The design of our hardware accelerator was driven by the need to both provide high performance for named elliptic curves and support point multiplications for arbitrary, less frequently used curves. It is based on an architecture for binary polynomial fields $GF(2^m)$, $m \leq 255$. We believe that this maximal field degree offers adequate security strength for commercial web traffic for the foreseeable future. We chose to represent elements of $GF(2^m)$ in polynomial basis, i.e. polynomials $a = a_{m-1}t^{m-1} + a_{m-2}t^{m-2} + \cdots + a_1 t + a_0$ are represented as bit strings $(a_{m-1}a_{m-2}\ldots a_1 a_0)$.

[3] For a detailed mathematical background on ECC the reader is referred to [3].

5.1 Architectural Overview

We developed a programmable processor optimized to execute ECC point multiplication. The data path shown in Figure 3 implements a 256-bit architecture. Parameters and variables are stored in an 8kB data memory DMEM and program instructions are contained in a 1kB instruction memory IMEM. Both memories are dual-ported and accessible by the host machine through a 64-bit/66MHz PCI interface. The register file contains eight general purpose registers R0-R7, a register RM to hold the irreducible polynomial and a register RC for curve-specific configuration information.

The arithmetic units implement division (DIV), multiplication (MUL) and

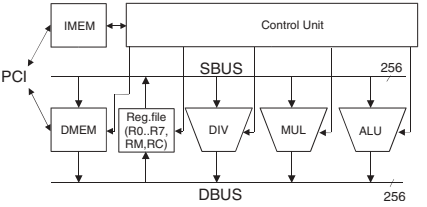


Fig. 3. Data Path and Control Unit.

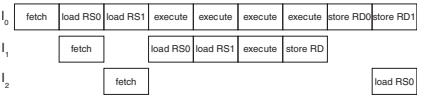


Fig. 4. Parallel Instruction Execution.

squaring/addition/shift left (ALU). Source operands are transferred over the source bus SBUS and results are written back into the register file over the destination bus DBUS.

Program execution is orchestrated by the Control Unit, which fetches instructions from the IMEM and controls the DMEM, the register file and the arithmetic units. As shown in Table 1, the instruction set is composed of memory instructions, arithmetic/logic instructions and control instructions.

Memory instructions LD and ST transfer operands between the DMEM and the register file. The arithmetic and logic instructions include MUL, MULNR, DIV, ADD, SQR and SL. We implemented a load/store architecture. That is, arithmetic and logic instructions can only access operands in the register file. The execution of arithmetic instructions can take multiple cycles and, in case of division and multiplication, the execution time may even be data-dependent. To control the flow of the program execution, conditional branches BMZ and BEQ, unconditional branch JMP and program termination END can be used.

The data path allows instructions to be executed in parallel or overlapped. The Control Unit examines subsequent instructions and decides on the execution model based on the type of instruction and data dependencies. An example for parallel and overlapped execution of an instruction sequence $I_0; I_1; I_2$ is given in Figure 4. Parallel execution of $I_0; I_1$ is possible if I_0 is a MUL or MULNR instruction and I_1 is an ADD or SQR instruction and no data dependencies exist between the destination register/s of I_0 and the source and destination register/s of I_1. Execution of I_1 and I_2 can be overlapped if source register $RS0$

Table 1. Instruction Set.

Instruction Type / Opcode		Name	Semantics	Registers	Cycles
Memory Instr.					
LD	DMEM,RD	Load	DMEM → RD	RD={R0..R7,RM,RC}	3
ST	RS,DMEM	Store	RS → DMEM	RS={R0..R7}	3
Arithmetic Instr.					
DIV	RS0,RS1,RD	Divide	(RS1/RS0) mod M → RD	RS0,RS1,RD={R0..R7}	$\leq 2m+4$
MUL	RS0,RS1,RD	Multiply	(RS0*RS1) mod M → RD	RS0,RS1,RD={R0..R7}	8 (7)
MULNR	RS0,RS1,RD	Multiply w/o Reduction	RS0*RS1 → RD0,RD1	RS0,RS1,RD0,RD1= {R0..R7}	8
ADD	RS0,RS1,RD	Add	RS0+RS1 → RD (RD==0) → EQ	RS0,RS1,RD={R0..R7}	3
SQR	RS,RD	Square	(RS*RS) mod M → RD (RD==0) → EQ	RS,RD={R0..R7}	3
SL	RS,RD	Shift Left	{RS[254..0],0} → RD RS[255] → MZ (RD==0) → EQ	RS,RD={R0..R7}	3
Control Instr.					
BMZ	ADDR	Branch	branch if MZ == 0		2
BEQ	ADDR	Branch	branch if EQ == 1		4
JMP	ADDR	Jump	jump		2
END		End	end program execution		

of I_2 is different from destination register $RD1$ of I_0, i.e. $RS0$ can be read over the SBUS while $RD1$ is written over the DBUS.

5.2 ALU

The ALU incorporates two arithmetic and one logic operation: Addition, squaring and shift left. The addition of two elements $a, b \in GF(2^m)$ is defined as the sum of the two polynomials obtained by adding the coefficients $mod\, 2$. This can be efficiently computed as the bitwise XOR of the corresponding bit strings.

Squaring is a special case of multiplication and is defined in two steps. First, the operand $a \in GF(2^m)$ is multiplied by itself resulting in a polynomial $c_0 = a^2$ of degree less than $2m-1$, i.e. $deg(c_0) < 2m-1$. c_0 may not be an element of the underlying field since its degree may be greater than $m-1$. Second, c_0 is reduced to a congruent polynomial $c \equiv c_0 \, mod\, M$, whereby $c \in GF(2^m)$ is defined as the residue of the polynomial divison of c_0 and the irreducible polynomial M. Squaring a does not require a full multiplication since all mixed terms $a_i a_j t^k, k = 1..2(m-1), k = i+j, i \neq j$ occur twice cancelling each other out. Therefore, $a^2 = a_{m-1} t^{2(m-1)} + a_{m-2} t^{2(m-2)} + \cdots + a_1 t^2 + a_0$ can be easily computed by inserting zeros into the corresponding bit string. For example, squaring $(t^3 + t^2 + t + 1)$ results in $(1111)^2 = 1010101$.

Reduction is based on the congruency

$$u \equiv u + vM \, mod\, M \tag{1}$$

for an irreducible polynomial M and arbitrary polynomials u and v. Since the degree of c_0 is less than $2m-1$, c_0 can be split up into two polynomials $c_{0,h}$ and $c_{0,l}$ with $deg(c_{0,h}) < m-1, deg(c_{0,l}) < m$ such that

$$c_0 = a^2 = c_{0,h} * t^m + c_{0,l} \tag{2}$$

Using $t^m \equiv M - t^m \, mod \, M$ as a special case of (1), the congruency $c_1 = c_{0,h} * (M - t^m) + c_{0,l} \equiv c_0 \, mod \, M$ is obvious. Given that $deg(c_{0,h}) < m - 1$ and $deg(M - t^m) < m$, it follows that $deg(c_1) < 2m - 2$. By iteratively splitting up c_j into polynomials $c_{j,h}$ and $c_{j,l}$ such that

$$c_{j+1} = c_{j,h} * (M - t^m) + c_{j,l} \quad \text{until} \quad c_{j,h} = 0 \Leftrightarrow c_j \in GF(2^m) \tag{3}$$

the reduced result $c = c_i$ can be computed in a maximum of $i \leq m-1$ reduction iterations. The minimum number of iterations depends on the second highest term in the irreducible polynomial M [22], [12]. For

$$M = t^m + t^k + \sum_{j=1}^{k-1} M_j t^j + 1, \; 1 \leq k < m \tag{4}$$

it follows that a better upper bound for $deg(c_1)$ is $deg(c_1) < m+k-1$. Applying (3), $deg(c_j)$ gradually decreases such that

$$deg(c_{j+1,h}) = \begin{cases} deg(c_{j,h}) + k - m & \text{if } deg(c_{j,h}) > m - k \\ 0 & \text{if } deg(c_{j,h}) \leq m - k \end{cases}$$

The minimum number of iterations i is given by

$$m - 1 - i(m - k) \leq 0 \Leftrightarrow \; i \geqslant \lceil \frac{m-1}{m-k} \rceil \tag{5}$$

To enable efficient implementations, M is often chosen to be either a trinomial M_t or pentanomial M_p:

$$M_t = t^m + t^{k_3} + 1, \quad M_p \; = t^m + t^{k_3} + t^{k_2} + t^{k_1} + 1, \quad m > k_3 > k_2 > k_1 > 1$$

Choosing M such that $k_3 \leq \frac{m-1}{2}$ apparently limits the number of reduction iterations to 2, which is the case for all irreducible polynomials recommended by NIST [18] and SECG [7]. The multiplications $c_{j,h} * (M - t^m)$ can be optimized if $(M - t^m)$ is a constant sparse polynomial.

In this case, the two steps of a squaring operation can be hard-wired and executed in a single clock cycle. As shown in Figure 5, the ALU implements hard-wired reduction for the irreducible polynomials $t^{163} + t^7 + t^6 + t^3 + 1, t^{193} + t^{15} + 1$ and $t^{233} + t^{74} + 1$, respectively. Moreover, the ALU can compute addition (XOR) and execute a shift left. It further computes the flags EQ and MZ used by the branch instructions BEQ and BMZ as specified in Table 1.

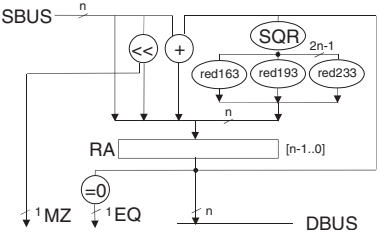


Fig. 5. ALU.

5.3 Multiplier

We studied and implemented several different architectures and, finally, settled on a digit-serial shift-and-add multiplier. Figure 6 gives a block diagram of the multiplier.

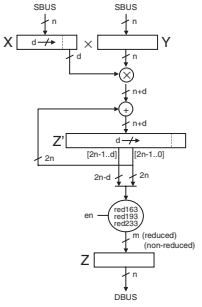


Fig. 6. Shift-and-Add Multiplier.

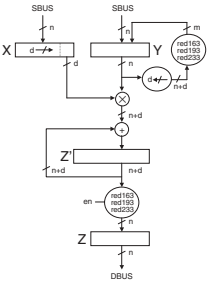


Fig. 7. Least-Significant-Digit-First Multiplier.

The result is computed in two steps. First, the product is computed by iteratively multiplying a digit of operand X with Y, and accumulating the partial products in Z'. Next, the product Z' is reduced by the irreducible polynomial. In our implementation, the input operands X and Y can have a size of up to $n = 256$ bits, and the reduced result Z has a size of $m = 163, 193, 233$ bits according to the specified named curve. The digit size d is 64. We optimized the number of iterations needed to compute the product Z' such that the four iterations it takes to perform a full 256-bit multiplication are only executed for $m = 193, 233$ whereas three iterations are executed for $m = 163$. To compensate for the missing shift operation in the latter case, a multiplexer was added to select the bits of Z' to be reduced. The reduction is hard-wired and takes another clock cycle.

The alternative designs we studied were based on the Karatsuba algorithm [14] and the LSD multiplier [22]. Applying the Karatsuba algorithm to Figure 6, we first split the 64-bit by 256-bit multiplication $X[63..0] * Y[255..0]$ into four 64-bit by 64-bit multiplications $X[63..0] * Y[63..0]$, $X[63..0] * Y[127..64]$, $X[63..0] * Y[191..128]$, $X[63..0] * Y[255..192]$ and then use the Karatsuba algorithm to calculate the four partial products. Compared with the shift-and-add algorithm the Karatsuba algorithm is attractive since it lowers the bit complexity from $O(n^2)$ to $O(n^{lg3})$ [6]. It does, however, introduce irregularities into the wiring and, as a result, additional wire delays. As we will show in Table 3, this design did not meet our timing goal.

We also implemented the LSD multiplier shown in Figure 7. When compared with the shift-and-add multiplier of Figure 6 the LSD multiplier is attractive since it reduces the size of the register used for accumulating the partial results

from $2n$ bits to $n+d$ bits. This is accomplished by shifting the Y operand rather than the product Z' and reducing Y every time it is shifted. The implementation cost is an additional reduction circuit. Since the two reduction operations of Y and Z' do not take place in the same clock cycle, it is possible to share one reduction circuit. However, considering the additional placement and routing constraints imposed by a shared circuit, two separate circuits are, nevertheless, preferred. An analysis of our FPGA implementation shows no advantage in terms of size or performance. The size of the multiplier is dominated by the amount of combinational logic resources and, more specifically, the number of look-up tables (LUTs) needed. Thus, there is no advantage in reducing the size of the register holding Z'. Note, that as the digit size d is reduced, the ratio of registers and LUTs changes; given the fixed ratio of registers and LUTs available on an FPGA device, the LSD multiplier, therefore, can be attractive for small digit sizes.

As it is our goal to process arbitrary curve types, we can rely on the hard-wired reducers only for the named curves. All other curve types need to be handled in a more general way, for example, with the algorithm presented in Section 5.5. We, therefore, need a multiplier architecture that either provides a way to reduce by an arbitrary irreducible polynomial or offers the option to calculate a non-reduced product. We opted for the latter option and added a path to bypass the reducer in Figure 6. Note that with the LSD multiplier a non-reduced product can not be offered thus requiring full multipliers to replace the reduction circuits.

5.4 Divider

The hardware accelerator implements dedicated circuitry for modular division based on an algorithm described by Shantz [21]. A block diagram of the divider is shown in Figure 8. It consists of four 256-bit registers A, B, U and V and a fifth register holding the irreducible polynomial M. It can compute division for arbitrary irreducible polynomials M and field degrees up to $m = 255$.

Initially, A is loaded with the divisor X, B with the irreducible polynomial M, U with the dividend Y, and V with 0. Throughout the division, the following invariants are maintained:

$$A * Y \equiv U * X \bmod M \qquad (6) \qquad\qquad B * Y \equiv V * X \bmod M \qquad (7)$$

Through repeated additions and divisions by t, A and B are gradually reduced to 1 such that U (respectively V) contains the quotient $\frac{Y}{X} \bmod M$. A polynomial is divisible by t if it is even, i.e. the least significant bit of the corresponding bit string is 0. Division by t can be efficiently implemented as a shift right operation. In contrast to the original algorithm, which included magnitude comparisons of registers A and B, we use two counters CA and CB to test for termination of the algorithm. CB is initialized with the field degree m and CA with $m-1$. The division algorithm consists of the following operations:

1. **Division by** t
   a) If even(A) and even(U): $A := \frac{A}{t}$, $CA := CA - 1$
   b) If even(B) and even(V): $B := \frac{B}{t}$, $CB := CB - 1$
2. **Congruent addition of** M
   a) If odd(U): $U := U + M$
   b) If odd(V): $V := V + M$
3. **Addition of** A **and** B
   a) If odd(A) and odd(B) and $CA > CB$: $A := A + B$, $U := U + V$
   b) If odd(A) and odd(B) and $CA \leq CB$: $B := A + B$, $V := U + V$

The preconditions ensure that for any configuration of A, B, U and V at least one of the operations can be executed. It is interesting to note that operations, whose preconditions are satisfied, can be executed in any order without violating invariants (6) and (7). The control logic of the divider chooses operations as preconditions permit starting with 1a and 2a. To ensure termination, 3a is executed if $CA > CB$ and 3b is executed if $CA \leq CB$. CA and CB represent the upper bound for the order of A and B. This is due to the fact that the order of $A + B$ is never greater than the order of A if $CA > CB$ and never greater than the order of B if $CA \leq CB$. Postconditions of 2a, 2b, 3a and 3b guarantee that either 1a or 1b can be executed to further decrease the order of A and B towards 1.

The division circuit shown in Figure 8 was designed to execute sequences of operations per clock cycle, e.g. 3a,2a and 1a could be executed in the same cycle. In particular, it is possible to always execute either 1a or 1b once per clock cycle. Therefore, a modular division can be computed in a maximum of $2m$ clock cycles.

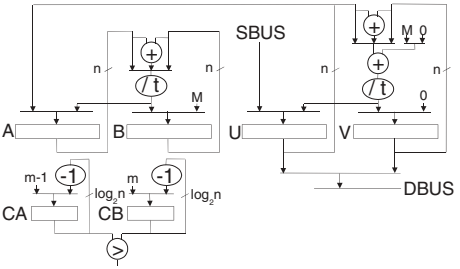


Fig. 8. Divider.

5.5 Point Multiplication Algorithms

We experimented with different point multiplication algorithms and settled on Montgomery Scalar Multiplication using projective coordinates as proposed by López and Dahab [16]. This choice is motivated by the fact that, for our implementation, multiplications can be executed much faster than divisions. Expensive divisions are avoided by representing affine point coordinates (x, y) as projective triples (X, Y, Z) with $x = \frac{X}{Z}$ and $y = \frac{Y}{Z}$. In addition, this algorithm is attractive since it provides protection against timing and power analysis attacks as each point doubling is paired with a point addition such that the sequence of instructions is independent of the bits in k.

A point multiplication kP can be computed with $\lfloor log_2(k) \rfloor$ point additions and doublings. Throughout the computation, only the X- and Z-coordinates of two points $P_{1,i}$ and $P_{2,i}$ are stored. Montgomery's algorithm exploits the fact that for a fixed point $P = (X, Y, 1)$ and points $P_1 = (X_1, Y_1, Z_1)$ and

$P_2 = (X_2, Y_2, Z_2)$ the sum $P_1 + P_2$ can be expressed through only the X- and Z-coordinates of P, P_1 and P_2 if $P_2 = P_1 + P$. P_1 and P_2 are initialized with $P_{1,\lfloor log_2(k) \rfloor} = P$ and $P_{2,\lfloor log_2(k) \rfloor} = 2P$. To compute kP, the bits of k are examined from left ($k_{\lfloor log_2(k) \rfloor}$) to right ($k_0$). For $k_i = 0$, $P_{1,i}$ is set to $2P_{1,i+1}$ (8) and $P_{2,i}$ is set to $P_{1,i+1} + P_{2,i+1}$ (9).

$$X_{1,i} = X_{1,i+1}^4 + bZ_{1,i+1}^4 \qquad Z_{2,i} = (X_{1,i+1} * Z_{2,i+1} + X_{2,i+1} * Z_{1,i+1})^2$$
$$Z_{1,i} = Z_{1,i+1}^2 * X_{1,i+1}^2 \quad (8) \qquad X_{2,i} = XZ_{2,i} + (X_{1,i+1}Z_{2,i+1})(X_{2,i+1}Z_{1,i+1}) \quad (9)$$

Similarly, for $k_i = 1$, $P_{1,i}$ is set to $P_{1,i+1} + P_{2,i+1}$ and $P_{2,i}$ is set to $2P_{2,i+1}$. The Y-coordinate of kP can be retrieved from its X- and Z-coordinates using the curve equation. In projective coordinates, Montgomery Scalar Multiplication requires $6\lfloor log_2(k) \rfloor + 9$ multiplications, $5\lfloor log_2(k) \rfloor + 3$ squarings, $3\lfloor log_2(k) \rfloor + 7$ additions and 1 division.

Named Curves. An implementation of Equations (8) and (9) for named curves over $GF(2^{163})$, $GF(2^{193})$ and $GF(2^{233})$ is shown in Table 2.

Table 2. Implementation and Execution of Projective Point Doubling and Addition.

`// register`	`R0, R1, R2, R3`	Code Execution
`// value`	`X1, Z1, X2, Z2`	
`MUL(R1, R2, R2)`	$R2 = Z_1 * X_2$	`MUL(R1, R2, R2); SQR(R1, R1)`
`SQR(R1, R1)`	$R1 = Z_1^2$	
`MUL(R0, R3, R4)`	$R4 = X_1 * Z_2$	`MUL(R0, R3, R4); SQR(R0, R0)`
`SQR(R0, R0)`	$R0 = X_1^2$	
`ADD(R2, R4, R3)`	$R3 = Z_1 * X_2 + X_1 * Z_2$	`ADD(R2, R4, R3)`
`MUL(R2, R4, R2)`	$R2 = Z_1 * X_2 * X_1 * Z_2$	`MUL(R2, R4, R2); SQR(R1, R4)`
`SQR(R1, R4)`	$R4 = Z_1^4$	
`MUL(R0, R1, R1)`	$R1 = Z_1^2 * X_1^2$	`MUL(R0, R1, R1); SQR(R3, R3)`
`SQR(R3, R3)`	$R3 = Z_3 = (Z_1 * X_2 + X_1 * Z_2)^2$	
`LD(data_mem_b, R5)`	$R5 = b$	`LD(data_mem_b, R5)`
`MUL(R4, R5, R4)`	$R4 = b * Z_1^4$	`MUL(R4, R5, R4); SQR(R0, R0)`
`SQR(R0, R0)`	$R0 = X_1^4$	
`LD(data_mem_Px, R5)`	$R5 = X$	`LD(data_mem_Px, R5)`
`MUL(R3, R5, R5)`	$R4 = X * (Z_1 * X_2 + X_1 * Z_2)^2$	`MUL(R3, R5, R5); ADD(R4, R0, R0)`
`ADD(R4, R0, R0)`	$R0 = X_1^4 + b * Z_1^4$	
`ADD(R2, R5, R2)`	$R2 = X * Z_3 + (Z_1 * X_2) * (X_1 * Z_2)$	`ADD(R2, R5, R2)`

The computation of the two equations is interleaved such that there are no data dependencies for any MUL/SQR or MUL/ADD instruction sequences. Hence, all MUL/SQR and MUL/ADD sequences can be executed in parallel. Furthermore, there are no data dependencies between subsequent arithmetic instructions allowing for overlapped execution.

Generic Curves. Squaring and multiplication require reduction, which can either be hard-wired or implemented for arbitrary field degrees through an instruction sequence of polynomial multiplications (MULNR) and additions (ADD) as shown in Section 5.2. Figure 9 shows a multiplication including reduction. Note

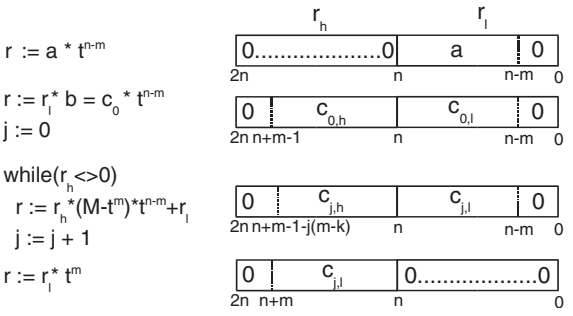


Fig. 9. Non-Hard-Wired Reduction through Multiplication and Addition.

that the multiplier includes registers r_l and r_h, which have a width $n = 256$ not equal to the field degree m. Therefore, the constant factor t^{n-m} is used to align multiplication results to the boundary between r_l and r_h. Computing $c_{j+1,h}$ and $c_{j+1,l}$ from $c_{j,h}$ and $c_{j,l}$ based on Equation (3) requires one MULNR and one ADD instruction. Hence, multiplication and squaring operations with reduction for arbitrary field degrees can be computed with $3 + i$ MULNR and i ADD instructions with i as in Equation (5). Looking at the code sequence of a point multiplication, optimization can be done by storing some multiplication results multiplied by t^{n-m} omitting the first and last step.

6 Implementation and Performance

We specified the hardware in Verilog and prototyped it in a Xilinx Virtex XCV2000E-FG680-7 FPGA using the design tools Synplify 7.0.2 and Xilinx Design Manager 3.3.08i. Area constraints were given for the ALU, the divider and the register file, but no manual placement had to be done. The prototype runs off the PCI clock at a frequency of 66.4 MHz.

Table 3 summarizes the cost and performance of the ALU, the divider and three multiplier design alternatives. The cost is given as the number of used 4-input look-up tables (LUTs) and flip-flops (FFs). The multiplier clearly dominates the design size with 73% of the LUTs and 46% of the FFs. However, multiplication is the single most time-critical operation as shown in Table 4. For point multiplication over $GF(2^{163})$, field multiplications account for almost 62% of the execution time. It is, therefore, justified to allocate a significant portion of the available hardware resources to the multiplier. Parallel and overlapped execution save more than 27% time compared to sequential execution. There is still room for improvements since instructions BMZ, BEQ, SL, JMP and END responsible for the flow control consume almost 21% of the execution time. This time could be saved by separating control flow and data flow.

To evaluate the performance of the divider, we implemented an inversion algorithm proposed by Itoh and Tsujii [13] based on Fermat's theorem. With this algorithm, an inversion optimized for $GF(2^{163})$ takes 938 cycles (0.01413 ms),

Table 3. Cost and Performance of Arithmetic Units.

Unit	LUTs	FFs	f[MHz]
Karatsuba Multiplier	9870	2688	52.2
LSD Multiplier	14347	2592	66.6
Shift-and-Add Multiplier	14241	2990	66.5
ALU	1345	279	80.4 (est.)
Divider	2678	1316	79.6 (est.)
Full Design	19508	6442	66.5

Table 4. Decomposition of the Execution Time for a $GF(2^{163})$ Point Multiplication.

Instruction	#Instr.	Cycles	ms
DIV	1	329	0.00495
ADD	333	666	0.01003
SQR	3	6	0.00009
MUL	10	60	0.00090
MULNR	1	7	0.00011
MUL + ADD	162	972	0.01464
MUL + SQR	810	4860	0.07319
ST	11	33	0.00050
LD	344	688	0.01036
BMZ	326	652	0.00982
BEQ	2	8	0.00012
JMP	162	324	0.00488
SL	326	978	0.01473
END	1	5	0.00008
total		9588	0.14440

while the divider is almost three times faster speeding up point multiplication by about 6.4%.

Table 5 shows hardware and software performance numbers for point multiplication on named and generic curves as well as execution times for ECDH and ECDSA with and without hardware support. The hardware numbers were obtained on a 360MHz Sun Ultra 60 workstation and all software numbers represent a generic 64-bit implementation measured on a 900MHz Sun Fire 280R server. For generic curves, the execution time for point multiplications depends on the irreducible polynomial as described in Sections 5.5 and 5.2. The obtained numbers assume irreducible polynomials with $k_3 \leq \frac{m-1}{2}$. Hard-wired reduction for named curves improves the execution time by a factor of approximately 10 compared to generic curves.

For ECDH-163, the hardware accelerator offers a 12.5-fold improvement in execution time over the software implementation for named curves. Overhead is created by OpenSSL and accesses to the hardware accelerator leading to a lower speedup than measured for raw point multiplication. A disproportionally larger drop in speedup can be observed for ECDSA-163 since it requires two point multiplications and one point addition executed in software. All numbers were measured using a single process on one CPU. The hardware numbers for ECDH and ECDSA could be improved by having multiple processes share the hardware accelerator such that while one processes waits for a point multiplication to finish, another process can use the CPU.

7 Conclusions

We have demonstrated a secure client/server system that employs elliptic curve cryptography for the public-key operations in OpenSSL. We have further presented a hybrid hardware accelerator architecture providing optimized performance for named elliptic curves and support for generic curves over arbitrary

Table 5. Hardware and Software Performance.

	Hardware		Software		Speedup
	ops/s	ms/op	ops/s	ms/op	
Named Curves					
$GF(2^{163})$	6987	0.143	322	3.110	21.7
$GF(2^{193})$	5359	0.187	294	3.400	18.2
$GF(2^{233})$	4438	0.225	223	4.480	19.9
Generic Curves					
$GF(2^{163})$	644	1.554	322	3.110	2.0
$GF(2^{193})$	544	1.838	294	3.400	1.9
$GF(2^{233})$	451	2.218	223	4.480	2.0
ECDH					
$GF(2^{163})$	3813	0.235	304	3.289	12.5
ECDSA (sign)					
$GF(2^{163})$	1576	0.635	292	3.425	5.4
ECDSA (verify)					
$GF(2^{163})$	1224	0.817	151	6.623	8.1

fields $GF(2^m)$, $m \leq 255$. Previous approaches such as presented in [11] and [20] focused on only one of these aspects.

The biggest performance gain was achieved by optimizing field multiplication. However, as the number of cycles per multiplication decreases, the relative cost of all other operations increases. In particular, squarings can no longer be considered cheap. Data transport delays become more critical and contribute to a large portion of the execution time. To make optimal use of arithmetic units connected through shared data paths, overlapped and parallel execution of instructions can be employed.

For generic curves, reduction has shown to be the most expensive operation. As a result, squarings become almost as expensive as multiplications. This significantly impacts the cost analysis of point multiplication algorithms. In particular, the Itoh-Tsujii method becomes much less attractive since it involves a large number of squaring operations.

Dedicated division circuitry leads to a performance gain over soft-coded inversion algorithms for both named and generic curves. However, the tradeoff between chip area and performance needs to be taken into account.

Although prototyped in reconfigurable logic, the architecture does not make use of reconfigurability. It is thus well-suited for an implementation in ASIC technology. For commercial applications this means lower cost at high volumes, less power consumption, higher clock frequencies and tamper resistance.

As for future work, we are in the process of setting up a testbed that will allow us to empirically study the performance of ECC-based cipher suites and compare it to conventional cipher suites. This includes measurements and analysis of the system performance at the web server level. As for the hardware accelerator, we intend to improve the performance of point multiplication on generic curves. Furthermore, we want to optimize the hardware-software interface to achieve higher performance at the OpenSSL level. We plan to discuss the results in a follow-on publication.

Acknowledgments. We would like to thank Jo Ebergen for suggesting to implement the divider with counters rather than with comparators and Marc Herbert for his help with the Solaris driver.

References

[1] Agnew, G.B., Mullin, R.C., Vanstone, S.A.: An Implementation of Elliptic Curve Cryptosystems Over $F_{2^{155}}$. In IEEE Journal on Selected Areas in Communications, 11(5):804–813, June 1993.

[2] Bednara, M., Daldrup, M., von zur Gathen, J., Shokrollahi, J., Teich, J.: Reconfigurable Implementation of Elliptic Curve Crypto Algorithms. Reconfigurable Architectures Workshop, 16th International Parallel and Distributed Processing Symposium, April 2002.

[3] Blake, I., Seroussi, G., Smart, N.: Elliptic Curves in Cryptography. London Mathematical Society Lecture Note Series 265, Cambridge University Press, 1999.

[4] Blake-Wilson, S., Dierks, T., Hawk, C.: ECC Cipher Suites for TLS. Internet draft, `http://www.ietf.org/internet-drafts/draft-ietf-tls-ecc-01.txt`, March 2001.

[5] Blum, T., Paar, C.: High Radix Montgomery Modular Exponentiation on Reconfigurable Hardware. To Appear in IEEE Transactions on Computers.

[6] Borodin, A., Munro, I.: The Computational Complexity of Algebraic and Numeric Problems. American Elsevier, New York, 1975.

[7] Certicom Research: SEC 2: Recommended Elliptic Curve Domain Parameters. Standards for Efficient Cryptography, Version 1.0, September 2000.

[8] Dierks, T., Allen, C.: The TLS Protocol - Version 1.0. IETF RFC 2246, January 1999.

[9] Ernst, M., Klupsch, S., Hauck, O., Huss, S.A.: Rapid Prototyping for Hardware Accelerated Elliptic Curve Public-Key Cryptosystems. 12th IEEE Workshop on Rapid System Prototyping, Monterey, CA, June 2001.

[10] Gao, L., Shrivastava, S., Lee, H., Sobelman, G.: A Compact Fast Variable Key Size Elliptic Curve Cryptosystem Coprocessor. Proceedings of the Seventh Annual IEEE Symposium on Field-Programmable Custom Computing Machines, 1998.

[11] Goodman, J., Chandrakasan, A.P.: An Energy-Efficient Reconfigurable Public-Key Cryptography Processor. In IEEE Journal of Solid-State Circuits, Vol. 36, No. 11, 1808–1820, November 2001.

[12] Halbutoğulları, A., Koç, Ç.K.: Mastrovito Multiplier for General Irreducible Polynomials. In IEEE Transactions on Computers, Vol. 49, No. 5, 503–518, May 2000.

[13] Itoh, T., Tsujii, S.: A Fast Algorithm for Computing Multiplicative Inverses in $GF(2^m)$ Using Normal Bases. In Information and Computation, 78:171–177, 1988.

[14] Karatsuba, A., Ofman, Y.: Multiplication of Many-Digital Numbers by Automatic Computers. Doklady Akad. Nauk, SSSR 145, 293-294. Translation in Physics-Doklady 7, 595–596, 1963.

[15] Koblitz, N.: Elliptic curve cryptosystems. Mathematics of Computation, 48:203–209, 1987.

[16] López, J., Dahab, R.: Fast multiplication on elliptic curves over GF(2m) without precomputation. In CHES '99 Workshop on Cryptographic Hardware and Embedded Systems, Springer-Verlag, Lecture Notes in Computer Science 1717, August 1999.

[17] Miller, V.: Uses of elliptic curves in cryptography. In Lecture Notes in Computer Science 218: Advances in Crytology - CRYPTO '85, pages 417–426, Springer-Verlag, Berlin, 1986.

[18] U.S. Department of Commerce / National Institute of Standards and Technology: Digital Signature Standard (DSS), Federal Information Processing Standards Publication FIPS PUB 186-2, January 2000.

[19] See `http://www.openssl.org/`.

[20] Orlando, G., Paar, C.: A High-Performance Reconfigurable Elliptic Curve Processor for $GF(2^m)$. In CHES '2000 Workshop on Cryptographic Hardware and Embedded Systems, Springer-Verlag, Lecture Notes in Computer Science 1965, August 2000.

[21] Chang Shantz, S.: From Euclid's GCD to Montgomery Multiplication to the Great Divide. Sun Microsystems Laboratories Technical Report TR-2001-95, `http://research.sun.com/`, June 2001.

[22] Song, L., Parhi, K.K.: Low-Energy Digit-Serial/Parallel Finite Field Multipliers. In IEEE Journal of VLSI Signal Processing Systems 19, 149–166, 1998.

A Low-Power Design for an Elliptic Curve Digital Signature Chip

Richard Schroeppel, Cheryl Beaver, Rita Gonzales, Russell Miller, and Timothy Draelos

Sandia National Laboratories** Albuquerque, NM 87185-0785
{rschroe, cbeaver, ragonza, rdmille, tjdrael}@sandia.gov

Abstract. We present a VHDL design that incorporates optimizations intended to provide digital signature generation with as little power, space, and time as possible. These three primary objectives of power, size, and speed must be balanced along with other important goals, including flexibility of the hardware and ease of use. The highest-level function offered by our hardware design is Elliptic Curve Optimal El Gamal digital signature generation. Our parameters are defined over the finite field $GF(2^{178})$, which gives security that is roughly equivalent to that provided by 1500-bit RSA signatures.
Our optimizations include using the point-halving algorithm for elliptic curves, field towers to speed up the finite field arithmetic in general, and further enhancements of basic finite field arithmetic operations. The result is a synthesized VHDL digital signature design (using a CMOS $0.5\mu m$, $5V$, $25^{\circ}C$ library) of 191,000 gates that generates a signature in 4.4 ms at 20 MHz.

Keywords: Digital Signature, Elliptic Curve, ECDSA, Optimal El Gamal, Characteristic 2, Field Towers, Trinomial Basis, Quadratic Equation, Qsolve, Almost-Inverse Algorithm, Point Halving, Signed Sliding Window, GF(2^{89}), GF(2^{178}), Hardware, VHDL, Low Power

1 Introduction

While the value of elliptic curve arithmetic in enabling public-key cryptography to serve in resource-constrained environments is well accepted, efforts in creative implementations continue to bear fruit. A particularly active area is that of hardware implementations of elliptic curve operations, including hardware description language developments, programmable hardware realizations, and fabricated custom circuits. Kim, et al, [1] introduce a hardware architecture to take advantage of a nonconventional basis representation of finite field elements to make point multiplication more efficient. Moon, et al, [2] address field multiplication and division, proposing new methods for fast elliptic curve arithmetic

** Sandia is a multiprogram laboratory operated by Sandia Corporation, a Lockheed Martin Company, for the United States Department of Energy under Contract DE-AC04-94AL85000.

B.S. Kaliski Jr. et al. (Eds.): CHES 2002, LNCS 2523, pp. 366–380, 2003.

appropriate for hardware. Goodman and Chandrakasan [3] tackle the broader problem of providing energy-efficient public-key cryptography in hardware while supporting multiple algorithms, including elliptic curve-based algorithms. Moving closer to applications of elliptic curve cryptography (ECC), Aydos, et al, [4] have implemented an ECC-based wireless authentication protocol that utilizes the elliptic curve digital signature algorithm (ECDSA).

We present a VHDL[1] design that incorporates optimizations intended to provide elliptic curve-based digital signature generation with as little power, space, and time as possible. These three primary objectives of power, size, and speed must be balanced along with other important goals, including flexibility of the hardware (e.g., support of a class of elliptic curves) and ease of use (e.g., doesn't require the user to supply or interpret complex parameters). Currently, the highest-level function offered by our hardware design is digital signature generation. Our elliptic curve parameters are defined over the finite field $GF(2^{178})$, which gives security roughly equivalent to that provided by 1500-bit RSA signatures[2]. As we don't currently have hardware and therefore explicit power measurements, the emphasis of this paper is on a design that reflects many choices promoting a low-power outcome. In addition to the obvious goal of minimizing the number of gates, the speed of execution is critical to power consumption since power can be removed from circuits as soon as they complete their functions.

ECC solutions are well-known for their suitability in smart-card applications and wireless communications security. Our work was motivated by the need to reduce the resources required to provide strong public-key authentication for sensor-based monitoring systems and critical infrastructure protection. For these applications, signature generation is often performed in highly constrained, battery-operated environments, whereas signature verification is performed on desktop systems with only the typical constraint of purchasing power. Hence, our hardware design focused primarily on the signature generation, with signature verification to follow. Here, we present a chip design represented in VHDL of the best to date, in our minds and for our applications, digital signature generation solution for low-power, resource-constrained environments.

In Section 2, we start with the selection of an El Gamal digital signature variant that minimizes the number of operations necessary for signature generation. Section 3 presents algorithmic optimizations of the computational elements necessary to compute a digital signature. We note that many of these elements

[1] VHDL stands for VHSIC Hardware Description Language, where VHSIC stands for Very High Scale Integrated Circuit.

[2] The number of computer instructions to factor a number, N, is estimated as $0.018e^{(1.923\sqrt[3]{logN(loglogN)^2})}$. The multiplier 0.018 is selected to give a figure of 10,000 MIPS-years to factor a 512-bit number. The number of elliptic curve operations (point addition or halving) to solve a discrete log problem (and discover a secret signing key, or forge a signature on a given message) is roughly the square root of the group size. Our group size is about 2^{177} so the breaking work is about $2^{88.5}$ curve operations. Assuming $1,000$ computer instructions per elliptic curve operation, this number of instructions would factor a 1570-bit number.

can be applied to other elliptic curve algorithms over $GF(2^m)$. The focus of this paper is on the implementation of highly-optimized versions of these core operations. Section 4 presents the VHDL implementation of the digital signature algorithm and elliptic curve arithmetic operations. In Section 5, we provide results of the number of gates and time required to generate a digital signature and perform many of the underlying primitive functions that might be used in an elliptic curve coprocessor.

2 The 'Optimal El Gamal' Authentication Algorithm

2.1 Optimal El Gamal Scheme

The signature algorithm is the Optimal El Gamal digital signature scheme adapted for use with elliptic curves (see [5], [6] for original description and security proofs). For an introduction to elliptic curves see [7]. This variant of the El Gamal algorithm was chosen because it avoids the computationally expensive modular reciprocal during signature generation and verification.

Parameters. The public parameters for the Optimal El Gamal Signature scheme are (E, G, W, r) where E is a choice of an elliptic curve, $G \in E$ is a point of large order, r, and $W = sG$ is the public key where s $(1 < s < r)$ is the long-term private key. We assume that the public key parameters and a common hash function are available to all relevant algorithms.

We denote by x_P (resp. y_P) the $x-$coordinate of a point $P \in E$ (resp. $y-$coordinate).

Alg. 1 Elliptic Curve Optimal El Gamal Signature Generation

Input: *Private Key, s; Message, M* **Output:** *Signature (c,d) of M*

1. *Generate a key pair* $(v, V = vG)$*, where* $v \neq 0$ *is a randomly chosen integer modulo r*
2. *Compute* $c \equiv x_V \pmod{r}$*; If* $c = 0$*, then go to Step 1*
3. *Let* $f = Hash(M)$*. Compute an integer* $d = (cfs + v) \pmod{r}$*; If* $d = 0$*, then go to Step 1*
4. *Output the pair* (c, d) *as the signature*

Although our current chip design does not include signature verification, we describe the algorithm for completeness. Most of the optimizations presented later in the paper will benefit signature verification as well as generation.

Alg. 2 Signature Verification
Input: *Signature (c,d) on Message M* **Output:** *Accept/Reject*

1. *If* $c \notin [1, ..., r-1]$, *or* $d \notin [1, ..., r-1]$, *output "Reject" and stop*
2. *Compute* $f = Hash(M)$
3. *Compute the integer* $h = cf \pmod r$
4. *Compute an elliptic curve point* $P = dG - hW$; *If* $P = \mathcal{O}$, *output "Reject" and stop*
5. *Compute* $c' \equiv x_P \pmod r$
6. *If* $c' = c$, *then output "Accept"*

We note that anyone can forge a signature on a message that hashes to 0. However, inverting the hash to find such a message is thought to be computationally infeasible.

3 Algorithmic Optimizations

The field of definition for the elliptic curve is important since it is the basis for all elliptic curve operations. Generally the curve is defined over either $GF(p)$ for some large prime p, or $GF(2^m)$. Since the arithmetic in the latter field is much faster, that was our choice. In particular, we use the field $GF(2^{178})$. One reason for choosing this field is to make use of optimizations that can be derived from the fact that it can be realized as a field tower: $GF(2^{178}) = GF((2^{89})^2)$. In the case of characteristic two fields, the equation for the elliptic curve can be given by $E : y^2 + xy = x^3 + ax^2 + b$. For simplicity and saving on storage, we assume that $a = (1, 0)$. This is useful since the point addition algorithms use a but not b so we don't need to store b (it is implied by the coordinates of the generating point). Further, we exploit properties of $GF(2^{89})$ to reduce some of the basic arithmetic operations (e.g. squaring, square root) to simple XOR gates which are very fast in hardware. The 'almost inverse' algorithm in [8] is especially fast for smaller degree fields. Finally, we modify our elliptic curve multiplication algorithm to use point halving [9,10] which offers a savings over the usual point doubling.

3.1 Finite Field Arithmetic and Field Towers

Our first optimization involves field towers, which simplifies all underlying operations. The finite field is

$$GF(2^m) \cong GF(2)[x]/f(x) = \{a_0 + a_1 x + \cdots + a_{m-1}x^{m-1} \pmod{f(x)} \mid a_i \in GF(2)\}$$

where $f(x)$ is an irreducible binary polynomial of degree m. An element $a \in GF(2^m)$ can therefore be represented as an $m-$tuple $a = (a_0, a_1, ..., a_{m-1})$ of zeros and ones. Addition of two elements is a bitwise exclusive-OR (XOR) operation:

$$a, b \in GF(2^m),\ a + b = (a_0 \oplus b_0, a_1 \oplus b_1, ..., a_{m-1} \oplus b_{m-1})$$

and multiplication is like a plain multiplication without any carries but with the XOR accumulation only. The result of the multiplication must, however, be reduced by the field polynomial $f(x)$. As the degree m of the field gets large, the multiplication can become time-consuming and the representation of the numbers can become cumbersome. For a general reference on finite field arithmetic, see [11].

If m is a composite number, we can use field towers to speed-up the computations. Suppose $m = ns$. Then we can think of $GF(2^m) = GF((2^n)^s)$ as a degree s extension of $GF(2^n)$. The elements are $a \in GF(2^m), a = (\alpha_0, \alpha_1, ..., \alpha_{s-1})$, where $\alpha_i \in GF(2^n)$.

For this work, we use the finite field $GF(2^{178})$ and the corresponding field tower $GF((2^{89})^2)$. Our choice of irreducible polynomial for $GF(2^{89})$ over $GF(2)$ is $f(u) = u^{89} + u^{38} + 1$, and the irreducible polynomial we use for $GF((2^{89})^2)$ over $GF(2^{89})$ is $g(V) = V^2 + V + 1$. We note that this field is not susceptible to known attacks on elliptic curves over composite degree fields (see [12]). Using a trinomial for the field polynomial over $GF(2^{89})$ makes the modular reduction easy and also helps with squaring, square root, Qsolve, and the finishing step in the almost inverse algorithm.

3.2 Finite Field Algorithms

As elements of $GF(2^{178})$ are represented as pairs of $GF(2^{89})$ elements, all algorithms can be described using the arithmetic over $GF(2^{89})$. While some of our optimizations are for general fields, some are specific to our chosen field. We first describe any optimizations over $GF(2^{89})$ and then give algorithms for the extension to $GF(2^{178})$.

Algorithms over $GF(2^{89})$. Unlike the situation with real numbers, squaring and square-rooting are one-to-one operations in characteristic 2 finite fields. Every field element has a single unique square root and square. The following algorithms are specific to $GF(2^{89})$ with field polynomial $f(u) = u^{89} + u^{38} + 1$. In the case of squaring, square root, and solving the equation $a = z^2 + z$ for z (which we call "QSolve"), we note that the algorithmic descriptions can be reduced to simple XORs of the input bits.

Alg. 3 *Squaring*
Input: $a = (a_{00}, ..., a_{88}) \in GF(2^{89})$
Output: $z = (z_{00}, ..., z_{88}) \in GF(2^{89})$ *where* $z = a^2$

z even bits:

$$z_{00} - z_{36} : z_{2n} = a_n \oplus a_{n+70}$$
$$z_{38} - z_{74} : z_{2n} = a_n \oplus a_{n+51}$$
$$z_{76} - z_{88} : z_{2n} = a_n$$

z odd bits:

$$z_{01} - z_{37} : z_{2n+1} = a_{n+45}$$
$$z_{39} - z_{87} : z_{2n+1} = a_{n+45} \oplus a_{n+26}$$

Alg. 4 *Square Root*
Input: $a = (a_{00}, ..., a_{88}) \in GF(2^{89})$
Output: $z = (z_{00}, ..., z_{88}) \in GF(2^{89})$ *where* $z = \sqrt{a}$

$$
\begin{aligned}
&z_{00} - z_{12} : z_n = a_{2n} \oplus a_{2n+51} \oplus a_{2n+13} \\
&z_{13} - z_{18} : z_n = a_{2n} \oplus a_{2n+51} \oplus a_{2n+13} \oplus a_{2n-25} \\
&z_{19} - z_{31} : z_n = a_{2n} \oplus a_{2n+13} \oplus a_{2n-25} \\
&z_{32} - z_{37} : z_n = a_{2n} \oplus a_{2n-63} \oplus a_{2n+13} \oplus a_{2n-25} \\
&z_{38} - z_{44} : z_n = a_{2n} \\
&z_{45} - z_{63} : z_n = a_{2n-89} \\
&z_{64} - z_{82} : z_n = a_{2n-89} \oplus a_{2n-127} \\
&z_{83} - z_{88} : z_n = a_{2n-89} \oplus a_{2n-127} \oplus a_{2n-165}
\end{aligned}
$$

Quadratic Solve. We developed a special circuit for computing QSolve with a relatively small number of XOR gates (387) and depth (35). The full circuit and detailed derivation are in [13].

Alg. 5 *Qsolve*
Input: $a = (a_{00}, ..., a_{88}) \in GF(2^{89})$
Output: $z = (z_{00}, ..., z_{88}) \in GF(2^{89})$ *where* $a = z^2 + z$

Except for odd z in the range $z_{01} - z_{19}$ *(which are computed directly), the bits of z are computed from the following equations:*

a even bits:

$$
\begin{aligned}
&a_{00} - a_{36} : a_{2n} = z_{2n} \oplus z_n \oplus z_{n+70} \\
&a_{38} - a_{74} : a_{2n} = z_{2n} \oplus z_n \oplus z_{n+51} \\
&a_{76} - a_{88} : a_{2n} = z_{2n} \oplus z_n
\end{aligned}
$$

a odd bits:

$$
\begin{aligned}
&a_{01} - a_{37} : a_{2n+1} = z_{2n+1} \oplus z_{n+45} \\
&a_{39} - a_{87} : a_{2n+1} = z_{2n+1} \oplus z_{n+45} \oplus z_{n+26}
\end{aligned}
$$

This derivation uses several observations to reduce the number of gates.

1. QSolve is linear, so we could precompute QSolve(u^N) for each N. The runtime circuit XORs together the appropriate subset for a general polynomial (see [14] for one method of doing the precomputation). This is fast, but uses a lot of gates. We traded speed for size, getting a slower but smaller circuit.
2. We reduced the number of required QSolve(u^N) values by removing some powers of u from the problem. For example, the substitution QSolve(u^{2N}) $\Rightarrow u^N + QSolve(u^N)$ eliminates even powers of u. The substitution $u^N \Rightarrow u^{N-38} + u^{N+51}$ removes some odd powers of u. After repeated substitutions like these, QSolve(u^N) is only needed for odd N in the range 1...19.
3. Only some of the answer bits are required: z_{odd} in the range $z_{01}...z_{19}$. This reduces the number of gates considerably. The remaining bits can be recovered by solving the bit equations for QSolve. For example, we compute z_{45} from the equation $a_{01} = z_{01} \oplus z_{45}$.

4. We assume that a_{00} is equal to a_{51}. The actual value of a_{00} is ignored. Furthermore, z_{00} is irrelevant, and is set equal to 0.

Our minimal size QSolve circuit used only 287 XOR gates, but had depth 65. We moved back from this extreme point on the speed-size tradeoff curve to a circuit with 387 XOR gates and depth 35.

Division. Inversion over $GF(2^{89})$ is performed with an "almost inverse" algorithm [8]. Division is a reciprocal followed by a multiply.

Algorithms over $GF(2^{178})$. We consider $GF(2^{178})$ as a degree two extension of $GF(2^{89})$ with field polynomial $V^2 + V + 1$. Elements of $\omega \in GF(2^{178})$ are pairs of elements from $GF(2^{89})$. So $\omega = (u_1, v_1)$ where $u_1, v_1 \in GF(2^{89})$; i.e. $\omega = u_1 V + v_1$ where $V^2 = V + 1$. The algorithms from $GF(2^{89})$ are extended to $GF(2^{178})$ in the obvious way. We give here some examples where some optimizations have been made.

Alg. 6 *Multiplication* $GF(2^{178})$
Input: $x = (u_1, v_1)$, $y = (u_2, v_2)$; **Output:** $z = x * y = (u_3, v_3)$

1. $u_3 = (u_1 + v_1)(u_2 + v_2) + v_1 v_2$
2. $v_3 = u_1 u_2 + v_1 v_2$

Alg. 7 *Inversion* $GF(2^{178})$
Input: $x = (u_1, v_1)$; **Output:** $x^{-1} = (u_2, v_2)$

1. $u_2 = \frac{u_1}{(u_1+v_1)^2+u_1 v_1}$
2. $v_2 = \frac{u_1+v_1}{(u_1+v_1)^2+u_1 v_1}$

Alg. 8 *Squaring* $GF(2^{178})$
Input: $x = (u_1, v_1)$; **Output:** $x^2 = (u_2, v_2)$

1. $u_2 = u_1^2$
2. $v_2 = u_1^2 + v_1^2$

Alg. 9 *Square Root* $GF(2^{178})$
Input: $x = (u_1, v_1)$; **Output:** $\sqrt{x} = (u_2, v_2)$

1. $u_2 = \sqrt{u_1}$
2. $v_2 = \sqrt{u_1} + \sqrt{v_1}$

Alg. 10 *Qsolve* $GF(2^{178})$
Input: $a = (u_1, v_1)$; **Output:** $z = (u_2, v_2)$ *such that* $a = z^2 + z$

1. $u_2 = Qsolve(u_1)$ *(per Alg. 5)*
2. *Set* $t = u_1 + v_1 + u_2 = t_0 t_1 ... t_{88}$
3. *If* $t_0 \oplus t_{51} = 1$, *then* $u_2 = u_2 + 1$ *and* $t = t + 1$
4. $v_2 = Qsolve(t)$

In both $GF(2^{89})$ and $GF(2^{178})$, only half of the field elements, a, have a corresponding solution, z. Moreover, when z is a solution, so is $z + 1$. In step 3 of Algorithm 10, we choose the Qsolve solution u_2 so that t can be Qsolved in step 4.

3.3 Point Halving Algorithm

The slowest part of the signature algorithm is the multiplication of points. We modified the point multiplication algorithm to use a point halving algorithm in place of a doubling algorithm. The idea of "halving" a point $P = (x_P, y_P)$ is to find a point $Q = (x_Q, y_Q)$ such that $2Q = P$. Note this is the inverse of the point doubling problem. The point halving can nevertheless be used in our algorithm by a simple adjustment on the base point of the elliptic curve used. The algorithm offers a speed-up in software of a factor of about two to three over the point doubling algorithm. We follow the algorithm of [9].

For this algorithm we sometimes write the coordinates of the points $P \in E$ as (x_P, r_P) where $r_P = x_P / y_P$. In fact, we use the (x_P, r_P) form whenever possible, but the input and output of the point addition algorithm need the Y coordinate, so the halving algorithm must handle Y outputs and inputs. When the y_Q output is not required, the point halving algorithm needs only one field multiplication. It is most efficient when point halvings are consecutive. Our signed sliding window multiplication method uses about five halvings between additions.

Alg. 11 *Point Halving over* $GF(2^m)$
Input: $P \in E$ **Output:** $Q = \frac{1}{2}P \in E$

1. $M_h = Qsolve(x_P + a)$, *where* a *is the curve parameter*
2. $T = x_P * (M_h + r_P)$ *or* $T = x_P * M_h + y_P$
3. *If parity(*T *and* t_m*)*$= 0$, *then* $M_h = M_h + 1$; $T = T + x_P$
   Here t_m *is a mask that depends upon the modulus polynomial. In our case,* $t_m = (u^{51} + 1, 0)$.
4. $x_Q = \sqrt{T}$
5. $r_Q = M_h + x_Q + 1$
6. *If needed,* $y_Q = x_Q * r_Q$

3.4 Sliding Window Multiplication with Precomputation

The computation of elliptic curve multiplication, nP, is performed using a 4-bit signed sliding window algorithm [8]. The table of precomputed values $\{1, 3, 5, 7\}G$ are stored and the circuit automatically computes the negatives $\{-1, -3, -5, -7\}G$ as needed on the fly. On average there are 5 halvings per addition.

4 Hardware Architecture and Design

The hardware design is a full VHDL implementation of the Elliptic Curve Optimal El Gamal Signature algorithm that can be targeted to a Field Programmable Gate Array (FPGA) or an Application Specific Integrated Circuit (ASIC). The implementation is a VHDL Register-Transfer-Level design. The goal is to maximize speed and functionality while conserving area and therefore power.

The overall strategy was first to develop a set of basic $GF(2^m)$ arithmetic blocks (in VHDL) that would be used throughout the design. Basic building blocks include addition, multiplication, reciprocal, squaring, etc. The design was then built with these blocks to create the full algorithm implementation for point addition, point halving, point multiplication, and signature generation. The VHDL implementation was created using a bottom-up approach. This allowed a great deal of flexibility throughout the development. As algorithms were improved and/or optimized, the design was easily adaptable.

4.1 Hardware Implementation

The VHDL implementation consists of mapping algorithms discussed in the previous sections to hardware functions and optimizing area and speed, where possible, while allowing user flexibility.

The hardware was organized into four functional design blocks (Fig. 1). The control block contains all the I/O interface circuitry and controls the flow of the digital signature algorithm. The remainder is used for modular reduction in the signature as well as in the pseudo-random number generation process. The SHA-1 hash function serves a dual purpose in hashing the input message and creating the pseudo-random number required for signature generation. The signature algorithm block controls and performs the actual signing of the message.

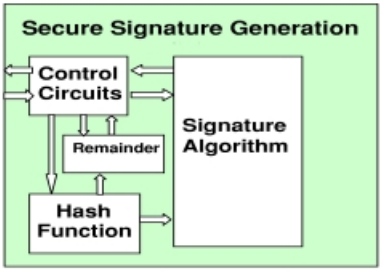


Fig. 1. Top Level Architecture.

4.2 Command, Configuration, and Control

The command, configuration, and control circuitry is responsible for all the high-level control and configuration of the device. It controls the external interface

to the chip, message input and signature output, random number generation control, power management, and algorithm flow control.

The external I/O interface to this chip is intended to hang off of a microprocessor bus. There is a 16-bit address bus, an 8-bit data bus, and control signals. The device is intended to be used as a memory-mapped device in which communication to the device is via a read and write interface similar to that of random access memory (RAM). In addition, there are interrupt signals that are used to indicate to the host system signature status, error status, and signature completion (Fig. 2).

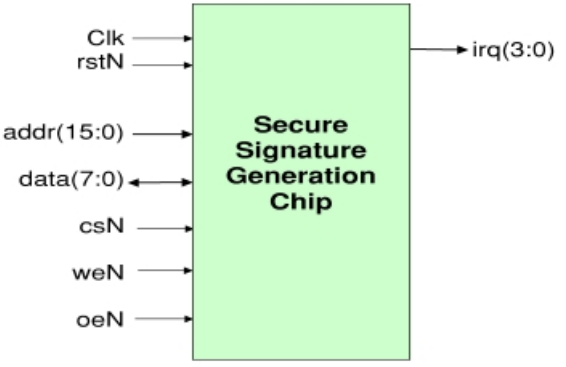


Fig. 2. Secure Signature Generation Chip Interface

The architecture gives the end user a great deal of flexibility. The device can be used in conjunction with any microprocessor that contains a memory-mapped interface. Within the chip, there is a memory map for a full suite of initialization, configuration, result, and status registers. In this respect, the aspects of the signature algorithm are programmable. The following parameters can be programmed (i.e. written) into the Secure Signature Generation Chip.

- Message (up to 512 bits at a time) or Message Digest (based on configuration)
- Generating Point on the Elliptic Curve, $G = (x_G, r_G)$
- Order, r, of the Point G
- Private Key, s
- Random Number (178 bits) or Random Seed (320 bits) (based on configuration), used to generate the per-message nonce
- Configuration Variables (message format, random number format, sleep mode, 1st Message, etc.)

In addition, the signature algorithm generates a set of variables that can be polled (i.e. read).

- Public Key Output
- Output Signature (c, d)

The power management circuitry (in the control section) is essentially a clock-gating circuit that controls when a certain functional area is receiving a clock. The power management is used on a function-by-function basis. That is, the clock-gating follows the circuit function, and, when a circuit is not calculating a value (i.e. idling), the clock to that respective circuit is disabled. This logic is used to reduce overall power consumption by controlling the switching capacitance of respective functional blocks.

4.3 Random Number Generation

There are two methods for generating the per-message nonce needed for the El Gamal signature generation. The first is to simply input the random number via the I/O memory mapped interface. This allows the user to use a true-random number if so desired, but has the obvious overhead of needing to input that random number for each message to be signed.

The second approach is to use the on-chip pseudo-random number generator. This method follows the updated pseudo-random number generation algorithm of the Digital Signature Standard [15]. This circuit uses the remainder circuit and the hash function to create the pseudo-random number. The methodology is to use two 160-bit seeds to create two independent 160-bit hash values. These values are fed back into the random seed registers for further creation of pseudo-random numbers. They are then concatenated together to produce a 320-bit value, from which the remainder (mod r) is extracted. This value is then used as a 178-bit random number.

4.4 Message Input

There are two methods for message input. The user can configure the device to accept a 160-bit message digest. This allows the user to generate the hash of the message and input the message digest via the memory-mapped interface. The hashing overhead would be under user control.

Alternatively, the user can have the on-chip circuitry hash a raw input message using SHA-1. The SHA-1 VHDL was implemented per FIPS Standard [16] and computes a 160-bit message digest from the incoming message.

4.5 Signature Algorithm

The VHDL implementation of the Elliptic Curve Optimal El Gamal Signature Algorithm is a direct implementation of the algorithm described in Section 2. As with the full-chip implementation, the control circuit is responsible for operation of the algorithm and data flow between the various blocks. The multiply and remainder functions exist to compute the products and modular reductions needed in the signature. They are both simple ripple/shift implementations of the mathematical operations. The block diagram is shown in Fig. 3.

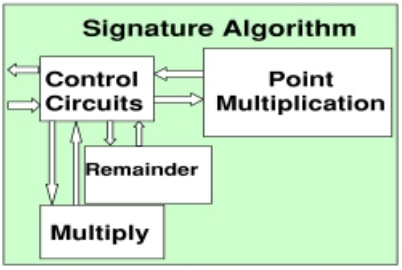


Fig. 3. Signature Algorithm Architecture

4.6 Hardware Optimizations

There were several design optimizations that were used to improve area and performance. Some of the more prominent and significant improvements are discussed below.

For multiplication in a finite field (Section 3.1), which operates with a simple shift and add, the radix of the multiply was increased to 16. This allowed us to perform the multiply in 4-bit fragments, which provided a dramatic speed increase with a slight area penalty. This operation was a bottleneck in the design, thus this improvement provided a speed-up of about a factor of two.

In the Almost Inverse Algorithm ([8], p.50), there were three optimizations that were implemented. The 1st is a parallel degree comparator circuit, which was optimized for both area and speed. The 2nd optimizes the search for a 1 in the LSB of a variable by using a "look ahead" technique with 4-bit blocks before defaulting to operating on the data 1 bit at a time. This increased speed with a very small area penalty. In a similar manner, the 3rd optimization is applied to the last step in the algorithm, which divides and shifts the result a variable number of times. This too performs a "look ahead" using 8-bit data blocks before defaulting to the single-bit implementation.

The Qsolve Algorithm 5 was parallelized and the depth of the XOR tree has been reduced to increase the speed as described in Section 3.2.

The implementation of the SHA-1 algorithm has been optimized to use a shift register for the main data storage, which reduced the area used, with a corresponding increase in speed.

5 Hardware Design Results

This design has not been realized in silicon. However, the design has been synthesized to a target CMOS $0.5\mu m$, $5V$ library. It has also undergone static timing analysis, timing simulations, and power analysis. The following is a summary of results for this target library.

Signature Generation Time Using a 20 Mhz System Clock:

- Initialization: 0.25 ms
  (Necessary any time the Generating Point is initialized and/or changed)
- Signature Generation: 4.4 ms

Synthesized Gate Count Approximations (target library $0.5\mu m$, $5V$, $25°C$) of major sub-blocks, where a gate is equivalent to a standard library NAND Cell.

- Chip: 191,000 Gates
  - Control: 27,000 Gates
  - SHA-1: 13,000 Gates
  - Remainder: 6,700 Gates
  - Signature Algorithm: 143,000 Gates
    - Control: 15,000 Gates
    - Multiply: 6,200 Gates
    - Remainder: 6,800 Gates
    - Point Multiplication: 112,000 Gates
      - Register & Control: 30,000 Gates
      - Point Addition: 52,000 Gates
      - Point Halving: 29,000 Gates

Critical Timing Path (Setup Timing) (target library $0.5\mu m$, $5V$, $25°C$):

- Critical Setup Timing Path (register to register): 48 ns

The critical timing path is located at the Signature Algorithm Level in the computation of $cfs + u \pmod r$. Specifically, it is located in the subtract circuit within the Remainder that computes the modulo r value for the signature. Optimizations are still being performed to improve timing critical paths that affect the overall performance of the device.

Power Analysis and Estimation Using 20 Mhz System Clock (target library $0.5\mu m$, $5V$, $25°C$):

- Dynamic Power Consumption Estimation: 150 mW
- Static (Idle) Power Consumption Estimation: 6 μW

The above numbers were generated using the Synopsys Power Compiler (power analysis tool) which uses gate switching data (based on typical simulation results) to generate power estimates. These estimates are library dependent and are based on the accuracy of the library models provided.

Please note that these design results (performance/speed, gate count, and power estimation) are only applicable to the target hardware process technology, which is not the most advanced technology available today, but was the

most accessible and complete for this analysis. If one were to target a more advanced technology, the design would certainly improve in performance (speed), area (gate count), and power consumption. Specifically, the Critical Setup Timing Path could significantly improve, thus improving the overall speed of the chip. Using power $P = V^2/R$, where V=operating voltage and R=operating resistance, which is fixed, lowering V from $5V$ to $3.3V(1.8V)$ would result in a 56%(87%) reduction in power consumption. At $1.8V$, the estimated dynamic power consumption is $19mW$.

6 Conclusions

Low-power hardware implementations of public-key cryptography continue to enable its use in resource-constrained environments. Wireless applications alone will further drive this market. In this paper, our VHDL design takes advantage of several optimizations of both finite field and elliptic curve arithmetic for the specific function of digital signature generation. We use hardware techniques to reduce the overall power consumption by switching the clock off to areas that are not currently being used. This reduces the power by reducing the effective switching capacitance of the clock. Our design has been successful in achieving performance attributes that are attractive to low-power applications requiring strong public-key authentication. Opportunities to further develop optimized implementations of elliptic curve-based signature algorithms include the following.

1. Further utilization of extension fields.
2. Additional improvements to point multiplication.
3. Improvement of the worst case setup timing path.

Finally, since our main focus was minimizing power consumption, we note that we have ignored the problem of side channel attacks. Countermeasures against such attacks are important and should be the subject of future work. Under the auspices of technology transfer, anyone interested in employing our current and future developments in their application is encouraged to contact the authors.

Acknowledgements. The authors would like to thank Mark Torgerson for many useful discussions and comments.

References

1. Kim, C., Oh S., and Lim, Jongin, "A new hardware architecture for operations in GF(2^n)", IEEE Transactions on Computers v 51 n 1 January 2002. p. 90–92.
2. S. Moon, J. Park, and Y. Lee, "Fast VLSI arithmetic algorithms for high-security elliptic curve cryptographic applications", Proceedings of ICCE. International Conference on Consumer Electronics, 19-21 June 2001, Los Angeles, CA.

3. J. Goodman and A. Chandrakasan, "An energy-efficient reconfigurable public-key cryptography processor", IEEE Journal of Solid-State Circuits, vol.36, no.11, p. 1808–20. Feb. 2001, San Francisco, CA.
4. M. Aydos, T. Yanik, and C. Koc, "High-speed implementation of an ECC-based wireless authentication protocol on an ARM microprocessor", in IEEE Proceedings-Communications, vol.148, no.5, p. 273–9, Oct. 2001.
5. Nyberg, K. and R. A. Rueppel, "Message recovery for signature schemes based on the discrete logarithm problem", Advances in Cryptography – Eurocrypt '94, Springer LNCS 950, 1994, p. 182–193.
6. L. Harn and Y. Xu, "Design of Generalised El Gamal Type Digital Signature Schemes Based on Discrete Logarithm", in Electronics Letters Online, No.19941398, September 30, 1994. IEEE.
7. J. Silverman, The Arithmetic of Elliptic Curves. Springer-Verlag, 1986.
8. R. Schroeppel, H. Orman, S. O'Malley, and O. Spatscheck, "Fast Key Exchange with Elliptic Curve Systems", in Advances in Cryptology – Crypto '95, Springer LNCS 963, 1995, p. 43–56.
9. R. Schroeppel, "Elliptic Curves – Twice as Fast", Midwest Algebraic Geometry Conference, Urbana, IL, November 2000.
10. E. Knudsen, "Elliptic Scalar Multiplication Using Point Halving", Advances in Cryptology – Asiacrypt '99, Springer LNCS 1716, 1999, p. 135–149.
11. IEEE P1363, Standard Specifications for Public Key Cryptography. Appendix A, 1997.
12. N. Smart, "How Secure Are Elliptic Curves over Composite Extension Fields?", in Eurocrypt 2001, LNCS 2045, May 2001, p. 30–39.
13. Schroeppel, R. "Circuits for Solving a Quadratic Equation in $GF[2^N]$", in preparation, 2002.
14. M. Rosing, Implementing Elliptic Curve Cryptography, Manning Publications, 1999.
15. FIPSPUB 186-2 + change notice 1.
16. FIPSPUB 180-1.

A Reconfigurable System on Chip Implementation for Elliptic Curve Cryptography over $\mathbb{GF}(2^n)$

M. Ernst[1], M. Jung[1], F. Madlener[1], S. Huss[1], and R. Blümel[2]

[1] Integrated Circuits and Systems Lab., Computer Science Department, Darmstadt University of Technology, Germany
{ernst|mjung|madlener|huss}@iss.tu-darmstadt.de
[2] cv cryptovision GmbH, Gelsenkirchen, Germany
rainer.bluemel@cryptovision.com

Abstract. The performance of elliptic curve based public key cryptosystems is mainly appointed by the efficiency of the underlying finite field arithmetic. This work describes two generic and scalable architectures of finite field coprocessors, which are implemented within the latest family of Field Programmable System Level Integrated Circuits FPSLIC from Atmel, Inc. The HW architectures are adapted from Karatsuba's divide and conquer algorithm and allow for a reasonable speedup of the top-level elliptic curve algorithms. The VHDL hardware models are automatically generated based on an eligible operand size, which permits the optimal utilization of a particular FPSLIC device.

Keywords. Elliptic Curve cryptography, $\mathbb{GF}(2^n)$ arithmetic, Karatsuba multiplication, VHDL model generator, coprocessor synthesis, FPGA hardware acceleration, Atmel FPSLIC platform.

1 Introduction

Today there is a wide range of distributed systems, which use communication resources that can not be safeguarded against eavesdropping or unauthorized data alteration. Thus cryptographic protocols are applied to these systems in order to prevent information extraction or to detect data manipulation by unauthorized parties. Besides the widely-used RSA method [1], public-key schemes based on elliptic curves (EC) have gained more and more importance in this context. In 1985 elliptic curve cryptography (ECC) has been first proposed by V. Miller [2] and N. Koblitz [3]. In the following a lot of research has been done and nowadays ECC is widely known and accepted. Because EC methods in general are believed to give a higher security per key bit in comparison to RSA, one can work with shorter keys in order to achieve the same level of security [4]. The smaller key size permits more cost-efficient implementations, which is of special interest for low-cost and high-volume systems. Because ECC scales well over the whole security spectrum, especially low-security applications can benefit from ECC.

B.S. Kaliski Jr. et al. (Eds.): CHES 2002, LNCS 2523, pp. 381–399, 2003.

Each application has different demands on the utilized cryptosystem (e.g., in terms of required bandwidth, level of security, incurred cost per node and number of communicating partners). The major market share probably is occupied by the low-bandwidth, low-cost and high-volume applications, most of which are based on SmartCards or similar low complexity systems. Examples are given by the mobile phone SIM cards, electronic payment and access control systems. In case of access control systems, ECC allows to use *one device and one key-pair per person* for the entire application. A very fine granular control is possible and in contrast to present systems, which are mostly based on symmetric ciphers, there is no problem regarding the key handling.

Depending on the application, the performance of genuine SW implementations of ECC is not sufficient. In this paper two generic and scalable architectures of *Finite Field* coprocessors for the acceleration of ECC are presented. The first one, which is mainly composed of a single combinational Karatsuba multiplier (CKM), allows for a significant speed-up of the finite field multiplication while spending only a small amount of HW resources. The second one is a finite field coprocessor (FFCP), implementing field multiplication, addition and squaring completely within HW. The proposed multi-segment Karatsuba multiplication together with a cleverly selected sequence of intermediate result computations permits high-speed ECC even on devices offering only approx. 40K system gates of HW resources. A variety of fast EC cryptosystems can be built by disposing the proposed system partitioning. Running the EC level algorithms in SW allows for algorithmic flexibility while the HW accelerated finite field arithmetic contributes the required performance.

Recently, Atmel, Inc. introduced their new AT94K family of FPSLIC devices (Field Programmable System Level Integrated Circuits). This architecture integrates FPGA resources, an AVR microcontroller core, several peripherals and SRAM within a single chip. Based on HW/SW co-design methodologies, this architecture is perfectly suited for System on Chip (SoC) implementations of ECC.

The mathematical background of elliptic curves and finite fields is briefly described in the following section. In Sec. 3 the architectures of the proposed finite field coprocessors are detailed. Sec. 4 introduces the FPSLIC hardware platform. Finally, we report on our implementation results give some performance numbers and conclusions.

2 Mathematical Background

There are several cryptographic schemes based on elliptic curves, which work on a subgroup of points of an EC over a finite field. Arbitrary finite fields are approved to be suitable for ECC. In this paper we will concentrate on elliptic curves over the finite field $\mathbb{GF}(2^n)$[1] and their arithmetics only. For further information we refer to [5] and [6].

[1] In the context of cryptographic applications n should be prime, in order to be safeguarded against *Weil decent* attacks [7].

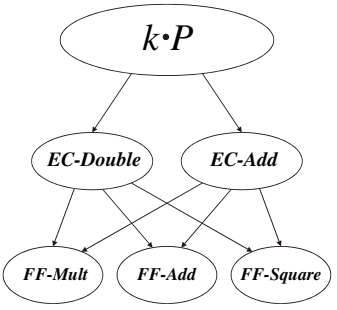


Fig. 1. EC arithmetic hierarchy

2.1 Elliptic Curve Arithmetic

An elliptic curve over $\mathbb{GF}(2^n)$ is defined as the cubic equation

$$E: \quad y^2 + xy = x^3 + ax^2 + b \tag{1}$$

with $a, b, x, y \in \mathbb{GF}(2^n)$ and $b \neq 0$. The set of solutions $\{(x, y) \mid y^2 + xy = x^3 + ax^2 + b\}$ is called the points of the elliptic curve E. By defining an appropriate addition operation and an extra point $\mathcal{O}$, called the point at infinity, these points become an additive, abelian group with $\mathcal{O}$ the neutral element. The EC point multiplication is computed by repeated point additions such as

$$\underbrace{P + P + \ \ldots \ + P + P}_{k \text{ times}} = k \cdot P = R$$

with $k \in \mathbb{N}$ and $P, R \in E$.

The hierarchy of arithmetics for EC point multiplication is depicted in Fig. 1. The top level $k{\cdot}P$ algorithm is performed by repeated *EC-Add* and *EC-Double* operations. The EC operations in turn are composed of basic operations in the underlying finite field (FF). The proposed FF coprocessor (see Sec. 3.2 and Sec. 5.2) is capable to compute the *FF-Add* and *FF-Square* operations within one clock cycle. The operation *FF-Mult* is more costly. The number of clock cycles for its computation depends on the particular architecture of the FF multiplier. Compared to *FF-Add*, *FF-Square* and *FF-Mult* the FF inversion is a very expensive operation and is therefore not implemented in the coprocessor as a basic operation. In order to avoid computing inverses, projective coordinates are used during the computation of the EC operations. By exploiting a projective coordinate representation only one FF inversion is required to perform a complete EC point multiplication. This single FF inversion has to be done at the end of the $k{\cdot}P$ algorithm for the conversion back to affine coordinates.

2.2 Finite Field Arithmetic

As previously mentioned, the EC arithmetic is based on a FF of characteristic 2 and extension degree n: $\mathbb{GF}(2^n)$, which can be viewed as a vector space of dimension n over the field $\mathbb{GF}(2)$. There are several bases known for $\mathbb{GF}(2^n)$. The most common bases, which are also permitted by the leading standards concerning ECC (IEEE 1363 [8] and ANSI X9.62 [9]) are *polynomial bases* and *normal bases*. The representation treated in this paper is a polynomial basis, where field elements are represented by binary polynomials modulo an irreducible binary polynomial (called reduction polynomial) of degree n. Given an irreducible polynomial $P(x) = x^n + \sum_{i=0}^{n-1} p_i x^i$, with $p_i \in \mathbb{GF}(2)$; an element $A \in \mathbb{GF}(2^n)$ is represented by a bit string $(a_{n-1}, \ldots, a_2, a_1, a_0)$, so that

$$A(x) = \sum_{i=0}^{n-1} a_i x^i = a_{n-1}x^{n-1} + \ldots + a_2 x^2 + a_1 x + a_0$$

is a polynomial in x of degree less than n with coefficients $a_i \in \mathbb{GF}(2)$. By exploiting a field of characteristic 2, the addition is reduced to just XOR-ing the corresponding bits. The sum of two elements $A, B \in \mathbb{GF}(2^n)$ is given by

$$C(x) = A(x) \oplus B(x) = \sum_{i=0}^{n-1} (a_i \oplus b_i) x^i \tag{2}$$

and therefore takes a total of n binary XOR operations. The multiplication of two elements $A, B \in \mathbb{GF}(2^n)$ is equivalent to the product of the corresponding polynomials:

$$C(x) = A(x) \cdot B(x) = \sum_{i=0}^{2n-2} c_i x^i \;\text{ denoting }\; c_k = \sum_{i=0}^{k} a_i b_{k-i} \;\text{ for }\; 0 \le k \le 2n-2, \tag{3}$$

with $a_i = 0$ and $b_i = 0$ for $i \ge n$. At the bit level the multiplication in $\mathbb{GF}(2)$ is performed with boolean AND operation. Squaring is a special case of multiplication. For $A \in \mathbb{GF}(2^n)$ the square is given by:

$$A^2(x) = \sum_{i=0}^{n-1} a_i x^{2i}. \tag{4}$$

In the case of multiplication and squaring a polynomial reduction step has to be performed, which is detailed in Sec. 2.

Karatsuba Multiplication. In 1963 A. Karatsuba and Y. Ofman discovered that multiplication of two n bit numbers can be done with a bit complexity of less than $O(n^2)$ using an algorithm now known as Karatsuba multiplication [10]. For multiplication in $\mathbb{GF}(2^n)$ the Karatsuba multiplication scheme can be applied as

well. Therefore, a polynomial $A \in \mathbb{GF}(2^n)$ is subdivided into two segments and expressed as

$$A = A_1 x^{n/2} \oplus A_0 \,.$$

For polynomials $A, B \in \mathbb{GF}(2^n)$ the n-bit multiplication $C = A \cdot B$ is subdivided into $n/2$-bit multiplications as follows:

$$\begin{aligned} C &= A \cdot B \\ &= (A_1 x^{n/2} \oplus A_0) \cdot (B_1 x^{n/2} \oplus B_0) \\ &= A_1 \cdot B_1 x^n \oplus (A_1 \cdot B_0 \oplus A_0 \cdot B_1) x^{n/2} \oplus A_0 \cdot B_0 \,. \end{aligned}$$

By defining some additional polynomials

$$\begin{aligned} T_1 &= A_1 \cdot B_1 \\ T_2 &= (A_1 \oplus A_0) \cdot (B_1 \oplus B_0) = A_1 B_0 \oplus A_0 B_1 \oplus A_1 B_1 \oplus A_0 B_0 \\ T_3 &= A_0 \cdot B_0 \end{aligned}$$

one gets $A \cdot B = T_1 x^n \oplus (T_2 \ominus T_1 \ominus T_3) x^{n/2} \oplus T_3$ and since $\ominus$ and $\oplus$ are equal in $\mathbb{GF}(2^n)$

$$A \cdot B = T_1 x^n \oplus (T_2 \oplus T_1 \oplus T_3) x^{n/2} \oplus T_3 \,. \tag{5}$$

This results in a total of three $n/2$-bit multiplications and some extra additions (XOR operations) to perform one n-bit multiplication.

Multi-segment Karatsuba Multiplication. The fundamental Karatsuba multiplication for polynomials in $\mathbb{GF}(2^n)$ is based on the idea of *divide and conquer*, since the operands are divided into two segments. One may attempt to generalize this idea by subdividing the operands into more than two segments. [11] reports on such an implementation with a fixed number of three segments denoted as *Karatsuba-variant multiplication*. The Multi-Segment Karatsuba (MSK) multiplication scheme, which is detailed subsequently, is more general because an arbitrary number of segments is supported. Disregarding some slight arithmetic variations, the Karatsuba-variant multiplication is a special case of the MSK approach.

Two polynomials in $\mathbb{GF}(2^n)$ are multiplied by a k-segment Karatsuba multiplication (MSK_k)[2] in the following way: It is assumed that $n \mod k = 0$; if not, the polynomials are padded with the necessary number of zero coefficients. A polynomial $A \in \mathbb{GF}(2^n)$ is divided into k segments such that $A = \bigoplus_{i=0}^{k-1} A_i \cdot \hat{x}^i$, with $\hat{x} = x^{n/k}$. With Eqn. 6 holds $C = A \cdot B = MSK_k(A, B)$ for any polynomials $A, B \in \mathbb{GF}(2^n)$:

$$MSK_k(A,B) = \left(\bigoplus_{i=1}^{k} S_{i,0}(A,B) \cdot \hat{x}^{i-1} \right) \oplus \left(\bigoplus_{i=1}^{k-1} S_{k-i,i}(A,B) \cdot \hat{x}^{i-1+k} \right) \tag{6}$$

with

[2] A k-segment Karatsuba multiplication is subsequently termed as MSK_k.

$$S_{m,l}(A,B) = \left(\bigoplus_{i=1}^{m-1} S_{i,l}(A,B)\right) \oplus \left(\bigoplus_{i=1}^{m-1} S_{i,l+m-i}(A,B)\right) \oplus M_{m,l}(A,B)\,, \quad (7)$$

$$S_{1,l}(A,B) = M_{1,l}(A,B) \text{ and } M_{m,l}(A,B) = \left(\bigoplus_{i=l}^{l+m-1} A_i\right) \cdot \left(\bigoplus_{i=l}^{l+m-1} B_i\right)$$

According to Eqn. 6 the entire product $C = A \cdot B = MSK_k(A,B)$ is composed of the partial sums $S_{m,l}(A,B)$. Each partial sum consists of partial products $M_{m,l}(A,B)$ according to Eqn. 7. The total number $N(k)$ of required n/k-bit multiplications in order to perform one n-bit multiplication using the MSK_k scheme is given by

$$N(k) = \sum_{i=1}^{k} i = \frac{(k+1)\cdot k}{2}\,. \quad (8)$$

The application of the above equations for a MSK_3 multiplication, made up of six $n/3$-bit multiplications, is illustrated in the appendix of this paper.

Polynomial Reduction. For $A, B \in \mathbb{GF}(2^n)$, the maximum degree of the resulting polynomial $C(x) = A(x) \cdot B(x)$ is $2n-2$. Therefore, in order to fit into a bit string of size n, $C(x)$ has to be reduced. The polynomial reduction process modulo $P(x)$ is based on the equivalence

$$x^n \equiv \sum_{i=0}^{n-1} p_i x^i \mod P(x). \quad (9)$$

Implementations of the reduction can especially benefit from hard-coded reduction polynomials with low Hamming weight such as trinomials, which are typically used in cryptographic applications. Given such a trinomial as prime polynomial $P(x) = x^n + x^b + 1$ the reduction process can be performed efficiently by using the identities:

$$\begin{aligned} x^n &\equiv x^b + 1 \mod P(x) \\ x^{n+1} &\equiv x^{b+1} + x \mod P(x) \\ &\vdots \\ x^{2n} &\equiv x^{b+n} + x^n \mod P(x) \end{aligned}$$

This leads to

$$\begin{aligned} C(x) &= \sum_{i=0}^{2n-2} c_i x^i \\ &\equiv \sum_{i=0}^{n-1} c_i x^i + \sum_{i=n}^{2n-2} c_i (x^{b+i-n} + x^{i-n}) \mod P(x) \end{aligned}$$

$$
\begin{aligned}
&= \sum_{i=0}^{n-1} c_i x^i + \sum_{i=0}^{n-1-b} c_{i+n} x^{b+i} + \sum_{i=n-b}^{n-1} c_{i+n} x^{b+i} + \sum_{i=0}^{n-1} c_{i+n} x^i \\
&\equiv \sum_{i=0}^{n-1} c_i x^i + \sum_{i=0}^{n-1-b} c_{n+i} x^{b+i} + \sum_{i=n-b}^{n-1} c_{n+i} (x^{2b+i-n} + x^{b+i-n}) + \sum_{i=0}^{n-1} c_{i+n} x^i \mod P(x) \\
&= \underbrace{\sum_{i=0}^{n-1} c_i x^i}_{(1)} + \underbrace{\sum_{i=0}^{n-1-b} c_{i+n} x^{b+i}}_{(2)} + \underbrace{\sum_{i=0}^{b-1} c_{2n-b+i} x^{b+i}}_{(3)} + \underbrace{\sum_{i=0}^{b-1} c_{2n-b+i} x^i}_{(4)} + \underbrace{\sum_{i=0}^{n-1} c_{n+i} x^i}_{(5)} \qquad (10)
\end{aligned}
$$

which results in a total of $2n+b$ binary XOR operations for one polynomial reduction. The particular terms (1...5) of the final equation are structured according to Fig. 2 in order to perform the reduction. With respect to the implementation in Sec. 3 a single n-bit register is sufficient to store the resulting bit string.

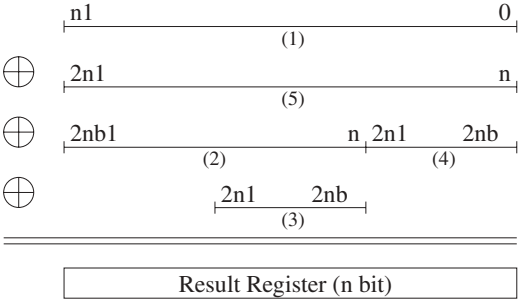


Fig. 2. Structure of the polynomial reduction

3 Hardware Architecture

An ideal HW/SW partitioning targeting a reconfigurable HW platform for an EC based cryptosystem depends on several parameters. As stated before, the FF arithmetic is the most time critical part of an EC cryptosystem. Depending on the utilized key size and the amount of available FPGA resources the FF operations can not inevitably be performed completely within HW. Therefore, flexibility within the HW design flow is essential, in order to achieve the maximum performance from a specific FPGA device. In order to ensure this flexibility, the HW design flow is based on the hardware description language VHDL, which is the *de-facto* standard for abstract modeling of digital circuits. A VHDL generator approach (similar to that one documented in [12]) was exploited to derive VHDL models for both of the subsequently described FF coprocessors. In Sec. 3.1 the combinational Karatsuba multiplier (CKM) is illustrated and Sec. 3.2 details the architecture of the entire finite field coprocessor (FFCP).

3.1 Combinational Karatsuba Multiplier (CKM)

As stated in Sec. 2.2 and shown in Fig. 3a the multiplication over $\mathbb{GF}(2)$ is computed by a single AND operation. According to Eqn. 5 the multiplication of two polynomials of degree m can be computed with three $m/2$-bit multiplications and some XOR operations to determine interim results and to accumulate the final result. This leads immediately to a recursive construction process, which builds CKMs of width $m = 2^i$ for arbitrary $i \in \mathbb{N}$ (see Fig. 3). With slight modifications this scheme can be generalized to support arbitrary bit widths. Exploiting the VHDL generator, CKM models for arbitrary $m \in \mathbb{N}$ can be automatically generated.

To determine the number of gates that constitute an m-bit CKM, we take a look at Fig. 4. In addition to the resources of the three $m/2$-bit multipliers, $2(m/2) = m$ 2-input XOR's are needed to compute the sub-terms $(A_1 \oplus A_0)$ and $(B_1 \oplus B_0)$ of T_2. As can be seen from Fig. 4, $2(m/2-1) = (m-2)$ 4-input XOR's (light gray) and one 3-input XOR (dark gray) are in addition necessary to sum up the product. Thus, we can calculate the number of gates of an m-bit CKM with the following recurrences:

$$XOR_2(m) = \begin{cases} 0 & m = 1 \\ m + 3 \cdot XOR_2(m/2) & m > 1 \end{cases} \quad XOR_3(m) = \begin{cases} 0 & m = 1 \\ 1 + 3 \cdot XOR_3(m/2) & m > 1 \end{cases}$$

$$AND_2(m) = \begin{cases} 1 & m = 1 \\ 3 \cdot AND_2(m/2) & m > 1 \end{cases} \quad XOR_4(m) = \begin{cases} 0 & m = 1 \\ m - 2 + 3 \cdot XOR_4(m/2) & m > 1 \end{cases}$$

With the master method [13] it can easily be shown that all of these recurrences belong to the complexity class $\Theta(m^{\log_2 3})$. Explicit gate counts for CKM of various bit widths are summarized in the Tab. 1.

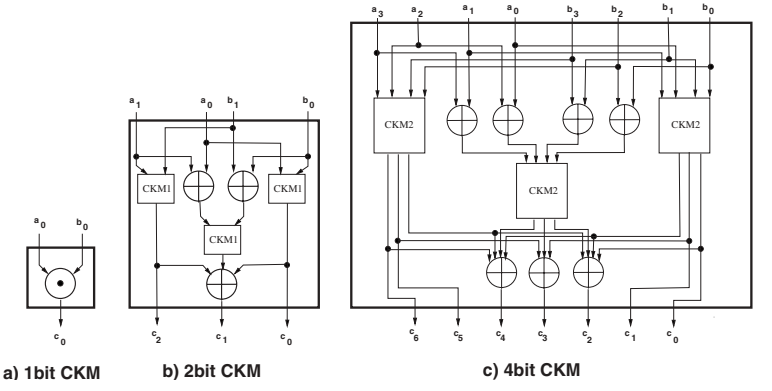


Fig. 3. Recursive construction process for CKM

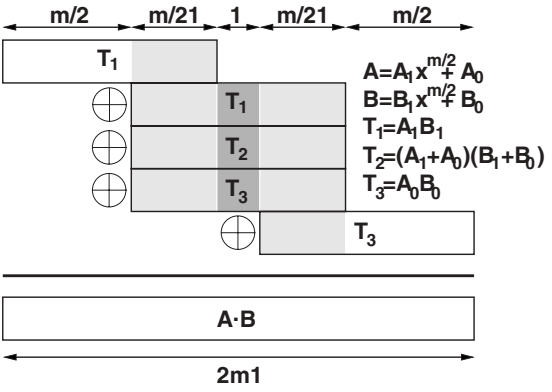


Fig. 4. Karatsuba multiplication

Table 1. CKM gate counts

Bit Width	1	2	4	8	16	32	64
AND_2	1	3	9	27	81	243	729
XOR_2	0	2	10	38	130	422	1330
XOR_3	0	1	4	13	40	121	364
XOR_4	0	0	2	12	50	180	602
SUM	1	6	25	90	301	966	3025

3.2 Finite Field Coprocessor (FFCP)

This section presents a generic and scalable FFCP architecture, which accelerates field multiplication, addition and squaring. Addition and squaring are operations, which require only a few logical resources and hence can be implemented by combinational logic. In contrast, the multiplication can not reasonably be implemented by combinational logic only. By the use of the proposed MSK multiplication scheme (see Sec. 2.2) and a cleverly selected sequence of intermediate result computations, the resulting datapath has only modest requirements on logic resources and at the same time a low cycle count for a complete field multiplication.

The datapath is build around a low complexity m-bit CKM as detailed in Sec. 3.1, but of course any other multiplier design would also do. By application of the sequential MSK_k multiplication algorithm, $k \cdot m$ bit wide operands can be processed. With respect to the implementation in Sec. 5.2 and for reasons of easy illustration we assume $k = 5$ in the following, but the scheme applies and scales in a nice way for arbitrary $k > 1$.

Eqn. 6 evaluated for $k = 5$ (MSK_5) is illustrated in Fig. 5a. Each rectangle denotes the result of an m-bit multiplication. As one would expect, these products are as wide as two segments. The labels in the rectangles determine the indices of the segments, whose sums have been multiplied. E.g., the label "234" represents the term $(A_2 \oplus A_3 \oplus A_4) \cdot (B_2 \oplus B_3 \oplus B_4)$, which is denoted $M_{3,2}(A, B)$

in Eqn. 7. The horizontal position of a rectangle denotes the exponent i of the associated factor $\hat{x}^i$. E.g., the rectangle in the lower left edge labeled "4" together with its position denotes the term $(A_4 \odot B_4) \cdot \hat{x}^8$. The result $A \odot B$ is computed by summing up (XORing) all the terms according to their horizontal position. This product is $2k$ segments wide, as one would expect. The partial products can be reordered as shown in Fig. 5b. This order was achieved in consideration of three optimization criteria.

First, most partial products are added two times in order to compute the final result. They can be grouped together and placed in one of three patterns, which are indicated in Fig. 5b. This is true for all instances of the multi-segment Karatsuba algorithm. In the datapath, these patterns are computed by some additional combinational logic, which is connected to the output signals of the CKM (see part (c) of Fig. 6).

Second, the resulting patterns are ordered by descending i of their factor $\hat{x}^i$. In this way, the product can be accumulated easily in a shift register.

As the third optimization criterion the remaining degree of freedom is taken advantage of in the following way: The patterns are once more partially reordered, such that when iterating over them from top to bottom, one of the two following conditions holds: Either the current pattern is constructed out of a single segment product (e.g. $A_4 \odot B_4$), or the set of indices of the patterns segments differs only by one index from its predecessor (as in the partial products $(A_0 \oplus A_1) \cdot (B_0 \oplus B_1)$ and $(A_0 \oplus A_1 \oplus A_2) \odot (B_0 \oplus B_1 \oplus B_2)$). In Fig. 5b this criterion is met for all but one iteration step (namely it is not met for the step from "23" to "1234"). Thus, based on the datapath in Fig. 6 the computation of the partial product "1234" takes a total of two clock cycles, which is one more compared to all other iteration steps. The number of additional clock cycles due to the fact that this third criterion can not be met increases slowly with the number of segments k.

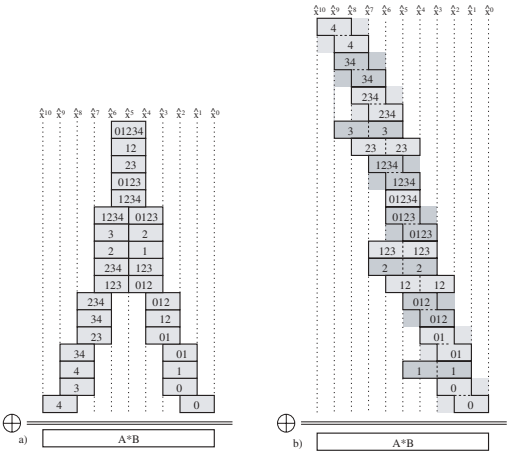


Fig. 5. 5-Segment Karatsuba multiplication and operand reordering

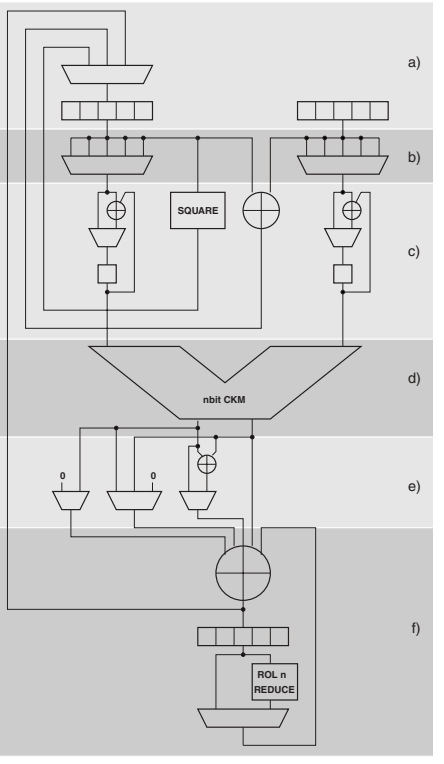


Fig. 6. Generic datapath architecture

By applying the third optimization criterion to the pattern sequence, the partial product computations can be performed as follows: By placing m-bit accumulator registers at the inputs of the CKM, the terms $M_{m,l}(A, B)$ can be computed iteratively. This results in a two stage pipelined design for the complete datapath and yields a total of 17 clock cycles to perform one field multiplication utilizing the MSK_5.

The complete datapath is depicted in Fig. 6. In part a) the two operand registers of width $l = k \cdot m$ are shown as well as their partitioning into five segments. Both are implemented as shift-registers in order to allow data exchange with the external controller. The multiplexors in part b) select one from the five segments of the operands. They can both be controlled by the same set of signals, since they are always operating synchronously. Besides the combinational addition and squaring blocks, part c) illustrates the two accumulator registers. Both can either be loaded with a new segment, or they can accumulate intermediate segment sums. Section d) of Fig. 6 consists of the CKM. Part e) covers the pattern generation stage, which is mainly composed of multiplexors. Finally, in part f) the multiplication accumulator register is shown. It can either hold its value or the current pattern can be added to it in each cycle. Each time the intermediate result is shifted left by m bit, an interleaved reduction step according to Eqn. 10

Table 2. FFCP gate count

Datapath	FF	XOR_2	AND_2[3]	$MUX_{2:1}$	$MUX_{3:1}$	$MUX_{4:1}$
part a)	$2l$				l	
part b)						$2m$
part c)	$2m$	$2m+l+$ $2n+b$		$2m$		
part e)		m	$2m$	$2m$		
part f)	n	$4m$		n		
SUM	$2m+$ $2l+n$	$7m+l+$ $2n+b$	$2m$	$4m+n$	l	$2m$

n, b: according to $P(x)=x^n+x^b+1$
m : CKM bit width
l : input Reg width

is performed. This way, the accumulator needs only to be n bits wide, where n is the degree of the reduction polynomial. Furthermore, the necessary number of logic elements for the reduction step is minimized and no additional clock cycle is needed.

In order to reduce the amount of communication between the controller and the FFCP, the result of the current computation is fed back to one of the operand registers. Thus, interim results need not inevitably be transferred several times between controller and FFCP.

Tab. 2 gives an overview of the amount of structural and logical components, which are required to implement the proposed datapath (excluding the CKM resources, please refer to Sec. 3.1 for the CKM implementation complexity). The number of states of the finite state machine, which controls the datapath, is in the order of Eqn. 8. Thus, logic resources for the FSM are negligibly small.

4 Atmel FPSLIC Hardware Platform

For the implementation of the previously detailed FF coprocessors the AT94K FPSLIC hardware platform from Atmel, Inc. is used within this work [14]. This product family integrates FPGA resources, an AVR 8-bit RISC microcontroller core, several peripherals and up to 36K Bytes SRAM within a single chip. The AVR microcontroller core is a common embedded processor, e.g., on SmartCards and is also available as a stand-alone device. The AVR is capable of 129 instructions, most of which can be performed within a single clock cycle. This results in a 20+ MIPS throughput at 25 MHz clock rate.

The FPGA resources within the FPSLIC devices are based on Atmel's AT40K FPGA architecture. A special feature of this architecture are FreeRam[4] cells which are located at the corners of each 4x4 cell sector. Using these cells results in minimal impact on bus resources and by that in fast and compact FPGA designs. The FPGA part is connected to the AVR over an 8-bit data bus. The amount of available FPGA resources ranges from about 5K system gates within the so-called μFPSLIC to about 40K system gates within the AT94K40.

[3] $MUX_{2:1}$ components with constant zero inputs have been optimized to AND_2 gates.
[4] Each FreeRam cell is organized as a 32x4 bit dual-ported RAM block.

Both, the AVR microcontroller core and the FPGA part are connected to the embedded memory separately. Up to 36K Bytes SRAM are organized as 20K Bytes program memory, 4K Bytes data memory and 12K Bytes that can dynamically be allocated as data or program memory.

Atmel provides a complete design environment for the FPSLIC including tools for software development (C Compiler), tools for hardware development (VHDL synthesis tools) and a HW/SW co-verification tool, which supports the concurrent development of hardware and software.

For the implementations detailed subsequently the Atmel ATSTK94 FPSLIC demonstration board is used. This board comes with a AT94K40 device and is running at 12 MHz clock rate. The FPGA part consists of 2304 logic cells and 144 FreeRam cells, which is equivalent to approx. 40K system gates.

5 Implementation

Three different prototype implementations were built in order to evaluate the architectures detailed in Sec. 3. Due to the restrictions in terms of available FPGA resources these implementations support 113 bit EC point multiplication only. This is certainly not sufficient for high-security applications, but can be applied in low-security environments.

The following sections present some implementation details and performance numbers for a purely software based implementation, a design that is accelerated with a 32-bit CKM and another one, which applies the FFCP. Furthermore an extension to the FFCP design is proposed and performance numbers for this extended version are estimated.

5.1 Pure Software without HW Acceleration

The software variant is entirely coded in assembler and has been optimized regarding the following design criteria:

Table 3. SW performance values

Operation	Bit Width	Clock Cycles Computation	Overhead	Total
FF-Mult	16	96	NA	96
FF-Mult	32	$3 * 96 = 288$	131	419
FF-Mult	64	$3 * 419 = 1.257$	383	1.640
FF-Mult	128	$3 * 1.640 = 4.920$	489	5.409
FF-Square	128	340	NA	340
FF-Add	128	160	NA	160
FF-Reduce	113	420	NA	420
EC-Double	113	15.300	NA	15.300
EC-Add	113	25.200	NA	25.200
$k \cdot P$	113	4.752.000	NA	4.752.000

- High performance.
- Resistance against side channel attacks.
- Easy SW/HW exchange of basic FF operations.

Concerning the performance, special effort has been spent at FF level in optimizing the field multiplication and reduction, which is the performance critical part of the entire $k{\cdot}P$ algorithm. At the EC level the so-called *2P Algorithm* documented in [15] is utilized to perform the EC point multiplication. This algorithm takes only 4 multiplications, 1 squaring and 2 additions in the underlying FF for one EC-Add computation. One EC-Double takes only 2 multiplications, 4 squaring and 1 addition. Summing up, this $k{\cdot}P$ implementation is about 2 times faster compared to standard *Double-and-Add* implementations. Furthermore, the *2P Algorithm* is inherently resistent against pertinent timing resp. power attacks, since in every iteration of its inner loop both operations (EC-Add and EC-Double) have to be computed, regardless of the binary expansion of k. Thus, besides some pre- and postprocessing overhead, one $k{\cdot}P$ computation over $\mathbb{GF}(2^n)$ takes exactly n EC-Add and n EC-Double operations. At the FF level countermeasures against side-channel attacks based on randomization and avoidance of conditional branches are applied as well [16].

Tab. 3 summarizes the performance of the implementation on FF level as well as on EC level. The analysis of the $k{\cdot}P$ algorithm identifies the field multiplication as the most time consuming operation, which amounts to about 85% of the overall cycle count.

5.2 Hardware Acceleration

The subsequently detailed FPGA designs have been implemented by using the design tools which are packaged with the utilized demonstration board. For hardware synthesis this is *Leonardo v2000.1b* from Mentor, Inc. The FPGA mapping is done with *Figaro IDS v7.5* from Atmel, Inc. Also from Atmel, Inc. there is the top-level design environment called *System Designer v2.1*, which is required to build up the entire design based on the AVR and the FPGA part.

Acceleration Based on CKM. The genuine SW implementation can be accelerated by utilizing a CKM as presented in Sec. 3.1, which is implemented in the FPGA part of the AT94K40 device. Matching to the particular bit width m of the raw CKM, two m-bit input registers and a $2m$-bit output register is added on the HW side. In order to allow a reasonable communication over the fixed 8-bit interface, the input registers are designed as 8-bit shift-in and parallel-out registers. Accordingly, the output register is parallel-in and 8-bit shift-out.

Tab. 4 summarizes the performance of the combined HW/SW implementation based on a 32-bit CKM. The 32-bit CKM takes about 53% of the FPGA resources. At the FF level this results in a speed-up of about 3 and for the $k{\cdot}P$ algorithm there is still a speed-up factor of about 2.2 compared to the values given in Tab. 3.

Table 4. 32-bit CKM performance values

Operation	Bit Width	Clock Cycles Computation	Overhead	Total
FF-Mult	32	1	16	17
FF-Mult	64	$3 * 17 = 51$	383	434
FF-Mult	128	$3 * 434 = 1.302$	489	1.791
EC-Double	113	8.100	NA	8.100
EC-Add	113	10.700	NA	10.700
$k \cdot P$	113	2.201.000	NA	2.201.000

Table 5. FFCP performance values

Operation	Bit Width	FFCP Clock Cycles best case	worst case	extended FFCP est. Clock Cycles
FF-Mult	113	32	152	19
FF-Add	113	16	136	3
FF-Square	113	1	91	3
EC-Double	113		493	53
EC-Add	113		615	85
$k \cdot P$	113		130.200	16.380

The CKM architecture is of special interest for HW platforms offering only a small amount of FPGA resources, such as the μFPSLIC (see Sec. 4). This device is still sufficient for the implementation of an 8-bit CKM, which results in 3384 cycles for one 128-bit field multiplication. This is still a speed-up of about 1.6 compared to the genuine SW implementation.

Acceleration Based on FFCP. Utilizing the FFCP architecture detailed in Sec. 3.2 instead of the stand-alone CKM design allows for a further significant performance gain. For the implementation presented here, the particular design parameters are fixed to 113-bit operand width, 24-bit CKM and 5-segment Karatsuba multiplication (MSK_5). This results in a FPGA utilization of 96% for the entire FFCP design.

Due to the fact that the result of each operation is fed back into one of the operand registers, the cycle count of a particular operation (I/O overhead plus actual computation) differs regarding to data dependencies. The corresponding best- and worst-case value for each FF operation is denoted in Tab. 5.

Tab. 5 unveils that the major part of cycles is necessary to transfer 113-bit operands over the fixed 8-bit interface between AVR and FPGA. These transfers can be avoided almost completely with an additional register file on the FFCP and an extended version of the finite state machine, which interprets commands given by the software running on the AVR. Assuming a 2-byte command format (4 bit opcode, 12 bit to specify the destination and the source registers) results

in cycle counts according to the right column of Tab. 5. With respect to the FPSLIC architecture and their special FreeRAM feature, such a register file can be implemented without demand on additional logic cells. The extended version of the FFCP is currently under development on our site.

5.3 Performance Comparison

There are several FPGA based hardware implementations of EC point multiplication documented in the literature [12] [17] [18] [19]. The performance values of these state-of-the-art implementations are given in Tab. 6. Additionally, Tab. 6 comprises the particular figures of the previously described FPSLIC based implementations.

A performance comparison of hardware implementations against each other is in general not straight forward. This is mostly because of different key sizes and due to the fact that different FPGA technologies are used for their implementation.

A basically scalable HW architecture is common to all implementations referenced in Tab. 6. In contrast to our SoC approach, the implementations in [12] [17] [18] and [19] are mainly focusing on high-security, server based applications. Their functionality is entirely implemented within a relatively large FPGA and no arrangements against side-channel attacks are documented.

In [12] and [17] the underlying field representation is an optimal normal basis. Both implementations are based on FPGAs from Xilinx, Inc. Furthermore, VHDL module generators are used in both cases to derive the particular HW descriptions. The approach in [17] allows for a parameterization of the key size only. Parallelization, which is essential in order to achieve maximum performance from a specific FPGA, is additionally supported by the design in [12]. For the implementation in [17] a XCV300 FPGA with a complexity of about 320K system gates is used. The design in [12] is based on a XC4085XLA device with approx. 180K system gates.

The implementations in [18] and [19] are both designed for polynomial bases and the field multiplications are in principle composed of partial multiplications. The design in [18] is based on an Altera Flex10k family device with a complexity of about 310K system gates. The architecture is centered around a w_1-bit$\times w_2$-bit partial multiplier. Due to the flexibility in w_1 and w_2 it is shown, that the architecture scales well, even for smaller FPGA platforms. The best performing implementation, representing the current benchmark with respect to $k{\cdot}P$ performance, is described in [19]. It is highly optimized, exploiting both pipelining and concurrency. The field multiplication is performed with a digit-serial multiplier. A Xilinx XCV400E FPGA with a complexity of about 570K system gates, running at 76.7 MHz is used for the implementation. Compared to our design this signifies a factor of more than 10 in space and a factor of about 6 in speed.

Table 6. Performance comparison

Target Platform	Bit Width	$k \cdot P$
FPGA (XCV300, 45 MHz) [17]	113	3.7 ms
FPGA (XC4085XLA, 37 MHz) [12]	155	1.3 ms
FPGA (EPF10K, 3 MHz) [18]	163	80.7 ms
FPGA (XCV400E, 76.7 MHz) [19]	167	210 μs
FPSLIC pure SW (AT94K40, 12 MHz)	113	396 ms
FPSLIC with 32-bit CKM (AT94K40, 12 MHz)	113	184 ms
FPSLIC with FFCP (AT94K40, 12 MHz)	113	10.9 ms
FPSLIC with ext. FFCP (AT94K40, 12 MHz)	113	1.4 ms (est.)

6 Conclusion

Speeding up the most time critical part of EC crypto schemes enables the use of these methods within combined HW/SW systems with relatively low computing power. Running the EC level algorithms in SW facilitates algorithmic flexibility while the required performance is contributed by dedicated coprocessors.

Two generic and scalable architectures of FF coprocessors (CKM and FFCP) which are qualified for SoC implementations have been illustrated in this paper. While CKM supports only multiplication, the FFCP architecture implements multiplication, addition and squaring completely within HW. The proposed multi-segment Karatsuba multiplication scheme, which is the core of the FFCP architecture, permits fast and resource saving HW implementations. By exploiting the presented coprocessor architectures a considerable speed-up of EC cryptosystems can be achieved.

Acknowledgment. This work was sponsored by and has been done in cooperation with cv cryptovison GmbH, Gelsenkirchen, Germany.

References

1. R. L. Rivest, A. Shamir and L. M. Adleman, "A Method for Obtaining Digital Signatures and Public-Key Cryptosystems," *Communications of the ACM*, Feb 1978.
2. V. Miller, "Use of elliptic curves in cryptography," *Advances in Cryptology, Proc. CRYPTO'85*, LNCS 218, H. C. Williams, Ed., Springer-Verlag, pp. 417–426, 1986.
3. N. Koblitz, "Elliptic Curve Cryptosystems," *Mathematics of Computation*, vol. 48, pp. 203–209, 1987.
4. A. Lenstra and E. Verheul, "Selecting Cryptographic Key Sizes," *Proc. Workshop on Practice and Theory in Public Key Cryptography*, Springer-Verlag, ISBN 3540669671, pp. 446–465, 2000.
5. A. J. Menezes, "Elliptic Curve Public Key Cryptosystems," Kluwer Akademic Publishers, 1993.
6. J. H. Silverman, "The Arithmetic of Elliptic Curves," *Graduate Texts in Mathematics*, Springer-Verlag, 1986.

7. S. Galbraith and N. Smart, "A cryptographic application of Weil descent," *Codes and Cryptography*, LNCS 1746, Springer-Verlag, pp. 191–200, 1999.
8. IEEE 1363, "Standard Specifications For Public Key Cryptography," http://grouper.ieee.org/groups/1363/, 2000.
9. ANSI X9.62, "Public key cryptography for the financial services industry: The Elliptic Curve Digital Signature Algorithm (ECDSA)," (available from the ANSI X9 catalog), 1999.
10. A. Karatsuba and Y. Ofman, "Multiplication of multidigit numbers on automata," *Sov. Phys.-Dokl (Engl. transl.), vol. 7, no. 7*, pp. 595–596, 1963.
11. D. V. Bailey and C. Paar, "Efficient Arithmetic in Finie Field Extensions with Application in Elliptic Curve Cryptography," *Journal of Cryptology*, vol. 14, no. 3, pp. 153–176, 2001.
12. M. Ernst, S. Klupsch, O. Hauck and S. A. Huss, "Rapid Prototyping for Hardware Accelerated Elliptic Curve Public-Key Cryptosystems," *Proc. 12th IEEE Workshop on Rapid System Prototyping (RSP01)*, Monterey, CA, 2001.
13. J. L. Bentley, D. Haken and J. B. Saxe, "A general method for solving divide-and-conquer recurrences," *SIGACT News*, vol. 12(3), pp. 36–44, 1980.
14. Atmel, Inc. "Configurable Logic Data Book," 2001.
15. J. Lopez and R. Dahab, "Fast multiplication on elliptic curves over $GF(2^m)$ without precomputation," *Workshop on Cryptographic Hardware and Embedded Systems (CHES 99)*, LNCS 1717, C.K. Koc and C. Paar Eds., Springer-Verlag, pp. 316–327, 1999.
16. J. Coron, "Resistance against Differential Power Analysis for Elliptic Curve Cryptosystems," *Workshop on Cryptographic Hardware and Embedded Systems (CHES 99)*, LNCS 1717, C.K. Koc and C. Paar Eds., Springer-Verlag, pp. 292–302, 1999.
17. K.H. Leung, K.W. Ma, W.K. Wong and P.H.W. Leong, "FPGA Implementation of a Microcoded Elliptic Curve Cryptographic Processor," *Proc. IEEE FCCM 2000*, pp. 68–76, Napa Valley, 2000.
18. S. Okada, N. Torii, K. Itoh and M. Takenaka, "Implementation of Elliptic Curve Cryptographic Coprocessor over $GF(2^m)$ on an FPGA," *Workshop on Cryptographic Hardware and Embedded Systems (CHES 2000)*, LNCS 1965, C.K. Koc and C. Paar Eds., Springer-Verlag, pp. 25–40, 2000.
19. G. Orlando and C. Paar, "A High-Performance Reconfigurable Elliptic Curve Processor for $GF(2^m)$," *Workshop on Cryptographic Hardware and Embedded Systems (CHES 2000)*, LNCS 1965, C.K. Koc and C. Paar Eds., Springer-Verlag, pp. 41–56, 2000.

Appendix: 3-Segment Karatsuba Multiplication

For any polynomials $A, B \in \mathbb{GF}(2^n)$ the product $C = A \cdot B = MSK_3(A, B)$ using the 3-segment Karatsuba multiplication is according to Eqn. 6 given by:

$$MSK_3(A,B) = \left(\bigoplus_{i=1}^{3} S_{i,0}(A,B) \cdot \hat{x}^{i-1} \right) \oplus \left(\bigoplus_{i=1}^{2} S_{3-i,i}(A,B) \cdot \hat{x}^{i+2} \right)$$

$$
\begin{aligned}
&= S_{1,0}(A,B) && \hat{x}^0 \oplus \\
&\quad S_{2,0}(A,B) && \hat{x}^1 \oplus \\
&\quad S_{3,0}(A,B) && \hat{x}^2 \oplus \\
&\quad S_{2,1}(A,B) && \hat{x}^3 \oplus \\
&\quad S_{1,2}(A,B) && \hat{x}^4
\end{aligned}
$$

$$
\begin{aligned}
&= M_{1,0}(A,B) && \hat{x}^0 \oplus \\
&\quad (S_{1,0}(A,B) \oplus S_{1,1}(A,B) \oplus M_{2,0}) && \hat{x}^1 \oplus \\
&\quad (S_{1,0}(A,B) \oplus S_{2,0}(A,B) \oplus S_{1,2}(A,B) \oplus S_{2,1}(A,B) \oplus M_{3,0}(A,B)) && \hat{x}^2 \oplus \\
&\quad (S_{1,1}(A,B) \oplus S_{1,2}(A,B) \oplus M_{2,1}(A,B)) && \hat{x}^3 \oplus \\
&\quad M_{1,2}(A,B) && \hat{x}^4
\end{aligned}
$$

$$
\begin{aligned}
&= M_{1,0}(A,B) && \hat{x}^0 \oplus \\
&\quad (M_{1,0}(A,B) \oplus M_{1,1}(A,B) \oplus M_{2,0}(A,B)) && \hat{x}^1 \oplus \\
&\quad (M_{1,0}(A,B) \oplus S_{1,0}(A,B) \oplus S_{1,1}(A,B) \oplus M_{2,0}(A,B) \oplus M_{1,2}(A,B) \oplus \\
&\quad S_{1,1}(A,B) \oplus S_{1,2}(A,B) \oplus M_{2,1}(A,B) \oplus M_{3,0}(A,B)) && \hat{x}^2 \oplus \\
&\quad (M_{1,1}(A,B) \oplus M_{1,2}(A,B) \oplus M_{2,1}(A,B)) && \hat{x}^3 \oplus \\
&\quad M_{1,2}(A,B) && \hat{x}^4
\end{aligned}
$$

$$
\begin{aligned}
&= M_{1,0}(A,B) && \hat{x}^0 \oplus \\
&\quad (M_{1,0}(A,B) \oplus M_{1,1}(A,B) \oplus M_{2,0}(A,B)) && \hat{x}^1 \oplus \\
&\quad (M_{1,0}(A,B) \oplus M_{1,0}(A,B) \oplus M_{1,1}(A,B) \oplus M_{2,0}(A,B) \oplus M_{1,2}(A,B) \oplus \\
&\quad M_{1,1}(A,B) \oplus M_{1,2}(A,B) \oplus M_{2,1}(A,B) \oplus M_{3,0}(A,B)) && \hat{x}^2 \oplus \\
&\quad (M_{1,1}(A,B) \oplus M_{1,2}(A,B) \oplus M_{2,1}(A,B)) && \hat{x}^3 \oplus \\
&\quad M_{1,2}(A,B) && \hat{x}^4
\end{aligned}
$$

$$
\begin{aligned}
&= M_{1,0}(A,B) && \hat{x}^0 \oplus \\
&\quad (M_{1,0}(A,B) \oplus M_{1,1}(A,B) \oplus M_{2,0}(A,B)) && \hat{x}^1 \oplus \\
&\quad (M_{2,0}(A,B) \oplus M_{2,1}(A,B) \oplus M_{3,0}(A,B)) && \hat{x}^2 \oplus \\
&\quad (M_{1,1}(A,B) \oplus M_{1,2}(A,B) \oplus M_{2,1}(A,B)) && \hat{x}^3 \oplus \\
&\quad M_{1,2}(A,B) && \hat{x}^4
\end{aligned}
$$

with

$$
\begin{aligned}
M_{1,0}(A,B) &= A_0 \cdot B_0 \\
M_{1,1}(A,B) &= A_1 \cdot B_1 \\
M_{1,2}(A,B) &= A_2 \cdot B_2 \\
M_{2,0}(A,B) &= (A_0 \oplus A_1) \cdot (B_0 \oplus B_1) \\
M_{2,1}(A,B) &= (A_1 \oplus A_2) \cdot (B_1 \oplus B_2) \\
M_{3,0}(A,B) &= (A_0 \oplus A_1 \oplus A_2) \cdot (B_0 \oplus B_1 \oplus B_2)
\end{aligned}
$$

Genus Two Hyperelliptic Curve Coprocessor

N. Boston, T. Clancy, Y. Liow, and J. Webster

Illinois Center for Cryptography and Information Protection*
Coordinated Science Laboratory, University of Illinois, Urbana-Champaign
1308 West Main Street, Urbana, IL 61801
{n-boston, tclancy, liow, jewebste}@uiuc.edu

Abstract. Hyperelliptic curve cryptography with genus larger than one has not been seriously considered for cryptographic purposes because many existing implementations are significantly slower than elliptic curve versions with the same level of security. In this paper, the first ever complete hardware implementation of a hyperelliptic curve coprocessor is described. This coprocessor is designed for genus two curves over $\mathbb{F}_{2^{113}}$. Additionally, a modification to the Extended Euclidean Algorithm is presented for the GCD calculation required by Cantor's algorithm. On average, this new method computes the GCD in one-fourth the time required by the Extended Euclidean Algorithm.

1 Introduction

Hyperelliptic curves (HEC) are a generalization of elliptic curves and the first suggestion of their cryptographic use was made by N. Koblitz at Crypto '88 ([10]). Elliptic curve cryptography (ECC) has received much attention because it offers several benefits over other public-key cryptosystems, such as RSA. With a higher security per bit than RSA, ECC allows for a comparable level of security with a smaller key size. Additionally, many have reported ECC hardware implementations require significantly fewer transistors.

This paper presents concrete performance results from a hardware-based genus two hyperelliptic curve coprocessor over $\mathbb{F}_{2^{113}}$. Additionally, these performance characteristics are compared to a software implementation over the same field and curve.

The hardware version was implemented on a Field Programmable Gate Array (FPGA). FPGAs allow programmers to input a logic structure which will be emulated using the extensive set of gates available on the FPGA. These logic structures are created using Hardware Description Language (HDL). In this implementation, Verilog, a popular HDL, was used to describe the hardware. From there, the Xilinx Integrated Software Environment was used to synthesize and implement the logic design for a Xilinx Virtex II FPGA. Additionally, the Modeltech Microsim simulator was used to verify the correctness of the design.

* Research supported by NSF EIA 00-88063

B.S. Kaliski Jr. et al. (Eds.): CHES 2002, LNCS 2523, pp. 400–414, 2003.

A software version written in C++ was compiled using Microsoft's Visual Studio and tested on a Pentium system. Further results on the software implementation will be available in a forthcoming publication.

While no other complete hardware-based HEC coprocessor has been previously completed, [20] presents many of the architectural requirements. [20] did not achieve a space-efficient implementation and consequently no accurate timing and area values were included. In contrast, this work presents a complete implementation with accurate timing and area requirements.

Several theoretic results are also included. An alternate for computing the GCD of three polynomials of small degree is presented, which significantly decreases processing time in Cantor's Algorithm. The paper also presents a bound on the expected computation time of point multiplication using a point adder and a point doubler in parallel.

2 Basic Algorithms

This paper assumes the reader possesses a familiarity with groups, rings, and fields, in addition to a basic understanding of elliptic curve cryptography. For a complete review of abstract algebra, see [7] or [5]. For a background on the mathematical concepts involved in elliptic curves, see [17] or [2].

Elements in the ECC group are pairs of finite field elements. The corresponding group based on the hyperelliptic curve is its Jacobian. The points are described by pairs of polynomials over a finite field. For a more rigorous treatment of hyperelliptic curves, see [12].

This section develops the various algorithms used to handle binary operations in finite fields, polynomial rings, and on the Jacobian of the hyperelliptic curve. Additionally, some theoretic results on the processing time of certain algorithms are derived.

2.1 Finite Fields

Using a *polynomial basis* over $\mathbb{F}_{2^n}$, any finite field element can be represented as coefficients of powers of x, which can be conveniently stored in memory as an n-bit vector. Another basis called *optimal normal basis* (ONB) is also commonly used because it allows for very efficient squaring; however, ONB is very slow when performing inversions. This implementation uses polynomial basis.

Field Addition. When adding two field polynomials, their sum is the sum of the coefficients. Under a characteristic two finite field, implementing addition requires bitwise XOR-ing together the two vectors representing the polynomial coefficients.

Field Multiplication. Multiplication can be efficiently achieved by using a slightly modified version of the standard grade-school algorithm. A method for

reducing the product as the algorithm progresses is required to prevent it from growing too large, and is presented in Algorithm 1. An alternate field multiplication algorithms originally presented in [19] multiplies D bits simultaneously, requiring $\lceil \frac{n}{D} \rceil$ clock cycles to complete; however, implementing such multipliers in a hyperelliptic environment would not be area efficient enough to fit on most FPGAs.

Algorithm 1: Field Multiplication

Input: $a, b \in \mathbb{F}_{2^n}$, and reduction polynomial f
Output: $c = a \times b$, with c reduced
1. $c \leftarrow 0$
2. for i from $n-1$ downto 1
3. if $(b_i = 1)$ then $c \leftarrow (c + a) \ll 1$ else $c \leftarrow c \ll 1$
4. if (shift carry $= 1$) then $c \leftarrow c + f$
5. if $(b_0 = 1)$ then $c \leftarrow c + a$
6. return c

Field Squaring. In a characteristic p finite field, $(x_1 + \cdots + x_n)^p = x_1^p + \cdots x_n^p$. Hence when squaring in characteristic two, the powers of the basis terms double, essentially spacing out the bits in the vector representation. For example, $(x^2 + x+1)^2 = x^4+x^2+1$. After spacing the bits, the vector is twice its original length, and the higher bits may need to be reduced. Algorithm 2 presents reassignment of the lower bits, and reduction of the upper bits.

Algorithm 2: Field Squaring

Input: $a \in \mathbb{F}_{2^n}$, and reduction polynomial f
Output: $b = a^2 \in \mathbb{F}_{2^n}$
1. $g \leftarrow f \ll 1$
2. Let $b_{2i} = a_i$, for $0 \leq i \leq \lfloor \frac{n}{2} \rfloor$
3. for i from $\lfloor \frac{n}{2} \rfloor$ to n-1
4. if $(a_i = 1)$ then $b \leftarrow b + g$
5. if $(g_{n-1} = 1)$ then $g \leftarrow (g \ll 1) + f$ else $g \leftarrow g \ll 1$
6. repeat step 5
7. return b

Field Inversion. Inversion of finite field elements uses a modified version of the Extended Euclidean Algorithm as reported in [6]. This version only keeps track

of the minimal set of required information, and uses bit shifting with XOR. The details are in Algorithm 3.

Algorithm 3: Field Inversion

Input: $a \in \mathbb{F}_{2^n}$, and reduction polynomial f
Output: $b = a^{-1} \in \mathbb{F}_{2^n}$
1. $b \leftarrow 1$, $c \leftarrow 0$, $u \leftarrow a$, $v \leftarrow f$
2. While $\deg(u) \neq 0$
3. $\quad j \leftarrow \deg(u) - \deg(v)$
4. $\quad$ if $(j < 0)$ then $u \leftrightarrow v$, $b \leftrightarrow c$, $j \leftarrow -j$
5. $\quad u \leftarrow u + (v \ll j)$, $b \leftarrow b + (c \ll j)$
6. return b

2.2 Polynomial Rings

The set of all polynomials with coefficients in $\mathbb{F}_{2^n}$ forms a ring, and is denoted $\mathbb{F}_{2^n}[u]$. This section discusses the mathematical aspects of dealing with these polynomials.

Ring Addition. Addition of two polynomials over a finite field equates to adding each term of each coefficient. That is, to add two polynomials of degree m, with $a_i, b_i \in \mathbb{F}_{2^n}$

$$\sum_{i=0}^{m-1} a_i u^i + \sum_{i=0}^{m-1} b_i u^i = \sum_{i=0}^{m-1} (a_i + b_i) u^i \tag{1}$$

where $a_i + b_i$ is field addition defined in Section 2.1.

Ring Squaring. Since a characteristic two finite field is being used, polynomial squaring has the same property as when performing field squaring, where all odd powers of u have a coefficient of zero. The result is $b_{2i} = a_i^2$, $\forall\ 0 \leq i \leq \deg(a)$ where a_i^2 is calculated as in Section 2.1.

Ring Multiplication. To multiply two ring elements, again defer to the grade-school method. When multiplying a polynomial by a scalar in the field, multiply each term of the polynomial by the scalar. To multiply two polynomials, extend these steps to Algorithm 4.

Ring Division. When dividing two polynomials a and b, a quotient q and remainder r are obtained, such that $a = q \times b + r$. The algorithm required to complete this is a slight modification of the Euclidean algorithm, and is presented in Algorithm 5.

Algorithm 4: Ring Multiplication

Input: $a, b \in \mathbb{F}_{2^n}[u]$
Output: $c = a \times b \in \mathbb{F}_{2^n}[u]$
1. $c \leftarrow 0$
2. for j from $\deg(a)$ downto 0
3. $\quad c \leftarrow (c \ll 1) + a_j \times b$
4. return c

$\times$ is scalar multiplication
$\ll$ is polynomial coefficient shift

Algorithm 5: Ring Division

Input: $a, b \in \mathbb{F}_{2^n}[u]$
Output: $(q, r) \in \mathbb{F}_{2^n}[u]$: $a = q \times b + r$
1. $i \leftarrow (b_{\deg(b)})^{-1}, r = a$
2. for j from $\deg(a) - \deg(b)$ downto 0
3. $\quad f \leftarrow (r_{\deg r} \times i) \ll j$
4. $\quad t \leftarrow b \times f$
5. $\quad r \leftarrow r + t$, $q \leftarrow q + f$
6. return (q, r)

2.3 Hyperelliptic Curves

Hyperelliptic curves are a special class of algebraic curves. The following equation defines a genus g hyperelliptic curve.

$$v^2 + H(u)v = F(u) \tag{2}$$

where $F(u)$ is a monic polynomial of degree $2g + 1$ and $H(u)$ is a polynomial with degree at most g. In this implementation, the ground field is $\mathbb{F}_{2^{113}}$, and it uses the curve definition

$$v^2 + uv = u^5 + u^2 + 1 \tag{3}$$

whence $g = 2$, $F(u) = u^5 + u^2 + 1$, and $H(u) = u$.

Divisors. Divisors are pairs denoted $\mathrm{div}(A, B)$, where A and B are polynomials that satisfy the congruence

$$B^2 + H(u)B \equiv F(u) \pmod{A} \tag{4}$$

Divisors are essentially points on the Jacobian of the hyperelliptic curve. Since these polynomials could have arbitrarily large degree and still satisfy the equation, the notion of a reduced divisor is needed. In a reduced divisor, the degree of

A is no greater than g, and the degree of B is less than the degree of A. Cantor's Algorithm includes a method for reducing divisors.

Additionally, $\mathrm{div}(A, B)$ can be normalized by multiplying A by α^{-1}, where α is the leading coefficient of A. Using normalized divisors speeds up several portions of Cantor's Algorithm.

Jacobian. The Jacobian of a hyperelliptic curve is the set of all reduced divisors. This set is a group under the binary operation defined in the following section. The largest prime dividing the size of this group determines the overall security of the cryptosystem. For the implemented curve the largest prime dividing the size of the group is sixty-eight digits long. This prime is much larger than the current recommended value, and is therefore considered secure.

A more space efficient implementation could be achieved using different curves over smaller fields, however the curve and field combination implemented here is significantly more secure than other combinations of similar complexity. A forthcoming publication, [4], expands the ideas presented here, and examines performance results of genus two curves over other field sizes.

Cantor's Algorithm. Originally presented in [3], Cantor's Algorithm has been the keystone of all computation on Jacobians of hyperelliptic curves. In [11], Cantor's original algorithm was modified for compatability with binary fields. The Koblitz's version of Cantor's Algorithm is presented in Algorithm 6.

Algorithm 6: Cantor's Algorithm

Input: reduced $D_1 = \mathrm{div}(a_1, b_1)$, $D_2 = \mathrm{div}(a_2, b_2)$
Output: reduced $\mathrm{div}(a_3, b_3) = D_1 + D_2$

1. Perform two extended GCD's to compute
   $d = \gcd(a_1, a_2, b_1 + b_2 + H) = s_1 a_1 + s_2 a_2 + s_3(b_1 + b_2 + H)$
2. $a_3 \leftarrow a_1 a_2 / d^2$
3. $b_3 \leftarrow (s_1 a_1 b_2 + s_2 a_2 b_1 + s_3(b_1 b_2 + F))/d \pmod{a_3}$
4. while $\deg(a_3) > g$
5. $\quad a_3 \leftarrow (F - H b_3 - b_3^2)/a_3$
6. $\quad b_3 \leftarrow -H - b_3 \pmod{a_3}$
7. return $\mathrm{div}(a_3, b_3)$

The algorithm can be broken down into three independent steps. The first step is computation of the extended GCD. The second is the composition step, corresponding to steps two and three above. Steps four through seven correspond to the reduction step. Improvements are discussed in each of the following sections.

Table 1. Genus Two GCD Computation Cases

d_1	d_2	d_3	**Result**
x	0	$-\infty$	$d = 1,\ s = (0, 1, 0)$
1	1	$-\infty$	If $\gamma_1 = \gamma_2$ then $d = u - \gamma_2,\ s = (0, 1, 0)$ Else $d = 1,\ (s_1, s_2) = L(a_1, a_2),\ s_3 = 0$
2	1	$-\infty$	If $a_1(-\gamma_2) = 0$ then $d = u - \gamma_2,\ s = (0, 1, 0)$ Else $d = 1,\ (s_1, s_2) = M(a_1, a_2),\ s_3 = 0$
2	2	$-\infty$	Let $D = d\Delta_b - \Delta_c^2$ where $d = \beta_1\gamma_2 - \beta_2\gamma_1,\ \Delta_b = \beta_2 - \beta_1,\ \Delta_c = \gamma_2 - \gamma_1$ If $D \neq 0$ then $d = 1,\ (s_1, s_2) = N(a_1, a_2),\ s_3 = 0$ ElseIf $\beta_1 = \beta_2$ then $d = a_1,\ s_1 = (0, 1, 0)$ Else $d = u + \Delta_c\Delta_b^{-1},\ s = (-\Delta_b^{-1}, \Delta_b, 0)$
x	x	0	$d = 1,\ s = (0, 0, 1)$
x	0	1	$d = 1,\ s = (0, 1, 0)$
1	1	1	If $\gamma_1 = \gamma_2 = \gamma_3$ then $d = u + \gamma_1,\ s = (0, 0, 1)$ ElseIf $\gamma_1 \neq \gamma_2$ then $d = 1,\ s_3 = 0,\ (s_1, s_2) = L(a_1, a_2)$ Else $d = 1,\ s_2 = 0,\ (s_1, s_3) = L(a_1, a_3)$
2	1	1	If $\gamma_2 \neq \gamma_3$ then $d = 1,\ s_1 = 0,\ (s_2, s_3) = L(a_2, a_3)$ ElseIf $a_1(-\gamma_2) \neq 0$ then $d = 1.\ s_1 \neq 0,\ s_3 = 0$ or $s_2 = 0.$ $(s_1, s_2) = M(a_1, a_2)$ or $(s_1, s_3) = M(a_1, a_3)$ Else $d = u + \gamma_2,\ s = (0, 0, 1).$
x	2	1	$a_3 = u + \gamma_3$ If $a_1(-\gamma_3) = a_2(-\gamma_3) = 0$ then $d = u + \gamma_3,\ s = (0, 0, 1)$ ElseIf $a_1(-\gamma_3) \neq 0$ then $d = 1,\ s_2 = 0,\ (s_1, s_3) = M(a_1, a_3)$ Else $d = 1,\ s_1 = 0,\ (s_2, s_3) = M(a_2, a_3)$
$L(a_1, a_2) = (-(\gamma_2 - \gamma_1)^{-1},\ \gamma_2 - \gamma_1)$ $M(a_1, a_2) = (-a_1(-\gamma_2)^{-1},\ a_1(-\gamma_2)^{-1}u + (\beta_1 - \gamma_2)a_1(-\gamma_2)^{-1})$ $N(a_1, a_2) = ((\Delta_b/D)u + (\beta_2\Delta_b - \Delta_c)/D,\ (-\Delta_b/D)u + (-\beta_1\Delta_b + \Delta_c)/D)$			

Extended GCD Calculation. Traditionally, the Extended Euclidean Algorithm (EEA) is used twice to calculate the greatest common divisor of three polynomials, resulting in $d = s_1a_1 + s_2a_2 + s_3a_3$ where $a_3 = b_1 + b_2 + H$. However, for genus two curves with $\deg(H) \leq 1$, the degrees of a_1, a_2, and a_3 are maximally (and most frequently) 2, 2, and 1, respectively. Since a_3 is of degree one, it cannot be factored. Hence, the GCD of the original three polynomials must be either 1 or a_3. Similar arguments can be made for the other possible degree cases. The complete proof of these results is not included in this paper, but will be available in [4].

Table 1 shows a list of all the different cases for the degrees of a_1, a_2, and $a_3 = b_1 + b_2 + H$, and the explicit solution to the GCD and s_i values. For the cases shown in Table 1, $d_i = \deg(a_i)$, $a_1(u) = u^2 + \beta_1 u + \gamma_1$, $a_2(u) = u^2 + \beta_2 u + \gamma_2$, and $a_3(u) = \beta_3 u + \gamma_3$. Additionally, assume that if $\deg(a_2) > \deg(a_1)$, a_1 and a_2 are swapped prior to computation.

Composition Step. The standard composition step in Cantor's Algorithm requires two polynomial divisions, in addition to a third in the form of modular

reduction. In general, polynomial divisions are very time consuming in hardware. However, for genus two curves, notice that d will almost always equal one. The only occurance of degree one is when $b_1 + b_2 + H$ exactly divides both a_1 and a_2. Therefore, most of the time the two polynomial divisions can be completely removed. Additionally, notice that four multiplications in the composition step can be completed in parallel with the GCD to further increase speed.

Reduction Step. Consider the following:

Proposition 1. *For any hyperelliptic curve of genus g, at most $\lceil \frac{g}{2} \rceil$ reduction iterations are required to completely reduce any semi-reduced divisor.*

Proof. See [15], proof 51.

Corollary 1. *Semi-reduced divisors of a genus two curve require at most a single reduction iteration.*

Given Corollary 1, the control logic for the reduction portion of Cantor's algorithm can be simplified. The *while* loop can be replaced with a single *if* block. In [18], Nigel Smart presents an alternate reduction algorithm, based on Tenner reduction. In a genus two case where only one reduction is required, this algorithm is identical to the standard algorithm.

2.4 HEC Cryptosystems

As with most public-key cryptosystems, HEC cryptosystems are usually only used for a symmetric key exchange, using the Diffie-Hellman protocol. Additionally, they can be used to sign messages using the Digital Signature Algorithm (DSA). For details on these algorithms, see [2].

Both algorithms involve scalar point multiplication (divisor P multiplied by scalar integer k) on the Jacobian of the HEC, which is defined as adding P to itself $k - 1$ times. Since $k - 1$ point additions is very slow, a square-and-add approach called *binary expansion* can be used. Given k is expressed as a binary vector, the bases can be calculated by repeatedly doubling P. Then for each 1 in the binary representation of k, add the appropriate basis to a running total. On average, this requires n doubles and $\frac{n}{2}$ adds, and can be efficiently implemented using a point doubler and a point adder operating in parallel.

A point doubler is essentially a special case of a point adder where both inputs are equal. Many multiplications can be replaced with squaring operations, which are significantly faster because field squaring can be implemented very quickly.

Given it takes α time units to complete an add, and β time units to complete a double, the statistical expected amount of time required to multiply a point by an n-bit integer can be computed. For $\alpha < 2\beta$, the time required is discussed in Proposition 2.

Proposition 2. *Let Θ be an endomorphism of group G (e.g. the doubling map for binary expansion). Let k_i be statistically independent and not equaling 0 with probability δ; α be the time to perform group addition; and β be the time required to compute $k_i\Theta^i P$ given $\Theta^{i-1}P$, with $\delta\alpha < \beta$ and the two devices operating in parallel. The expected time T to compute*

$$\left(\sum_{i=0}^{n-1} k_i\Theta^i\right) P = \sum_{i=0}^{n-1} k_i(\Theta^i P) \tag{5}$$

has a sharp upper bound of $\frac{\alpha^2\delta(1-\delta)}{2(\beta-\alpha\delta)} + \alpha\delta + (n-1)\beta$.

Proof. The overall system can be modeled as a discrete time D/G/1 queue (see [9]). Evaluation of the endomorphism corresponds to queue arrivals with deterministic interarrival times X equal to β. For the service time process Y, each queue departure is nonzero with probability δ and hence has service time α. The following are the first and second order statistics for stochastic processes X and Y:

$$E[X] = \beta \quad \mathrm{Var}(X) = 0 \quad E[Y] = \alpha\delta \quad \mathrm{Var}(Y) = \alpha^2\delta(1-\delta) \tag{6}$$

For a reasonably large n, the mean system waiting time, W, in equilibrium converges in distribution to the mean system waiting time after $(n-1)\beta$ time units due to the basic principles of renewal theory (see [9]). Therefore, $T \stackrel{d}{=} W + (n-1)\beta$. W can be easily bounded above by the Kingman Moment Bound (see [1]).

$$W \leq \frac{(1/E[X])(\mathrm{Var}(X) + \mathrm{Var}(Y))}{2(1 - E[Y]/E[X])} + E[Y] \tag{7}$$

Substituting the statistics:

$$T \stackrel{d}{=} W + (n-1)\beta \leq \frac{\frac{1}{\beta}(\alpha^2\delta(1-\delta))}{2(1-\frac{\alpha\delta}{\beta})} + \alpha\delta + (n-1)\beta = \frac{\alpha^2\delta(1-\delta)}{2(\beta-\alpha\delta)} + \alpha\delta + (n-1)\beta \tag{8}$$

Notice that as $\alpha\delta$ approaches β, the bound on the processing time goes to infinity. This is because the queue length builds up indefinitely and the system is no longer positive recurrent; the waiting time after $(n-1)\beta$ time units cannot be approximated by the equilibrium distribution, and no general solution is possible using this model.

Corollary 2. *In the case where $\theta = [2]$ and G is the Jacobian of a hyperelliptic curve: $\delta = \frac{1}{2}$, α is the time to perform point addition, and β is the time to complete point doubling, with $\alpha < 2\beta$. Therefore the mean time T to perform a point multiplication has a sharp upper bound of $\frac{\alpha^2}{8\beta-4\alpha} + \frac{\alpha}{2} + (n-1)\beta$*

3 Coprocessor Implementation

In order to effectively utilize FPGA area, the final coprocessor includes two polynomial multipliers and one each of the other polynomial computation blocks. A control unit is responsible for channeling data in and out of the computation blocks, implementing Cantor's algorithm.

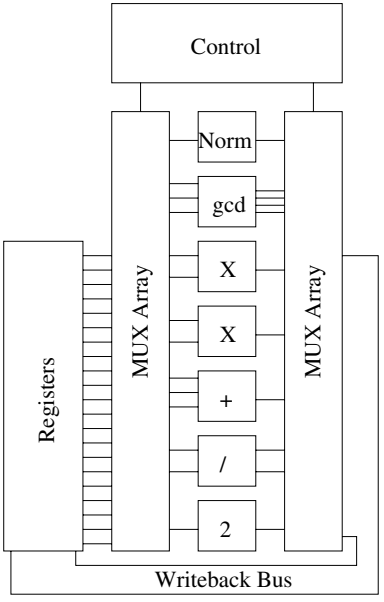


Fig. 1. Overall Coprocessor Architecture

3.1 Field Calculation Blocks

Since field addition is such a simple operation, a separate Verilog module was not created to implement it.

Field multiplication is at the heart of most calculations, and therefore must be done quickly and efficiently. Implemented in hardware, Algorithm 1 uses combinatorial logic to compute $((c+a) \ll 1) + f$, $(c+a) \ll 1$, $(c \ll 1) + f$, and $c \ll 1$ and a multiplexor to select the correct result for a given iteration.

While an alternate multiplication design which processes multiple bits at a time could be implemented, effectively making the coprocessor operate two to three times faster, the extra area requirements are not physically realizable on a Xilinx FPGA. These are known as $D = X$ multipliers where X is the number of bits that can be simultaneously computed.

For field squaring, Algorithm 2 step two is a rewiring, using no gates. If the upper bits are zero (such as when squaring 1), the algorithm is complete.

Otherwise, during each loop iteration, updating b and g can occur independently and consequently during the same clock cycle. By considering two bits of g rather than one, steps five and six can be combined using a multiplexor.

Field inversion is used in the polynomial GCD and polynomial division blocks. Algorithm 3 is implemented as a finite state machine.

Table 2. Field Implementation Results

Module	Clock Cycles	Slices	Max Speed
Field Multiplication	2 or 115 [a]	399	96 MHz
Field Squaring	2 or 59 [b]	186	124 MHz
Field Inversion	395 (avg)	1631	98 MHz

[a] When multiplying by zero or one, the result is immediate

[b] When squaring a field element of degree $\lfloor \frac{n}{2} \rfloor$ or less, only reassignment is required

3.2 Polynomial Calculation Blocks

Polynomial blocks are collections of the appropriate field blocks combined to implement various algorithms. The control structure only has access to polynomial units; therefore, all functionality required by Cantor's Algorithm must be available through polynomial calculation blocks. Additionally, polynomials have no maximum degree. The input size for each polynomial block was determined by the maximal value of the intermediate states, and ensures there will be no overflow problems.

The addition block accepts three input polynomials of maximum degree six, and returns their sum. This requires 1582 XOR gates and is completely combinatorial.

The maximum degree ever encountered during multiplication for the first input is five, and the second input is two. Therefore, the polynomial multiplier must accommodate multiplication of a degree five polynomial by a degree two polynomial. This requires each of six field multipliers to complete three multiplications, as demonstrated by Algorithm 4. The version implemented in this design includes cases to check for multiplication by both zero and one, which results in a two clock-cycle calculation.

The ring squarer accepts inputs with degree up to three, and returns polynomials of maximal degree six. Its implementation consists of four field squarers, operating in parallel.

Ring division is by far the most complex and time-consuming operation. The number of required clock cycles greatly depends on the degrees of the two input polynomials, which can range from zero to six for the numerator, and zero to four for the denominator. The greater the difference in degree, the longer

the processing takes. A special case of scalar multiplication by the inverse was incorporated for a zero degree denominator.

Ring normalization makes a degree two polynomial monic by multiplying its terms by the inverse of its leading coefficient. It is implemented using a field inverter and two field multipliers.

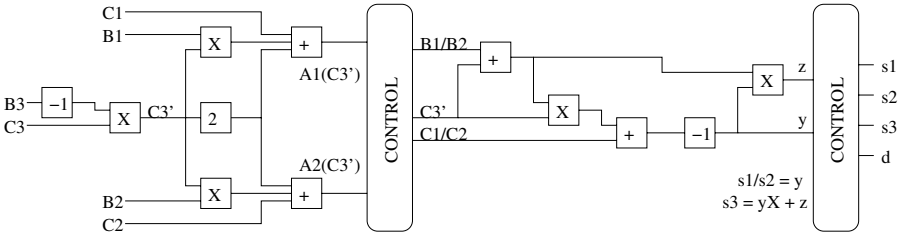


Fig. 2. Dual Degree Two Case GCD Calculation Block

While all the cases described in Table 2 are significantly faster than using the Extended Euclidean Algorithm for the GCD computations, the chip area required to implementation all the cases is prohibitive. Therefore, only the final case where $d_1 = d_2 = 2$ is implemented as shown in Figure 2, using two field multipliers, a field squarer, and a field inverter. For all other cases, the coprocessor control subsystem uses the already existing polynomial logic blocks to perform the Extended Euclidean Algorithm. Since the probability of having $d_1 < 2$ or $d_2 < 2$ is small, the adverse performance effects are trivial.

Table 3. Ring Implementation Results

Module	Clock Cycles	Slices	Max Speed
Ring Addition	1	791	83 MHz
Ring Multiplication	2 to 353	1,561	64 MHz
Ring Squaring	2 or 59	515	55 MHz
Ring Division	2 to 2,300	8,337	80 MHz
Ring GCD	1,270 (avg)	3,515	96 MHz
Ring Norm	615 (avg)	2,488	71 MHz

3.3 Control Implementation

Using the available polynomial computation blocks, a control block operated by a finite state machine decides which data should enter each polynomial ring unit, decides when the unit is done processing, and keeps track of intermediate variables as Cantor's algorithm progresses. The scheduling is programmed into

the state machine. Since some computation blocks require a variable number of clock cycles to complete, the control mechanism does not try to use a result that has not yet been obtained, and it also does not waste time waiting for a result that has already returned.

Two different control mechanisms were designed. The first is a finite state machine which performs general point addition. The second is a finite state machine which performs point doubling. The point doubler is not as simple as the one described in [18] because the implemented curve uses $H(u) = u$ rather than $H(u) = 1$. The implementation results of both are included in the next section.

4 Performance

Since the emphasis of this paper is on the hardware implementation, the software implementation is not extensively discussed. The original intent of the software version was to verify the correctness of the hardware version; however, an effort was made to minimize both memory usage and runtime. On a Pentium III, 1.2 GHz, point addition, doubling, and multiplication can be completed in 1.97, 1.01, and 222 milliseconds, respectively.

Using the Xilinx ISE tools with target chip configured as the Virtex II FPGA, the final implementation results for the hardware version are stated in Table 4. Through both experimental tests and application of Corollary 2, point multiplication can be achieved in 10.1 milliseconds. The smallest Xilinx device supporting the full point multiplication architecture using a parallel adder and doubler is the Virtex II 2VP30, which has 30,816 slices.

Table 4. Hardware Implementation Results

Operation	Clock Cycles	Slices	Max Speed	Time
Point Addition	4,750	16,600	45 MHz	105 μs
Point Doubling	4,050	15,100	45 MHz	90 μs

5 Conclusion

By using many standard finite field algorithms, and an alternate method for computing the GCD, this implementation is proof that HECC can be implemented at speeds comparable to ECC, and realizable in areas suited for embedded applications.

An important thing to keep in mind when comparing these results to those of ECC coprocessors implementing an equal level of security is the algorithm complexity. Cantor's algorithm operates over polynomials and not individual field elements, which results in much higher FPGA area requirements. The timing

results in Table 4 could easily be reduced by a factor of 10 by using a $D = 16$ field multiplier (such as in [16]); however, such a chip would have unreasonable space requirements and not be physically realizable on most FPGA's.

Future work on this project is expected at both the hardware and software levels. We are interested in carrying the implementation forward and completing the ASIC level design and eventual fabrication of the chip. Additionally, we are extending the implementation to use the τ-adic method for point multiplication, as described in [8] and more extensively in [13]. We are also examining implementations of a genus three curve over $\mathbb{F}_{2^{61}}$, on a 64-bit system.

References

1. D. Bertsekas, R. Gallager. *Data Networks.* Prentice Hall: New Jersey. 1992.
2. I. Blake, G. Seroussi, N. Smart. *Elliptic Curves in Cryptography.* London Mathematical Society Lecture Notes Series, Volume 265. University Press: Cambridge. 1999.
3. D. Cantor. "Computing the Jacobian of a Hyperelliptic Curve." *Mathematics of Computation*, v48, pp 95–101. January 1987.
4. T. Clancy, Y. Liow. "FPGA-based Hyperelliptic Curve Cryptosystems." *to appear.*
5. D. Dummit, R. Foote. *Abstract Algebra.* John Wiley and Sons: New York. 1999.
6. D. Hankerson, J. Hernandez, A. Menezes. "Software Implementation of Elliptic Curve Cryptosystems Over Binary Fields." *CHES 2000*, LNCS, Volume 1965, pp 1–24. Springer-Verlag: New York. 2000.
7. T. Hungerford. *Algebra.* Graduate Texts in Mathematics, Volume 73. Springer-Verlag: New York. 1974.
8. C. Gunther, T. Lange, A. Stein. "Speeding up the Arithmetic on Koblitz Curves of Genus Two." *SAC 2000*, LNCS, Volume 2012, pp 106–117. Springer-Verlag: New York. 2001.
9. L. Kleinrock. *Queueing Systems, Volume I: Theory.* John Wiley and Sons: New York. 1975.
10. N. Koblitz. "A Family of Jacobians Suitible for Discrete Log Cryptosystems." *CRYPTO 1988*, LNCS, Voume 403, pp 94–99. Springer-Verlag: New York. 1988.
11. N. Koblitz. "Hyperelliptic Cryptosystems." *Journal of Cryptology*, Volume 1, pp 139–150. 1989.
12. N. Koblitz. *Algebraic Aspects of Cryptography.* Algorithms and Computation in Mathematics, Volume 3. Springer: New York. 1998.
13. T. Lange. "Fast Arithmetic on Hyperelliptic Curves." PhD Thesis, Institute for Information Security and Cryptography, Ruhr-Universität Bochum. 2002.
14. A. Menezes, P. van Oorschot, S. Vanstone. *Handbook of Applied Cryptography.* CRC Press. 1997.
15. A. Menezes, Y. Wu, R. Zuccherato. "An Elementary Introduction to Hyperelliptic Curves." appendix in *Algebraic Aspects of Cryptography* N. Koblitz, Springer-Verlag, 1998.
16. G. Orlando, C. Paar. "A High Performance Reconfigurable Elliptic Curve Processor for GF(2^m)." *CHES 2000*, LNCS, Volume 1965, pp 41–56. Springer-Verlag: New York. 2000.
17. J. Silverman. *Arithmetic of Elliptic Curves.* Graduate Texts in Mathematics, Volume 106. Springer-Verlag: New York. 1986.

18. N. Smart. "On the Performance of Hyperelliptic Cryptosystems." HPL-98-162. HP Extended Enterprise Laboratory. September 1998.
19. L. Song, K. Parhi, "Low-Energy Digit-Serial/Parallel Finite Field Multipliers," *Journal of VHDL Signal Processing Systems.* pp 1–17. 1997.
20. T. Wollinger. "Computer Architectures for Cryptosystems Based on Hyperelliptic Curves." Master's Thesis, Worcester Polytechnic Institute. April 2001.

True Random Number Generator Embedded in Reconfigurable Hardware

Viktor Fischer[1] and Miloš Drutarovský[2]

[1] Laboratoire Traitement du Signal et Instrumentation,
Unité Mixte de Recherche CNRS 5516, Université Jean Monnet,
Saint-Etienne, France
fischer@univ-st-etienne.fr
[2] Department of Electronics and Multimedia Communications,
Technical University of Košice,
Park Komenského 13, 041 20 Košice, Slovak Republic
Milos.Drutarovsky@tuke.sk

Abstract. This paper presents a new True Random Number Generator (TRNG) based on an analog Phase-Locked Loop (PLL) implemented in a digital Altera Field Programmable Logic Device (FPLD). Starting with an analysis of the one available on chip source of randomness - the PLL synthesized low jitter clock signal, a new simple and reliable method of true randomness extraction is proposed. Basic assumptions about statistical properties of jitter signal are confirmed by testing of mean value of the TRNG output signal. The quality of generated true random numbers is confirmed by passing standard NIST statistical tests. The described TRNG is tailored for embedded System-On-a-Programmable-Chip (SOPC) cryptographic applications and can provide a good quality true random bit-stream with throughput of several tens of kilobits per second. The possibility of including the proposed TRNG into a SOPC design significantly increases the system security of embedded cryptographic hardware.

1 Introduction

Random number generators represent basic cryptographic primitives. They are widely used for example as confidential key generators for symmetric key and public-key crypto-systems (e. g. RSA-moduli) and as password sources. In some algorithms (e.g. DSA) or protocols (e.g. zero-knowledge), random numbers are intrinsic to the computation [1]. In all these applications, security depends greatly on the randomness of the source.

Because security algorithms and protocols rely on the unpredictability of the keys they use, random number generators for cryptographic applications must meet stringent requirements. Unfortunately computers and digital hardware can implement only pseudo-random generators. A Pseudo-Random Number Generator (PRNG) is a deterministic polynomial time algorithm that expands short (hopefully true random and well distributed) seeds into long bit sequences, this

B.S. Kaliski Jr. et al. (Eds.): CHES 2002, LNCS 2523, pp. 415–430, 2003.

distribution is polynomially indistinguishable from the uniform probability distribution. PRNGs rely on complexity and their use in cryptography, for example to generate keys, is very critical. An alternative solution is to get true random numbers, hence true security for crypto-systems, using a True Random Number Generator (TRNG) based on a random physical phenomenon. Even an ideal PRNG relies upon, and is limited by, the quality of its input seed data. Good TRNG is designed to generate high-quality random numbers directly or as a seed for PRNG. Current modern high-density Field Programmable Logic Devices (FPLDs) provide a suitable hardware platform for a complete System-On-a-Programmable-Chip (SOPC). This SOPC can be used for cryptographic applications, even for system-level integration of embedded algorithms. Unfortunately, high quality embedded TRNGs were not realizable in FPLDs. Most hardware TRNGs follow unpredictable natural processes, such as thermal (resistance or shoot) noise or nuclear decay. Such TRNGs are not compatible with modern FPLDs and cannot provide a SOPC solution. The fact that TRNG cannot be implemented inside the FPLD represents significant security and system disadvantages in embedded cryptographic applications.

TRNGs can be produced using any non deterministic process. The fundamental probabilistic phenomena utilized by proposed TRNG is the frequency instability of electronic oscillator. The use of this phenomena to generate truly random numbers is not new and was used e.g. in [2], [3]. These implementations used two free running oscillators with relatively high instability at least one of them.

This paper describes implementation of new analog Phase-Locked Loop (PLL) based TRNG that uses on-chip resources of recent Altera FPLD families (e. g. APEX E [4], APEX II [5], etc.). Described TRNG uses two coupled oscillators that are not free running and originally designed to be as stable as possible. Proposed method reliably extracts intrinsic randomness from low-jitter clock signals synthesized by on-chip analog PLL circuits and to our best knowledge it is the first TRNG implementation that uses only on-chip FPLD resources. This paper extends the description of the proposed method first announced in [6], provides new results of tested output TRNG signals, reveals some deviations from ideal TRNG, and discusses system aspects of proposed TRNG. It is organized as follows: a brief overview of jitter performance of analog PLL circuits embedded in recent FPLDs is given in Sect. 2. In Sect. 3, a proposed new method of reliable true randomness extraction from low jitter on-chip PLL synthesized clock signal is presented. The experimental TRNG hardware used for the testing of the proposed method is described in Sect. 4. In Sect. 5, statistical evaluations of output TRNG signals are made. Finally, concluding remarks are presented in Sect. 6.

2 PLL – Source of Randomness in Recent FPLDs

Recent FPLDs use often on-chip PLLs to increase performance of clock distribution and to provide on-chip clock-frequency synthesis. There are two fundamental

approaches to implement PLL in FPLDs - one uses digital delay lines, or DLL, (e.g. in XILINX Virtex FPLDs [7]) and the second one uses true analog PLL (e.g. in Altera APEX E [4] and APEX II [5] FPLDs). Both approaches have some system advantages and disadvantages but we believe that analog PLL is a better candidate for cryptographic TRNG design since it contains analog source of unpredictable randomness.

2.1 Analog PLL in Altera FPLD

To support high-speed designs, new Altera FPLD devices offer ClockLock, Clock-Boost and ClockShift circuitry containing several integrated on-chip analog PLL circuits. Block diagram of enhanced PLL sub-circuit available in latest versions of APEX E and APEX II FPLDs is depicted in Fig. 1 [4], [5].

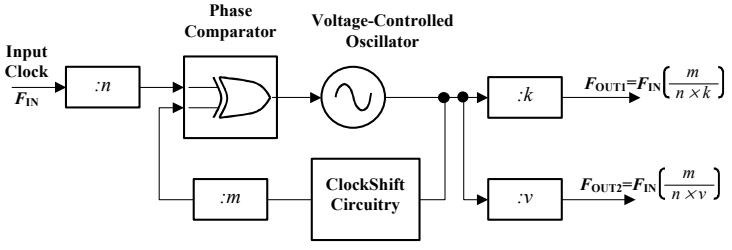


Fig. 1. Block diagram of enhanced Altera PLL circuit

In analog PLLs, various noise sources cause the internal voltage controlled oscillator (VCO) to fluctuate in frequency. The internal control circuitry adjusts the VCO back to the specified frequency and this change is seen as jitter. Under ideal conditions, the jitter is caused only by analog (non-deterministic) internal noise sources. Such jitter is called an intrinsic jitter. Other possible frequency fluctuations are caused by variations of supply voltage, temperature, external interference through the power, ground, and even by the internal noisy environment generated by internal FPLD circuits [7]. From cryptographic point of view, these sources should be regarded as deterministic and the function of TRNG must not be deteriorated by them. In other words, the output TRNG must in any case depend also on the non-deterministic intrinsic jitter. Any additional disturbing deterministic jitter is possible as far as dependency of the output signal on intrinsic jitter is guaranteed.

2.2 Jitter Characteristics of Altera PLL Circuitry

Parameters of the proposed TRNG depend on the jitter characteristics of Altera embedded PLLs. Real measurements of jitter parameters requires the use of special equipment which was not available, so we had to rely on the parameters

given in the Altera data sheets [4] and the application note [8]. Some of these parameters have been independently confirmed by Xilinx and the results are available in [7]. Since these parameters are fundamental for our TRNG design, they are summarized and discussed in this subsection.

Altera tries to minimize the clock jitter[1] by a proper design, for example their typical analog intrinsic PLL jitter in an APEX FPLD has 1-sigma value of $\sigma_{\text{jit}} \approx 15$ ps (under Gaussian approximation, the peak-to-peak jitter value is approximately $t_{\text{JITTER}} = 6\sigma_{\text{jit}}$) for a $F = 66.6$ MHz synthesized clock signal and multiplication factor of $2\times$ [7]. Actual distribution of jitter values is depicted in Fig. 2 [7]. These results were acquired under "ideal conditions", with only a minimal amount of occupied FPLD resources and minimal input/output activities.

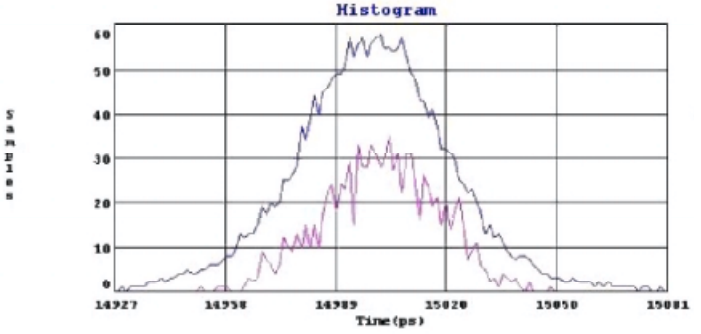


Fig. 2. APEX intrinsic jitter performance for 1,000 clock samples (bottom curve, peak-to-peak value 97.0 ps, $\sigma_{\text{jit}} \approx 15.9$ ps) and 1,000,400 clock samples (upper curve, peak-to-peak value 151.4 ps, $\sigma_{\text{jit}} \approx 15.7$ ps)

In [7] it was shown that the clock jitter in APEX FPLD is significantly higher, when internal FPLD flip-flops are switching with different clock frequencies. It was shown that when 35 % of the total available flip-flops were clocked with a 33.3 MHz clock and 35 % of the flip-flops with a 66.6 MHz clock, jitter is much higher than that specified in the data-sheet. These conditions simulated an internal noisy environment generated by internal FPLD circuits and jitter distribution was split into two peaks with a 665 ps total peak-to-peak value [7]. Although overall jitter performance exceeds data sheet specification, true intrinsic jitter is still present and it is clearly visible as two approximated Gaussian peaks have around 150 ps. We can conclude that under real conditions the clock jitter

[1] There are two types of jitter described in [7], [8], period jitter and cycle-to-cycle jitter. Period jitter is the deviation in time of any clock period from the ideal clock period (also known as "edge-to-edge" jitter). Peak-to-peak jitter defines an upper bound on the jitter. Cycle-to-cycle jitter is the deviation in clock period between adjacent or successive clock cycles.

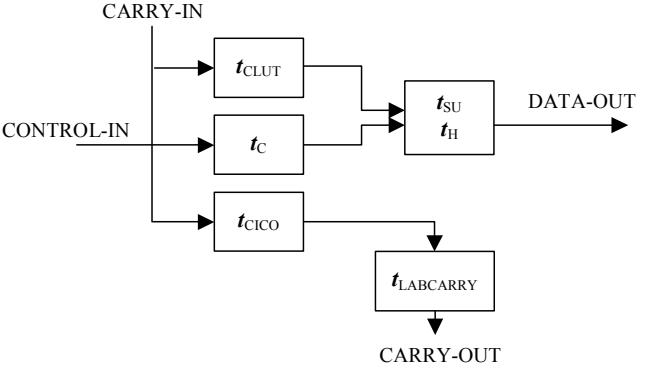
always contains intrinsic jitter and only the overall jitter distribution is changed. Such behavior is expected, since the intrinsic jitter cannot be removed by any interference.

3 Randomness Extraction from an Intrinsic Jitter

The principle behind our method is to extract the randomness from the jitter of the clock signal synthesized in the embedded analog PLL. The jitter is detected by the sampling of a reference (clock) signal using a correlated (clock) signal synthesized in the PLL. The fundamental problem lies in the fact that the reference signal has to be sampled near the edges influenced by the jitter. From the previous section we know that clock edges of a synthesized signal can vary under ideal conditions in the range of several tens of ps. This value is significantly lower than the smallest delay obtainable in APEX FPLDs and our method must overcome this problem.

3.1 Timing Analysis of the Logic Cell in Altera FPLD

The smallest possible delay in Altera FPLDs is obtainable between the carry-in and carry-out of the Logic Cell (LC). A simplified timing model of the logic cell is depicted in Fig. 3.



t_{CLUT} – Look-up-table (LUT) delay for carry-in
t_{SU} – Logic cell (LC) register setup time for data
t_H – LC register hold time for data
t_C – LC register control signal delay
t_{CICO} – Carry-in to carry-out delay
$t_{LABCARRY}$ – Routing delay for the carry-out signal of a LC driving the carry-in signal of a different LC in a different Logic Array Block (LAB).

Fig. 3. Altera simplified logic cell timing model

We have taken the parameters obtained from the Quartus II [9] version 2.0 timing analyzer as the basis for our method. From the result of this analysis we can conclude that the smallest obtainable delay in APEX FPLDs is $\tau \approx t_{\mathrm{CICO}} = 500\,\mathrm{ps}$. The delay τ is only a statistical value and its real size can vary with time, temperature and supply voltage.

3.2 Basic Principle of Randomness Extraction

The basic principle of the proposed randomness extraction is illustrated in Fig. 4 [6].

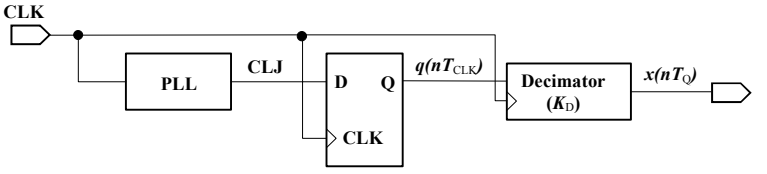


Fig. 4. Basic principle of randomness extraction from low-jitter clock signal

Let CLK be a system clock signal with the frequency F_{CLK}. In an actual implementation CLK can be either an external signal or it can be internally synthesized by an additional on-chip PLL. Let CLJ be an on-chip PLL-synthesized rectangular waveform with the frequency $F_{\mathrm{CLJ}} = F_{\mathrm{CLK}} K_{\mathrm{M}}/K_{\mathrm{D}}$. Let values of multiplication factor K_{M} and division factor K_{D} be relative primes, so

$$\mathrm{GCD}\,(K_{\mathrm{M}}, K_{\mathrm{D}}) = 1 \tag{1}$$

where GCD is an abbreviation for Greatest Common Divisor. Equation (1) ensures that the maximum guaranteed distance between the closest edges of CLK and CLJ (denoted as $\mathrm{MAX}\,(\Delta T_{\mathrm{min}})$) is minimized. As it is discussed in Sect. 2, signal CLJ certainly includes intrinsic analog PLL jitter σ_{jit} and it can also contain other "deterministic" jitter components from external or internal environment. This signal is sampled into the D flip-flop using a clock signal with frequency F_{CLK}. The sampled signal $q\,(nT_{\mathrm{CLK}})$ contains certain random values. Their exact position is not known and potentially it can vary in time. Random values can be easily extracted by a standard XOR decimator [10], [11]. In the proposed design the decimator produces one output bit per K_{D} input values $q\,(nT_{\mathrm{CLK}})$ (one period T_{Q}). The next paragraphs analyze more deeply the functionality of the proposed circuit.

Let us consider the output signal in two different conditions: ideal conditions without jitter and real conditions when jitter is included in the synthesized clock signal. Under ideal conditions when the jitter is zero ($\sigma_{\mathrm{jit}} = 0$), signal $q(nT_{\mathrm{CLK}}),\ n = 0, 1, \ldots$ is deterministic and under condition (1) periodic with the period

$$T_{\mathrm{Q}} = K_{\mathrm{D}} T_{\mathrm{CLK}} = K_{\mathrm{M}} T_{\mathrm{CLJ}}\ . \tag{2}$$

Therefore decimated output signal $x\,(nT_{\mathrm{Q}})\,,\ n = 0, 1, \ldots$

$$x\,(nT_{\mathrm{Q}}) = q\,(nT_{\mathrm{Q}}) \oplus q\,(nT_{\mathrm{Q}} - T_{\mathrm{CLK}}) \quad \oplus \ldots \oplus q\,(nT_{\mathrm{Q}} - (K_{\mathrm{D}} - 1)\,T_{\mathrm{CLK}}) \quad (3)$$

which represents bit-wise addition modulo 2 of K_{D} input samples, is also deterministic. The situation is completely different under real conditions when the jitter is nonzero ($\sigma_{\mathrm{jit}} > 0$). If K_{D} is chosen so, that the jitter σ_{jit} is comparable with the maximum distance $\mathrm{MAX}\,(\Delta T_{\mathrm{min}})$ between the two closest edges of CLK and CLJ, we can guarantee that during T_{Q} the rising edge of CLK will fall at least once into edge zone of CLJ (edge zone means the time interval around the edge including jitter[2]). The value $\mathrm{MAX}\,(\Delta T_{\mathrm{min}})$ can be computed as

$$\mathrm{MAX}\,(\Delta T_{\mathrm{min}}) = T_{\mathrm{CLK}} \frac{\mathrm{GCD}\,(2K_{\mathrm{M}}, K_{\mathrm{D}})}{4K_{\mathrm{M}}} = T_{\mathrm{CLJ}} \frac{\mathrm{GCD}\,(2K_{\mathrm{M}}, K_{\mathrm{D}})}{4K_{\mathrm{D}}}\ . \quad (4)$$

The during current period of T_{Q}, K_{D} values of CLJ will be sampled into D flip-flop and at least one of them will depend on the random jitter. The decimated signal $x\,(nT_{\mathrm{Q}})$ will not be deterministic anymore and its value will depend on this jitter. In Fig. 5-7 we analyze different possibilities for small values of K_{M} and K_{D} that demonstrate the validity of (4).

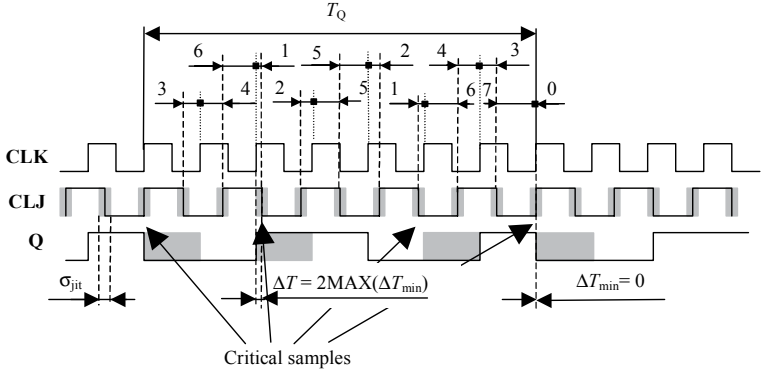


Fig. 5. Clock relation for $K_{\mathrm{M}} = 5$, $K_{\mathrm{D}} = 7$ ($F_{\mathrm{CLJ}} < F_{\mathrm{CLK}}$)

Figure 5 shows the case when $\mathrm{GCD}\,(2K_{\mathrm{M}}, K_{\mathrm{D}}) = 1$ and $F_{\mathrm{CLJ}} < F_{\mathrm{CLK}}$. In real implementation it is not possible to guarantee the position of CLJ in relationship to CLK. In this example the minimum distance ΔT_{min} is 0 (the last sample of the period T_{Q}). The worst case (maximum value of the minimum distance - $\mathrm{MAX}\,(\Delta T_{\mathrm{min}})$) will be the event when CLJ will be shifted by a half

[2] For qualitative analysis we can assume that the width of the edge zone is for example $6\sigma_{\mathrm{jit}}$. Therefore there is some non-zero probability that the jitter will influence the sampled signal value.

step (the step is equal to[3] $T_{\mathrm{CLJ}}/2K_{\mathrm{D}}$) to the left or to the right. In that case the minimum difference will be the half step in at least one of critical samples (they are indicated by arrows) and the output value Q will be nondeterministic during one period T_{CLK} (gray zones in Q output signal). Conclusion: the jitter should be comparable with the half step, therefore $\sigma_{\mathrm{jit}} \approx T_{\mathrm{CLJ}}/4K_{\mathrm{D}}$ and so

$$\mathrm{MAX}\left(\Delta T_{\mathrm{min}}\right) = \frac{T_{\mathrm{CLK}}}{4K_{\mathrm{M}}} = \frac{T_{\mathrm{CLJ}}}{4K_{\mathrm{D}}} = \frac{T_{\mathrm{CLJ}}}{28} \tag{5}$$

Figure 6 shows the case when $\mathrm{GCD}\left(2K_{\mathrm{M}}, K_{\mathrm{D}}\right) = 1$ and $F_{\mathrm{CLJ}} > F_{\mathrm{CLK}}$. Following the previous study it can be found that $\mathrm{MAX}\left(\Delta T_{\mathrm{min}}\right)$ can be expressed in the same way as in (5), so (4) is valid, too.

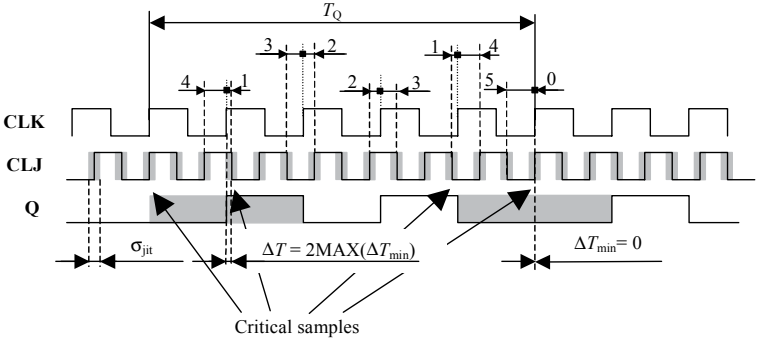


Fig. 6. Clock relation for $K_{\mathrm{M}} = 7$, $K_{\mathrm{D}} = 5$ ($F_{\mathrm{CLJ}} > F_{\mathrm{CLK}}$)

Figure 7 shows the case when K_{D} is even so $\mathrm{GCD}\left(2K_{\mathrm{M}}, K_{\mathrm{D}}\right) = 2$ and $F_{\mathrm{CLJ}} < F_{\mathrm{CLK}}$. It can be found that $\mathrm{MAX}\left(\Delta T_{\mathrm{min}}\right)$ can be expressed as

$$\mathrm{MAX}\left(\Delta T_{\mathrm{min}}\right) = \frac{T_{\mathrm{CLJ}}\mathrm{GCD}\left(2K_{\mathrm{M}}, K_{\mathrm{D}}\right)}{4K_{\mathrm{M}}} = \frac{2T_{\mathrm{CLJ}}}{4K_{\mathrm{D}}} = \frac{T_{\mathrm{CLJ}}}{2K_{\mathrm{D}}} = \frac{T_{\mathrm{CLJ}}}{16} \tag{6}$$

and (4) is valid also in this case.

Following this analysis we can conclude that according to (4) it is better (if it is possible from system point of view) to choose relative primes K_{M}, K_{D} in such a way that K_{D} is odd. This choice will decrease $\mathrm{MAX}\left(\Delta T_{\mathrm{min}}\right)$ by the factor of 2.

[3] Note that there are $K_{\mathrm{D}} = 7$ clock periods in interval T_{Q}. The longest distance ΔT is 7Δ. 7Δ is a half period of CLJ. So the longest distance ΔT is the half period of CLJ. The worst case of the largest distance $\mathrm{MAX}\left(\Delta T_{\mathrm{min}}\right)$ is $0.5\Delta = 1/14$ of the half period of CLJ. That means $1/28 = 1/\left(4K_D\right)$ of the full period of CLJ.

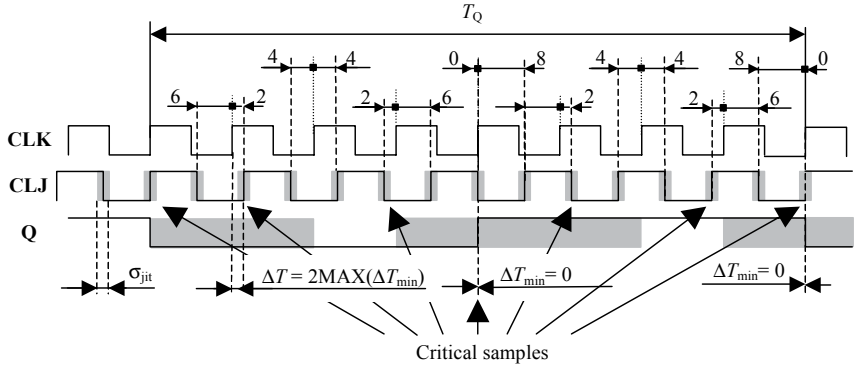


Fig. 7. Clock relation for $K_M = 7$, $K_D = 8$ ($F_{CLJ} < F_{CLK}$)

3.3 TRNG Realization

Under real conditions D flip-flop in Fig. 4 produces signal $q(nT_{CLK})$ that is sampled K_D times during the time interval T_Q. Based on the analysis in Sect. 3.2 it is possible to express the decimated output signal $x(nT_Q)$ as

$$\begin{aligned} x(nT_Q) = [&q(nT_Q) \oplus q(nT_Q - T_{CLK}) \oplus \dots \\ &\dots \oplus q(nT_Q - (J-1)T_{CLK}) \oplus q(nT_Q - (J+1)T_{CLK}) \oplus \dots \\ &\dots \oplus q(nT_Q - (K_D - 1)T_{CLK})] \quad \oplus \quad q(nT_Q - JT_{CLK}) \end{aligned} \tag{7}$$

where the first term in (7) contains all values not influenced by the jitter (therefore they are deterministic) and the second term[4] $q(nT_Q - JT_{CLK})$ is influenced by the jitter. In general, values $q(nT_Q - JT_{CLK})$ are statistically biased random bits that have expectation (long run average) $p = E[q(nT_Q - JT_{CLK})]$ different from the ideal value of $1/2$ by a bias $b = p - 1/2$. Under Gaussian approximation the bias for intrinsic jitter can be computed by

$$|b| = \left| \frac{1}{\sqrt{2\pi}} \int_{-\infty}^{\frac{\Delta T_{min}}{\sigma_{jit}}} e^{\frac{-x^2}{2}} dx - \frac{1}{2} \right| \leq \frac{1}{2} \mathrm{erf}\left(\frac{\mathrm{MAX}(\Delta T_{min})}{\sigma_{jit}\sqrt{2}} \right) < \frac{1}{2} \tag{8}$$

where erf() is the Error function [15]. Since the exact value of J is not known (it can be influenced by a non-deterministic jitter described in Sect. 2 or by the temperature and supply voltage variations that influence delays in the D flip-flop) it is necessary to use a decimator that produces $x(nT_Q)$ according to (7) with the output sample rate T_{CLK}/K_D. This ensures that randomness included in the value $q(nT_Q - JT_{CLK})$ is also included in $x(nT_Q)$ without precise knowledge of

[4] In principle more terms could be influenced by the jitter but according to the previous analysis, choosing proper values K_M and K_D we can guarantee that at least one sample will be influenced by the jitter.

the actual value J (position of the sample influenced by jitter in the frame of one period T_{Q}).

Good TRNG should produce binary outputs with equal probability, so $b \to 0$. Signal $x\,(nT_{\mathrm{Q}})$ generally does not fulfill this requirement. One common way to reduce statistical bias is to use a XOR corrector [10], [11]. The simplest XOR corrector takes non-overlapped pairs of bits[5] from the input stream and XORs them to produce an output stream with the half bit-rate of the input stream. If input stream bits are statistically independent then the bias at the output (decimated) stream is $b_{\mathrm{out}} = -2b_{\mathrm{in}}^2$ and $|b_{\mathrm{out}}| < |b_{\mathrm{in}}|$ since $|b_{\mathrm{in}}| < 1/2$ [10]. There are two XOR operators needed in the complete TRNG realization (see Fig. 8): the XOR decimator implied by the basic principle of the method (described above) and a XOR corrector of N_{d} samples

$$q_i\,(nT_{\mathrm{CLK}}) = q\left(nT_{\mathrm{CLK}} - \sum_{j=0}^{i} j\tau_j\right),\ i = 0, 1, \ldots, N_{\mathrm{d}} - 1,\ \tau_j \approx \tau\ . \qquad (9)$$

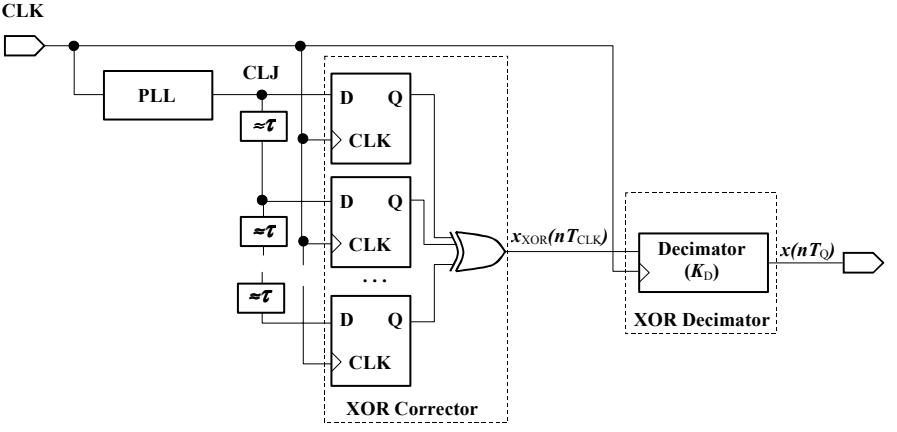


Fig. 8. Simplified block diagram of complete TRNG

To increase the probability of overlapping CLK and CLJ edge zones during the T_{Q} period, the signal CLJ is delayed in $N_{\mathrm{d}} - 1$ delay elements. Outputs of these elements are synchronously sampled with the frequency F_{CLK} and XOR-ed together to produce signal

$$x_{\mathrm{XOR}}\,(nT_{\mathrm{CLK}}) = q_0\,(nT_{\mathrm{CLK}}) \oplus q_1\,(nT_{\mathrm{CLK}}) \oplus\ \ldots\ \oplus q_{N_{\mathrm{d}}-1}\,(nT_{\mathrm{CLK}})\ . \qquad (10)$$

Output of the complete TRNG can be written in the form:

[5] This principle can be applied also to more non-overlapped bits that are XORed together.

$$x\left(nT_{\mathrm{Q}}\right) = x_{\mathrm{XOR}}\left(nT_{\mathrm{Q}}\right) \oplus x_{\mathrm{XOR}}\left(nT_{\mathrm{Q}} - T_{\mathrm{CLK}}\right) \oplus \ldots \\ \ldots \oplus x_{\mathrm{XOR}}\left(nT_{\mathrm{Q}} - \left(K_{\mathrm{D}} - 1\right) T_{\mathrm{CLK}}\right) \ . \tag{11}$$

The minimal signal delay obtainable in Altera APEX family is $\tau \approx 500$ ps and actual values $\tau_i,\ i = 0, 1, \ldots, N_{\mathrm{d}} - 1$ fluctuate around this value and are influenced by the supply voltage and the temperature. This mechanism causes fluctuation of biases b_i of individual outputs of the delay line (since corresponding values $\Delta T^i_{\mathrm{min}},\ i = 0, 1, \ldots, N_{\mathrm{d}} - 1$ influence b_i according to (8)). In order to decrease the output bias it is necessary to use a N_{d} which is as large as possible. We propose to use about N_{d} delay elements, where the maximal value of N_{d} is limited by

$$N_{\mathrm{d}} \leq T_{\mathrm{CLK}}/\tau = 1/(F_{\mathrm{CLK}}\tau) \ . \tag{12}$$

The sum of the delays thus spans one period T_{CLK} and ensures (for $T_{\mathrm{CLK}} < T_{\mathrm{CLJ}}$) that if the edge of the CLJ signal is in the current T_{CLK} window, the edge zone is sampled only once with the probability $\sigma_{\mathrm{jit}}/\tau$. Larger values of N_{d} are not recommended, since sampling one edge of CLJ signal two or more times can create an undesired statistical dependency. Therefore at each output-sampling interval nT_{Q}, the signal $x\left(nT_{\mathrm{Q}}\right)$ is the result of XOR-ing

$$N_{\mathrm{XOR}} = K_{\mathrm{D}} N_{\mathrm{d}} \tag{13}$$

individual bits. There are $2K_{\mathrm{M}}$ edges of CLJ signal over T_{Q} period, so approximately N_{bit} bits, N_{bit} being calculated by

$$N_{\mathrm{bit}} \approx 2K_{\mathrm{M}}\sigma_{\mathrm{jit}}/\tau, \tag{14}$$

are influenced by the intrinsic jitter and these bits are used by XOR corrector for a bias reduction. Although value (14) is just statistical estimation, it provides information about the applicability of some statistical rules.

If the input bits were statistically independent, the decimated output sequence $x\left(nT_{\mathrm{Q}}\right)$ would quickly converge to an unbiased binary sequence that is uncorrelated. Since the binary stream $x_{\mathrm{XOR}}\left(nT_{\mathrm{CLK}}\right)$ is influenced by an analog part of the PLL, we can expect that obtained values will be statistically independent. This hypothesis is tested in Sect. 5.

4 Experimental Hardware Implementation

To measure the real performance of our proposed TRNG, an Altera NIOS development board was selected. This development board was chosen to eliminate concerns about proper board layout technique. The same board was also used in [7] for the reference PLL measurements so we can expect that jitter characteristics presented in Sect. 2 can be directly applied to our design. The board features a PLL-capable APEX EP20K200-2X with four on-chip analog PLLs. In order to use as large output data rate as possible, the two[6]

[6] It is possible to create TRNG based only on one PLL, but it requires a different crystal than the NIOS board actualy uses.

on-chip PLLs shown in Fig. 9 were used for generating CLJ and CLK signals. The external clock source was 33.3 MHz, on-chip synthesized clocks were $F_{\mathrm{CLK}} = 33.3 \times 159/60 = 88.245$ MHz and $F_{\mathrm{CLJ}} = F_{\mathrm{CLK}}\,(785/1272) \approx 54.459$ MHz, so $K_{\mathrm{M}} = 785$ and $K_{\mathrm{D}} = 1272$. These values were chosen as a compromise of minimal $\mathrm{MAX}\,(\Delta T_{\mathrm{min}})$ for actual NIOS board constraints. According to (4) they ensure that $\mathrm{MAX}\,(\Delta T_{\mathrm{min}}) \approx 7.2$ ps $< \sigma_{\mathrm{jit}}$. The TRNG was implemented for $N_{\mathrm{d}} = 22$ in VHDL using standard Altera megafunction for embedded PLL configuration.

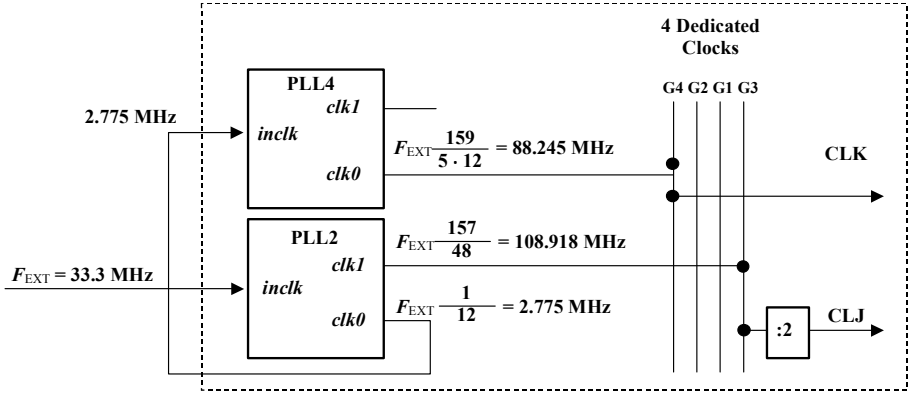


Fig. 9. Actual PLL configuration used in experimental hardware

There are two problems related to the random number generator implementation in an FPLD:

- the function of the generator cannot be verified using simulation (jitter is not simulated),
- since detection of the jitter is based on a repetition of a small signal delay using a carry chain, placement and routing has a significant impact on the generator operation (for example, to guarantee the correct operation of the generator, D flip-flops have to be implemented in the same logic array block as the carry chain delay).

For a proper operation the design must use resource locking (assignments) and the design must be verified and tested on a real hardware. Generator blocks have been designed using both Altera Hardware Description Language (AHDL) and VHDL. Since its implementation is hardware-specific, it seems to be more practical to use AHDL instead of VHDL (at least for the jitter detector block), because AHDL is closer to the hardware and the implementation can be better controlled on a low level basis (assignments of hardware elements).

The FPLD resource requirements of the proposed TRNG block as well as the supporting logic (FIFO, control logic) of the experimental hardware implementation is shown in Table 1. The first four columns show resource requirements

(in Logic Cells and Embedded System Blocks (ESB)) of the generator, as it is presented in Fig. 8. The second four columns give resource requirements of the complete TRNG circuit including 8 bits wide 1024-byte FIFO and a data bus controller. Presented results have been obtained using Altera Quartus II v. 2.0 [9]. Values $x\left(nT_{\mathrm{Q}}\right)$ generated with bit-rate $1/T_{\mathrm{Q}} \approx$ 69375 bits/s were saved on the hard disk for further analysis.

Table 1. APEX FPLD resource requirements

Device	TRNG only				TRNG + FIFO			
	LCs #	LCs %	ESBs #	ESBs %	LCs #	LCs %	ESBs #	ESBs %
EP20K200EFC484-2X	48	0.6	0	0	121	1.5	4	7.7

5 Statistical Evaluation of TRNG

Testing a hardware random number generator differs from testing a PRNG [12]. In particular, if we know the design of the generator we can tailor some of the tests. However, the random number generator (either random or pseudorandom) might pass the test and still not be a good generator. There are some well documented general statistical tests that can be used to look for deviations from an ideal TRNG [13], [14], [15]. A good TRNG should pass all kinds of tests.

5.1 Testing of Basic Statistical Assumption

A potential problem of using XOR decimation technique for bias removing is that XOR decimation should be used only with statistically independent bits. Our XOR corrector performs XOR-ing of $N_{\mathrm{XOR}} = 1272 * 22 = 27984$ input bits. According to (14) there are about $N_{\mathrm{bit}} \approx 47$ input bits per one output bit that are influenced by a non-deterministic jitter. Under ideal assumption (statistically independent biased jitter values) the output signal must converge to an almost unbiased binary sequence ($B \to 0$) with probability of 1's and 0's equal to $1/2 \pm B$, where the total bias B can be computed as

$$B = E\left[x\left(nT_{\mathrm{Q}}\right)\right] = E\left[f\left(q_0\left(nT_{\mathrm{Q}}\right), \ldots q_{N_{\mathrm{d}}-1}\left(nT_{\mathrm{Q}}\right), b_0, b_1 \ldots b_{N_{\mathrm{d}}-1}\right)\right] + \\ + (-2)^{N_{\mathrm{d}}-1} \prod_{i=0}^{N_{\mathrm{d}}-1} b_i \,. \tag{15}$$

For statistically independent values the first term of (15) is zero and the second term of (15) very quickly converges to a low value since $|b|_i < 1/2$, $i = 0, 1, \ldots, N_{\mathrm{d}} - 1$. Table 2 shows the results of the mean value computation for several 1-Gigabit TRNG output records acquired from two available NIOS boards.

It is clear that there is a certain small difference from an ideal TRNG. This difference is caused by a certain small non-zero statistic dependency in the first term of (15). This is the first[7] detected difference between our TRNG and ideal one.

Table 2. Mean values computed for several 1-Gigabit records

Record	1	2	3	4	5
NIOS	(Board A)	(Board B)	(Board B)	(Board B)	(Board B)
Mean	0.500109	0.499917	0.499911	0.499896	0.499872

5.2 The NIST Statistical Tests

A large number of generalized statistical tests for randomness have been proposed. It seems that the NIST statistical test suite [15] is currently the most comprehensive tool publicly available. Our NIST statistical tests were performed on 1-Gigabit of continuous TRNG output records and followed the testing strategy, general recommendations and result interpretation described in [15]. We have used a set of $m = 1024$ 1-Megabit sequences produced by the generator and we have evaluated the set of P-values (some typical values are shown in Table 3 [6]) at a significance level $\alpha = 0.01$. The total number of acceptable sequences was within the expected confidence intervals [15] for all performed tests and P-values were uniformly distributed over the $(0, 1)$ interval.

We have performed the same tests for several 1-Gigabit records and have uncovered certain deviations in the FFT statistical test results. For ideal TRNG the distribution of P-values is uniform in the interval $(0, 1)$. For tested TRNG this uniformity is checked by using a χ^2 test distribution of P-values in subintervals C1-C10. If the P-value shown in Table 4 (more precisely a P-value of the P-values [15]) is lower than 0.0001 the test fails and indicates a detectable difference from the ideal TRNG.

6 Conclusions

In this paper we have evaluated a new method of true random numbers generated in SOPC based on a reconfigurable hardware. The randomness of the sequence of numbers has been extensively tested and only small differences from an ideal TRNG have been detected. We believe that intrinsic analog PLL noise is a good source of true randomness and at least for typical cryptographic keys with the length from hundreds to several thousands bits, our TRNG is not distinguishable from the ideal TRNG. For very critical cryptographic applications the proposed

[7] Note that this difference is really detectable only for long streams and we believe that proposed TRNG can be used for key generation in typical cryptographic applications.

Table 3. NIST test results (uniformity of P-values and proportion of passing sequence) for 1-Gigabit record that passed all tests

C1	C2	C3	C4	C5	C6	C7	C8	C9	C10	P-value	Proportion	Statistical Test
112	103	114	95	98	105	91	95	104	107	0.827418	0.9873	Frequency
111	103	103	91	104	110	101	108	93	100	0.920212	0.9922	Block-Freq.
103	113	89	100	92	104	107	97	92	127	0.242375	0.9873	Cusum
97	81	97	117	114	91	93	106	115	113	0.144842	0.9941	Runs
86	108	102	92	93	94	122	99	125	103	0.106543	0.9922	Long-Run
99	92	116	110	90	115	103	93	104	102	0.582174	0.9902	Rank
83	110	116	110	112	108	120	87	79	99	0.027813	0.9951	FFT
117	107	90	95	108	98	102	99	105	103	0.830876	0.9824	Periodic-Template
130	95	111	112	99	91	97	92	111	86	0.072399	0.9863	Universal
91	114	118	102	85	94	108	96	112	104	0.327204	0.9951	Apen
95	107	105	126	99	94	94	96	104	104	0.510619	0.9932	Serial
110	90	104	127	94	96	78	107	114	104	0.056616	0.9863	Lempel-Ziv
105	108	96	96	103	114	106	87	108	101	0.807953	0.9893	Linear-Complexity

Table 4. NIST FFT test results (uniformity of P-values and proportion of passing sequence) for all tested 1-Gigabit records

#	C1	C2	C3	C4	C5	C6	C7	C8	C9	C10	P-value	Proportion	Statistical Test
1	83	110	116	110	112	108	120	87	79	99	0.027813	0.9951	FFT
2	105	136	100	113	111	99	101	79	88	92	0.010138	0.9932	FFT
3	**96**	**113**	**125**	**143**	**96**	**96**	**118**	**86**	**82**	**69**	**0.000002**	***0.9951**	**FFT**
4	**107**	**132**	**133**	**110**	**117**	**95**	**71**	**93**	**86**	**80**	**0.000010**	***0.9971**	**FFT**
5	91	132	115	128	101	93	99	109	78	78	0.000301	0.9941	FFT

TRNG can be used at least as an useful internal source of entropy or efficiently combined with one-way hash functions or PRNGs.

The proposed solution is very cheap. It uses very small amounts of FPLD resources and it is fast enough for typical embedded cryptographic applications. The advantage of our solution lies in the fact that the proposed TRNG block together with symmetrical and asymmetrical algorithms can fit into one FPLD chip and significantly increase the system security of an embedded cryptographic SOPC system.

The bias reduction of the TRNG can be further improved by a proper choice of parameters K_{M} and K_{D} and using more sophisticated XOR corrector. This solution is currently in development and will be presented in a future paper.

References

1. Menezes, J.A., Oorschot, P.C., Vanstone, S.A.: Handbook of Applied Cryptography, CRC Press, New York (1997)
2. Faifield, R.C., Mortenson, R.L., Coulthart, K.B.: An LSI Random Number Generator (RNG). Lecture Notes in Computer Science, Vol. 0196. Springer-Verlag, Berlin Heidelberg New York (1984) 203–230
3. Jun, B., Kocher, P.: The INTEL Random Number Generator. Cryptography Research, Inc., White Paper prepared for Intel Corporation, April 1999, 1–8, http://www.intel.com
4. APEX 20K Programmable Logic Device Family. Data Sheet, February 2002, ver. 4.3, 1–116, http://www.altera.com
5. APEX II Programmable Logic Device Family. Data Sheet, December 2001, ver. 1.3, 1-96, http://www.altera.com
6. Fischer, V., Drutarovský, M.: True Random Number Generator in Field Programmable Logic Devices. Submitted to Electronic Letters, Paper Number ELL 32365, April 2002
7. Superior Jitter management with DLLs. Virtex Tech Topic VTT013 (v1.2), January 21, 2002, 1–6, http://www.xilinx.com
8. Jitter comparison analysis: APEX 20KE PLL vs. Virtex-E DLL. Technical Brief 70, January 2001, ver.1.1, 1–7, http://www.altera.com
9. Quartus II - Programmable Logic Design Software. January 2002, ver.2.0, 1–45, http://www.altera.com
10. Davies, R.B.: Exclusive OR (XOR) and Hardware Random Number Generators. February 28, 2002, 1–11, http://webnz.com/robert/
11. Eastlake, D., Crocker, S., Schiller, J.: Randomness Recommendations for Security. Request for Comments 1750, December 1994, http://www.ietf.org/rfc/rfc1750.txt
12. Davies, R.: Hardware Random Number Generators. Paper presented to the 15th Australian Statistics Conference, July 2000, 1–13, http://statsresearch.co.nz
13. Marsaglia, G.: DIEHARD: A Battery of Tests of Randomness. http://stat.fsu.edu/ geo/diehard.html
14. Security Requirements for Cryptographic Modules. Federal Information Processing Standards Publication 140-2, U.S. Department of Commerce/NIST, 1999, http://www.nist.gov
15. Rukhin, A., Soto, J., Nechvatal, J., Smid, M., Barker, E., Leigh, S., Levenson, M., Vangel, M., Banks, D., Heckert, A., Dray, J., Vo, S.: A Statistical Test Suite for Random and Pseudorandom Number Generators for Cryptographic Applications. NIST Special Publication 800-22, May 15, 2001, 1–153, http://www.nist.gov

Evaluation Criteria for True (Physical) Random Number Generators Used in Cryptographic Applications

Werner Schindler[1] and Wolfgang Killmann[2]

[1] Bundesamt für Sicherheit in der Informationstechnik (BSI)
Godesberger Allee 185–189
53175 Bonn, Germany
Werner.Schindler@bsi.bund.de
[2] T-Systems ISS GmbH
Rabinstr. 8
53111 Bonn, Germany
Wolfgang.Killmann@t-systems.com

Abstract. Random number generators are essential components of many cryptographic systems. Inappropriate random number generators may weaken the security properties of the system considerably. This paper considers evaluation criteria for true (physical) random number generators. General objectives are formulated and possible criteria and measures are discussed which shall ensure these goals. Central parts of the mathematical-technical reference of the German evaluation guidance document AIS 31 ([19,2]) are cited and rationale is given.

Keywords: True random number generator, entropy, evaluation criteria, online test.

1 Introduction

Many security mechanisms need nonrecurring and / or unpredictable data as nonces or secrets. While the non-recurrence property is fulfilled for nearly all random number generators (RNGs) with overwhelming probability (provided that the length of the (pseudo-)random strings is sufficiently large) unpredictability is more difficult to assure. An RNG with this property which can neither be observed nor controlled by an attacker generates ideal secrets like cryptographic keys. Random numbers are needed by many cryptographic applications. They are used for the generation of random session keys, signature keys and signature parameters, challenges and zero knowledge proofs, for instance.

Ideal random number generators are characterized by the property that the generated random numbers are independent and uniformly distributed on a finite range. An ideal random number generator, however, is a fiction. The class of (real world) RNGs falls into three subclasses. First, random microcosmic processes may cause physical effects applicable as random noise sources. Examples

B.S. Kaliski Jr. et al. (Eds.): CHES 2002, LNCS 2523, pp. 431–449, 2003.

are quantum random processes, time between emissions during radioactive decay, inherent semiconductor thermal noise, shot noise from Zener diodes or free-running oscillators. These processes are chaotic and non-deterministic by their nature. Normally, the random noise source produces a time-continuous analog signal which is digitised after uniform time intervals (e.g. by a comparator) which yield the so-called *digitized analog signals*, briefly denoted as *das random numbers*. Such RNGs are called true or physical (TRNG) because of their random noise source. In many designs the das random numbers are algorithmically post-processed in order to reduce or at least to mask potential weaknesses. This gives the so-called *internal random numbers*. Upon external call the TRNG outputs *external random numbers*. TRNG are implemented e.g. in smart cards.

Pseudorandom number generators (or synonymously: *deterministic random number generators, DRNGs*) form the second subclass. They generate pseudo-random numbers deterministically from a randomly chosen seed. The third subclass are the *'hybrid generators'* which refresh their seed regularly, for instance by means of random numbers derived from user's interaction (mouse movement or key strokes) and / or register values of the used PC. Applying an appropriate transformation (usually a compression algorithm) yields the desired random numbers.

In the past a lot of research work has been devoted to the development of good physical noise sources (see [4,21,14], for instance), and a variety of deterministic and hybrid random number generators have been proposed. Less work has been spent in the development of suitable tests and assessment criteria.

Random numbers are not only needed for cryptographic applications but also for Monte Carlo methods and, above all, in stochastic simulations. Stochastic simulations treat probabilistic problems which cannot be solved analytically because they are too complex (consider the service times in complex multiuser systems, for example). Roughly speaking, one generates pseudorandom numbers ([12,20]) with statistical properties depending on the specific simulation problem. Repeating this process many times one hopes to get reliable estimators for the unknown solution of the underlying probabilistic problem (typically a distribution of a random variable or a random vector or any restriction of it). A large number of statistical tests and whole test suites have been proposed to assess the statistical properties of the generated pseudorandom numbers ([20,22]) as unsuitable pseudorandom number generators may suggest false conclusions. These test suites are also often applied to random numbers used for sensitive cryptographic applications. However, besides statistical properties sensitive cryptographic applications demand that the used random numbers should have a backward and forward unpredictability property (cf. Sect. 2). As will be explained below this property cannot be assured by applying statistical blackbox tests. However, also [24,25,27] provide blackbox test suites. (Note that in [27] support is given how to apply the statistical tests, and in Sect. 1.1.2 it is noted explicitly that random numbers should be unpredictable. However, similarly as in [13], for instance, clear criteria are missing which assure this goal.) In [19] an approach for the evaluation of physical random number generators (due to CC (Common Crite-

ria; cf. [8]) or ITSEC (Information Technology Security Evaluation Criteria; cf. [15])) is given which takes the construction of the TRNG explicitly into account. [19] is the mathematical-technical reference of the AIS 31 ([2]) which has been effective in the German evaluation and certification scheme since September 2001. In the present paper we will explain the main items of [19] and the central ideas behind them. We point out that there is a similar document for deterministic random number generators ([1,29]; cf. Section 2 for a brief discussion).

In Sect. 2 the fundamental differences between true, deterministic and hybrid generators are explained. In Sect. 3 the general objectives of a TRNG evaluation are briefly pointed out whereas Sects. 4, 5 and 6 go into detail. Generic requirements are formulated and concrete measures and criteria from [19] are given which shall ensure these requirements. The paper ends with considerations concerning the vulnerablity analysis and final remarks.

2 True, Deterministic, and Hybrid Random Number Generators: Main Differences

RNGs generally consist of a random source and a deterministic postprocessing. Of course, if a TRNG has an appropriate noise source a deterministic postprocessing after the digitization of the analog signal is not necessary. Design and analysis of an RNG is based on the understanding of randomness.

We already mentioned physical processes as random sources. Another kind of randomness is based of the intersection of causal chains: the combination of events of several processes which are independent of each other may behave randomly. Consider, for example, system data as time of interrupts, hard-disk seek times or user interactions. The randomness of the combined random source will increase with the number of the processes, their 'degree' of independence and the randomness within each of the processes. These processes are part of the RNG and cannot be neglected in its analysis even if the RNG is mostly implemented in software. An example of such a software RNG can be found in OpenSSL implementations under Windows operating systems.

The postprocessing may transform the digitized random signals into uniformly distributed random numbers even if the initial signal has significant statistical defects. Moreover, the postprocessing may collect entropy of the random noise source but it does not stretch the digitized random signal into longer strings.

Deterministic random number generators rely on a completely different concept. Viewed as an autonomous automaton a DRNG has an internal state whose initial value is called the *seed*, a state transition function and a output function. Such automatons are very cheap to implement as they merely require some additional lines of code. Their drawback lies in the fact that the seed (generated by a true or hybrid generator) contains the overall entropy of all pseudorandom numbers which are generated until a new seed is chosen. Therefore the evaluation methodology [29], for instance, requires a clear description of how the seed is generated together with a rationale why this will induce a specified distribution.

Depending on the applications the DRNG is designated for this methodology distinguishes four DRNG classes K1 to K4. Class K1 generators only have to produce mutually different vectors with high probability whereas their statistical properties are irrelevant. Such vectors might be used for challenge-response protocols, for instance. The higher classes extend the possible applications of the random numbers to the generation of key or signature parameters. The requirements increase from mutually different outputs, through statistical features of the output (class K2), minimum bounds for the entropy of the seed and the practical impossibility for an adversary to work out preceding and following random numbers of a known part of the output sequence (class K3). Class K4 additionally demands that an adversary shall not be able to determine preceding random numbers even if he knows the actual internal state of the DRNG. An example of a K4-DRNG is the NIST approved DRNG for the DSA (see [26], Annex 3 and Change Notice 1) without the optional user input (provided that the seed is generated in an appropriate manner).

While the overall entropy of the output sequence of a true RNG increases with each random number, the entropy of the output of a DRNG is limited by the entropy of the seed initializing the internal state. Hybrid RNGs try to compensate the drawback of DRNGs by regular re-seeding the internal state by a random source (like user's interaction, register values of a PC or, more favourably, by a TRNG) and stretching the internal state like a DRNG. We note in this context that the randomness induced by the interaction of a user or by register values of a PC is difficult to assess. For example LINUX implements a hybrid RNG with a combined random source as /dev/urandom. The ANSI X9.17 RNG [3] refreshes the seed by means of the time which has only little entropy and might be guessed. Interestingly, the RNG [26], Annex 3, may be weak in hybrid mode, namely if an attacker is able to control the optional user input (see [18] for details).

It is important to note that a finite set of statistical tests may detect defects of a random source. On the other hand these tests cannot verify the randomness of the source. Moreover, the needs of IT security and especially of cryptography are different from those of stochastic simulations.

Remark 1. The Kolmogorov complexity theory defines a sequence as random if the length of the shortest program that generates a segment of the sequence arises with the length of this segment. A cryptographic strong definition of randomness requires pseudo-random sequences to be computationally indistinguishable from a randomly chosen uniformly distributed sequence (cf. [7]).

In the context of cryptography we may be faced with an active attacker who does not only analyze the RNG design and its output but additionally tries to affect or control the RNG. We hence should not only have a theoretical abstraction of an RNG in mind but also its concrete realization.

3 General Objectives on a TRNG Evaluation

Normally, random number generators are part of an IT security product whose overall security (or particular aspects thereof) has to be assessed. The random number generation may be an externally visible security function, e.g. provided by a smart card chip to the operating system for the key generation. However, requirements on the TRNG often depend on the internal function they provide for other cryptographic components.

Roughly speaking, a TRNG evaluation falls into two phases. At first, the suitability of the random number generation process has to be checked at hand of some prototypes. In Sects. 4 and 5 this phase is considered in detail.

However, even if the design of the TRNG has turned out to be convenient this does not necessarily imply that each TRNG of the same production run has the same pleasant properties as the prototypes in the lab all the time when it is in operation. Tolerances of components of the random noise source maybe responsible that the 'quality' of the actually produced random numbers is worse than that of the carefully investigated prototypes. Also aging of the components or even the impact of an attacker may affect the characteristics of the generated das random numbers. In a worst case scenario the noise source may totally break down, maybe for natural reasons or as a consequence of a successful attack. The das random numbers are constant from then on, and the total entropy of the das random number sequence does not increase any more.

In order to detect such defects TRNGs should perform start-up tests, online test and so-called tot tests while they are in operation (cf. Sect. 6). The evaluation should give evidence that these measures are appropriate to assure this goal.

ITSEC (Information Technology Security Evaluation Criteria) and CC (Common Criteria; cf. [8]) provide evaluation criteria. ITSEC and CC *'will permit comparability between the results of independent security evaluations. It does so by providing a common set of requirements for the security functions of IT products and systems and for assurance measures applied to them during a security evaluation. The evaluation process establishes a level of confidence that the security functions of such products and systems and the assurance measures applied to them meet these requirements.'* ([8], Part 1, Scope). A product or system which has been successfully evaluated is awarded with an internationally recognized IT security certificate.

Although random numbers play an important role in numerous cryptographic applications, ITSEC, Common Criteria ([8]) and the corresponding evaluation manuals do not specify any uniform evaluation criteria for random number generators. However, rules are needed for the evaluation of true (physical) random number generators. The AIS 31 ([2,19]) supports this goal. It is mandatory in the German evaluation and certification scheme if the TRNG affects the security properties of the target of evaluation (TOE).

In the following sections we will work out the general objectives formulated above. We will often refer to and cite important parts of [19] where concrete criteria, statistical tests and decision rules are given that a TRNG should fulfil.

4 Assessing the Random Number Generation I (Standard Case)

The minimal requirements the external random numbers should at least fulfil depend essentially on the particular application. In general the application reads the external random numbers from the RNG asynchronous to the generation of the internal random numbers. Therefore a TRNG evaluation can only consider the properties of the das random numbers or the internal numbers. However, as the external random numbers usually are obtained by concatenating internal random numbers this is no serious restriction.

If the external random numbers serve as challenges or as openly transmitted initialization vectors for symmetric block ciphers, for instance, it is fully sufficient if the statistical behaviour of the internal random numbers is similar to that of random numbers generated by an ideal source. Generically, in [19] (P1.c) this reads as follows: *'The statistical behaviour of the internal random numbers should be inconspicious. This shall prevent replay and correlation attacks against cryptographic algorithms and protocols that are based on statistical weaknesses in the external random numbers used.'* More concrete, a TRNG is said to be a P1-TRNG in the sense of [2] if its internal random numbers pass a particular statistical test suite (cf. [19], Requirements P1.d(i),(ii) and P1.i(i),(ii)).

Good statistical properties of the internal random numbers are clearly not sufficient for sensitive applications as the generation of session keys, signature key pairs or signature parameters, for example. Note that even output sequences of linear feedback shift registers (LFSR) should pass the P1-specific statistical tests (and many others, cf. Example 1) unless the length m of the LFSR is extremely small. The overall entropy of a pseudorandom number sequence generated by a LFSR is contained in its seed. If a potential attacker knows about m output bits (e.g. from random numbers used as an openly transmitted challenge) he can easily compute all random numbers ever generated. All he has to do is to solve a system of linear equations.

The key criterion is not the statistical behavior of the internal numbers but their entropy. Consequently, in [19] a subclass of the P1-TRNGs is introduced, called P2. For a P2-TRNG evidence has to be given that the increase of entropy per internal random number is sufficiently large. In [19], P2.c, the aim of the P2-specific requirements (cf. [19], P2.d)) is formulated generically: *'In addition to the P1-specific aim P1.c), the prospects of success for systematic guessing of the external random numbers (realised through sytematic exhaustion attacks) - even if external random number subsequences are known - should at best be negligibly higher than would be the case if the external random numbers had been generated by an ideal random number generator.'*

To assure this goal the increase of entropy per internal random number must be sufficiently large. Unlike the computational complexity for deterministic random number generators (cf. Sect. 2) entropy yields a theoretical security bound. However, entropy is a property of random variables but not of their realizations, in our context, of the observed internal random numbers. Unfortunately, there is no statistical test or reliable estimator (and it is hardly to believe that there

might exist one) for the case where nothing is known about the distribution of the underlying random variables.

Remark 2. The adjective 'universal' in the title of [23] has led to misunderstanding and confusion in the past. To apply Maurer's test the random numbers are concatenated and interpreted as a bit stream which in turn is segmented into blocks of equal size. If the block size tends to infinity Maurer's test value yields an estimator for the increase of entropy per random bit *provided that the bits were generated by a stationary binary random source with finite memory* (cf. also [11]). If this assumption is not fulfilled, however, Maurer's test value need not yield a reliable estimator for the entropy. For pseudorandom bits generated by a LFSR, for example, the increase of entropy per bit obviously equals zero whereas the test value 'suggests' a considerable amount of entropy per bit. We point out that [19] indeed applies a test introduced by Coron (cf. [10]) which is closely related with Maurer's test but yields information on the entropy per bit for *fixed* block size. As will be explained below Coron's test is applied to the das random numbers not until the respective random variables have shown to be (at least approximately) Markovian.

Definition 1. *A* realization *of a random variable X is a value assumed by X. We call a random variable binary if it only assumes the values* 0 *and* 1*. If X assumes values on a finite set Ω then*

$$H(X) := -\sum_{x\in\Omega} \mathrm{Prob(X = x)} \log_2(\mathrm{Prob(X = x)})$$

is the entropy of X. In the context of random variables iid *stands for 'independent and identically distributed'. Applying a statistical test to a sample $x_1, x_2, \ldots, x_N$ delivers a numerical value called* test value *or* test statistic *t. The test value itself may be interpreted as a realization of a random variable T, the so-called test variable.*

Mathematical Model. In the following we interpret the das random numbers $b_1, b_2, \ldots \in \Omega_{\mathrm{das}}$ (usually $\Omega_{\mathrm{das}} = \{0,1\}^k$ for $k \geq 1$) as realizations of random variables $B_1, B_2, \ldots$. Similarly, the internal random numbers $r_1, r_2, \ldots$ are viewed as realizations of random variables $R_1, R_2, \ldots$. The random variables $R_1, R_2, \ldots$ result from $B_1, B_2, \ldots$ by applying the postprocessing algorithm.

Remark 3. The term

$$H(B_n \mid B_1, \ldots, B_{n-1}) := -\sum_{b_1,\ldots,b_{n-1}\in\Omega_{\mathrm{das}}} \mathrm{Prob(B_1 = b_1, \ldots, B_{n-1} = b_{n-1})} \quad \times$$

$$\sum_{b_n\in\Omega_{\mathrm{das}}} \mathrm{Prob(B_n = b_n \mid B_j = b_j \,for\, j < n)} \log_2(\mathrm{Prob(B_n = b_n \mid B_j = b_j \,for\, j < n)})$$

quantifies the increase of the total entropy of $B_1, \ldots, B_{n-1}$ by B_n. If the random variables $B_1, B_2, \ldots$ are iid the conditional probabilities do not depend on

the conditions. If $B_1, B_2, \ldots$ form a homogeneous Markov chain the conditional probabilities do only depend on the preceding value b_{n-1}, and for sufficiently large n we have $\mathrm{Prob}(\mathrm{B_{n-1}} = \mathrm{b_{n-1}}) \approx \nu(\mathrm{b_{n-1}})$ where ν stands for the stationary distribution (under the natural assumption that the Markov chain is ergodic). If the context is clear, i.e. if we consider the whole sequence, we also use the imprecise formulation 'the entropy of B_n'. By abuse of language we will often speak of 'the increase of entropy per das random number' or shortly 'the entropy of a das random number'.

A principle question is whether the das random numbers or the internal random numbers should be considered for testing. The latter seems to be near at hand since the internal random numbers are output. Example 1 (cf. [30], Example 1, or [19], Example E.1), however, underlines that this approach may be misleading.

Example 1. Suppose that the TRNG produces binary das random numbers and let a LFSR of length 63 with primitive feedback polynomial be synchronized with the digitization of the analog noise signal. In each time step the feedback shift register outputs an internal random number (a single bit). The actually generated das random number is XOR-ed to the feedback value, and this sum is fed back into the LFSR. This mathematical postprocessing is a one-to-one mapping for each initial value of the LFSR and hence cannot increase the average entropy per bit. Consequently, weaknesses of the das random numbers cannot be reduced but only transferred into others. If, for example, the das random numbers are independent but not equidistributed (i.e., if the probability for "0" is not 0.5) the internal random numbers are basically equidistributed but dependent. Unless its linear complexity profile is considered statistical tests applied to the internal random number sequence will presumably not even detect the worst case when the physical noise source has totally broken down. In fact, from this moment on the das random numbers are constant, and the internal random numbers are generated deterministically.

Example 1 underlines an important fact: Even if the internal random numbers pass certain statistical tests which the das random numbers do not this does not necessarily imply that the mathematical postprocessing improves the quality of the das random numbers. Weaknesses may be merely masked and transformed into others. Clearly, an increase of entropy per bit can only be achieved by a data compression which in turn lowers the bit rate.

Of course, also the das random numbers may not be equidistributed and there may exist dependencies on predecessors. However, in contrast to the internal random numbers there will not exist complicated algebraic dependencies. Consequently, the das random numbers should always be tested if this is possible. As demonstrated above internal random numbers may pass statistical tests even if their overall entropy equals zero.

The aim of a P2-evaluation is clear: Evidence shall be given that the increase of entropy per bit is sufficiently large. The crucial requirement a TRNG has to fulfil in the context of entropy is Requirement P2.d)vii): '*Digitised noise signal*

sequences (das random numbers) *meet particular criteria or pass statistical tests intended to rule out features such as multi-step dependencies. Moreover, the entropy test T8 is passed. The tests and evaluation rules are specified in subsection P2.i)...'*. For the normal case where the TRNG generates a single das bit per time unit in [19] (P2.i)vii)) five tests and an overall decision rule are specified which shall assure this goal. Test T7 represents a special case of Test 76 (a test for the equality of multinomial distributions) from [17]. In our context (comparison of two bit sequences $a_1, \ldots, a_n$ and $a'_1, \ldots, a'_n$) the test value is given by

$$t_7 := \sum_{t=0,1} \frac{(h[t] - h'[t])^2}{h[t] + h'[t]} \tag{1}$$

where $h[t] := |\{j \leq n \mid a_j = t\}|$ and $h'[t] := |\{j \leq n \mid a'_j = t\}|$. Under the null hypothesis, i.e. that the bit sequences $a_1, \ldots, a_n$ and $a'_1, \ldots, a'_n$ are realizations of iid binary-valued random variables $A_1, \ldots, A_n$ and $A'_1, \ldots, A'_n$, the test variable $T7$ is asymptotically χ^2-distributed with one degree of freedom. Note that the random variables $A_1, \ldots, A_n, A'_1, \ldots, A'_n$ need not be equidistributed on $\{0, 1\}$. Especially, for significance level $\alpha := 0.0001$ the null hypothesis is rejected if $t_7 > 15.13$.

In [19] the following test procedure is specified:

P2.i)(vii.a) *The evaluator generates a digitised noise signal (bit) sequence* $w_1, \ldots, w_{n_0}$ *with* $n_0 := 100000$. *Let* $\mu_{emp} = (\mu_{emp}(0), \mu_{emp}(1))$ *be its empirical distribution (i.e.* $\mu_{emp}(1) := \sum_{j=1}^{n_0} w_j/n_1$). *Property (vii.a) is fulfilled if* $|\mu_{emp}(1) - 0.5| < a_0 := 0.025$.

P2.i)(vii.b) *The evaluator generates a further digitised noise signal sequence* $w_1, w_2, \ldots$ *which he splits into* 2 *disjoint sub-sequences* $\mathrm{TF}_{(0)}$ *and* $\mathrm{TF}_{(1)}$. *Here, the tuple* (w_{2j+1}, w_{2j+2}) *belongs to sub-sequence* $\mathrm{TF}_{(r)}$ *if and only if* $w_{2j+1} = r$. *The initial sequence* $w_1, w_2, \ldots$ *must be sufficiently long that both sub-sequences contain at least* $n_1 := 100000$ *elements. If we project the first* n_1 2*-tuples of sub-sequence* $\mathrm{TF}_{(r)}$ *onto the second component, we obtain the one-dimensional sample* $\mathrm{St}_{(r)}$. *If we divide the frequencies at which individual values* (0 or 1) *are assumed by the size of the sample* n_1, *we obtain the empirical* 1*-step transition distribution* $\nu_{emp(r)}(\cdot)$ *for predecessor* r. *Property (vii.b) is fulfilled if* $|\nu_{emp(0)}(1) + \nu_{emp(1)}(0) - 1| < a_1 := 0.02$.

P2.i)(vii.c) *The evaluator generates a further digitised noise signal sequence* $w_1, w_2, \ldots$ *which he splits into* $2^2 = 4$ *disjoint sub-sequences* $\mathrm{TF}_{((0)-(0))}, \ldots, \mathrm{TF}_{((1)-(1))}$. *Here, the triple* $(w_{3j+1}, w_{3j+2}, w_{3j+3})$ *belongs to sub-sequence* $\mathrm{TF}_{((r)-(s))}$ *if and only if* $(w_{3j+1}, w_{3j+2}) = (r, s)$. *The initial sequence* $w_1, w_2, \ldots$ *must be sufficiently long that each of these four sub-sequences contains at least* $n_2 := 100000$ *elements. If we project each of the first* n_2 3*-tuples of sub-sequence* $\mathrm{TF}_{((r)-(s))}$ *onto the third component, we obtain the one-dimensional sample* $\mathrm{St}_{((r)-(s))}$. *For each* $s \in \{0, 1\}$ *the evaluator compares the underlying distributions of the two samples* $\mathrm{St}_{((0)-(s))}$ *and* $\mathrm{St}_{((1)-(s))}$ *with test T7* (cf. (1)) *at the significance level* $a_2 := 0.0001$ *for equality. Property (vii.c) is fulfilled if both tests are passed. Otherwise Property (vii.c) is considered not to be fulfilled.*

P2.i)(vii.d) *The evaluator generates a further digitised noise signal sequence* $w_1, w_2, \ldots$ *which he splits into* 8 *disjoint sub-sequences* $\mathrm{TF}_{((0)-(0)-(0))}, \ldots, \mathrm{TF}_{((1)-(1)-(1))}$. *Here, the quadruple* $(w_{4j+1}, w_{4j+2}, w_{4j+3}, w_{4j+4})$ *belongs to sub-sequence* $\mathrm{TF}_{((r)-(s)-(t))}$ *if and only if* $(w_{4j+1}, w_{4j+2}, w_{4j+3}) = (r, s, t)$. *The initial sequence* $w_1, w_2, \ldots$ *must be sufficiently long that each of these eight sub-sequences contains at least* $n_3 := 100000$ *elements. If we project each of the first* n_3 *quadruples of sub-sequence* $\mathrm{TF}_{((r)-(s)-(t))}$ *onto the fourth component, we obtain the one-dimensional sample* $\mathrm{St}_{((r)-(s)-(t))}$. *For each pair* $(s, t) \in \{0, 1\}^2$ *the evaluator compares the underlying distributions of the two samples* $\mathrm{St}_{((0)-(s)-(t))}$ *and* $\mathrm{St}_{((1)-(s)-(t))}$ *with test T7 at the significance level* $a_3 := 0.0001$ *for equality. Property (vii.d) is fulfilled if all four tests are passed. Otherwise Property (vii.d) is considered not to be fulfilled.*

P2.i)(vii.e) *The evaluator generates a further digitised noise signal sequence* $w_1, w_2, \ldots$ *and applies to it the entropy test* (test T8 in [19]; cf. [10]) *with the parameters* $L = 8$, $Q = 2560$ *and* $K = 256000$. *Property (vii.e) is fulfilled if the test variable* $f > 7.976$.

Decision rule: If properties P2.i)(vii.a) - (vii.e) are fulfilled, then Property P2.d)(vii) is considered to be fulfilled. If more than one sub-property is not fulfilled, then Property P2.d)(vii) is considered not to be fulfilled. If precisely one sub-property is not fulfilled, P2.i)(vii.a) - (vii.e) are applied to another sample. If all sub-properties P2.i)(vii.a) - (vii.e) are fulfilled upon repetition, then Property P2.d)(vii) is considered to be fulfilled. A further repetition is not allowed.

We give a brief rationale for the criteria P2.i)(vii.a) to P2.i)(vii.e) and the decision rule. Criterion P2.i)(vii.a) compares the one-dimensional distribution of the das bits with the uniform distribution on $\{0, 1\}$ where any dependencies from predecessors are not taken into account. If the sequence $B_1, B_2, \ldots$ is iid and if $\mu(1) := \mathrm{Prob}(\mathrm{B_j} = 1) \in [0.475, 0.525]$ then the entropy per das bit is > 0.998. The sample size n_0 was chosen so large that criterion P2.i)(vii.a) is met with a probability of at least $1 - 0.00078$ if $\mu(1) \in [0.48, 0.52]$. Criterion P2.i)(vii.b) considers the one-step transition probabilities. If the random variables $B_1, B_2, \ldots$ form a homogeneous Markov chain and if the exact transition probabilities $\nu_{(r)}(s) := \mathrm{Prob}(\mathrm{B_n} = \mathrm{s} \mid \mathrm{B_{n-1}} = \mathrm{r})$ meet the inequality $|\nu_{(0)}(1) + \nu_{(1)}(0) - 1| < 0.02$ the dependencies reduce the entropy at most by 0.00057 per bit (compared with an iid sequence $B'_1, B'_2, \ldots$ having the same one-dimensional marginal distribution μ). Note that $\nu_{(0)}(1) + \nu_{(1)}(0) = \nu_{(0)}(1) + \nu_{(0)}(0) = 1$ if the B_j are iid. If $|\nu_{(0)}(1) + \nu_{(1)}(0) - 1| < 0.012$ then P2.i)(vii.b) is satisfied with a probability of at least $1 - 0.00017$.

Remark 4. The das random numbers shall not have deeper than 1-step dependencies from the predecessors. If such dependencies are significant the generation rate of the das bits must be reduced. Slight one-step dependencies are tolerated as those might be caused by dead times of components (e.g. by flip-flops).

Criteria P2.i)(vii.c) and P2.i)(vii.d) shall detect (resp. exclude) possible 2- or 3-step dependencies. Test T7 compares the conditional distributions $\mathrm{Prob}(\mathrm{B_{n+2}} \mid$

$\mathrm{B_n} = 0, \mathrm{B_{n+1}} = \mathrm{s})$ and $\mathrm{Prob}(\mathrm{B_{n+2}} \mid \mathrm{B_n} = 1, \mathrm{B_{n+1}} = \mathrm{s})$ and $\mathrm{Prob}(\mathrm{B_{n+3}} \mid \mathrm{B_n} = 0, \mathrm{B_{n+1}} = \mathrm{s}, \mathrm{B_{n+2}} = \mathrm{t})$ and $\mathrm{Prob}(\mathrm{B_{n+3}} \mid \mathrm{B_n} = 1, \mathrm{B_{n+1}} = \mathrm{s}, \mathrm{B_{n+2}} = \mathrm{t})$, resp., where $r, s \in \{0, 1\}$. If the random variables $B_1, B_2, \ldots$ form a homogeneous Markov chain the respective distributions are equal.

The applicant for an IT certificate (typically the producer of the TRNG) has to *'give clear description how the noise signal is generated, together with an explanation of why a digitised noise signal is to be induced in this way'* ([19], C.1(iii)). This description shall enable to detect or at least suspect (or, in the positive case, rule out) long-term dependencies of the das random numbers (e.g. caused by a beat). Detecting long-term dependencies by using any blackbox test suite seems to be almost impossible. Similarly, the stationarity assumption shall be made plausible in this way (eventually supported by adapted tests). The Criteria P2.i)(vii.b) to P2.i)(vii.d) shall detect possible short-term dependencies from predecessors. If these tests are passed this is viewed as an evidence that the sequence $B_1, B_2, \ldots$ is (at least approximately) a stationary Markov chain. (The stationarity assumption shall follow from the explanations mentioned at the beginning of this paragraph.) In particular, this induces the entropy bounds from above for the das random numbers, and the assumptions of Coron's entropy test are fulfilled. To be precise, if $B_1, B_2, \ldots$ were iid the expectation of Coron's test value equals the entropy per L-bit das block.

As the evaluation result is based on statistical tests it cannot be reproduced with certainty. However, it is 'quasireproducible' in the sense that for 'reasonable' TRNGs it is extremely unlikely that Requirement P2.d)vii) is considered not to be fulfilled. The probability for an ideal random number generator not to meet the particular criteria is 0, 0, $2 \cdot 10^{-4}$, $2 \cdot 10^{-4}$, and 0, resp. The probability that an ideal random number generator is not being attributed Property P2.d)(vii) is about $6 \cdot 10^{-7}$ (cf. P2.i)(vii.e)).

Remark 5. Clearly, the mathematical postprocessing must not reduce the average entropy per bit (Property P2.d)(viii)). (The adjective 'average' is due to the fact that the particular bits of the internal random numbers need not be identically distributed.)

Up to now we have exclusively considered the case that the das random numbers are binary-valued. However, there are TRNGs which generate k-bit das random numbers for $k > 1$. Unlike for the case $k = 1$ Reference [19] does not provide a concrete test suite for $k > 1$. However, [19] does not exclude those TRNGs. The applicant rather has to specify appropriate alternative tests which shall not be weaker than those for $k = 1$, i.e. they must ensure at least the same entropy bounds per bit. Depending on the noise source and the precise form of digitisation the individual bits of the das random numbers need not be identically distributed and there might exist dependencies between particular bits. A stationary $\{0, 1\}^k$-valued sequence need not necessarily induce a stationary binary valued sequence when the k-bit-values are interpreted as binary subsequences of length k. As a consequence, it is not sufficient just to apply the tests from P2.i)(vii.a) to P2.i(vii.e) to the derived binary sequence (interpreting the k-bit values as binary subsequences of length k) without further justifications.

5 Assessing the Random Number Generation II (Alternative Criteria)

In the previous section we claimed and justified that for P2 evaluations the das random numbers should be investigated but not the internal random numbers. The aim of Criteria P2.i)(vii.a)-(vii.e) (or the TRNG-individual criteria if $k > 1$, resp.; cf. the final paragraph of the preceding section) was to verify with negligible error probability that the average entropy per das bit exceeds an acceptable minimum limit. Further, the mathematical postprossing shall not reduce the entropy per bit (cf. Remark 5). Note that there exist TRNGs for which the das random numbers do not meet these requirements (Case 1), or due to the construction of the TRNG the evaluator may not have access to the das random numbers (Case 2). However, TRNGs of this kind need not necessarily be inappropriate. For Case 1, for instance, a suitable postprocessing might increase the entropy per bit. The AIS 31 does not automatically exclude such TRNGs. It is nonetheless conceded that a TRNG meets Requirement P2.d)(vii) (cf. Sect. 4) if the applicant is able to specify and justify suitable alternative evaluation criteria.

In Case 1 the tests destined for the das random numbers have to be applied to the internal random numbers. Additionally, and this is the crucial point which may turn out to be difficult or even impossible for a concrete TRNG, in [19] (Alternative criteria for P2.d)(vii); type 1)) it is specified that the applicant has to give *'Clear Proof that the internal random numbers achieve the goal set with criterion P2.d)(vii). The proof must be provided taking into account the mathematical postprocessing and on the basis of the empirical properties of the digitised noise sequence'.*

The 'clear proof' can be based on statistical tests of the internal random numbers if their suitability is justified. Depending on the type of postprocessing this may be rather difficult or even impossible (cf. Example 2 below). In Case 2 the situation is even more difficult. Again, the tests destined for the das random numbers have to be applied to the internal random numbers. Additionally, in [19] (Alternative criteria for P2.d)(vii); type 2)) it is specified that the applicant has to give *'Comprehensible and plausible description of a mathematical model of the physical noise source and the statistical properties of the digitised noise signal sequence derived from it.'* Further, he has to give a *'Specification of statistical tests that guarantee the goal defined in criterion P2.d)(vii) insofar as the internal random numbers pass these tests. It shall be comprehensibly justified that these tests are suitable. The proof must be provided taking into account the mathematical postprocessing and on the basis of the statistical properties of the noise signal sequence derived from the mathematical model of the noise source.'*

Reference [19] explicitly permits alternative criteria since no reasonable random number generator should be excluded from getting a certificate. It is not the intention of [19] to favour or discriminate particular types of TRNGs. The applicant himself has to specify and justify alternative criteria since it should be reasonable to expect that he is able to give evidence why his TRNG design

is suitable for sensitive cryptographic applications. If the evaluator has access to the das random numbers (single bits) and if the das random numbers fulfil the Criteria P2.i)(vii.a)-(vii.e) his work has already been done in [19] (cf. Sect. 4). The following example (cf. [19], Example E.4) illustrates the procedure and general difficulties when alternative criteria are applied.

Example 2. Throughout this example we assume that the noise source generates binary-valued das random numbers. We consider three different mathematical postprocessings.

a) The das random numbers (bits) are XORed to the feedback value of an LFSR (cf. Example 1).
b) Non-overlapping pairs of consecutive das bits are XORed.
c) The das bit sequence $b_1, b_2, \ldots$ is segmented into non-overlapping blocks $y_1, y_2, \ldots$ of length 128. The internal random numbers are given by $r_j := \mathrm{AES}(y_{2j-1}, y_{2j})$, i.e. block y_{2j-1} is encrypted with key y_{2j}.

(i) (Case 1) Suppose that extensive investigations of TRNG prototypes have shown that the das bits may be viewed as realizations of iid random variables but the (prototype-dependent) probability for assuming the value "1" lies in the interval $[0.45, 0.47]$. Obviously, the das random numbers do not meet criterion P2.i(vii.a).
Postprocessing a) does not increase the entropy per bit and hence an evaluation with alternative criteria is definitely not possible. Now assume that variant b) is applied. As there is evidence that the das random numbers are independent the internal random numbers should also behave like realizations of independent random variables, and the probability for a "1" lies in the interval $[0.49875, 0.50125]$. This argumentation provides a comprehensible proof sufficient for a Case 1 evaluation with alternative criteria. Consequently, the TRNG is conceded to meet Property P2.d)(vii). For variant c) statistical tests of the internal random numbers cannot deliver any useful information because such tests had at least to take the one-dimensional distribution of 128-bit blocks into consideration. An evaluation with alternative criteria would instead require a theoretical proof that the mathematical postprocessing leads to a sufficient increase of entropy per bit.
(ii) (Case 2) Suppose that a careful analysis of the precise realization of the physical noise source (taking into account the switching times and dead times of individual building blocks, sampling rates etc.) made it plausible to assume that the das random numbers are independent. (A comprehensible justification for deriving this mathematical model is extremely important!)
Postprocessing a) can easily be back-calculated, and Criteria P2i).(vii.a)-(vii.e) can be applied to the das random numbers. As the postprocessing does not increase the entropy per bit a Case 2 evaluation is only possible if the das bits fulfil these criteria. Similarly as above we can argue that variant b) transforms the das bits into independent internal random numbers. Consequently, it is necessary and sufficient that the internal random numbers fulfil the Criteria P2.i)(vii.a)-(vii.e). For c) the situation is even more complicated than in (i). An evaluation with alternative criteria seems hardly be possible.

6 Startup Test, Online Test, Tot Tests

In the previous sections we have intensively considered criteria for assessing the suitability of the random number generation process. However, even if Property P2.d)(vii) is conceded, i.e. if the prototypes in the lab generated 'high-quality' random numbers this may not be true for any TRNG of the same production run all the time when it is in operation. Tolerances of components of the noise source or aging effects may be responsible for this. In the worse case the noise source breaks totally down so that the das random numbers are constant from that moment on. Therefore, the developer should implement measures which shall detect defects of this kind in case they occur. We distinguish between *tot tests* ('tot' stands for 'total failure of the noise source'), *startup tests*, and *online tests*. As their names indicate a tot test shall detect a total breakdown of the noise source, the startup test is used to verify the principle functionality of the noise source when the TRNG has been started whereas the online test should detect if the quality of the random numbers is not sufficient for this particular TRNG (due to tolerances of components) or deteriorates in the course of the time.

Remark 6. The properties of the random number generation determines the class (P1,P2) to which the TRNG belongs. Besides, ITSEC and CC consider the 'strength of mechanisms and functions' as resistance against direct attacks. It is distinguished between 'low', 'medium' and 'high' and is assessed by means of the attack potential which is a function of expertise, resources and motivation required for a success ([9], Annex B.8). In our context the conceded strength of mechanisms or functions depends on the suitability to resist direct attacks exploiting weaknesses in the implementation. The tot, startup and online test are countermeasures to prevent negative effects of errors or failure of the TRNG or of malicious external influences on the quality of the generated random numbers (cf. Sect. 7). In such cases the random number generation should be stopped.

If the strength of mechanism (cf. [15]) or functions (cf. [8]) is low [19] does not demand any tot test, startup test or online test. If the strength is medium or high these tests are required. For class P1 these tests shall consider the internal random numbers (cf. [19], P1.d)). We will not pursue this aspect in the following. Instead, we will concentrate on class P2 with strength of mechanisms high.

The requirement on the startup test are rather mild. It only has to verify statistical minimum properties of the das random numbers when the TRNG has been started ([19], P2.d)(ix)). The tot test has to prohibit that random number sequences are output for which the underlying das random numbers have been generated completely after a total failure of the noise source ([19], P2.d)(x); cf. also Remark 7(i)). The online test has to check the quality of the das random numbers. The TRNG has to trigger the online test itself. The online test and the call schema must be suitable for detecting unacceptable statistical defects of the das random numbers or the deterioration of their statistical properties within an acceptable period of time. For ideal random number generators the probability

that at least one random alarm occurs in the course of a year of typical use of the particular TRNG should be $\geq 10^{-6}$ ([19], P2.d)(xi),(xiii)).

Remark 7. (i) Ideally, no external random number should be output if any of the used das random numbers has been generated after a total breakdown of the noise source. This aim can be assured by using only internal random numbers where an online test (applied to the das random numbers) has been passed after these internal numbers had been generated. The internal random numbers should be generated from the das random numbers of the beginning of the test sample. However, this is not always possible. The minimal requirement P2.d)(x) in [19] needs concrete analysis for specific technical solutions.
(ii) If the internal random numbers are tested instead of the das random numbers an extra justification for the suitability of the tests is necessary (cf. [19], Comment 2).
(iii) A minimum probability for an erroneous noise alarm was specified in order to rule out weak statistical tests; e.g. if a noise alarm was released iff 64 consecutive das random numbers are equal. (However, this might be an appropriate decision rule for the tot test.)
(iv) The tot test, startup test and the online test are usually a part of the TRNG implementation. In exceptional cases they may be realized as external security measures, e.g. by software calling the TRNG. In this case the applicant has to provide an exact specification of these measures and a reference implementation ([19], Comments 1 and 3).
(v) Due to the small alarm probabilities the effectiveness of the online test has to be justified on a theoretical basis.

The AIS 31 does not prescribe any concrete solutions. Reference [19] merely demands properties tot tests, startup tests and online tests should have. Moreover, various examples are discussed ([19], Examples E.5-E.7).

Remark 8. (i) For the sake of efficiency the tot test, startup test and online test may use the same statistical test, although with different decision rules (cf. [30] or [19], Example E.7).
(ii) Many TRNGs release a noise alarm if a single test value exceeds a certain limit which is expected to occur with an extremely small probability (e.g., $\leq 10^{-8}$). As pointed out in [30] (Sect. 4) this approach has two drawbacks: Even for ideal random number generators usually only an approximation of the distribution of the test variable is known (e.g. for the χ^2-test). At the tail of these distributions the relative error may be very large. Moreover, those approximations normally give only little information on the rejection probability if the random numbers are not independent and equidistributed.
(iii) In [30] (cf. also [19], Example E.7) a new online test procedure was proposed where it is *practically feasible* to determine the expected number of noise alarms within a time interval, even if the tested random numbers are not independent and equidistributed. Moreover, the system designer can vary a whole parameter set and hence can fit the test to the very special requirements of the intended applications. Compared with the widely used online tests mentioned in (ii) the

proposed solution does only need little more memory, some additional lines of code and slightly more running time. These aspects are of particular importance if the TRNG is integrated in a smart card.

7 Vulnerability Analysis

The evaluation process according to CC (cf. [8] and [9]) or ITSEC (cf. [15] and [16]) will provide evidence for the correct implementation and the understanding of the design. The statistical tests and the criteria described in Sects. 4 and 5 check the properties of the random numbers generated by prototypes. The effectiveness of the tot, startup and online test is verified mostly by theoretical arguments. The goal of the vulnerability analysis is to determine whether vulnerabilities have been identified during the evaluation of the construction and anticipated operations of the TOE, or whether other methods (e.g. flaw hypotheses) could allow users to violate the security policy of the target of evaluation. Vulnerabilities of the RNG may allow the ability to interfere with or alter the random numbers. The analysis starts with obvious vulnerabilities which are open to exploitation and requires a minimum of understanding of the TOE, skill, technical sophistication, and resources in the public domain.

Note that the evaluation guidelines [19] themselves directly address countermeasures against some obvious vulnerabilities of the TRNG, e.g. aging effects and the total failure of the noise sources by requirements for tot, startup and online tests. Obvious vulnerabilities might be suggested by the RNG external and internal interface description. One easily sees the vulnerability of an hybrid RNG which uses external random data which are under the control of an attacker. Noise sources based on inherent semiconductor thermal noise may be vulnerable to environmental conditions like temperature. The attacker may try to run the RNG at very low or very high temperature which could affect the quality of the random numbers. Requirement [19], P2.d)(xii) demands that Requirement P2.d)(vii) (cf. Sects. 4 and 5) are fulfilled under the intended external usage conditions (temperature, power supply etc.) insofar as these can influence the function of the noise source. If the environmental conditions are hostile, it may be necessary to extend the tests under the aspect of the strength of mechanisms and the analysis of the weaknesses. Smart card chips are normally protected by means of a temperature sensor against operation out of range. The evaluator should examine whether this range of temperature is valid for the normal operation of the RNG. In some cases the online tests may detect deviations from normal RNG operations more effective than the sensor, especially if aging of the random source is taken into account. The obvious vulnerability get more specific if the concrete product is analyzed.

Another example of potential vulnerabilities of a TRNG (at least if it is implemented on a smart card chip) are side channels caused by the physical random source or the postprocessing. A vulnerability analysis of pseudo-random number generators may be found in [18]. A very specific but very instructive example of a DRNG vulnerability is described in [5]. The authors show that if

the random parameter k in the Digital Signature Algorithm (DSA) is generated by means of a linear congruential DRNG then the private key can quickly be recovered from a few signatures even though this generator passes all statistical tests and none of the pseudo-random numbers is ever revealed. D. Bleichenbacher [6] discovered a flaw in the RNG described in the Appendix 3 of FIPS 186-2 that result in the non-uniformity of the generated pseudo-random numbers. The RNG is revised in the change notice to FIPS 186-2 ([26]).

The vulnerability analysis has to consider all information obtained during the evaluation process of the TOE so far. All vulnerabilities have to be assessed whether they might be used for practical attacks. One aspect is the expected number of trials which are necessary to guess random numbers. The optimal strategy in an exhaustive key search begins with the most likely bit string and continues guessing in order of decreasing probabilities. Let Z denote a random variable assuming values in $\{z_1, z_2, \ldots, z_{2^n}\}$, the set of n-bit strings, ordered with respect to their probabilities, i.e. $P(z_1) \geq P(z_2) \geq \cdots P(z_{2^n})$. For $0 \leq \delta \leq 1$ let $\mu(n, \delta)$ denote the minimum number of strings which must be guessed in order to find the correct string with a probability of at least δ. Then clearly

$$\mu(n, \delta) := \min \left(k \mid \sum_{i=1}^{k} P(z_i) \geq \delta \right).$$

Assume that an RNG may be viewed as an ergodic stationary binary-valued source with p denoting the probability for the value 1. In [23] it is pointed out that then

$$\lim_{n \to \infty} \frac{\log_2 \mu(n, \delta)}{n} = H(p).$$

Generally the RNG need not generate binary-valued random numbers. In the most extreme case it generates each n-bit string at one go. Then the strong assumption of the Maurer's formula does not hold. More generally, Pliam's formula ([28]) is valid:

$$\left\lfloor \frac{1}{2 \max_{1 \leq i \leq 2^n} P(z_i)} \right\rfloor \leq \mu(n, 0.5) \leq \left\lceil 2^n \left(1 - \sum_{i=1}^{2^n} \left| P(z_i) - 2^{-n} \right| \right) \right\rceil.$$

Depending on the concrete numerical values the evaluator may use the lower bound to show that the RNG is secure against a guessing attack and the upper bound to point at a weakness.

8 Conclusions

A comprehensive introduction in the evaluation of physical random number generators was given. Fundamental differences to deterministic random number generators and hybrid number generators were pointed out and general principles for the evaluation of TRNGs have been explained. Concrete measures and criteria specified in [19] were cited and explained.

The AIS 31 (cf. [2,19]) has been effective in the German evaluation and certification scheme since September 2001. It is mandatory if a physical random number generator influences the security properties of an IT product and if a German IT security certificate is applied for. The AIS 31 does not favour or exclude particular TRNG design principles. It prescribes properties and criteria a TRNG should fulfil and the evaluation task to be performed.

References

1. AIS 20: Functionality Classes and Evaluation Methodology for Deterministic Random Number Generators. Version 1 (02.12.1999) (mandatory if a German IT security certificate is applied for; English translation). www.bsi.bund.de/zertifiz/zert/interpr/ais20e.pdf
2. AIS 31: Functionality Classes and Evaluation Methodology for Physical Random Number Generators. Version 1 (25.09.2001) (mandatory if a German IT security certificate is applied for; English translation). www.bsi.bund.de/zertifiz/zert/interpr/ais31e.pdf
3. ANSI X9.17-1985, Financial Institution Key Management (Wholesale).
4. V. Bagini and M. Bucci: A Design of Reliable True Number Generators for Cryptographic Applications. In: Ç.K. Koç and C. Paar (eds.): Cryptographic Hardware and Embedded Systems — CHES 1999. Springer, Lecture Notes in Computer Science, Vol. **1717**, Berlin (1999), 204–218.
5. M. Bellare, S. Goldwasser, D. Micciancio: "Pseudo-Random" Number Generation Within Cryptographic Algorithms: The DSS Case. In: B. Kaliski (ed.): Crypto'97. Springer, Lecture Notes in Computer Science, Vol. **1294**, Berlin (1997), 277–291.
6. Lucent Technologies, Bell Labs: Scientist discovers significant flaw that would have threatened the integrity of on-line transactions, press article at www.lucent.com/press/0201/010205.bla.html.
7. M. Blum and S. Micali: How to Generate Cryptographically Strong Sequences of Pseudo-Random Bits. SIAM J. Comput., Vol. **13** (1984), 850–864.
8. Common Criteria for Information Technology Security Evaluation, Part 1-3; Version 2.1, August 1999 and ISO 15408:1999.
9. Common Methodology for Information Technology Security Evaluation CEM-99/045, Part 2: Evaluation Methodology, Version 1.0 (August 1999).
10. J.-S. Coron: On the Security of Random Sources. In: H. Imai and Y. Zheng (eds.): Public Key Cryptography — PKC 99. Springer, Lecture Notes in Computer Science, Vol. **1560**, Berlin (1999), 29–42.
11. J.-C. Coron, D. Naccache: An Accurate Evaluation of Maurer's Universal Test. In: S. Tavares and H. Meijer (eds.): Selected Areas in Cryptography — SAC '98. Springer, Lecture Notes in Computer Science, Vol. **1556**, Berlin (1999), 57–71.
12. L. Devroye: Non-Uniform Random Variate Generation. Springer, New York (1986).
13. IEEE: IEEE P1363 / D8 (Draft Version 8), Annex D, Standard Specifications for Public Key Cryptography (05.10.1998). www.informatik.tu-darmstadt.de/TI/Veroeffentlichung/Artikel/Kryptographie/PublicKeyAllgemein/P1363-D-10-05-98.pdf
14. Intel Platform Security Division: The Intel Random Number Generator. Intel Corporation (1999).
15. Information Technology Security Evaluation Criteria (ITSEC); Provisional Harmonised Criteria, Version 1.2, June 1991

16. Information Technology Security Evaluation Manual (ITSEM); Provisional Harmonised Methodology, Version 1.0, September 1993
17. G.K. Kanji: 100 Statistical Tests. Sage Publications, London (1995).
18. J. Kelsey, B. Schneier, D. Wagner, C. Hall: Cryptanalytic Attacks on Pseudorandom Number Generators. In: S. Vaudenay (ed.): Fast Software Encryption — FSE 1998, Springer, Lecture Notes in Computer Science, Vol. **1372**, Berlin (1998), 168-188.
19. W. Killmann, W. Schindler: A Proposal for: Functionality Classes and Evaluation Methodology for True (Physical) Random Number Generators. Version 3.1 (25.09.2001), mathematical-technical reference of [2] (English translation); www.bsi.bund.de/zertifiz/zert/interpr/trngk31e.pdf
20. D.E. Knuth: The Art of Computer Programming. Vol. 2, Addison-Wesley, London (1981).
21. D.P. Maher, R.J. Rance: Random Number Generators Founded on Signal and Information Theory. In: Ç.K. Koç, C. Paar (eds.): Cryptographic Hardware and Embedded Systems — CHES 1999. Springer, Lecture Notes in Computer Science, Vol. **1717**, Berlin (1999), 219–230.
22. G. Marsaglia: Diehard (Test Suite for Random Number Generators). www.stat.fsu.edu/˜geo/diehard.html
23. U. Maurer: A Universal Statistical Test for Random Bit Generators. J. Crypt. **5** (1992), 89–105.
24. NIST: Security Requirements for Cryptographic Modules. FIPS PUB 140-1 (11.04.1994). www.itl.nist.gov/fipspubs/fip140-1.htm
25. NIST: Security Requirements for Cryptographic Modules. FIPS PUB 140-2 (25.05.2001) and Change Notice 1 (10.10.2001). csrc.nist.gov/publications/fips/fips140-2/fips1402.pdf
26. NIST: Digital Signature Standard (DSS). FIPS PUB 186-2 (27.01.2000) with Change Notice 1 (5.10.2001). csrc.nist.gov/publications/fips/fips186-2/fips186-2-change1.pdf
27. A. Rukhin et al.: A Statistical Test Suite for Random and Pseudorandom Number Generators for Cryptographic Applications. NIST Special Publication 800–22 with revisions dated 15.05.2001. csrc.nist.gov/rng/SP800-22b.pdf
28. J.O. Pliam: The Disparity Between the Work and the Entropy in Cryptology (01.02.1999). eprint.iacr.org/complete/
29. W. Schindler: Functionality Classes and Evaluation Methodology for Deterministic Random Number Generators. Version 2.0 (02.12.1999), mathematical-technical reference of [1] (English translation); www.bsi.bund.de/zertifiz/zert/interpr/ais20e.pdf
30. W. Schindler: Efficient Online Tests for True Random Number Generators. In: Ç.K. Koç, D. Naccache, C. Paar (eds.): Cryptographic Hardware and Embedded Systems — CHES 2001. Springer, Lecture Notes in Computer Science, Vol. **2162**, Berlin (2001), 103–117.

A Hardware Random Number Generator

Thomas E. Tkacik

Motorola
2900 South Diablo Way
Tempe, AZ 85282
Tom.Tkacik@motorola.com

Abstract. Some of the desirable properties a cryptographic random number generator should have are lack of bias, bit independence, unpredictiability and nonrepeatability. In this paper, we discuss how a hardware random number generator formed from simple components can provide these properties. The components include two state machines with different structures, and free-running oscillators. The generated numbers pass the DIEHARD battery of tests.

The main uses of random numbers are in simulation and for cryptography. For simulation, the main requirement on the quality of the numbers is on their statistical properties; that they appear random.

Random number generators are used in many cryptographic algorithms and protocols. Their uses include generation of session keys and private keys, as a challenge against a replay attack, and as padding material for short messages. Weak random number generators can be targets for breaking into a cryptographic system [1]. When used for cryptographic purposes, random numbers must be unpredictable as well as have good statistical properties. We describe a hardware random number generator, used at Motorola, which passes Marsaglia's DIEHARD battery of tests [2], as well as FIPS-140 [3] and Crypt-X [4].

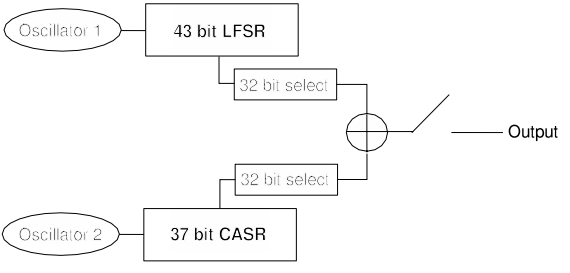


Fig. 1. Hardware random number generator block diagram.

The 32-bit hardware random number generator is based on a linear feedback shift register (LFSR), and a cellular automata shift register (CASR). Figure 1

B.S. Kaliski Jr. et al. (Eds.): CHES 2002, LNCS 2523, pp. 450–453, 2003.

shows a simplified block diagram of the generator. Each shift register is clocked by an independent ring oscillator, and the output is sampled only when a new number is requested. The LFSR has 43 bits, and a characteristic polynomial of $X^{43}+X^{41}+X^{20}+X+1$. This is a primitive polynomial and gives a cycle length of $2^{43}-1$, (the all zero pattern is missing).

Wolfram [5] describes using 1-dimensional cellular automata (CA) with a neighborhood of three for generating random sequences. He defines CA rules as one of the 256 functions of three variables to define the next state of each cell site. The number of the rule is given by the decimal value of its eight row truth table. Pries [6] describes a hybrid CA using rules CA90 and CA150. The important property of this hybrid CA is that, with the appropriate selection of the CA90 and CA150 rules for each cell site, it gives a maximal length sequence. The CA90 rule is defined by the equation

$$a_i(t+1) = a_{i-1}(t) \oplus a_{i+1}(t)$$

while the CA150 rule is defined by the equation

$$a_i(t+1) = a_{i-1}(t) \oplus a_i(t) \oplus a_{i+1}(t)$$

The hardware random number generator uses a 37-bit CASR with a CA150 at cell site 28, and CA90s at all other cell sites. Hortensius [7] states "the hybrid CA's of maximal length that we have found all require null boundary conditions", and our CASR is no exception. The CASR has a maximal length of $2^{37}-1$, (again the all zero pattern is missing).

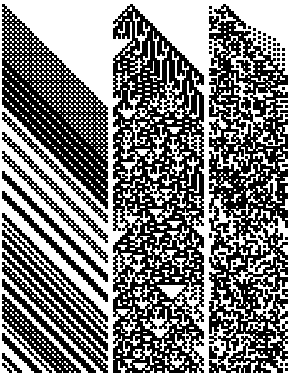


Fig. 2. State-time diagram of the LFSR, CASR and combined generator.

To generate a random number, 32 bits of the LFSR and CASR are selected and permuted, and then XORed together. Because the cycle lengths of the two state machines are relatively prime, the cycle length of the combined generator

is close to 2^{80} (actually $2^{80} - 2^{43} - 2^{37} + 1$). Because of the missing all zero pattern from both the LFSR and the CASR, there is a slight bias in the output bits, on the order of 2^{-43} and 2^{-37} respectively. In the combined generator, this bias drops to close to 2^{-80}. We have never generated enough random output to actually measure this bias.

Figure 2 shows a state-time diagram for the LFSR, CASR and combined generator. The initial states are shown at the top, and time progresses downward for 150 steps. The LFSR shows significant patterns, as the data in the LFSR is simply shifted right each step, and the only new bit is inserted at the left. The CASR is better, but triangular artifacts can readily be seen. The combination of the LFSR and CASR is much better (at least visually), but further testing will show that used this way, the combination still is not of acceptable quality.

Hortensius [7] also describes "site spacing" and "time spacing" as means to reduce the correlations between bits of the CA. The hardware random number generator uses site spacing in the selection and permutation of the LFSR and CASR bits it combines. Time spacing is also used in that there are two free running oscillators used as clocks for the two state machines, and a variable sampling period to capture the output data. The oscillators' frequencies vary with temperature, voltage and processing parameters. The state machines cycle through their states at different rates from each other and from the system clock. Even when not used, the hardware random number generator is active, keeping the internal state unpredictable. When multiple words are requested, there is a minimum sampling time which allows both state machines to clock at least twice their length (i.e., the LFSR is allowed to clock through at least 86 clock cycles). This minimum number of system clock cycles is determined by the lowest expected frequency of the free running oscillators.

Because this hardware random number generator has internal state, it is critical that the sequence of numbers it generates is not repeatable. The frequencies of the two oscillators are not controlled, and they drift with variations in temperature and voltage. Also, the state registers are not reset at power up, so that the intial state may take different values. These features allow the random number generator to cycle through a different sequence each time it is restarted.

The different components of the hardware random number generator were tested using DIEHARD. Figure 3 presents the results of running DIEHARD on individual pieces of the hardware random number generator, as well as actual silicon. DIEHARD is a collection of 15 tests, most of which give several results. In total, there are 234 p-values generated by DIEHARD. P-values are uniform over the range [0,1), for true random numbers. If uniform p-values are sorted and plotted, the result is a straight line, shown in Figure 2 as Ideal. The results for truely random data should approximate this line. The next three plots assume a single clock and sample every clock cycle. Both the LFSR and CASR fail the tests miserably, their p-values are not uniform. For the combination of the LFSR and CASR, there are 15 p-values equal to 1.0000, showing that there are still flaws in this generator, but it is a significant improvement over either individually. The final graph is from output of the actual hardware random number generator.

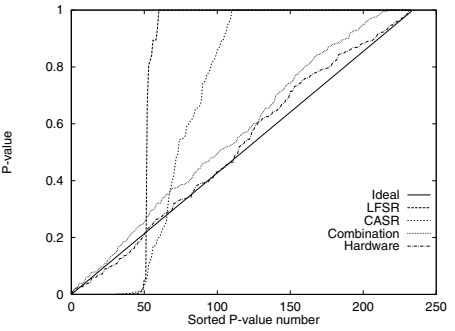


Fig. 3. DIEHARD results of the LFSR, CASR and combined generator

With variable time sampling, the hardware random number generator passes these tests.

The entire design of the hardware random number generator is written in Verilog RTL, with the exception of the ring oscillators, which are netlists of a number of inverters. The selected lengths of the inverter chains depend on the process technology and system clock frequency. If the random number generator will not be used for some time, the oscillators can safely be turned off to reduce power, thus allowing its use in lower power applications.

Acknowledgements. This random number generator has been used within Motorola for a number of years, and has gone through several minor variations in that time. Ezzy Dabbish and Steve Tugenberg developed the original hardware design.

References

1. I. Goldberg and D. Wagner. Randomness and the Netscape Browser. *Dr. Dobb's Journal*, January 1996.
2. G. Marsaglia. DIEHARD Statistical Tests. `http://stat.fsu.edu/~geo/diehard.html`
3. FIPS PUB 140. Security requirements for cryptographic modules. May 2001.
4. Crypt-X. `http://www.isrc.qut.edu.au/resource/cryptx`.
5. S. Wolfram. Random sequence generation by cellular automata. *Advances in Applied Mathematics*, 7:123–169, June 1986.
6. W. Pries, A. Thanailakis, and H. C. Card. Group properties of cellular automata and VLSI applications. *IEEE Transactions on Computers*, 35(12):1013–1024, December 1986.
7. P. D. Hortensius, R. D. McLeod, and H. C. Card. Parallel random number generation for VLSI systems using cellular automata. *IEEE Transactions on Computers*, 38(10):1466–1473, October 1989.

RFID Systems and Security and Privacy Implications

Sanjay E. Sarma, Stephen A. Weis, and Daniel W. Engels

Auto-ID Center
Massachusetts Institute of Technology
Cambridge, MA 02139
www.autoidcenter.org

Abstract. The Auto-ID Center is developing low-cost radio frequency identification (RFID) based systems with the initial application as next generation bar-codes. We describe RFID technology, summarize our approach and our research, and most importantly, describe the research opportunities in RFID for experts in cryptography and information security. The common theme in low-cost RFID systems is that computation resources are very limited, and all aspects of the RFID system are connected to each other. Understanding these connections and the resulting design trade-offs is an important prerequisite to effectively answering the challenges of security and privacy in low-cost RFID systems.

1 Introduction

Automatic Identification (Auto-ID) systems have become commonplace in access control and security applications, in industries requiring the tracking of products through the supply chain or manufacturing process, and in industries requiring the identification of products at the point of sale or point of service. Perhaps the most widely recognized Auto-ID system is the bar code system developed during the early 1970's. More recently, Radio-Frequency Identification (RFID) systems have begun to find greater use in automatic identification applications. RFID systems consist of Radio Frequency (RF) tags, or transponders, and RF tag readers, or transceivers. The transponders themselves typically consist of integrated circuits connected to an antenna [8]. The use of silicon-based microchips enables a wide range of functionality to be integrated into the transponder. Typical functionality ranges from large read/write memories to integrated temperature sensors to encryp! tion and access control functionality. The transceivers query the transponders for information stored on them. This information can range from static identification numbers to user written data to sensory data.

The potential applications for RFID systems are numerous. Consider, for example, supply chain management applications and the use of EAN-UCC bar codes. Today, over 5 billion bar codes are scanned daily world-wide [6]. Yet, most bar codes are scanned only once during the lifetime of the item, namely at the check out. RFID systems, if strategically deployed, are a single platform on which a number of supply chain management applications can be simultaneously

B.S. Kaliski Jr. et al. (Eds.): CHES 2002, LNCS 2523, pp. 454–469, 2003.

implemented, benefiting all parties involved in a commercial transaction: the manufacturers, the retailers, the users, and even regulatory bodies (such as the Food and Drug Administration (FDA) in the United States). Automated item level inventory identification using RFID systems will revolutionize supply chain management by enabling applications such as automated real-time inventory monitoring (at the shelf and in the warehouse), automated quality control, and automatic check-out.

The significant benefits that an inexpensive, open standards-based RFID system can provide are widely understood and acknowledged. At the same time, typical low-cost transponders are priced in the range of US$0.50-US$1.00, and RFID systems lack widely accepted and implemented standards for communication and functionality, thereby limiting their practical usefulness and keeping their system costs too high for many applications. In order to achieve significant item-level penetration within most supply chain applications, transponders will need to be priced well under US$0.10, and preferably under US$0.05. These cost targets cannot be achieved without a system-level approach that encompasses every aspect of the RFID technology, from IC design to RF protocols, from reader design to back-end data systems, and from IC manufacturing to antenna manufacturing. The challenge has been to develop a complete open standards-based system that enables the design and manufacture of lo! w-cost RFID systems.

The Auto-ID Center, an industry sponsored research center with laboratories at Massachusetts Institute of Technology, USA, Cambridge University, UK, and the University of Adelaide, AU, has designed, developed, and deployed within a large-scale field trial an open standards-based system that enables the unique identification of and retrieval of information on ubiquitously tagged items. The Center, in conjunction with its sponsors, has also undertaken projects to design and manufacture open standard low-cost RFID transceivers and transponders capable of little more than communicating a unique identifier stored within them. Low-cost transponders enable the tagging and unique identification of virtually all man-made items.

The commercial availability of low-cost, Auto-ID Center standards-based RFID systems by mid-2003 has poised these systems to be one of the earliest and perhaps most explosive opportunities in ubiquitous computing. As these systems leave the industrial applications and enter our daily lives, privacy and security related issues will play an increasingly important role in their use and ubiquity. The pupose of this paper is to explain the technology, the challenges, and the opportunities ubiquitous RFID systems present to the security and privacy communities.

2 A Brief Introduction to RFID Systems

2.1 Basic System Components

All RFID systems are comprised of three main components:

– the RFID tag, or *transponder,* which is located on the object to be identified and is the data carrier in the RFID system,

- the RFID reader, or *transceiver,* which may be able to both read data from and write data to a transponder, and
- the *data processing subsystem* which utilizes the data obtained from the transceiver in some useful manner.

Typical *transponders* (*trans*mitters/re*sponders*) consist of a microchip that stores data and a coupling element, such as a coiled antenna, used to communicate via radio frequency communication. Transponders may be either active or passive. Active transponders have an on-tag power supply (such as a battery) and actively send an RF signal for communication while passive transponders obtain all of their power from the interrogation signal of the transceiver and either reflect or load modulate the transceiver's signal for communication. Most transponders, both passive and active, communicate only when they are interrogated by a transceiver.

Typical *transceivers* (*trans*mitter/re*ceivers*), or RFID readers, consist of a radio frequency module, a control unit, and a coupling element to interrogate electronic tags via radio frequency communication. In addition, many transceivers are fitted with an interface that enables them to communicate their received data to a data processing subsystem, e.g., a database running on a personal computer. The use of radio frequencies for communication with transponders allows RFID readers to read passive RFID tags at small to medium distances and active RFID tags at small to large distances even when the tags are located in a hostile environment and are obscured from view.

The basic components of an RFID system combine in essentially the same manner for all applications and variations of RFID systems. All objects to be identified are physically tagged with transponders. The type of tag used and the data stored on the tag varies from application to application.

Transceivers are strategically placed to interrogate tags where their data is required. For example, an RFID-based access control system locates its readers at the entry points to the secure area. A sports timing system, meanwhile, locates its readers at both the starting line and the finish line of the event. The readers continuously emit an interrogation signal. The interrogation signal forms an interrogation zone within which the tags may be read. The actual size of the interrogation zone is a function of the transceiver and transponder characteristics. In general, the greater the interrogation signal power and the higher the interrogation signal frequency, the larger the interrogation zone. Sending power to the transponders via the reader-to-tag communication signal is the bottleneck in achieving large read range with passive tags. Active tags do not suffer from this drawback; thus, they typically have larger communication ranges than an otherwise equivalent passive tag.

The transceivers and transponders simply provide the mechanism for obtaining data (and storing data in the case of writable tags) associated with physical objects.

Passive RFID systems are the most promising to provide low-cost ubiquitous tagging capability with adequate performance for most supply chain management applications. These low-cost RFID systems are, of necessity, very resource

limited, and the extreme cost pressures make the design of RFID systems a highly coupled problem with sensitive trade-offs. Unlike other computation systems where it is possible to abstract functionality and think modularly, almost every aspect of an RFID system affects every other aspect. We present a brief overview of the critical components of RFID technology and summarize some of these trade-offs in passive RFID design.

2.2 Transceiver-Transponder Coupling and Communication

Passive RFID tags obtain their operating power by harvesting energy from the electromagnetic field of the reader's communication signal. The limited resources of a passive tag require it to both harvest its energy and communicate with a reader within a narrow frequency band as permitted by regulatory agencies. We denote the center of this frequency band by f, and we refer to RFID systems operating at frequency f with the understanding that this is the center frequency of the band within which it operates.

Passive tags typically obtain their power from the communication signal either through inductive coupling or far field energy harvesting. Inductive coupling uses the magnetic field generated by the communication signal to induce a current in its coupling element (usually a coiled antenna and a capacitor). The current induced in the coupling element charges the on-tag capacitor that provides the operating voltage, and power, for the tag. In this way, inductively coupled systems behave much like loosely coupled transformers. Consequently, inductive coupling works only in the near-field of the communication signal. The near field for a frequency f extends up to $1/(2\pi f)$ meters from the signal source.

For a given tag, the operating voltage obtained at a distance d from the reader is directly proportional to the flux density at that distance. The magnetic field emitted by the reader antenna decreases in power proportional to $1/d^3$ in the near field. Therefore, it can be shown that for a circularly coiled antenna the flux density is maximized at a distance d (in meters) when $R \cong \sqrt{2} \cdot d$, where R is the radius of the reader's antenna coil. Thus, by increasing R the communication range of the reader may be increased, and the optimum reader antenna radius R is 1.414 times the demanded read range d.

Far field energy harvesting uses the energy from the interrogation signal's far field signal to power the tag. The far field begins where the near field ends, at a distance of $1/(2\pi f)$ from the emitting antenna. The signal incident upon the tag antenna induces a voltage at the input terminals of the tag. This voltage is detected by the RF front-end circuitry of the tag and is used to charge a capacitor that provides the operating voltage for the tag.

There is a fundamental limitation on the power detected a distance d away from a reader antenna. In a lossless medium, the power transmitted by the reader decreases as a function of the inverse square of the distance from the reader antenna in the far field.

A reader communicates with and powers a passive tag using the same signal. The fact that the same signal is used to transmit power and communicate data creates some challenging trade-offs. First, any modulation of the signal causes a

reduction in power to the tag. Second, modulating information onto an otherwise spectrally pure sinusoid spreads the signal in the frequency domain. This spread, referred to as a "side band," along with the maximum power transmitted at any frequency, is regulated by local government bodies in most parts of the world. These regulations limit the rate of information that can be sent from the reader to the tag. RFID systems usually operate in free bands known as Industrial-Scientific-Medical (ISM) bands, where the emitted power levels and the side band limits tend to be especially stringent.

The signaling from the tag to the reader in passive RFID systems is not achieved by active transmission. Since passive tags do not actively transmit a signal, they do not have a regulated limit on the rate of information that can be sent from the passive tag to the reader. In the near field, tag to reader communication is achieved via *load modulation.* Load modulation is achieved by modulating the impedance of the tag as seen by the reader. In the far field, tag to reader communication is achieved via *backscatter.* Backscatter is achieved by modulating the radar cross-section of the tag antenna. Comprehensive reviews of the operation of tags and readers are available in [8] and [17].

The powering of and communication with passive tags with the same communication signal places restrictions on the functionality and transactions the tags are capable of. First, there is very little power available to the digital portion of the integrated circuit on the tag. This limits the functionality of the tag. Second, the length of transactions with the tag is limited to the time for which the tag is expected to be powered and within communication range. Governmental regulations can further limit communication timings. In the US in the 915 MHz ISM band, regulations require that, under certain operating conditions, the communication frequency change every 400 ms. Since every change in frequency may cause loss of communication with a tag, transponders must not be assumed to communicate effectively for longer than 400 ms. Finally, it is important to minimize state information required in passive tags. In many practical situations, power supplied to the tag may be errat! ic, and any long-term reliance on state in the tag may lead to errors in the operation of a communication protocol.

2.3 Data Coding

The data, consisting of ones and zeroes, communicated between tags and readers must be sent in a reliable manner. There are two critical steps to reliable communication, the encoding of the data and the transmission of the encoded data, that is, the modulation of the communication signal. The combination of coding and modulation schemes determines the bandwidth, integrity, and tag power consumption.

The coding and modulation used in RFID communications is limited by the power and modulation/demodulation capabilities of the tags. Another limiting factor is the bandwidth occupied by the signal. Readers are capable of transmitting at high power but are limited to narrow communication bands by communications regulations; therefore, the encoding used from reader to tag usually

needs to occupy a low bandwidth. Passive tags, however, do not actively transmit a signal; therefore, the encoding used for tag to reader communication can occupy a high bandwidth.

There are two broad categories of codes used in RFID: level codes and transition codes. Level codes represent the bit with their voltage level. Transition codes capture the bit as a change in level. Level codes, such as Non-Return-to-Zero (NRZ) and Return-to-Zero (RZ), tend to be history independent; however, they are not very robust. Transition codes can be history dependent, and they can be robust. Figure 1 illustrates several codes.

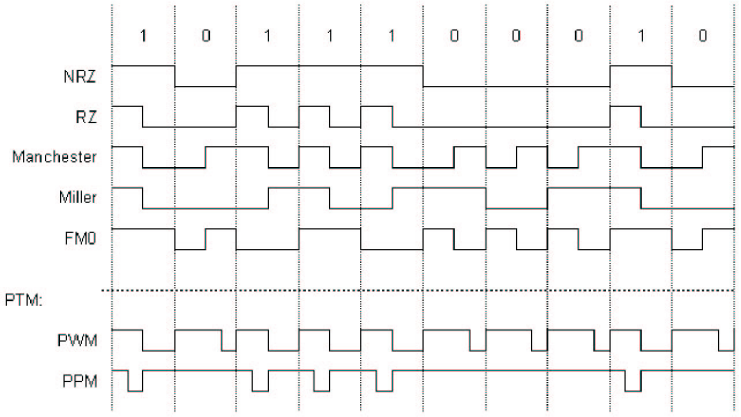


Fig. 1. Examples of several coding schemes.

The simplest code is Pulse Pause Modulation (PPM) in which the length between pulses is used to convey the bit. PPM codes provide low bit rates but occupy only a small bandwidth and are very easy to implement. In addition, these encodings can be adapted easily to ensure uninterrupted power supply since the signal does not change for long periods of time.

The Manchester code is a higher bandwidth transition code that represents a 1 as a negative transition at the half period and a 0 as a positive transition at a half period. The Manchester Code provides for efficient communication since the bit rate is equal to the bandwidth of the communication.

In RFID, the coding technique must be selected with three considerations in mind: 1) the code must maintain power to the tag as much as possible, 2) the code must not consume too much bandwidth, and 3) the code must permit the detection of collisions. The collision detection ability of a code is discussed further in Section 2.5. Depending on the bandwidth available, most RFID systems use PPM or PWM to communicate from reader to tag and Manchester or NRZ to communicate from tag to reader.

2.4 Modulation

The data coding scheme determines how the data is represented in a continuous stream of bits. How that stream of bits is communicated between the tag and the reader is determined by the modulation scheme. For convenience, RF communications typically modulate a high frequency carrier signal to transmit the baseband code. The three classes of digital modulation are Amplitude Shift Keying (ASK), Frequency Shift Keying (FSK) and Phase Shift Keying (PSK). The choice of modulation is based on power consumption, reliability requirements, and bandwidth requirements. All three forms of modulation may be used in the return signal although ASK is most common in load modulation at 13.56 MHz, and PSK is most common in backscatter modulation.

A problem unique to RFID systems is the vast difference in power between the signal outgoing from the reader and that returning to the reader as reflected from the tag. In some situations, this difference may be in the range of 80-90 dB [8], and the return signal may be impossible to detect. To avoid this problem, the return signal is sometimes modulated onto a sub-carrier, which is then modulated on to the carrier. For example, in the ISO 15693 standard for RFID, a sub-carrier of 13.56/32 (= 423.75 KHz) is used.

2.5 Tag Anti-collision

When multiple tags respond simultaneously to a reader's signal, their communication signals can interfere with one another. This interference is referred to as a collision and typically results in a failed transmission. In order for a reader to communicate with multiple tags, a method for collision free tag communication must be employed. These methods are referred to as anti-collision methods. An anti-collision method must be employed if an application will typically have more than one tag communicating with a reader at the same time.

Anti-collision methods, or algorithms, in tags have similarities to anti-collision algorithms in networking. Unlike standard networking however, RFID tags pose a number of problems that arise from the very limited resources that they are provided with. First, they can afford only limited computation power. Second, state information, such as what portion of the tags identifier has already been read, may be unreliable. Third, collisions may be difficult to detect due to widely varying signal strengths from the tags. Finally, as in most wireless networks, transponders cannot be assumed to be able to hear one another.

A common classification of anti-collision algorithms, either *probabilistic* or *deterministic,* is based upon how the tags respond during the anti-collision algorithm. In probabilistic algorithms, the tags respond at randomly generated times. There are several variations of probabilistic protocols depending on the amount of control the reader has over the tags. Many probabilistic algorithms are based on the Aloha scheme in networking [3]. The times at which readers can respond can be slotted or continuous. The ISO 15693 protocol, for example, supports a slotted Aloha mode of anti-collision.

Deterministic schemes are those in which the reader sorts through tags based on their unique identification number. The simplest deterministic scheme is the

binary tree-walking scheme, in which the reader traverses the tree of all possible identification numbers. At each node in the tree, the reader checks for responses. Only tags whose identifier is a child of the checked node respond. The lack of a response implies that the sub-tree is empty. The presence of a response gives the reader an indication as to where to search next.

The performance metrics that are traded-off by these algorithms and their variants include: 1) the speed at which tags can be read, 2) the outgoing bandwidth of the reader signal, 3) the bandwidth of the return signal, 4) the amount of state that can be reliably stored on the tag, 5) the tolerance of the algorithm to different types of noise in the field, 6) the cost of the tag, 7) the cost of the reader, 8) the ability to tolerate tags which enter and leave the field during the inventory-taking process, 9) the desire to count tags exactly as opposed to sampling them, and finally, 10) the range at which tags can be read.

The impact of regulated reader-to-tag bandwidth on the anti-collision protocol can be severe. In the US, for example, two common operating frequencies for RFID systems are the 13.56 MHz and the 915 MHz ISM bands. The regulations on the 13.56 MHz band offer significantly less bandwidth in the communication from the reader to the tag than do the regulations on the 915 MHz band. For this reason, Aloha-based anti-collision algorithms are more common in systems that operate in the 13.56 Mhz band and deterministic anti-collision algorithms are more common in the 915 Mhz band.

In practice, most RFID anti-collision algorithms tend to be an amalgam of probabilistic and deterministic concepts. Almost all require a unique ID to sort through the tags. This in itself has implications on privacy, as we will discuss later. The interplay between the anti-collision algorithm, the identifier, and the bandwidth available has an impact on all transactions between the reader and the tag. Approaches to security and privacy must therefore be geared to these very subtle trade-offs. Protocols to secure the tag at 13.56 Mhz, for example, must use far less signaling from reader-to-tag than at 915 Mhz. Either way, when there are several tags in the field, it is best to leverage the anti-collision algorithms as much as possible for efficiency.

2.6 Reader Anti-collision

RFID systems have traditionally been used in sparse applications where the readers tend to be far apart. In the applications we have explored, particularly those in supply chain management, the density of readers will often be very high, creating a new class of problems related to reader interference. We first reported the *Reader Collision Problem* in [7]. The solution to a reader collision problem allocates frequencies over time to a set of readers. The solution may be obtained in either a distributed or centrally controlled manner.

Reader collision problems have some similarities to frequency assignment problems in mobile telephone systems. However, the approaches that work in mobile telephones do not translate to RFID systems due to the limited functionality in RFID tags. The inability of the transponders to aid in the communication process means that they are unable to discriminate between two readers

communicating with them simultaneously. As a result, two readers that may communicate with the same tag must communicate at different times.

In a cooperative, trusted environment, reader collisions can be handled in a fairly seamless way. However, complications may arise in the execution of commands that change the state of the tag. If the reader executing a series of state changing actions is interrupted by another reader, it may be forced to relinquish control over the tag. The new reader that acquires the tag may further change the state of the tag without the cooperation of the first reader. Transactions between readers and tags must therefore be brief and atomic.

2.7 Frequencies and Regulations

The operation of RFID systems worldwide is regulated by local governmental bodies which control the electromagnetic spectrum in a region. Most RFID systems operate in so-called Industrial-Scientific-Medical (ISM) bands. These bands are freely available for use by low-power, short-range systems. The ISM bands are designated by the International Telecommunications Union (ITU) [11]. A comprehensive summary of the standards is available in [17]. The most commonly used ISM frequencies for RFID are 13.56 MHz and 902-928 MHz (in the US only). In addition, the low frequency band 9kHz-135 kHz is available for unlicensed use in most regions, and the 868MHz-870MHz band is available for use by nonspecific short-range devices in Europe. Each band has its own radiation power and bandwidth regulations.

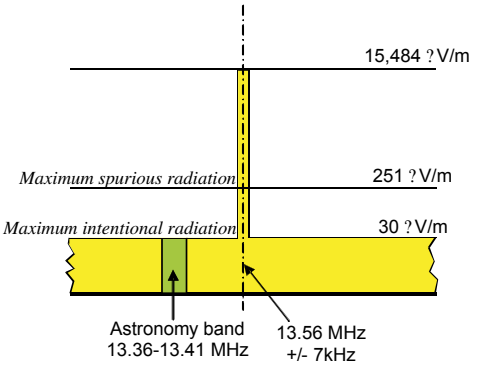


Fig. 2. The 13.56 MHz ISM band US power regulations.

Each frequency band brings its own challenges and advantages in terms of operation. The 13.56 MHz band shown in Figure 2 offers a great deal of asymmetry between the forward and reverse communication. Since readers must power the tags in passive RFID systems, the reader-to-tag communication must be at maximum power for maximum communication range. This limits the bandwidth

in reader-to-tag communication to a total of 14kHz. However, there is a great deal more bandwidth available for the low-power return communication.

The 915 MHz ISM band in the US, in contrast, allows multiple reader-to-tag communication options. The option that enables the longest communication range, the most commonly used mode in RFID systems, requires the reader to change its communication frequency every 0.4 seconds. The reader may 'hop' between 50 or more channels, each with a maximum bandwidth of 250 kHz. Frequency hopping imposes several limitations on RFID systems. The most severe of these limitations is that a tag cannot be assumed to be in continuous communication across a frequency hop. This means that transactions with 915 MHz RFID systems in the US should be limited to within the 0.4 second intervals allocated to any single frequency sub-band. Constraints of this type also point to the need for limited length, atomic transactions in RFID systems, a requirement which must be respected in the design of security and privacy systems.

3 The EPC System: A Minimalist Approach

At the Auto-ID Center, we have developed and implemented a system that enables all physical objects to be connected in real-time to the Internet by affixing an RFID tag to the object [14]. The scale of the system (essentially a several quadrillion node network), combined with the trade-offs in RFID design, created an intriguing design challenge. We utilized a minimalist strategy for the RFID tags (the most replicated component of the system) to enable extremely low-cost RFID systems. The result is a system that minimizes the functionality on the tag by moving that functionality to the 'network.'

The four key components of this system are the Electronic Product Code (EPC), the Object Name Service (ONS), the Savant, and the RFID transponders.

The EPC. The Electronic Product Code (EPC) is an identification scheme designed to enable the unique identification of all physical objects. This is the only data required to be stored on a tag, for once the unique identity of an object is established, information about that object can be obtained from the network. As such, the EPC acts like a pointer to this information.

The ONS. The Object Name Service (ONS) is a directory service that maps the EPC to an IP (Internet Protocol) address where information about the associated object can be written and/or accessed. The ONS is based entirely on the Domain Name Service (DNS), the directory service used on the Internet today to map a domain name (e.g., www.mit.edu) to an IP address (e.g., 18.181.0.31). At the IP address pointed to by the ONS, data about the particular object is stored in XML [20] format, and can be accessed by standard methods like HTTP and SOAP [19].

ONS reduces the burden on the transponders, and provides several advantages simultaneously. First, it reduces the memory and power requirements on the tag. Second, by transferring much of the data communication to a much higher-bandwidth back-end network, it saves precious wireless bandwidth. Third,

it makes the system more robust: while it is difficult to store and recover information from a failed tag, it is possible to back up databases inexpensively. Finally, this approach significantly reduces the footprint of the tag's microchip, reducing the cost of the transponder. (The cost of the microchip is proportional to its area [15].)

Savant. The Savant system is a hierarchical control and data management building block that can be used to provide automated control functionality and manage the large volumes of data generated by the RFID readers. A Savant enables the distributed creation of a reader network by acting as a gateway to the next higher level in the Savant hierarchy, effectively isolating the reader sub-network. The use of Savants enables distributed security by providing convenient points for network isolation.

The Savant network further reduces the burden on the tags while providing several advantages. First, it reduces the memory and power requirements on the tags by transferring the computationally intensive functionality to a powered system. Second, it makes the system more robust: any single point of failure has local effects. Third, it enables the entire system to be scalable as systems and reader sub-networks may be added seemlessly. Finally, the Savant network significantly reduces the footprint of the tag's microchip, reducing the cost of the transponder.

RFID Transponders. RFID transponders are the most numerous and cost sensitive of our system components. We have designed RFID protocols for both 13.56 MHz and 915 MHz, both with the aim of having minimum cost identification tags with acceptable performance for supply chain applications. Both transponders are designed to store a unique identifier, an EPC, and have that identifier retrieved as part of the anti-collision algorithm. The 915 MHz, UHF, transponder utilizes a directed tree search anti-collision algorithm, while the 13.56 MHz, HF, transponder utilizes a slotted Aloha-based anti-collision algorithm. Both transponders also implement a password protected `Self Destruct` command, that enables the owner of the tag to electrically and permanently destroy the tag.

The implementation cost of securing the `Self Destruct` command was weighed against the benefits of that security. It was determined that a secret key must be used to execute the `Self Destruct` command; therefore, requiring the destruction of a single tag at a time. The secret key is meant only to discourage the unauthorized destruction of tags. In a pervasive reader environment, unauthorized `Self Destruct` commands can be detected by the readers, enabling a proper reaction to the issuance of these commands.

We have taken a building-block approach to RFID transponder design in that these minimal functionality tags form the foundation of the functionality that will be found in higher-functionality tags. These higher functionality tags may be used in applications that can afford the additional cost of the transponder and require the transponder to implement the functionality.

4 RFID Security Benefits and Threats

Universally deploying RFID tags offers many potential security benefits, yet may expose new privacy threats. Otherwise intrusive or cumbersome security practices, such as airline passenger and baggage tracking, can be made practical by using RFID systems. Authentication systems already take advantage of RFID technology, for example car key-less entry systems. Embedding RFID tags as seals of authenticity in documents, designer products, and currency may discourage forgery. While RFID tags improve certain security properties in these applications, they may exacerbate privacy threats or pose new security risks.

RFID systems are different from other means of identification because RF communication is non-contact and non-line-of-sight, whereas other means of identification are either contact-based or require line-of-sight. In other words, it is more difficult for the owner of the RF tag to physically impede communication with the tag. The promiscuity of RF tags is not unique; magnetic stripe cards, for example, are promiscuous, but we assume that the owner of the card takes the physical responsibility of preventing unauthorized users from physically accessing the card. Of course, the propagation characteristics of electromagnetic fields do limit the range from which passive RFID cards can be read. In fact, most tags operating at 13.56 MHz cannot be read from more than a meter away, and 915 MHz tags are difficult to read through most materials. Yet, as the information stored on the tag becomes more and more valuable, it is necessary to think through some of the security and privacy! related issues in RFID. We present such a discussion in this section, ending with a proposed approach.

4.1 Previous Work

The contactless interface and constrained computational resources of RFID devices present a unique set of characteristics most closely related to smart cards. Many relevant lessons may be gleaned from the wealth of smart card and tamper resistant hardware research. [1] discusses a range of smart card protocols and analyzes cost and security trade-offs. Many RFID tags will operate in hostile environments and may be subject to intense physical attacks. Analysis of smart cards operation in hostile environments is presented in [9], while [18] provides an excellent overview of many physical attacks and countermeasures. Several specific lower cost physical attacks are detailed in [2] and are part of ongoing research at the University of Cambridge's TAMPER Lab [16]. Many results pertaining to implementation of cryptographic primitives on smart cards apply to RFIDs. Cautionary information regarding implementation of AES i! n smart cards appears in [5]. Being contactless and passively powered may make RFID devices especially susceptible to fault induction or power analysis attacks. Both [4] and [12] highlight many of these issues in cryptographic devices.

4.2 Security Goals

It is useful to state clear security goals when discussing security properties of various RFID designs. Tags must not compromise the *privacy* of their holders. Information should not be leaked to unauthorized readers, nor should it be

possible to build long-term tracking associations between tags and holders. To prevent tracking, holders should be able to detect and disable any tags they carry. Publicly available tag output should be randomized or easily modifiable to avoid long-term associations between tags and holders. Private tag contents must be protected by access control and, if interrogation channels are assumed insecure, encryption.

Both tags and readers should *trust* each other. Spoofing either party should be difficult. Besides providing an access control mechanism, mutual authentication between tags and readers also provides a measure of trust. Session hijacking and replay attacks are also concerns. Fault induction or power interruption should not compromise protocols or open windows to hijack attempts. Both tags and readers should be resistant to replay or man-in-the-middle attacks.

4.3 Low-Cost RFID Issues

With these security goals in mind, consider the security properties of passive factory-programmed, read-only tags. Each tag contains a unique identifier such as an EPC. While no more "promiscuous" than an optical bar code, automated monitoring of RF tags is possible. This basic design clearly violates the privacy goal since tracking tag holders and reading tag contents are possible if the tag is properly presented to a reader's interrogation field. Neither tags nor readers are authenticated; therefore, no notion of trust exists either.

To address these deficiencies, suppose we adopt a policy of erasing unique serial numbers at the point of sale. Consumer held tags would still contain product code information, but not unique identification numbers. Unfortunately, tracking is still possible by associating "constellations" of particular tag types with holder identities. For example, a unique penchant for RFID-tagged Gucci shoes, Rolex watches and Cohiba cigars may betray your anonymity. Furthermore, this design still offers no trust mechanism.

Providing the stated security goals requires implementing access control and authentication. Public key cryptography offers a solution. A particular (type of) reader's public key and a unique private key may be embedded into each tag. During interrogation, tags and readers may mutually authenticate each other with these keys using well understood protocols. To prevent eavesdropping within the interrogation zone, tags may encrypt their contents using a random nonce to prevent tracking. Unfortunately, supporting strong public key cryptography is beyond the resources of low cost (US$0.05-0.10) tags, although solutions do exist for more expensive tags [13] .

Symmetric message authentication requires each tag to share a unique key with a reader or for a key to be shared among a batch of tags. To support a unique key per tag, a complex key management overhead is necessary. If keys are to be shared, tags must be resilient to physical attacks described in [18]; otherwise, compromising a single tag effective compromises an entire a batch. Implementing secure memory on a low cost tag with a logic gate count in the hundreds is a daunting task, especially in light of the difficulty in securing memory on relatively resource abundant smart cards. Even supporting strong symmetric encryption is a challenge in the short term.

4.4 Some Approaches to RFID Protection

Accepting short-term limitations on low-cost tag resources, we discuss a simple RFID security scheme based on a one-way hash function. In practice, a hardware optimized cryptographic hash function would suffice, assuming it may be implemented with significantly fewer resources than symmetric encryption. In this design, each hash-enabled tag contains a portion of memory reserved for a "meta-ID" and operates in either an unlocked or locked state. While unlocked, the full functionality and memory of the tag are available to anyone in the interrogation zone.

To lock a tag, the owner computes a hash value of a random key and sends it to the tag as a lock value, i.e. *lock=hash(key)*. In turn, the tag stores the lock value in the meta-ID memory location and enters the locked state. While locked, a tag responds to all queries with the current meta-ID value and restricts all other functionality. To unlock a tag, the owner sends the original key value to the tag. The tag then hashes this value and compares it to the lock stored under the meta-ID. If the values match, the tag unlocks itself.

Each tag always responds to queries in some form and thus always reveals its existence. Tags will be equipped with a physical self-destruct mechanism and will only be unlocked during communication with an authorized reader. In the event of power loss or transmission interruption, tags will return to a default locked state. A trusted channel may be established for management functions, such as key management, tag disabling or even tag writes, by requiring physical contact between a control device and a tag. Requiring physical contact for critical functionality helps defend against wireless sabotage or denial of service attacks.

The hash-based lock mechanism satisfies most of our privacy concerns. Access control to tag contents is restricted to key holders. Individuals may both locate and disable tags they may be carrying since tags always respond to queries. Long-term associations can be avoided since locked tags only respond with the correct meta-ID. One caveat is that stale meta-ID values may be used to build tracking associations over time. This necessitates periodically refreshing meta-ID values by unlocking and re-locking tags.

Although authenticating readers and providing a trusted channel satisfies some of our trust requirements, this design does sacrifice several security properties to save costs; specifically tag authentication. Tag MAC functionality would allow tags to authenticate themselves, but is beyond current low-cost tag resources. Lacking authentication exposes tags to man-in-the-middle attacks since an attacker can query tags for meta-IDs, rebroadcast those values to a legitimate reader, and later unlock the tags with the reader's response keys. Many key-less car entry systems currently possess the same vulnerability. Regardless, attackers without access to an authorized reader cannot access tag contents outside physical channels.

4.5 Future Research Directions

While this candidate design partially satisfies some desired security properties, more secure implementations require several developments. One key line of research is the further development and implementation of low cost cryptographic

primitives. These include hash functions, random number generators and both symmetric and public key cryptographic functions. Low cost hardware implementations must minimize circuit area and power consumption without adversely affecting computation time. RFID security may benefit from both improvements to existing systems and from new designs. More expensive RFID devices already offer symmetric encryption and public key algorithms such as NTRU [10,13]. Adaptation of these algorithms for the low-cost (US$0.05-0.10), passive RFID devices should be a reality in a matter of years.

Protocols utilizing these cryptographic primitives must be resilient to power interruption and fault induction. Compared to smart cards, RFID tags possess more vulnerabilities to these types of attacks. Protocols must account for disruption of wireless channels or communication hijack attempts. Tags themselves must gracefully recover from power loss or communication interruption without compromising security.

Continually improving technology will steadily blur the line between RFID devices, smart cards and ubiquitous computers. Research benefiting the security of RFID devices will help pave the way for a universal, secure ubiquitous computing system. Developments related to RFID tags and other embedded systems may all contribute to the creation of a robust and secure infrastructure offering many exciting potential applications.

5 Conclusions

This article is a summary of a research effort underway by three universities, more than 60 companies, and more than 50 researchers world-wide.

The effort has been fueled by the potential economic impact of inexpensive, ubiquitous item identification in the supply chain. The roadmap towards cheap tags has been laid out, but like any research effort, uncertainty is a part of the challenge. Several technology alternatives will need to be tested for each component of the system before the optimal one is determined. Even after the first cheap tags have been manufactured, scaling production to the volumes needed to meet expected demand will be a challenge. It may be years before the supply meets the enormous demand that a technology of this type is projected to generate. However, it is these very volumes that make it necessary for the technology to be carefully thought out to save every fraction of a cent in the cost of a tag and to ensure the security and privacy of its future users.

Acknowledgments. The authors wish to thank Ron Rivest and Peter Cole for their continued support of this work.

References

1. M. Abadi, M. Burrows, C. Kaufman, and B. W. Lampson. Authentication and delegation with smart-cards, In *Theoretical Aspects of Computer Software*, pages 326–345, 1991.

2. R. Anderson and M. Kuhn. Low cost attacks on tamper resistant devices. In *IWSP: International Workshop on Security Protocols, LNCS*, 1997.
3. B. Bing. *Broadband Wireless Access,* Boston, Kluwer Academic Publishers, 2000.
4. D. Boneh, R.A. DeMillo, and R.J. Lipton. On the importance of checking cryptographic protocols for faults. In *EUROCRYPT'97*, volume 1233, pages 37–51. Lecture Notes in Computer Science, Advances in Cryptology, 1997.
5. S. Chari, C. Jutla, J.R. Rao, and P. Rohatgi. A cautionary note regarding evaluation of AES candidates on smart-cards. In *Second Advanced Encryption Standard (AES) Candidate Conference*, Rome, Italy, 1999.
6. EAN International and the Uniform Code Council, Note to Editors, http://www.ean-int.org/index800.html
7. D. Engels. The Reader Collision Problem. Technical Report. MIT-AUTOID-WH-007, 2001. http://www.autoidcenter.org/research/MIT-AUTOID-WH-007.pdf.
8. K. Finkenzeller. *RFID Handbook,* John Wiley & Sons. 1999.
9. H. Gobioff, S. Smith, J.D. Tygar, and B. Yee. Smart cards in hostile environments. In *2nd USENIX Workshop on Elec. Commerce*, 1996.
10. J. Hoffstein, J. Pipher, and J.H. Silverman. NTRU: A ring-based public key cryptosystem. *Lecture Notes in Computer Science*, volume 1423, 1998.
11. International Telecommunications Union. Radio Regulations, Vol. 1, 1998.
12. B.S. Kaliski Jr. and M.J.B. Robshaw. Comments on some new attacks on cryptographic devices. RSA Laboratories' Bulletin No. 5, July 14, 1997. Available from http://www.rsasecurity.com/rsalabs/bulletins/.
13. NTRU. GenuID. http://www.ntru.com/products/genuid.htm.
14. S. Sarma, K. Ashton, D. Brock. The Networked Physical World, Technical Report MIT-AUTOID -WH-001, 1999. http://www.autoidcenter.org/research/MIT-AUTOID-WH-001.pdf.
15. S. Sarma. Towards the 5 cent Tag, Technical Report MIT-AUTOID -WH-006, 2001. http://www.autoidcenter.org/research/MIT-AUTOID-WH-006.pdf.
16. TAMPER Lab. University of Cambridge Tamper and Monitoring Protection Engineering Research Lab, http://www.cl.cam.ac.uk/Research/Security/tamper.
17. T. Scharfeld. An Analysis of the Fundamental Constraints on Low Cost Passive Radio-Frequency Indentification System Design. MS Thesis, Department of Mechanical Engineering, Massachusetts Institue of Technology, Cambridge, MA 02139, 2001.
18. S.H. Weigart. Physical security devices for computer subsystems: A survey of attacks and defences. *CHES 2000*, *Lecture Notes in Computer Science*, volume 1965, pages 302–317, 2000.
19. World Wide Web Consortium. http://www.w3c.org/SOAP/
20. World Wide Web Consortium. http://www.w3c.org/XML/

A New Class of Invertible Mappings

Alexander Klimov and Adi Shamir

Computer Science department, The Weizmann Institute of Science
Rehovot 76100, Israel
{ask,shamir}@wisdom.weizmann.ac.il

Abstract. Invertible transformations over n-bit words are essential ingredients in many cryptographic constructions. When n is small (e.g., $n = 8$) we can compactly represent any such transformation as a lookup table, but when n is large (e.g., $n = 64$) we usually have to represent it as a composition of simpler operations such as linear mappings, S-P networks, Feistel structures, etc. Since these cryptographic constructions are often implemented in software on standard microprocessors, we are particularly interested in invertible univariate or multivariate transformations which can be implemented as small compositions of basic machine instructions on 32 or 64 bit words. In this paper we introduce a new class of provably invertible mappings which can mix arithmetic operations (negation, addition, subtraction, multiplication) and boolean operations (not, xor, and, or), are highly efficient, and have desirable cryptographic properties. In particular, we show that for any n the mapping $x \to x + (x^2 \vee C)$ $(mod\ 2^n)$ is a permutation with a single cycle of length 2^n iff both the least significant bit and the third least significant bit in the constant C are 1.

1 Introduction

Block ciphers are among the most useful constructions in modern cryptography. They are usually constructed by repeatedly applying a simple invertible round function, which should be nonlinear with good avalanche properties. Since most block ciphers are now implemented in software on standard microprocessors, it is crucial to choose a round function which can be efficiently implemented with very few machine instructions. The importance of software-based efficiency was clearly demonstrated during the AES selection process [9] [11].

Since modern microprocessors have extremely fast arithmetic and boolean operations on 32 or 64 bit words, we would like to choose round functions which use such operations on complete words rather than operations that manipulate individual bits or bytes. In addition, it is a good idea to mix arithmetic and boolean operations in order to avoid pure algebraic or bit-oriented attacks, and to reduce the probabilities of linear and differential attacks.

The invertibility of round functions is of course a necessary condition in block ciphers, but it is also important in other constructions such as stream ciphers and hash functions: If we repeatedly apply a round function to the internal state, we want to prevent an incremental loss of entropy which can lead to early looping

B.S. Kaliski Jr. et al. (Eds.): CHES 2002, LNCS 2523, pp. 470–483, 2003.

of the output stream or to undesirable collisions among the hashed values. In these applications we do not actually want to run the round function backwards — we just need a guarantee that the forward mapping is one-to-one. In other applications such as the encryption of data stored on backup tapes, we have to encrypt huge amounts of data once a day, but rarely decrypt the result. In these cases the software efficiency of the backward mapping is relatively unimportant, and we can use slightly asymmetric schemes in which the process of decryption is slower (by a small factor) than the process of encryption. Such an asymmetry can even help against some cryptanalytic attacks if the opponent only knows the ciphertext and tries to perform exhaustive decryption under all possible keys.

The goal of this paper is to design new types of provably invertible round functions with good cryptographic properties and extremely efficient software implementations. They mix a small number of arithmetic and boolean operations on full machine words, and cannot be inverted by executing the same program backwards with inverse operations. Instead, we can use the mathematical proof of invertibility in order to construct a completely different (and somewhat slower) inversion algorithm, if we ever have to run the mapping backwards.

The best way to introduce the new idea is to describe several simple examples which demonstrate the fine line between invertible and noninvertible mappings. Let 1 denote the constant word $0 \cdots 01$, and assume that all the operations are carried out on n-bit words. Then the following univariate mappings are invertible:

$$x \to x + 2x^2, \quad x \to x + (x^2 \vee 1), \quad x \to x \oplus (x^2 \vee 1)$$

whereas the following closely related variants are noninvertible:

$$x \to x + x^2, \quad x \to x + (x^2 \wedge 1), \quad x \to x + (x^3 \vee 1)$$

The same techniques can be used to prove the invertibility of multivariate mappings such as:

$$(x, y) \to (x \oplus 2(x \wedge y), (y + 3x^3) \oplus x).$$

2 Notations and Definitions

In this section we introduce our notations and basic definitions. We denote the set $\{0, 1\}$ by $\mathbb{B}$. We use the same symbol x to denote the n-bit vector $([x]_{n-1}, [x]_{n-2}, \cdots, [x]_0)$ in $\mathbb{B}^n$ and an integer modulo 2^n, with the usual conversion rule: $x \longleftrightarrow \sum_{i=0}^{n-1} 2^i [x]_i$.

The basic operations we allow in our constructions are the following arithmetic and boolean operations:

Definition 1. *Let x and y be n-bit input variables. A function $\phi : \mathbb{B}^{k \times n} \to \mathbb{B}^n$ is called a* **primitive function** *if $k = 1$ and $\phi(x)$ is one of the operations of negation: $\phi(x) = -x \pmod{2^n}$ and complementation: $[\phi(x)]_i = \overline{[x]}_i$, or if $k = 2$ and $\phi(x, y)$ is one of the operations of addition: $\phi(x, y) = x + y \pmod{2^n}$, subtraction: $\phi(x, y) = x - y \pmod{2^n}$, multiplication: $\phi(x, y) = x \cdot y \pmod{2^n}$, xor: $[\phi(x, y)]_i = [x]_i \oplus [y]_i$, and: $[\phi(x, y)]_i = [x]_i \wedge [y]_i$, or: $[\phi(x, y)]_i = [x]_i \vee [y]_i$.*

Note that left shift is allowed (since it is equivalent to multiplication by a power of two), but right shift and circular rotations are not allowed in our framework, even though they are available as basic machine instructions in most microprocessors.

When we deal with an $m \times n$ bit matrix p, we start numbering its rows and columns from zero, and refer to its i-th row as $p_{i-1,\star}$ and to its j-th column as $p_{\star,j-1}$. Our multivariate transformations operate on inputs represented by the bit matrix $p_{\star,\star}$ (for plaintexts) and produce outputs represented by the matrix $c_{\star,\star}$ (for ciphertexts), where the value of the i-th variable is represented by the successive bits in row $i-1$ in the matrix.

Definition 2. Parametric functions *are functions* $g(x_1, \cdots, x_a; \alpha_1, \cdots, \alpha_b)$ *whose arguments are split by a semicolon into* **inputs** *(the* x_i*'s) and* **parameters** *(the* α_j*'s). A* **Parametric Invertible Function (PIF)** *is a parametric function whose input/output relationship is invertible for any fixed value of the parameters:*

$$\forall \alpha \; \forall x, y : g(x;\alpha) = g(y;\alpha) \iff x = y$$

3 Previous Constructions of Invertible Mappings

To test the applicability of previous construction techniques, we will try to use each technique in order to prove the claim that the univariate mapping $x \rightarrow x \oplus (x^2 \vee 1)$ over n-bit words is invertible for any n.

The simplest way to construct a large class of invertible mappings is to compose other invertible operations in various orders. A typical example is an S-P network in which we alternately apply invertible substitution and permutation operations. However, in our running example the global invertibility of the mapping $x \rightarrow x \oplus (x^2 \vee 1)$ cannot be explained by the local invertibility of its basic operations, since squaring and or'ing with 1 are non-invertible operations.

A second technique for constructing invertible transformations was proposed by Feistel [3] in 1973 and used in many block ciphers such as DES [7]. Unlike the first technique, it can use arbitrary non-invertible functions f in the construction by using the PIF $g(l;r) = l \oplus f(r)$. The basic Feistel construction maps two inputs (l, r) into $(r, l \oplus f(r))$, and the full Feistel construction iterates this mapping any number of times. This idea was generalized in several directions, and in [13] Schneier and Kelsey gave "a taxonomy of Feistel networks" which include *unbalanced Feistel networks* (UFN), *homogeneous* and *heterogeneous* UFNs, *incomplete* and *inconsistent* networks, etc. However, the Feistel construction requires at least two variables, and thus it also fails to explain the invertibility of our running example $x \rightarrow x \oplus (x^2 \vee 1)$ $(mod\ 2^n)$.

A third method of constructing invertible multivariate transformations is called triangulation, and is motivated by the way we use Gauss elimination to solve a system of linear equations by first triangulating its matrix of coefficients, and then computing the values of the variables in a sequential way. For example, in 1993 Shamir proposed [14] two constructions of birational transformations, in

which both the mapping and its inverse are defined by ratios of small polynomials. The core of his first scheme was the fact that the mapping

$$(x_0, \ldots, x_{k-1}) \mapsto (f_0(x_0, \ldots, x_{k-1}), \ldots, f_{k-1}(x_0, \ldots, x_{k-1})) \tag{1}$$

is uniquely invertible for almost all inputs if each f_i has the following form:

$$f_i(x_0, \ldots, x_{k-1}) = g_i(x_0, \ldots, x_{i-1})x_i + h_i(x_0, \ldots, x_{i-1}) \pmod N, \tag{2}$$

where g_i and h_i are arbitrary non-zero polynomial functions modulo a large RSA modulus N. The inversion process solves a series of trivial linear equations in one variable, as demonstrated in the following mapping from (x, y, z) to (x', y', z'):

$$\begin{aligned} x' &= 7x \pmod N \\ y' &= (x^2 + 12x + 7)y + x \pmod N \\ z' &= (xy + 4x^3 + y)z + xy^2 + y^3 \pmod N \end{aligned} \tag{3}$$

Given the output (x', y', z'), we can derive $x = \frac{x'}{7} \pmod N$ from the first equation, then use it to derive $y = \frac{y'-x}{x^2+12x+7}$ from the second equation, and finally use x and y to derive $z = \frac{z'-(xy^2+y^3)}{xy+4x^3+y}$ from the third equation. The derivation fails only when one of the numeric denominators is not relatively prime to N, which is an extremely rare event. This approach could easily handle arbitrary functions g_i and h_i which mix arithmetic and boolean operations, since these functions have to be evaluated only in the forward direction. However, it cannot explain the invertibility of our univariate running example, or the invertibility of the bivariate example $(x, y) \rightarrow (x \oplus 2(x \wedge y), (y + 3x^3) \oplus x)(mod\ 2^n)$, which can not be simplified into a triangular form.

A fourth approach is to concentrate on mappings which contain only the arithmetic operations of addition, subtraction and multiplication. Such polynomial mappings have a rich algebraic structure, and their invertible cases are called permutation polynomials. The problem of characterizing permutation polynomials over various domains is a very well studied problem in mathematics. For example, Hermite made considerable progress in characterizing univariate permutation polynomials modulo a prime p, and Dickson found an effective characterization for all univariate polynomials with degrees smaller than 5. However, even today the problem is not completely resolved for high degree polynomials modulo a large prime p.

Over the ring of integers modulo 2^n the problem seems to be much simpler. In 1997, the designers of the AES candidate RC6 [11] were looking for an invertible mapping with good mixing properties which could be implemented in software with a small number of arithmetic operations on 32 bit words. They chose the mapping $x \rightarrow x + 2x^2\ (mod\ 2^{32})$, and used an ad-hoc algebraic technique to prove that it is a permutation polynomial. In 1999, Rivest [12] generalized this proof technique and provided the following complete characterization of all the univariate permutation polynomials modulo 2^n:

Theorem 1. *Let* $P(x) = a_0 + a_1x + \cdots + a_dx^d$ *be a polynomial with integral coefficients. Then* $P(x)$ *is a permutation polynomial modulo* 2^n*,* $n > 2$ *if and only if* a_1 *is odd,* $(a_2 + a_4 + \cdots)$ *is even, and* $(a_3 + a_5 + \cdots)$ *is even.*

Unfortunately, his algebraic proof technique cannot be generalized to mappings which mix arithmetic and boolean operations, and thus it can not prove the invertibility of $x \to x \oplus (x^2 \vee 1)$ or of $(x, y) \to (x \oplus 2(x \wedge y), (y + 3x^3) \oplus x)$.

4 The New Construction

To understand the new construction, assume that each input variable has n bits, and we place the m input variables in the m rows of an $m \times n$ bit matrix. To compute the mapping in the forward direction, we apply to the rows of this matrix a sequence of primitive machine instructions. To compute the mapping in the backward direction, we cannot sequentially undo the effects of the row operations on the bit matrix, since some of them can be non-invertible. Instead, we derive the backward mapping by analysing the overall effect of the forward mapping on the columns of the bit matrix. Note that our goal is not to compute the inverse mapping, but only to prove that it is uniquely defined for any output matrix. Consequently, we don't actually have to perform the time consuming operation of changing the way bits are packed into words from row major to column major representation.

The idea of switching from row operations to column operations was used in 1997 by Eli Biham [2] to speed up the computation of DES. He placed 64 plaintexts in a 64×64 bit matrix, and encrypted all of them simultaneously on an alpha microprocessor (with 64 bit words) by operating on the vertical bit slices instead of on the horizontal plaintexts in the matrix. The row-oriented and column-oriented algorithms look completely different (in particular, one of them uses table lookups while the other uses only boolean operations to implement the S-boxes), but they have the same overall effect. In Biham's paper both the encryption and the decryption operations are carried out on the columns of the matrix, whereas in our case the forward mapping is performed on the rows and the backward mapping is performed on the columns of the matrix. The crucial observation which makes our construction possible is the fact that all the primitive functions can be applied to truncated versions of their inputs and provide truncated versions of the original outputs. To formalize this notion, we need the following definition:

Definition 3. *A function* f *from* $\mathbb{B}^{m \times n}$ *to* $\mathbb{B}^{l \times n}$ *is called a* **T-function** *if the* k*-th column of the output* $[\phi(x)]_{\star,k-1}$ *depends only on the first* k *columns of the input:* $[x]_{\star,0}, \ldots, [x]_{\star,k-1}$.

Consider, for example, the addition function: $x + y = z \pmod{2^n}$. The least significant bit of the result depends only on the least significant bits of the operands: $[z]_0 = [x]_0 \oplus [y]_0$. The second bit depends on the first and second bits of the operands $[z]_1 = [x]_1 \oplus [y]_1 \oplus \alpha$, where α is the carry into the second bit position which is defined by the least significant bits of x and y. The same holds

for all the other bits — in order to calculate bit number k of the result we only need to know bits $0, 1, \ldots, k$ of the operands. This is also true for subtraction and multiplication, and trivially true for all the bit-oriented boolean operations. We can thus conclude that all the primitive functions we allow in our constructions are T-functions, whereas the excluded operations of bit rotations or right shifts are not T-functions. The composition of two T-functions is also a T-function, and thus any mapping defined by a sequence of primitive functions applied to the input bit matrix is also a T-function.

The name T-function refers to the triangular dependence between the columns of the operands. Note that this **implicit triangulation** is different from the **explicit triangulation** in multivariate mappings which was mentioned in the previous section. In fact, explicit and implicit triangulations are row/column dual properties: A mapping has an explicit triangular shape if its i-th variable (row) is combined only with previous variables in the given expressions, and has an implicit triangular shape if its i-th bit slice (column) is combined only with previous bit slices by the T-functions.

To demonstrate the new construction, let us prove the following result:

Theorem 2. *For any composition f of primitive functions, the mapping $x \rightarrow x + 2 \cdot f(x) \ (mod\ 2^n)$ is invertible.*

Note that this class includes the RC6 function $x \rightarrow x + 2x^2 \ (mod\ 2^n))$, along with more complicated examples such as $x \rightarrow x + 2 \cdot ((x \wedge x^2) \vee (\bar{x} \wedge x^3))$ in which the bits of x are used to select between the bits of x^2 and x^3.

Proof. Our goal is to recover x from a given value of $y = x + 2 \cdot f(x) \ (mod\ 2^n)$. Assume that we already know bits $0, 1, \ldots, i-1$ of x, and our goal is to compute bit number i. Since f is a composition of primitive functions, we can calulate bits $0, 1, \ldots, i-1$ of $f(x)$ by evaluating all the arithmetic operations in f modulo 2^i, and truncating all the boolean operations after bit $i-1$. By leftshifting this result, we obtain all the values of bits $0, 1, \ldots, i$ of $2 \cdot f(x)$. When we reduce the equation $y = x + 2 \cdot f(x)$ modulo 2^{i+1}, the only unknown bit in the truncated equation is the i-th bit in the first occurence of x in this expression, and thus we can easily derive it from the equation. By repeating this process, we can recover all the bits of x from lsb to msb in a unique way for any f which is a composition of primitive functions.

Mappings defined by T-functions make it possible to consider vertical bit slices in sequential order, but this notion does not guarantee invertibility (for example, the mapping $x \rightarrow x + x^2 \ (mod\ 2^n)$ is a T-function but it maps both -1 and 0 to 0). To guarantee invertibility, we combine the notions of Feistel networks and triangulation in the following way:

Definition 4. *A* **triangular Feistel Network (TFN)** *is any mapping over bit matrices described by:*

$$\begin{aligned}&(p_{\star,0}, p_{\star,1}, \ldots, p_{\star,n-1}) \rightarrow \\ &(g_0(p_{\star,0}), g_1(p_{\star,1}; p_{\star,0}), \ldots, g_{n-1}(p_{\star,n-1}; p_{\star,0}, p_{\star,1}, \ldots, p_{\star,n-2})),\end{aligned} \tag{4}$$

in which g_0 is an invertible function and the g_i's for $i > 0$ are PIF's.

In fact, it is easy to prove that *any* T-function which is invertible is a TFN, and thus this is the most general construction of invertible T-functions. We prove this claim for bivariate mappings:

Lemma 1. *Let $\mathcal{A}$ and $\mathcal{B}$ be finite sets, and let $\phi : \mathcal{A}\times\mathcal{B} \to \mathcal{A}\times\mathcal{B}$ be an invertible mapping of the form: $(a, b)\to(f(a), g(b, a))$. Then f is invertible and $g(b; a)$ is a PIF.*

Proof. Suppose that f is not invertible, so $|\{f(a)|a \in \mathcal{A}\}|$ is strictly smaller than $|\mathcal{A}|$. Then $|\{(f(a), \phi(a, b))|a \in \mathcal{A}, b \in \mathcal{B}\}| < |\mathcal{A}|\cdot|\mathcal{B}|$, which contradicts the invertibility of ϕ. If g is not a PIF, then there exists a, b_1 and b_2, s.t. $g(b_1; a) = g(b_2; a)$. Consequently, ϕ maps (a, b_1) and (a, b_2) to the same pair of values.

5 Testing the Invertibility of Parametric Functions

In this section we consider the related issues of how to construct invertible mappings from primitive functions, and how to test the invertibility of a given mapping of this type. The general problem is quite difficult, but in practice we will be interested mostly in univariate or bivariate mappings which are defined by a small number of machine instructions, and in such cases our techniques seem to work very efficiently.

Given such a mapping, we would like to test whether it is a TFN. We first have to test whether the mapping of the least significant bit slice is invertible. The simplest technique is to try the 2^m possible combinations of the least significant bits of the m variables, which is trivial for $m \leq 20$. Next we have to test whether for each $i > 0$, the parametric mapping $g_i(p_{\star,i}; p_{\star,0}, p_{\star,1}, \ldots, p_{\star,i-1})$ is a PIF. This cannot be tested by exhaustive search, since the number of cases we have to consider grows exponentially with i. Instead, we ignore the actual values of additive parameters when we evaluate the primitive functions by using the following recursive rules:

Lemma 2. *If $\circ$ is a boolean operation, a, b are words, and $i > 0$, then*

$$[a \circ b]_0 = [a]_0 \circ [b]_0 \tag{5}$$
$$[a \circ b]_i = [a]_i \circ [b]_i \tag{6}$$
$$[a + b]_0 = [a]_0 \oplus [b]_0 \tag{7}$$
$$[a + b]_i = [a]_i \oplus [b]_i \oplus \alpha \tag{8}$$
$$[a - b]_0 = [a]_0 \oplus [b]_0 \tag{9}$$
$$[a - b]_i = [a]_i \oplus [b]_i \oplus \alpha \tag{10}$$
$$[a \cdot b]_0 = [a]_0 [b]_0 \tag{11}$$
$$[a \cdot b]_i = [a]_i [b]_0 \oplus [a]_0 [b]_i \oplus \alpha \tag{12}$$
$$\left[a^k\right]_0 = [a]_0 \ \ for\ any\ k > 0 \tag{13}$$
$$\left[a^k\right]_i = [a]_i [a]_0 \oplus \alpha\ for\ any\ odd\ k > 0 \tag{14}$$
$$\left[a^k\right]_i = \alpha\ for\ any\ even\ k > 0 \tag{15}$$

where the α's are parameters denoting carries from previous bit positions.

Proof. We only prove the last claim — all the other cases use similar ideas. Since k is even $t = k/2$ is an integer, and thus we can use the claim about $[a \cdot b]_i$ to get $\left[a^k\right]_i = [a^t \cdot a^t]_i = [a^t]_i [a^t]_0 \oplus [a^t]_0 [a^t]_i \oplus \alpha = \alpha$. Consequently, any bit position in a^k (except the least significant one) depends only on lower bit positions in a, whose combination is considered as an additive parameter α.

To use these rules, we consider each expression in the multivariate mapping as a tree with variables at the leaves, primitive functions at the internal nodes, and one of the outputs at the root. We then compute the i-th bit of the output by traversing the tree and evaluating its slice operations. This process is algorithmically similar to formal differentiation of algebraic expressions. To demonstrate the process, consider the following bivariate mapping from the introduction:

$$x' = x + 2(x \wedge y) \tag{16}$$
$$y' = (y + 3x^3) \oplus x \tag{17}$$

It is not clear, a-priori, whether this mapping is invertible. To test it, we first compute the i-th bit (for any $i > 0$) of the first output as a parametric function:

$$\begin{aligned}[x']_i &= [x]_i \oplus [2(x \wedge y)]_i \\ &= [x]_i \oplus [2]_i [x \wedge y]_0 \oplus [2]_0 [x \wedge y]_i \oplus \delta \\ &= [x]_i \oplus [2]_i \alpha \oplus 0 \oplus \delta = [x]_i \oplus \theta\end{aligned}$$

where δ, α and θ are parameters which can depend on the previous bit slices but not on the i-th bit of any one of the input variables. In a similar way, we can derive the parametric representation of the i-th bit (for any $i > 0$) of the second output as

$$\begin{aligned}[y']_i &= \left[y + 3x^3\right]_i \oplus [x]_i = [y]_i \oplus \left[3x^3\right]_i \oplus \gamma \oplus [x]_i \\ &= [y]_i \oplus [3]_i \left[x^3\right]_0 \oplus [3]_0 \left[x^3\right]_i \oplus \delta \oplus \gamma \oplus [x]_i \\ &= [y]_i \oplus \alpha \oplus [x]_i [x]_0 \oplus [x]_i \oplus \epsilon = [y]_i \oplus \beta [x]_i \oplus \tau\end{aligned}$$

It is now easy to see that the two parametric slice mappings:

$$\begin{aligned}[x']_i &= [x]_i \oplus \theta \\ [y']_i &= [y]_i \oplus \beta [x]_i \oplus \tau\end{aligned} \tag{18}$$

are invertible for any $i > 0$ and for any choice of the multiplicative parameter β and the additive parameters θ and τ, and thus this mapping is a PIF. In addition, it is easy to test that the mapping defined by the least significant slice is invertible, and thus the original bivariate transformation over n bit words is invertible.

As another example, let us reprove in our framework Rivest's characterization in Theorem 1 of permutation polynomials modulo 2^n. First we check the case of $i = 0$: Since all the powers of x are equal to x modulo 2, we can simplify $[P(x)]_0 = \left[a_0 + a_1 x + \cdots + a_d x^d\right]_0 = [a_0]_0 + [(a_1 + \cdots + a_d)]_0 [x]_0$, which is invertible modulo 2 iff $a_1 + \cdots + a_d$ is odd.

Next we have to check which polynomials have invertible slice mappings for $i > 0$. By using our recursive simplification technique we get the following parametric representations:

$$\left[a_1 x^1\right]_i = [a_1]_i [x]_0 \oplus [a_1]_0 [x]_i \oplus \alpha_1 = [a_1]_0 [x]_i \oplus \alpha_2$$

For higher even powers of x we get:

$$\left[a_k x^k\right]_i = [a_k]_i \left[x^k\right]_0 \oplus [a_k]_0 \left[x^k\right]_i \oplus \alpha_3 = \alpha_4$$

and for higher odd powers of x we get:

$$\left[a_k x^k\right]_i = [a_k]_i \left[x^k\right]_0 \oplus [a_k]_0 \left[x^k\right]_i \oplus \alpha_5$$

$$= [a_k]_i [x]_0 \oplus [a_k]_0 ([x]_0 [x]_i \oplus \alpha_6) \oplus \alpha_5 = [a_k]_0 [x]_0 [x]_i \oplus \alpha_7$$

Bringing it all together, $[P(x)]_i = [a_1]_0 [x]_i \oplus \bigoplus_{odd\ k \geq 3} [a_k]_0 [x]_0 [x]_i \oplus \alpha$. To make sure that it is a PIF, this slice mapping should be invertible for both values of $[x]_0$. For $[x]_0 = 0$ we get the condition $[a_1]_0 = 1$, and for $[x]_0 = 1$ we get the condition $a_1 \oplus a_3 \oplus a_5 \oplus \cdots = 1$. Together with the condition $a_1 \oplus a_2 \oplus a_3 \cdots = 1$ from slice $i = 0$ we get Rivest's characterization of permutation polynomials in an almost mechanical way. As a bonus, we can use exactly the same proof to get exactly the same characterization of invertible mappings of the form $P(x) = a_0 \oplus a_1 x \oplus \cdots \oplus a_d x^d$ (which could not be handled by Rivest's algebraic technique), since there is no difference between + and $\oplus$ in the parametrized slice mappings.

As demonstrated by these examples, the invertibility of many types of mappings can be reduced to the invertibility of systems of linear equations modulo 2 with additive and multiplicative parameters (even when the original mapping is highly nonlinear and combines arithmetic and boolean operations). If we have only additive parameters, this invertibility does not depend on the actual values of the parameters and can be decided by evaluating a single determinant. Multiplicative parameters are more problematic, as we have to test the invertibility of parametrized matrices (modulo 2) for all the possible 0/1 values of the parameters. Since the number of parameters is usually small, exhaustive search is feasible, but this test can have a one-sided error: If all the parametrized matrices are invertible then the original mapping is invertible, but in the other direction noninvertible matrices may occur only for combinations of parameter values which can never happen for actual inputs. Consequently, this test provides a sufficient but not a necessary condition for invertibility.

An interesting special case happens when the parametrized coefficient matrix is triangular with 1's along its diagonal. Such matrices are invertible modulo 2 regardless of how parameters occur in its off-diagonal entries. This is demonstrated in the system of equations 18, which is invertible for any choice of the multiplicative parameter β.

When we consider the dual problem of constructing invertible mappings, we should choose primitive operations whose slice mappings are linear systems of equations with either no multiplicative parameters or with a triangular collection of multiplicative parameters. Alternatively, we can start with any simple

combination of primitive functions and use our technique to diagnose its invertibility. A failed test can indicate how the mapping should be modified in order to make it invertible. As a typical example, consider the mapping $x \to x \oplus x^2$. Its i-th slice mapping for any $i > 0$ is the invertible $[x']_i = [x]_i \oplus \alpha$, but for $i = 0$ we get the noninvertible $[x']_0 = [x]_0 \oplus [x]_0 = 0$. To solve this problem, all we have to do is to tweak the first slice mapping without affecting the other slice mappings. The simplest way to do this is to "or" the constant 1 to x^2, which changes only the first slice mapping to the invertible $[x']_0 = [x]_0 \oplus 1$. This leads in a natural way to the interesting mapping $x \to x \oplus (x^2 \vee 1)$, which is invertible for any word size.

6 Cryptographic Applications

So far we have defined a large class of complex mappings which can use a mixture of standard arithmetic and boolean operations on arbitrarily long words. They can be implemented very efficiently in software, and their invertibility can be easily tested. In this section we describe one of their cryptographic applications.

Since the actual inversion of these invertible mappings is quite inefficient, we would like to use them in situations where they are only executed in the forward direction. A typical example of such an application is their iterated use on the state S of a pseudo random number generator (PRNG). The initial state S_0 of the generator is derived from the key k, and after each iteration the PRNG outputs some simple function $\mathcal{O}(S_i)$ of its current state (e.g., its most significant bit or byte). It is essential to use invertible update operations to avoid an incremental loss of state entropy, but the update function is never run backwards in this application.

Our goal is to choose a good state update function which uses the smallest possible number of primitive functions, in order to achieve the highest possible speed in software implementation. An exhaustive analysis of all the mappings defined by up to two primitive functions indicate that there are only 8 families of invertible mappings of this type: $x \cdot C'$, $x + C$, $x \oplus C$, $(x + C_1) \oplus C_2$, $x \cdot C' + C$, $x \cdot C' \oplus C$, $x \cdot C'' \oplus x$, $(x \oplus C) \cdot C'$, where C, C_1 and C_2 are arbitrary constants, C' is an arbitrary odd constant, and C'' is an arbitrary even constant. Unfortunately, all these families are useless, either because they have a simple linear structure, or because they have very short cycles. For example, the states defined by the mapping $x \oplus C$ just oscillate between two possible states, and thus the output of the PRNG is highly predictable. Consequently, we have to use at least 3 primitive operations to update the state of the PRNG.

A very desirable property of an invertible state update mapping in this application is that all the 2^n possible states should be connected by a single cycle of length 2^n. In the case of linear feedback shift registers, this is almost achieved by using a primitive feedback polynomial (which leaves only the all-zero state in a second cycle). In general, it is very difficult to determine the cycle structure of complex invertible mappings, but the possible cycle structures of T-functions are severely limited by the following observation:

Lemma 3. *Let $f_n(x) = f(x) \pmod{2^n}$ be an invertible T-function. Then for each cycle in f_{n-1} of length l, there are either two cycles of length l or one cycle of length $2l$ in f_n. Consequently, if f has any fixed point then for any $i > 0$ there are at least 2^{i+1} points that belong to cycles of length at most 2^i.*

For $n = 0$ the only possible forms of f_0 are $f_0(x) = x$ or $f_0(x) = x \oplus 1$ which have cycles of length 1 and 2. Consequently, the length of any cycle of an invertible T-function is a power of 2. Another trivial corollary is that we should only use functions which have no fixed points at all, since otherwise there are at least two fixed points, at least four points on cycles of length bounded by 2, etc.

Consider, for example, the well known RC6 function $f_n(x) = x(2x + 1) \pmod{2^n}$. Unfortunately, it has a fixed point $x = 0$, and thus it has several small cycles. In fact, its cycle structure is much worse than the general bound given above (for example, for $n = 3$, 4 out of the 8 possible inputs are fixed points). It is easy to show that for any n this f_n has exactly $2^{\frac{n}{2}}$ fixed points, exactly $2^{\frac{n}{2}-2}$ points on cycles of length 2, etc. This makes it very unsuitable for our PRNG application.

A particularly interesting family of mappings is $x \to x + (x^2 \vee C)$ for various constants C and word sizes n, for which we can prove:

Theorem 3. *The mapping $f(x) = x + (x^2 \vee C)$ over n bit words is invertible if and only if the least significant bit of C is 1. For $n \geq 3$ it is a permutation with a single cycle if and only if both the least significant bit and the third least significant bit of C are 1.*

Proof. The first claim is trivial. For any bit slice $i > 0$ $[f(x)]_i = [x]_i \oplus \alpha$ (where α is a parameter) is invertible, and thus the only possible problem is the least significant bit slice $i = 0$. There are two cases to consider: If $[C]_0 = 0$, $[f(x)]_0 = [x]_0 \oplus [x]_0 = 0$, whereas if $[C]_0 = 1$, $[f(x)]_0 = [x]_0 \oplus 1$. Consequently, the invertibility of $f(x)$ depends only on the least significant bit of the constant C.

The second claim is more complicated, and we prove it by induction on the word length n. For $n = 3$ we can show by simple enumeration that the function $f(x) = x + (x^2 \vee C) \pmod{2^3}$ has a single cycle if and only if $C = 5$ or $C = 7$. Note that this already implies that for any $n \geq 3$ $f(x)$ has more than one cycle whenever $C \notin \{5, 7\} \pmod 8$, and thus we only have to prove that the permutation has a single cycle whenever $C \in \{5, 7\} \pmod 8$ (i.e., whenever both the least significant bit and the third least significant bit of C are 1).

The length of any cycle of any T-permutation is of the form 2^k for some k. Our strategy is to show that $\forall n \exists x : \left[f^{(2^{n-1})}(x)\right]_{n-1} = [x]_{n-1} \oplus 1$, since this implies the existence of some cycle of length larger than 2^{n-1}, and the only possible cycle length is then the full 2^n.

Assume by induction that f has only one cycle for $n-1$. Consider the sequence x_0, $x_1 = f(x_0) = x_0 + (x_0^2 \vee C)$, $x_2 = f^{(2)}(x_0) = f(x_1) = x_1 + (x_1^2 \vee C) = x_0 + (x_0^2 \vee C) + (x_1^2 \vee C)$, …, $x_{2^{n-1}} = f^{(2^{n-1})}(x) = x_0 + \sum_{i=0}^{2^{n-1}-1}(x_i^2 \vee C)$. From the assumption that there is only one cycle for $n - 1$, we know that $\{x_i \bmod 2^{n-1}\}_{i=0}^{2^{n-1}-1}$ is just a permutation of $\{i\}_{i=0}^{2^{n-1}-1}$. Since $\left[x^2\right]_{n-1}$ does not depend

on $[x]_{n-1}$ the set $\{(x_i^2) \bmod 2^n\}_{i=0}^{2^{n-1}-1}$ is the same as $\{((x_i \bmod 2^{n-1})^2) \bmod 2^n\}_{i=0}^{2^{n-1}-1}$. So $f^{(2^{n-1})}(0) \bmod 2^n = \sum_{j=0}^{2^{n-1}-1}(j^2 \vee C) \bmod 2^n$. An expression $x \vee 2^i$ is equal either to x if $[x]_i = 1$ or to $x + 2^i$ otherwise. Similarly $j^2 \vee C = j^2 + \sum_{i:[C]_i=1 \wedge [j^2]_i=1} 2^i$ and $\sum_{j=0}^{2^{n-1}-1}(j^2 \vee C) = 2^{n-1}(\sum_{i:[C]_i=1} 2^j z_j)$. Here z_i denotes the probability of '0' in bit number i of x^2 for random x, i.e., $\frac{\#\{[x^2]_i=0\}_{x=0}^{2^i-1}}{2^i}$. It is easy to show that $z_0 = \frac{1}{2}$, $z_1 = 1$ and $\forall i > 1 : z_i = \frac{1}{2}(1 + 2^{-\lfloor \frac{i}{2} \rfloor})$. [1] The formula for the sum of squares is $\sum_{j=0}^{k} j^2 = \frac{k^3}{3} + \frac{k^2}{2} + \frac{k}{6}$, so $\sigma = f^{(2^{n-1})}(0) = (\frac{2^{3n}}{3} + \frac{2^{2n}}{2} + \frac{2^n}{6}) - 2^{2n} + 2^{n-1}(\sum_{j:[C]_j=1} 2^j z_j)$ and we want to prove that $[\sigma]_{n-1} = 1$. This is determined by the least significant bit of $\frac{\sigma}{2^{n-1}} = \frac{1}{3}(2^{2n+1} + 2^n + 1) - 2^{n+1} + \sum_{j:[C]_j=1} 2^j z_j$. In binary $\frac{1}{3} = 0.(10)_2$ and $\frac{2^k}{3}$ is either $1010\ldots10.(10)_2$ or $1010\ldots1.(01)_2$, so the infinite fraction $\frac{1}{3}(2^{2n+1} + 2^n)$ has either the form $0.(10)_2$ (if i is odd) or the form $1.(1)_2$ (if i is even). In both cases it represents a number which is equal to '0' modulo 2. In addition $2^j z_j = 0 \pmod 2$ for all j except '0' and '2', for which the values are $\frac{1}{2}$ and 3, respectively. Finally $\frac{\sigma}{2^{n-1}} \pmod 2 = \frac{1}{3} + \frac{1}{2} + 3 \pmod 2 = 1$ and thus for $x = 0$, $\left[f^{(2^{n-1})}(0)\right]_{n-1} = 1 = [0]_{n-1} \oplus 1$ which completes the proof.

In particular, $x + (x^2 \vee 1)$ is invertible but has multiple cycles of various sizes, whereas $x + (x^2 \vee 5)$ has only one cycle of length 2^n and thus it is a better choice for a PRNG application.

V. S. Anashin [1] published a somewhat related analysis of the cycle structure of functions defined over p-adic numbers. He used notions of ergodicity and measure preservation in order to obtain topological characterization of such mappings. In particular, he proved a theorem which could be translated into our notation in the following way: if $c, r \neq 0 \pmod p$ then $f(x) = d + cx + pv(x) \pmod{p^n}$ is invertible and $g(x) = c + rx + p(v(x+1) - v(x)) \pmod{p^n}$ has only one cycle. For example, the invertibility of the RC6 function $x(2x+1)$ could be shown by this theorem, but neither the invertibility nor the cycle structure of $x + (x^2 \vee 5)$ could be determined by his techniques.

The pure PRNG's produced by invertible T-functions are not cryptographically secure by themselves, but they can serve as excellent building blocks in software based generators (in the same way that linear feedback shift registers are insecure as stand-alone designs, but they can serve as excellent components in hardware based generators). They can have provably long cycles, and unlike the case of LFSR's, there is no need to worry about weak keys that fill the shift

[1] For $i \leq 2$ this could be checked by direct calculation. Every n-bit number x' could be represented as either $2x$ or $2x+1$, where x has $n-1$ bits. In the first case $\Pr[[x'^2]_i = 0 \,|\, [x']_0 = 0] = z_{i-2}$, and in the second $\Pr[[x'^2]_i = 0 \,|\, [x']_0 = 1] = \Pr[[4x^2 + 4x + 1]_i = 0] = \Pr[[x^2]_{i-2} \oplus [x]_{i-2} \oplus \alpha = 0] = \frac{1}{2}$ (where α is a carry), since $[x]_{i-2}$ is independent of other bits (again note that $[x^2]_i$ does not depend on $[x]_i$ for $i > 0$) and equally likely to be 0 and 1. Combining both cases we get an inductive proof of the claim.

register with zeroes and get stuck in a fixedpoint. Even the simplest generators can have excellent statistical properties. For example, we used the statistical test suite [8] which was developed for the AES candidates [10], and we found that the sequence of upper halves (top 32 bits) of the 64 bit numbers defined by the iterated use of $x_{i+1} = x_i + (x_i^2 \vee 5) \pmod{2^{64}}$ is statistically random with significance level $\alpha = 0.01$ (which is better than for some of the AES candidates themselves!). Since each iteration requires only 3 machine instructions, we can use it as an exceptionally fast source of (weak) pseudorandomness.

Note that unlike the case of LFSR's, when we iterate a T-function there is no propagation of information from left to right in the state word. To overcome this problem, we can alternately apply a T-function and a cyclic rotation to the state. The combined mapping is clearly invertible, has both left-to-right and right-to-left information propagation, but it is very difficult to analyse its cycle structure. Consequently, such combinations have to be studied very carefully before they can be adopted in new cryptographic schemes, and in particular one should consider the linear and differential properties of these mappings, the best way to combine them into stronger schemes, the best attacks against such combinations, how many output bits can be extracted from the current state in each iteration, etc.

7 Concluding Remarks

This paper proposes a new cryptographic building block which mixes boolean and arithmetic operations, and analyses some of its cryptographic properties. In particular, we use the multiplication operation (which on many modern microprocessors take about the same time as addition) to thoroughly mix the input bits in a nonlinear way and to enhance the statistical properties of the mapping. In addition, we use the boolean operations to mask the algebraic weaknesses of low degree polynomial mappings, and to control their cycle structures. The resultant mappings have very compact and extremely fast software implementations, and thus they could replace large S-boxes or LFSR's in many software based cryptographic schemes. We believe that as we move from 32 to 64 (and later to 128) bit processors, such wide-word mappings over mixed algebraic structures will become an increasingly natural and attractive alternative to traditional bit and byte oriented designs.

References

1. V. S. Anashin, "Uniformly distributed sequences over p-adic integers", Proceedings of the Int'l Conference on Number Theoretic and Algebraic Methods in Computer Science (A. J. van der Poorten, I. Shparlinsky and H. G. Zimmer, eds.), World Scientific, 1995.
2. E. Biham, "A Fast New DES Implementation in Software", Fast Software Encryption Workshop, 1997
3. H. Feistel, "Cryptography and Computer Privacy," Scientific American, v. 228, n. 5, May 1973, pp. 15–23.

4. V. Furman, "Differential Cryptanalysis of Nimbus", Fast Software Encryption Workshop, 2001
5. H. Lipmaa, S. Moriai, "Efficient Algorithms for Computing Differential Properties of Addition", 2001. Available from `http://citeseer.nj.nec.com/lipmaa01efficient.html`
6. A. W. Machado, "The nimbus cipher: A proposal for NESSIE", NESSIE Proposal, 2000.
7. National Bureau of Standards, NBS FIPS PUB 46, "Data Encryption Standard," National Bureau of Standards, U.S. Department of Commerce, Jan 1977.
8. The NIST Statistical Test Suite. Available from `http://csrc.nist.gov/rng/rng2.html`
9. J. Daemen, V. Rijmen, "AES Proposal: Rijndael", version 2, 1999
10. Randomness Testing of the AES Candidate Algorithms. Available from `http://csrc.nist.gov/encryption/aes/round1/r1-rand.pdf`
11. R. Rivest, M. Robshaw, R. Sidney, and Y. L. Yin, "The RC6 block cipher". Available from `http://www.rsa.com/rsalabs/rc6/`
12. R. Rivest, "Permutation Polynomials Modulo 2^{ω}", 1999.
13. B. Schneier and J. Kelsey, "Unbalanced Feistel Networks and Block Cipher Design", in Proceedings of the Third International Workshop on Fast Software Encryption, Cambridge, UK, February 1996, Springer, LNCS 1039, pp.121–144.
14. A. Shamir, "Efficient Signature Schemes Based on Birational Permutations", in Proceedings of CRYPTO 93, LNCS 773, 1–12.

Scalable and Unified Hardware to Compute Montgomery Inverse in GF(p) and GF(2^n)

Adnan Abdul-Aziz Gutub[1], Alexandre F. Tenca, Erkay Savaş[2], and Çetin K. Koç

Department of Electrical & Computer Engineering
Oregon State University, Corvallis, Oregon 97331, USA
{gutub,tenca,savas,koc}@ece.orst.edu

Abstract. Computing the inverse of a number in finite fields GF(p) or GF(2^n) is equally important for cryptographic applications. This paper proposes a novel scalable and unified architecture for a Montgomery inverse hardware that operates in both GF(p) and GF(2^n) fields. We adjust and modify a GF(2^n) Montgomery inverse algorithm to accommodate multi-bit shifting hardware, making it very similar to a previously proposed GF(p) algorithm. The architecture is intended to be scalable, which allows the hardware to compute the inverse of long precision numbers in a repetitive way. After implementing this unified design it was compared with other designs. The unified hardware was found to be eight times smaller than another reconfigurable design, with comparable performance. Even though the unified design consumes slightly more area and it is slightly slower than the scalable inverter implementations for GF(p) only, it is a practical solution whenever arithmetic in the two finite fields is needed.

1 Introduction

The modular inversion is an essential arithmetic operation for many cryptographic applications, such as Diffe-Hellman key exchange algorithm, decipherment operation of RSA algorithm, elliptic curve cryptography (ECC) [1,5], and the Digital Signature Standard as well as the Elliptic Curve (EC) Digital Signature algorithm [4,5]. The arithmetic performed in cryptographic applications consists mainly in modular computations of addition, subtraction, multiplication, and inversion. Although inversion is not as performance critical as all the others, it is the most time consuming arithmetic operation [1,2,8-10,12,13]. Therefore, most of the practical implementations try to avoid the use of inversion as much as possible. However, it is not possible to avoid it completely [1,2,5], what motivates the implementation of inversion as a hardware module in order to gain speed. In addition to that, hardware implementations provide an increased level of security for cryptographic systems, as discussed in [15].

Cryptographic inverse calculations are normally defined over either prime or binary extension fields [5], more specifically *Galois Fields GF(p)* or *GF(2^n)*. All

[1] Now with King Fahd University, Dhahran, Saudi Arabia, gutub@kfupm.edu.sa
[2] Now with Sabanci University, Istanbul, Turkey, erkays@sabanciuniv.edu

B.S. Kaliski Jr. et al. (Eds.): CHES 2002, LNCS 2523, pp. 484-499, 2003.

available application-specific integrated circuit (ASIC) implementations for inversion computation [8-10,12,13] are modeled strictly for one finite field, either GF(p) or GF(2^n). If the hardware at hand is for GF(2^n) calculations, such as [9,10,12,13], and the application this time needs GF(p) computation, a completely different hardware is required [5]. It is inefficient to have two hardware designs (one for GF(p) and another for GF(2^n)) when only one is needed each time. This issue motivated the search for a single unified hardware architecture used to compute inversion in either finite field GF(p) or GF(2^n), similar, in principle, to the multiplier idea proposed in [4].

Cryptography is heavily based on modular multiplication [4,5], which involves division by the modulus in its computations. Division, however, is a very expensive operation [6]. P. Montgomery proposed an algorithm to perform modular multiplication [7] that replaces the usual complex division with divisions by two, which is easily performed in the binary representation of numbers. The cost behind using Montgomery's method is paid in some extra computations to convert the numbers into Montgomery domain and vice-versa [7]. Once the numbers are transformed into Montgomery domain, all operations (addition, subtraction, multiplication, and inversion) are performed in this domain. The result is then converted back to the original integer values. Few methods were aimed to compute the inverse in the Montgomery domain [1-3] and are named *Montgomery modular inverse algorithms* [1].

The GF(p) Montgomery inverse (MonInv) algorithm [18] is an efficient method for doing inversion with an odd modulus. The algorithm is particularly suitable for implementation on application specific integrated circuits (ASICs). For GF(2^n) inversion, the original inverse procedure (presented in [17]) has been extended to the finite field GF(2^n) in [16]. It replaces the modulus (p) by an irreducible polynomial ($p(x)$), and adjusts the algorithm according to the properties of polynomials. We implemented the inversion algorithms in hardware based on the observation that the Montgomery inverse algorithm for both fields GF(p) and GF(2^n) can be very similar. We show that a unified architecture computing the Montgomery inversion in the fields GF(p) and GF(2^n) is designed at a price only slightly higher than the one for only the field GF(p), providing major savings when both types of inverters are required.

A scalable Montgomery inverter design methodology for GF(p) was introduced in [18]. This methodology allows the use of a fixed-area Montgomery inverter ASIC design to perform the inversion of unlimited precision operands. The design tradeoffs for best performance in a limited chip area were also analyzed in [18]. We use the design approach as in [14,18] to obtain a scalable hardware module. Furthermore, the scalable inverter described in this paper is capable of performing inversion in both finite fields GF(p) and GF(2^n) and is for this reason called a *scalable and unified Montgomery inverter*.

There are two main contributions of this paper. First, we show that a unified architecture for inversion can be easily designed without compromising scalability and without significantly affecting delay and area. Second, we investigate the effect of word length (w) and the actual number of bits (n) on the hardware area, based on actual implementation results obtained by synthesis tools. We start with a brief explanation of scalability in Section 2. In Section 3, we propose the GF(2^n) extended Montgomery inverse procedure that has several features suitable for an efficient hardware implementation. The unified architecture and its operation in both types of

finite fields, GF(p) and GF(2^n), are described in Section 4. Section 5 presents the area/time tradeoffs and appropriate choices for the word length of the scalable module. Finally, a summary and conclusions are presented in Section 6.

2 Scalable Architecture

Hardware architectures are generally designed for an exact number of operand bits. If this number of bits needs to be increased, even by one bit, the complete hardware needs to be replaced. In addition to that, if the design is implemented for a large number of bits, the hardware will be huge and usually slow. These issues motivated the search for the scalable inversion hardware proposed in [14].

The scalable architecture [14] solves the previous problems with the following three hardware features. First, the design's longest path should be short and independent of the operands' length. Second, it is designed in such a way that it fits in restricted spaces (flexible area). Finally, it can handle the computation of numbers in a repetitive way, up to a certain limit that is usually imposed by the size of the memory in the design. If the amount of data exceeds the memory capacity, the memory unit is replaced while the scalable computing unit may remain the same. Therefore, the scalable hardware design is built of two main parts, a memory unit and a computing unit. The memory unit is not scalable because it has a limited storage that imposes an upper bound on the number of bits that can be handled by the hardware (n_{max}). The computing unit read/write the data bits using another word size of w bits, normally much smaller than n_{max}. The computing unit is completely scalable. It is designed to handle w bits every clock cycle. The computing unit does not know the total number of bits that the memory is holding. It computes until the actual number of operand bits (n) is processed.

3 Montgomery Inverse Procedures for *GF(p)* and *GF(2^n)*

In order to design a unified Montgomery inverse architecture, the GF(p) and GF(2^n) algorithms need to be very similar and this way consume the least amount of extra hardware. Extending the GF(p) Montgomery inverse algorithm to GF(2^n) is practical due to the removal of carry propagation required in GF(p) and simple adjustments of test conditions. In other words, the GF(2^n) algorithm is like a simplification of the GF(p) algorithm. The converse (modifying GF(2^n) algorithms for GF(p)), on the other hand, is very difficult [4,5,16].

The scalable GF(p) Montgomery inverse (*MonInv*) procedure suitable for this work consists in two phases: the almost Montgomery inverse (*AlmMonInv)* and the correction phase (*CorPh*) [18]. The *AlmMonInv* has $a2^m$ as input and produces r and k, where $r = a^{-1}2^{k-m} \bmod p$, $2^{n-1} \le p < 2^n$ and $n < k < 2n$. The factor 2^m (of the AlmMonInv input $a2^m$) is related to Montgomery arithmetic [4,5,16]. The only restriction on the value of m is that it should not be less than the number of bits (n), i.e., $m \ge n$, as discussed in [1]. The *CorPh* takes r and k to generate the Montgomery inverse $a^{-1}2^m \bmod p$. Both GF(p) AlmMonInv and CorPh algorithms were mapped to hardware features and further modified for multi-bit shifting, a concept discussed in [18], which

resulted in an efficient implementation of the GF(p) Montgomery inverse. The GF(p) multi-bit shifting AlmMonInv and CorPh hardware algorithms (HW-Alg1 and HW-Alg2, respectively), are outlined in Figure 1.

GF(p) Multi-Bit Shifting AlmMonInv HW Algorithm (HW-Alg1)

Registers: u, v, r, s, x, y, z, and p (all registers hold n_{max} bits)
Input: $a2^m \in [1, p-1]$; Where p = modulus, and $m \geq n$ $(2^{n-1} \leq p < 2^n)$
Output: $result \in [1, p-1]$ & k; Where $result = a^{-1}2^{k-m} mod\ p$ & $n < k < 2n$
1. u = p; v = $a2^m$; r = 0; s = 1; x = 0; y = 0; z = 0; k = 0
2. if ($u_2u_1u_0$=000) then {u=ShiftR(u,3);s=ShiftL(s,3);k=k+3}; goto 8
2.1. if ($u_2u_1u_0$=100) then {u=ShiftR(u,2);s=ShiftL(s,2);k=k+2}; goto 8
2.2. if ($u_2u_1u_0$=110) then {u=ShiftR(u,1);s=ShiftL(s,1)}; goto 7
3. if ($v_2v_1v_0$=000) then {v=ShiftR(v,3);r=ShiftL(r,3);k=k+3}; goto 8
3.1. if ($v_2v_1v_0$=100) then {v=ShiftR(v,2);r=ShiftL(r,2);k=k+2}; goto 8
3.2. if ($v_2v_1v_0$=110) then {v=ShiftR(v,1);r=ShiftL(r,1)}; goto 8
4. x = Subtract(u, v); y = Subtract(v, u); z = Add(r, s)
5. if (x_{borrow}=0) then {u=ShiftR(x,1); r=z; s=ShiftL(s,1)}; goto 7
6. s = z; v = ShiftR(y,1); r = ShiftL(r,1)
7. k = k + 1
8. if (v ≠ 0) goto step 2
9. x = Subtract(p, r); y = Subtract($2p$, r)
10. if (x_{borrow} = 0) then {result=x}; else{result = y}

GF(p) Multi-Bit Shifting CorPh HW Algorithm (HW-Alg2)

Registers: r, u, v, x, y, z, and p (all registers hold n_{max} bits)
Input: r, p, n, k; Where $r\ (r=a^{-1}2^{k-m} mod\ p)$&$k$ from *HW-Alg1*
Output: result; Where result = $a^{-1}2^m\ (mod\ p)$.
11. $j = 2m-k$; x = 0; y = 0; z = 0
12. $v = 2p$; $u = 3p$
13. While $j > 0$
14. if $j = 1$ then {r = ShiftL(r,1); j=j-1}
15. else {r = ShiftL(r,2); j=j-2}
16. x=Subtract(r,p); y=Subtract(r,v); z=Subtract(r,u)
17. if (z_{borrow} = 0) then {r = z}
18. else if (y_{borrow} = 0) then {r = y}
19. else if (x_{borrow} = 0) then {r = x}
20. result = r

Fig. 1. Montgomery inverse hardware algorithm for GF(p)

Differently from what normally happens in a full-precision hardware design, the scalable hardware, as in [4,14,18], has multi-precision operators for shifting, addition, subtraction, and comparison. Observe the AlmMonInv algorithm in Figure 1, for example, the scalable subtraction (step 4) is also used for comparison ($u > v$), which is performed on a word-by-word basis (w-bit words) until all the actual data words (all n bits) are processed. Then, borrow-out bit of the most-significant word is used to decide on the result. Also, depending on the subtraction's completion, variable r or s has to be shifted. All variables, u, v, r and s, need to remain as is until the subtraction process is complete, and the borrow-out bit appears. For this reason, eight registers are required, as shown in Figure 1.

3.1 Representation and Manipulation of Elements in GF(2^n)

The inversion algorithm for GF(2^n) used in this work was presented in [16]. Although prime and binary extension fields, GF(p) and GF(2^n), have different properties, the elements of either field are represented using similar data structures. The elements of the field GF(2^n) can be represented in several different ways [5]. The polynomial representation, however, is a useful and appropriate form to the unified implementation, as used for the unified multiplier in [4]. According to the GF(2^n) polynomial representation, an element $a(x) \in GF(2^n)$ is a polynomial of length n, i.e., of degree less than or equal to $n-1$, written as $a(x)=a_{n-1}x^{n-1}+a_{n-2}x^{n-2}+ \ldots +a_2x^2+a_1x+a_0$, where $a_i \in GF(2)$. These coefficients a_i are represented as bits in the computer and the element $a(x)$ is represented as a bit vector $a = (a_{n-1}\ a_{n-2} \ldots a_2\ a_1\ a_0)$.

The addition/subtraction of two elements $a(x)$ and $b(x)$ in GF(2^n) is performed by adding/subtracting the polynomials $a(x)$ and $b(x)$, where the coefficients are added/subtracted in the field GF(2). As a consequence, both addition and subtraction operations are exactly the same and equivalent to bit-wise XOR operations on the bit-vectors a and b ($a_i \oplus b_i$). In order to compute the inverse of element $a(x)$ in GF(2^n), we need an irreducible polynomial of degree n. Let the irreducible polynomial be $p(x)= x^n+p_{n-1}x^{n-1}+p_{n-2}x^{n-2}+ \ldots +p_2x^2+p_1x+p_0$. Whenever the degree of a polynomial obtained in intermediate inversion calculations equals n, the polynomial is reduced (XORed) by $p(x)$. For example, if $\|r(x)\| = \|p(x)\|$ (degree of $r(x)$ equals degree of $p(x)$) then r is replaced by $p \oplus r$. Note that in some cases $\|r(x)\| = \|p(x)\|$ while $r < p$. These cases restrict the comparison of r to 2^n only (x^n not $p(x)$) to indicate if $r(x)$ needs to be reduced by $p(x)$ ($r = p \oplus r$); where 2^n is the binary representation of x^n.

3.2 Montgomery Inverse in GF(2^n)

The GF(2^n) Montgomery inverse of $a(x)x^m \bmod p(x)$ is $a(x)^{-1}x^m \bmod p(x)$ [5]. The Montgomery factor 2^m of GF(p) is replaced by x^m in GF(2^n), which is exactly equal to 2^m in a binary representations [4,5,16], where $m \geq n$. The elements of GF(p) and GF(2^n) are represented using similar binary data structures, a for both GF(p) and GF(2^n) equals ($a_{n-1}\ a_{n-2}\ \ldots\ a_2\ a_1\ a_0$) while $p = (p_{n-1}\ p_{n-2} \ldots p_2\ p_1\ p_0)$ for GF(p) and $p=(1\ p_{n-1}\ p_{n-2} \ldots p_2\ p_1\ p_0)$ for GF(2^n) [5]. Our adjusted binary GF(2^n) Montgomery inverse (MonInv) procedure consists in a GF(2^n) AlmMonInv and a GF(2^n) CorPh routines as outlined in Figure 2.

For more clarification of the GF(2^n) MonInv computation, see the numerical example in Figure 3. It takes as inputs the polynomial $a(x)= x^3+1$, represented into Montgomery domain as $a(x)x^9 \bmod p(x)= x^4+x^2$ ($m=9 \geq n=5$), and $p(x)= x^5+x^2+1$ as the irreducible polynomial. All the data are shown in its binary representation ($a=1001$, $a2^m=10100$, and $p=100101$). The example (Figure 3) follows the convention:

Met condition ➔ *affected registers with their updated values.*

The AlmMonInv routine generates the results $a^{-1}2^{k-m} = 1000$, and $k = (10)_{10}$ (k is a normal decimal counter), which are used by the CorPh to provide the Montgomery inverse result 111 (x^2+x+1 in the polynomial form). The reader is referred to the Appendix for checking the result of this example.

GF(2^n) AlmMonInv Algorithm

Input: $a2^m \in GF(2^n)$ & p; (p=irreducible polynomial & $m \geq n$)
Output: $result \in [1, p\text{-}1]$ & k ($result = a^{-1}2^{k-m} mod\ p$ & $n < k < 2n$)
1. u = p; v = $a2^m$; r = 0; s = 1; k = 0
2. While (v > 0)
3. if u_0 = 0 then {u = u/2; s = 2s}
4. else if v_0 = 0 then {v = v/2; r = 2r}
5. else if u>v then {u = (u⊕v)/2; r = r⊕s; s = 2s}
6. else {v = (u⊕v)/2; s = r⊕s; r = 2r}
7. k=k+1
8. if r ≥ 2^{n+1} (||r|| > ||p||) then {result = 2p⊕r}
9. else if r ≥ 2^n (||r|| = ||p||) then {result = p⊕r}
10. else result = r

GF(2^n) CorPh Algorithm

Input: r, p, m, & cowherd r & k from *AlmMonInv*
Output: *result*, where $result = a^{-1}2^m \ (mod\ p)$
11. j = 2m-k
12. While j > 0
13. r = 2r
14. if r ≥ 2^n (||r|| = ||p||) then {r = p⊕r}
15. j = j-1
16. result = r

Fig. 2. GF(2^n) Montgomery inverse algorithm in its binary representation

Observe on Figure 2 the several hardware operations applied to compute the MonInv in finite field GF(2^n). For example, the division and multiplication by two are equivalent to one bit shifting the binary representation of polynomials to the right and to the left, respectively. Checking the condition of step 5, if $u>v$, is performed through normal (borrow propagate) subtraction and test of the borrow-out bit. The subtraction result is completely discarded, only the borrow bit is observed. If the borrow bit is zero, then $u(x)$ is greater than $v(x)$. Similarly, the conditions in steps 8, 9, and 14 demand normal subtraction. However, the subtraction this time is used to check $\|r(x)\|$, which requires the availability of x^n (2^n in binary).

3.3 Multi-bit Shifting

A further improvement on the GF(2^n) MonInv algorithm is performed based on a multi-bit shifting method making it similar to the GF(p) algorithm in Figure 1. After comparing different multi-bit shifting distances applied to reduce the number of iterations of the GF(p) MonInv algorithm [18,19], the best maximum distance for multi-bit shifting was found to be three, as clarified in [18,19]. The GF(2^n) inverse algorithm (Figure 2) is mapped to hardware involving multi-bit shifting and making it very similar to the GF(p) algorithm (Figure 1) as shown in Figure 4. Note that x^n is required in the GF(2^n) algorithm as an extra variable that is needless in the GF(p) MonInv algorithm; x^n (2^n) is saved in register y in HW-Alg3 (used in step 9), and in register s in HW-Alg4 (used in step 16.1). These registers (y in HW-Alg3 and s in HW-Alg4) are not changed during the algorithms' execution.

GF(2^n) AlmMonInv Numerical Example

$a = 1001 \in GF(2^5)$, $p=100101$, $m= 9$, $n=5$
$a2^m \bmod p = 10100 \in GF(2^6)$ (a in Montgomery domain)
$u = p = 100101$, $v = a2^m = 10100$, $s = 1$, $r = k = 0$
$v_0 = 0 \rightarrow v = 1010$, $r = 0$, $k=1$
$v_0 = 0 \rightarrow v = 101$, $r = 0$, $k=2$
$u > v \rightarrow u = 10000$, $r = 1$, $s = 10$, $k=3$
$u_0 = 0 \rightarrow u = 1000$, $s = 100$, $k=4$
$u_0 = 0 \rightarrow u = 100$, $s = 1000$, $k=5$
$u_0 = 0 \rightarrow u = 10$, $s = 10000$, $k=6$
$u_0 = 0 \rightarrow u = 1$, $s = 100000$, $k=7$
$v > u \rightarrow v = 10$, $s = 100001$, $r = 10$, $k=8$
$v_0 = 0 \rightarrow v = 1$, $r = 100$, $k=9$
$u = v \rightarrow v = 0$, $r = 1000$, $s = 100101$, $k=10$
$||r|| < ||p|| \rightarrow$ result = r

GF(2^n) CorPh Numerical Example

$p=100101$, $m= 9$, $n= 5$
$r = 1000 \in GF(2^6)$, $k=10$ (from AlmMonInv)
$j = 8$
$r = 10000$, $j = 7$
$r = 100000$, $||r|| = ||p|| \rightarrow r = 101$, $j = 6$
$r = 1010$, $j = 5$
$r = 10100$, $j = 4$
$r = 101000$, $||r|| = ||p|| \rightarrow r = 1101$, $j = 3$
$r = 11010$, $j = 2$
$r = 110100$, $||r|| = ||p|| \rightarrow r = 10001$, $j = 1$
$r = 100010$, $||r|| = ||p|| \rightarrow r = 111$, $j = 0$

$\therefore GF(2^n)$ MonInv of $10100 = 111$ ($a^{-1}2^m$); Where $m=9$ & $n = 5$

Fig. 3. $GF(2^n)$ MonInv computation numerical example

For both GF(p) and $GF(2^n)$ MonInv hardware algorithms (Figure 1 and Figure 4, respectively), the AlmMonInv algorithm needs to finish its computation completely before the CorPh begins processing. This data dependency allows the use of the same hardware to execute both algorithms, i.e., both the AlmMonInv and CorPh. The algorithms are implemented in the unified and scalable hardware architecture as described in the following section.

4 The Unified and Scalable Inverter Architecture

Taking into account the amount of effort, time, and money that must be invested in designing an inverter, a scalable and unified architecture that can perform arithmetic in two commonly used algebraic finite fields is clearly advantageous. In this section, we present the hardware design of a Montgomery inverse architecture that can be used for both types of fields following the design methodology presented in [14]. The proposed unified architecture is obtained from the scalable architecture given in [14] but with some modifications, which slightly increases the longest path propagation delay and chip area. The scalable GF(p) Montgomery inverse architecture presented in [14] consisted in two main units, a non-scalable memory unit and a scalable computing unit. The memory unit is not scalable because it has a limited storage defined by the value of n_{max}. The data values of a and p are first loaded in the memory

unit. Then, the computing unit read/write (modify) the data using a word size of w bits. The computing unit is completely scalable. It is designed to handle w bits every clock cycle. The computing unit does not know the total number of bits, n_{max}, the memory is holding. It computes until the controller indicates that all operands' words were processed. Note that the precision of the actual numbers used may be way smaller than n_{max} bits. The user needs to identify the type of finite field his application needs at the beginning of the computation. An input signal *FSEL* (field select) is used to tell the architecture weather GF(p) or GF(2^n) is the desired arithmetic domain.

GF(2^n) Multi-Bit Shifting AlmMonInv HW Algorithm (HW-Alg3)

Registers: u, v, r, s, x, y, z, & p (all registers hold n_{max} bits)
Input: $a2^m$, $2^n \in [1,p\text{-}1]$ (p=irreducible polynomial & $m \geq n$)
Output: result$\in [1, p\text{-}1]$ & k (result=$a^{-1}2^{k-m} mod\ p$ & $n<k<2n$)

1. u = p; v = $a2^m$; r = 0; s = 1; x = 0; y = 2^n; z = 0; k = 0
2. if ($u_2u_1u_0$=000) then{u=ShiftR(u,3);s=ShiftL(s,3);k=k+3}; goto 8
2.1. if ($u_2u_1u_0$=100) then{u=ShiftR(u,2);s=ShiftL(s,2);k=k+2}; goto 8
2.2. if ($u_2u_1u_0$=110) then{u=ShiftR(u,1);s=ShiftL(s,1)}; goto 7
3. if ($v_2v_1v_0$=000) then{v=ShiftR(v,3);r=ShiftL(r,3);k=k+3}; goto 8
3.1. if ($v_2v_1v_0$=100) then{v=ShiftR(v,2);r=ShiftL(r,2);k=k+2}; goto 8
3.2. if ($v_2v_1v_0$=110) then{v=ShiftR(v,1);r=ShiftL(r,1)}; goto 8
4. S1 = Subtract(u, v); x = v$\oplus$u; z = r$\oplus$s
5. if ($S1_{borrow}$=0) then {u=ShiftR(x,1); r=z; s=ShiftL(s,1)}; goto 7
6. s = z; v = ShiftR(x,1); r = ShiftL(r,1)
7. k = k + 1
8. if (v $\neq$0) go to step 2
9. x = $p\oplus$r ; z = $2p\oplus$r ; S1 = Subtract (y,x); S2 = Subtract (y,z)
10. if ($S1_{borrow}$=0) then {result=x}
10.1 else if ($S2_{borrow}$=0) then {result=z}
10.2 else {result = r}

GF(2^n) Multi-Bit Shifting CorPh HW Algorithm (HW-Alg4)

Input: r, p, m, 2^n & k; Where r ($r=a^{-1}2^{k-m} mod\ p$)& k from *HW-Alg3*
Output: result; Where result = $a^{-1}2^m$ *(mod p).*

11. $j = 2m\text{-}k\text{-}1$; x = 0; y = 0; z = 0
12. $v = 2p$; $u = 3p$; s = 2^n
13. While $j > 0$
14. if $j = 1$ then {r = ShiftL(r,1); j=j-1}
15. else {r = ShiftL(r,2); j=j-2}
16. x = $p\oplus r$; y = $u\oplus r$; z = $u\oplus r$
16.1 S1=Subtract(s,x); S2=Subtract(s,y); S3=Subtract(s,z)
17. if ($S3_{borrow}$ = 0) then {r = z}
18. else if ($S2_{borrow}$ = 0) then {r = y}
19. else if ($S1_{borrow}$ = 0) then {r = x}
20. result = r

Fig. 4. Montgomery inverse hardware algorithm for GF(2^n)

The block diagram for the Montgomery inverter hardware is shown in Figure 5. The memory unit is connected to the computing unit components. The memory unit is not changed from what is presented in [14]. It contains a counter to compute variable k and eight first-in-first-out (FIFO) registers used to store the inversion algorithm's variables. All registers, u, v, r, s, x, y, z and p, are limited to hold at most n_{max} bits. Each FIFO register has its own reset signal generated by the controller. They have counters to keep track of n (the number of bits actually used by the application).

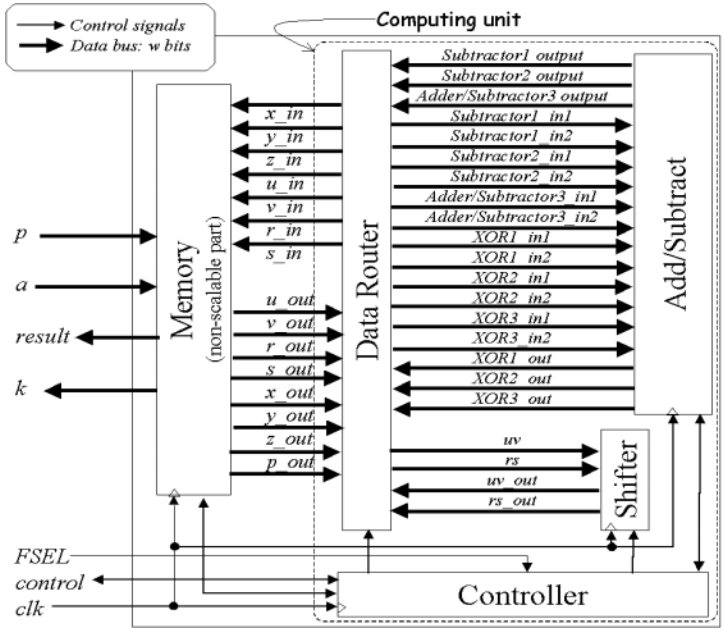


Fig. 5. Scalable and unified inverter hardware

The computing unit is made of four hardware blocks: add/subtract, shifter, data router, and controller block. The GF(p) add/subtract unit and the data router are the only components that need to be adjusted to make the inverter hardware unified for GF(p) and GF(2^n) finite fields.

The GF(p) add/subtract unit is originally built of two *w*-bit subtractors, a *w*-bit adder/subtractor, four flip-flops, one multiplexer, a *w*-bit comparator, and logic gates, as detailed in [14]. This unit is adjusted to operate for GF(2^n) by adding a set of *3w* parallel XOR gates used for steps 4 and 9 of HW-Alg3 and step 16 of HW-Alg4. The new add/subtract unit is shown in Figure 6. The signal *Control* makes the unit perform either two subtractions plus one addition (step 4 of HW-Alg1), or three subtractions (step 16 of HW-Alg2 and step 16.1 of HW-Alg4). Three flip-flops are used to hold the intermediate borrow bits of the subtractors and the carry bit of the adder to implement the multi-precision operations. The fourth flip-flop is used to store a flag that keeps track of the comparison between *u* and *v*, which is used to perform step 8 of HW-Alg1 and HW-Alg3. The subtractors borrow-out bits are connected to the controller through signals that are useful only at the end of each multi-precision addition/subtraction operation. Subtractor1 borrow-out bit will affect the flow of the operation to choose either step 5 or step 6 of both HW-Alg1 and HW-Alg3. It is also essential in electing the result observed in step 10 of HW-Alg1 and of HW-Alg2. The three subtractors borrow-out bits ($S1_{borrow}$, $S2_{borrow}$, $S3_{borrow}$) are likewise necessary for selecting the correct solution of the 'if' condition to be one of the steps 17, 18, or 19, from the HW-Alg2 and from the HW-Alg4 algorithms.

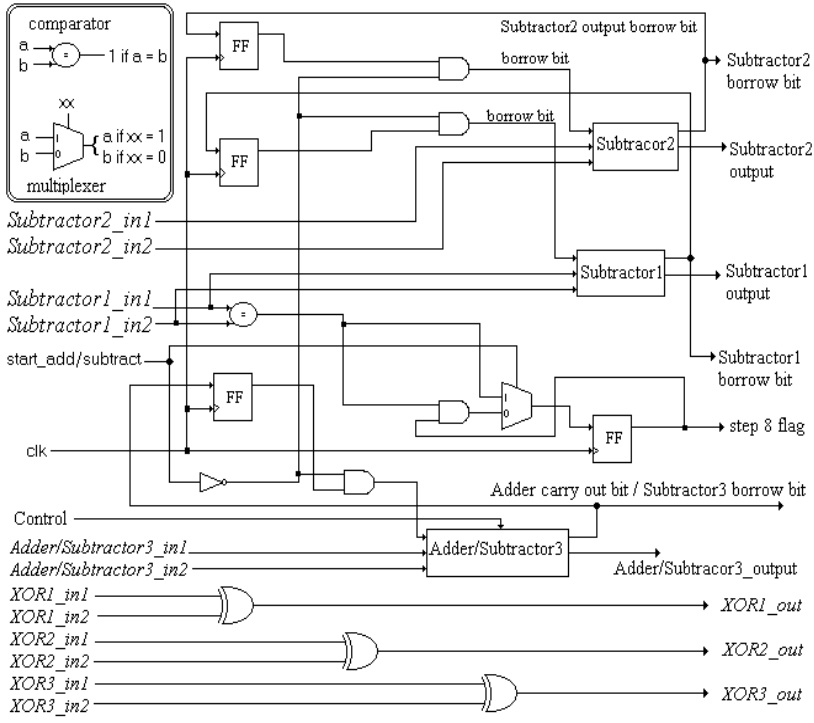


Fig. 6. Add/Subtract unit of the scalable and unified hardware

The shifter is made of two multiplexers and two registers with special mapping of some data bits, as shown in Figure 7. Depending on the controller signal *Distance*, the shifter acts as a one, two, or three-bit shifter. Two types of shifting operations are needed in the HW-Alg1 and the HW-Alg3 algorithms, shifting an operand (u or v) through the uv bus one, two, or three bits to the right, and shifting another operand (r or s) through the rs bus by a similar number of bits to the left. Shifting u or v is performed through Register1, which is of size w-1 bits. For each word, all the bits of uv are stored in Register1 except for the least significant bit(s) to be shifted, it is (or they are) read out immediately as the most significant bit(s) of the output bus *uv_out*. Shifting r or s to the left is performed via Register2, which is of size w+3 bits similar to shifting uv but to the other direction. When executing the HW-Alg2 or HW-Alg4, the shifting is performed either to one or two bits to the left only, which is via MUX2 and Register2 ignoring MUX1 and Register1.

The data router capabilities are extended to satisfy the unified architecture requirements. It interconnects the memory, add/subtract, and shifter units. The possible configurations of the data router are shown in Figure 8.

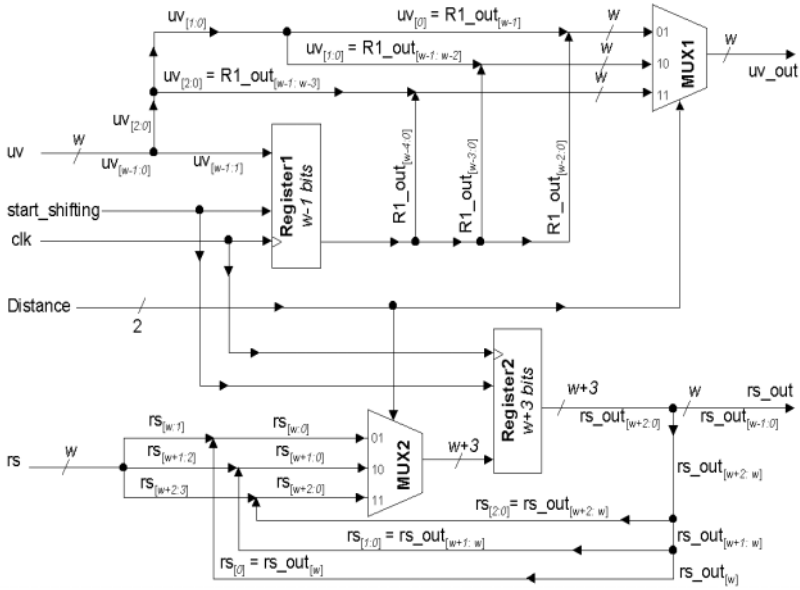


Fig. 7. Shifter unit hardware

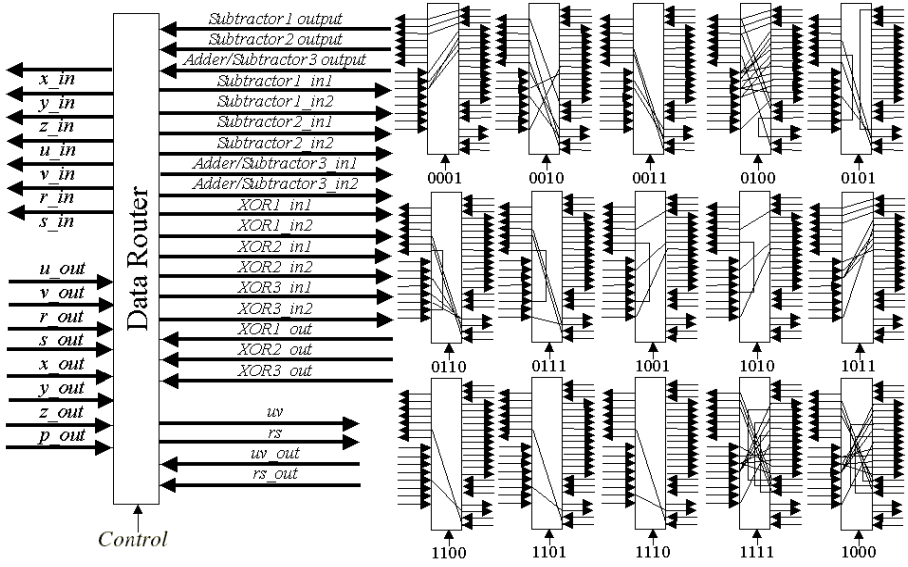


Fig. 8. Data router configurations

5 Modeling and Analysis

The unified and scalable inverter was modeled and simulated in VHDL. Previously, a fixed design (full precision) and other scalable inverter designs for inversion in GF(p) were also described in VHDL. All VHDL descriptions of the scalable designs, including the new unified ones, have two main parameters, namely n_{max} and w. The fixed hardware, however, is parameterized by n_{max} only. Their area and speed are presented in this section. Also a reconfigurable hardware [16] that can perform the inversion in both GF(p) and GF(2^n), besides other functions, is considered in the comparison. We didn't define a specific architecture for the adders and subtractors used in our VHDL implementations. Thus, the synthesis tool chooses the best option in terms of area from its library of standard cells. As a result, all proposed designs use the same type of adders and subtractors.

5.1 Area Comparison

The exact area of any design depends on the technology and minimum feature size. For technology independence, we use the equivalent number of NOT-gates as an area measure [6]. A CAD tool from Mentor Graphics (Leonardo) was used. Leonardo takes the VHDL design code and provides a synthesized model with its area and longest path delay. The target technology is a *0.5μm* CMOS defined by the 'AMI0.5 fast' library provided in the ASIC Design Kit (ADK) from the same Mentor Graphics Company [11]. It has to be mentioned here that the ADK is developed for educational purposes and cannot be thoroughly compared to technologies adopted for marketable ASICs. It however, provides a framework to contrast all scalable hardware designs together and with the fixed one. The sizes of the designs are compared in Figure 9. Observe that the fixed design has a better area if the maximum number of bits used (n_{max}) is small which is useless in cryptographic applications [5]. The unified designs are larger than the GF(p) ones with a calculated average of 8.4% more hardware area.

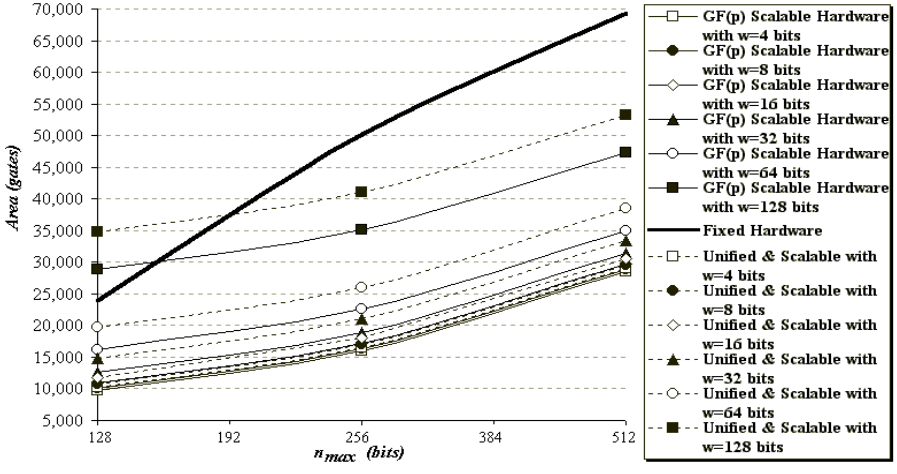


Fig. 9. Area comparison

The area of the unified designs were also compared with the reconfigurable hardware [16], but not shown in Figure 9. The reconfigurable design core is built of 880,000 devices [16]. Assume a device is corresponding to a transistor and our NOT-gate is equivalent to two transistors [6], so the reconfigurable hardware core is equivalent to 440,000 gates, which means that the reconfigurable design is eight times greater than the largest unified hardware shown in Figure 9. Of course, the design in [16] does more than inversion, but its datapath is responsible for most of the area, and would be used anyway for the inversion computation.

5.2 Speed Comparison

The total computation time is a product of the number of clock cycles the algorithm takes and the clock period of the final implementation. This clock period changes with the value of w in the unified and scalable hardware, and changes with the value of n_{max} in the fixed hardware. This is because $w = n_{max}$ in the fixed hardware. All VHDL coded designs clock cycle periods are generated automatically by Leonardo, which determines the longest path delay of the hardware circuits. The clock period of the reconfigurable design was considered as being 20ns/cycle (operates at 50MHz clock rate frequency) [16].

The number of clock cycles depends completely on the data and the algorithm. A probabilistic study described in [18] is used to estimate the average number of clock cycles. For the fixed design, the average number of clock cycles equal to $C_f = 1.525n$. For all scalable designs, the average number of clock cycles is $C_s=(2.4125n+1)\lceil n/w \rceil$, which is exactly the same for the unified designs presented in this paper. Hence, adjusting the scalable designs to be unified did not change the number of clock cycles of the inverse computation. However, the clock cycle period of the unified designs increased slightly, making the total computation time of the unified hardware different than what was given in [18]. The number of clock cycles for the reconfigurable hardware to complete the inversion process is $C_r=14.5n$ [16].

Similar to the GF(p) scalable hardware of [18], the unified and scalable hardware can have several designs for each n_{max}, depending on w. For example, Figure 10 shows the delay of several designs of the unified and scalable hardware compared to the reconfigurable, GF(p) scalable, and fixed hardware designs, all modeled for $n_{max}=512$ bits, which is a practical number for future cryptographic applications [5]. Observe how the actual data size (n) plays a big role on the speed of the designs. In other words, as n reduces and w is small, the number of clock cycles decrease significantly, which considerably reduces the overall computing time of all scalable designs (including the unified ones) compared to the others. This is a major advantage of the scalable hardware over the fixed [14,18] and reconfigurable ones.

The new unified designs when compared to the scalable design for GF(p) only have very similar characteristics. Overall, it needs an average of 19.8% more time than the designs for GF(p) [18]. Another observation from Figure 10 is that the unified designs are faster than the fixed one as long as:

$$n < \begin{cases} (\log_2 w) n_{max}/8 & when\ w < n_{max}/8 \\ n_{max} & when\ w \geq n_{max}/8 \end{cases}$$

which is generalized for different n_{max} values. Several experimental tests were done for n_{max}= *32, 64, 128, 512* and *1024* bits. Figure 10 also shows that the unified designs are comparable to the reconfigurable one giving better performance when:

$$n < \begin{cases} (\log_2 w)\, n_{max}/32 & when \;\; w < n_{max}/8 \\ (\log_2 w)\, n_{max}/25 & when \;\; w \geqslant n_{max}/8 \end{cases}$$

Consider the case when $n=n_{max}=512$ bits in Figure 10, the unified design with $w=64$ bits has almost the same speed as the fixed one, but the ones with $w=128$ bits remain faster. In fact, as w gets bigger the total time decreases, which is also true when comparing among the different unified designs while $n \geq w$, as also proven before in [18] for the GF(p) scalable designs. Whenever $n < w$ considering the unified and scalable designs, the scalability advantage of these designs is reduced since the number of words to be processed reached its lower limit, but still the unified and scalable designs are faster than the fixed one.

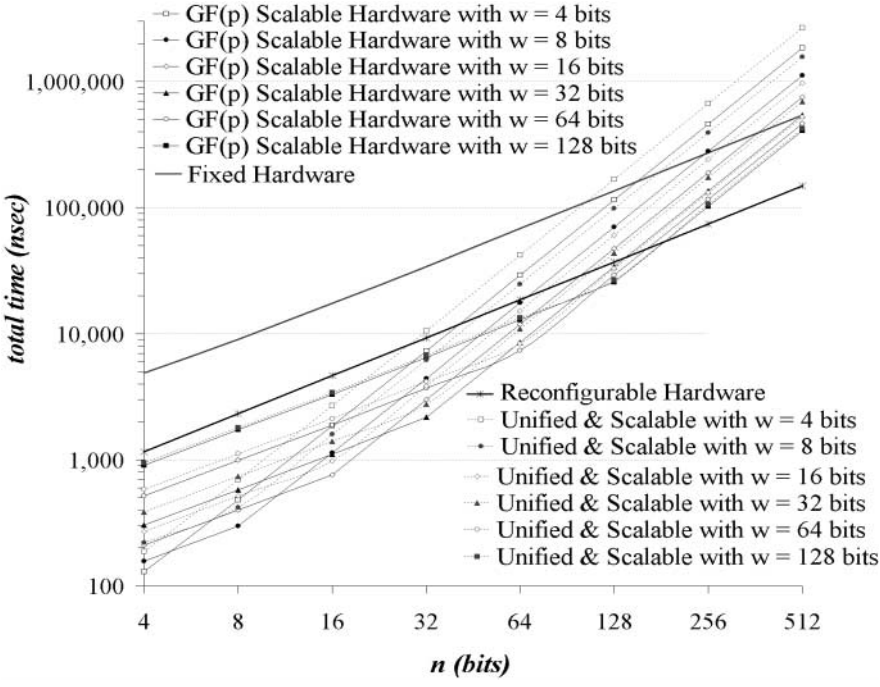


Fig. 10. Delay comparison of designs with n_{max} = 512 bits

6 Conclusion

This paper presents a scalable inverter for both finite fields GF(p) and GF(2^n) in a unified hardware module that applies the design approach proposed in [14,18,19]. The

primary contribution of this research is to show that it is possible to design a unified hardware without compromising scalability and area efficiency. The unified inverter hardware is built of two main units, a memory unit and a computing unit. The memory unit defines the upper bound of the number of bits that the hardware can handle. The computing unit is the real scalable hardware, it is designed to fit in constrained areas and perform the computation of numbers in a repetitive way. Our analysis shows that as the word size of the scalable computing unit reduces, the hardware area decreases and the possible clock frequency increases. However, if we increase the computing unit word size, the clock frequency is reduced, but for $n > w$ the overall computing time is also reduced, which is considered a normal area-time tradeoff.

Several configurations of the proposed inverter hardware (different word lengths) were described and synthesized using Mentor Graphics CAD tools. They were compared with equivalent configurations of a previously proposed inversion hardware design for inversion in GF(p) only. The comparisons show that this unified and scalable structure is very attractive for cryptographic systems, particularly for ECC where there is a need for modular inversion of large numbers in both finite fields GF(p) and GF(2^n) depending on the application usage.

Acknowledgments. The authors would like to thank KFUPM-Saudi Arabia and NSF under the CAREER grant CCR-0093434-"Computer Arithmetic Algorithms and Scalable Hardware Designs for Cryptographic Applications" for providing financial support toward this research.

References

1. E. Savas and C. K. Koç. The Montgomery Modular Inverse – Revisited. *IEEE Trans. on Computers*, 49(7): 763-766, July 2000.
2. T. Kobayashi and H. Morita. Fast Modular Inversion Algorithm to Match Any Operation Unit. *IEICE Trans. Fundamentals*, E82-A(5):733-740, May 1999.
3. B. S. Kaliski. The Montgomery Inverse and its Applications. *IEEE Trans. on Computers*, 44(8):1064-1065, Aug. 1995.
4. E. Savas, A. F. Tenca, and C. K. Koç. A Scalable and Unified Multiplier Architecture for Finite Fields GF(p) and GF(2^k). *In Cryptographic Hardware and Embedded Systems*, Lecture notes in Computer Science. Springer, Berlin, Germany, 2000.
5. I. Blake, G. Seroussi, and N. Smart. Elliptic Curves in Cryptography. *Cambridge University Press*: New York, 1999.
6. M. D. Ercegovac, T. Lang, and J. H. Moreno. Introduction to Digital System. *John Wiley & Sons, Inc.*, New York, 1999.
7. P. Montgomery. Modular Multiplication without Trail Division. *Mathematics of Computation*, 44(170): 519-521, April 1985.
8. N. Takagi. Modular Inversion Hardware with a Redundant Binary Representation. *IEICE Trans. on Information and Systems*, E76-D(8): 863-869, Aug. 1993.
9. J.-H. Guo, and C.-L. Wang. Hardware-Efficient Systolic Architecture for Inversion and Division in GF(2^m). *IEE Proceedings: Computers and Digital Techniques*, 145(4): 272-278, July 1998.
10. Choudhury, Pal, and Barua. Cellular Automata Based VLSI Architecture for Computing Multiplication and Inverses in GF(2^m). *Proceedings of the 7th IEEE International Conference on VLSI Design*, Calcutta, India, January 5-8 1994.
11. Mentor Graphics Co., *ASIC Design Kit*, http://www.mentor.com/partners/hep/AsicDesign Kit/dsheet/ami05data book.html

12. M. A. Hasan. Efficient Computation of Multiplicative Inverses for Cryptographic Applications. *Proceeding of the 15th IEEE Symposium on Computer Arithmetic*, June 2001.
13. M. Feng. A VLSI Architecture for Fast Inversion in GF(2^m). *IEEE Trans. on Computers*, 38(10):1383-1386, Oct. 1989.
14. A. A.-A. Gutub, A. F. Tenca, and C. K. Koç. Scalable VLSI Architecture for GF(p) Montgomery Modular Inverse Computation. *ISVLSI 2002: IEEE Computer Society Annual Symposium on VLSI*, Pittsburgh, Pennsylvania, April 25-26 2002.
15. J. R. Michener and S. D. Mohan. Clothing the E-Emperor. *IEEE Compute*, 34(9):116-118, Sep. 2001.
16. J. Goodman and A. P. Chandrakasan. An Energy-Efficient Reconfigurable Public-Key Cryptogrphy Processor. *IEEE Journal of Solid-State Circuits*, 36(11):1808-1820, Nov. 2001.
17. D. Knuth. The Art of Computer Programming – Seminumerical Algorithms, 2nd ed. Vol. 2, Reading, MA: *Addison-Wesley*, 1981.
18. A. A.-A. Gutub and A. F. Tenca. A Scalable VLSI Architecture for Montgomery Inversion in GF(p). *Submitted for publication in March 2002 to IEEE Trans. on VLSI.*
19. A. A.-A. Gutub, New Hardware Algorithms and Designs for Montgomery Modular Inverse Computation in Galois Fields GF(p) and GF(2^n), *Ph.D. thesis, Oregon State University, 2002.*

Appendix

This Appendix details the computations and verifies the results used in the GF(2^n) MonInv numerical example shown in Figure 3. The example defines $m=9$ and $n=5$; where n is the degree of the irreducible polynomial and m (of the Montgomery constant 2^m) is any number as long as $m \geq n$. To simplify the arithmetic lets only use the binary representation of polynomials. The MonInv takes the inputs $a=1001$ and $p=100101$. However, a is represented into Montgomery domain as $a2^m$, which is calculated as follows:

$$a=1001 \Rightarrow a2^m=a2^9=1001000000000$$

but since 1001000000000 needs to be reduced by p or a multiple of p until the number of significant bits of $a2^9$ is less or equal to n (the degree of polynomial $a(x)x^m \bmod p(x)$ should be less than the degree of the irreducible polynomial ($p(x)$)), so

$$a2^9 \oplus 2^7 p=1001000000000 \oplus 1001010000000=10000000$$

and 10000000 also needs reduction

$$10000000 \oplus 2^2 p = 10000000 \oplus 10010100 = 10100.$$

So

$$a2^m \bmod p = a2^9 \bmod p = 1001000000000 \bmod p \equiv 10100.$$

The fact that GF(2^n) MonInv of 10100 is $a^{-1}2^m$=111, can be similarly verified. The MonInv numerical example in Figure 3 calculated that $a^{-1}2^9 = 111 \Rightarrow a^{-1} = 111/2^9$.

Any congruent polynomial can be XORed with the irreducible polynomial, such as:

$$a^{-1}2^9=111 \equiv 111 \oplus 100101 = 100010 \rightarrow a^{-1}2^8=10001$$
$$a^{-1}2^8=10001 \equiv 10001 \oplus 100101=110100 \rightarrow a^{-1}2^6=1101$$
$$a^{-1}2^6=1101 \equiv 1101 \oplus 100101=101000 \rightarrow a^{-1}2^3=101$$
$$a^{-1}2^3=101 \equiv 101 \oplus 100101=100000 \rightarrow a^{-1}=100$$

To confirm that the GF(2^n) MonInv of 10100 is 111, when $m=9$ and $n=5$, it is enough to show that $a \cdot a^{-1} \bmod p = 1$, as follows:

$$a \cdot a^{-1} = 1001 \cdot 100 = 100100$$
$$100100 \bmod p = 100100 \oplus 100101 = 1$$

Dual-Field Arithmetic Unit for $GF(p)$ and $GF(2^m)^\star$

Johannes Wolkerstorfer

Institute for Applied Information Processing and Communications,
Graz University of Technology, Inffeldgasse 16a, 8010 Graz, Austria
Johannes.Wolkerstorfer@iaik.at, http://www.iaik.at

Abstract. In this article we present a hardware solution for finite field arithmetic with application in asymmetric cryptography. It supports calculation in $GF(p)$ as well as in $GF(2^m)$. Addition and multiplication with interleaved modular reduction are the main functionality of the unit. Additional functions—like shift operations and integer incrementation—allow the calculation of the multiplicative inverse and covering all operations required to implement Elliptic Curve Cryptography. Redundant number representation and efficient modular reduction make it ready for future cryptographic bitlengths and allow operation at high clock frequency on moderate hardware resources.

Keywords: Finite field arithmetic, multiplication, modular reduction, inversion, redundant number representation, hardware implementation.

1 Introduction

Finite field arithmetic is the backbone for nearly all public-key algorithms currently used. Widespread techniques like RSA encryption and Diffie-Hellman key agreement operate on finite fields with modular integer arithmetic. These algorithms have bitlengths up to 2048-bit to ensure information security for the next decade [3]. The calculation of these algorithms relies on exponentiation, which is computational intensive and demands dedicated hardware solutions when throughput is of concern. More recently, Elliptic Curve Cryptography (ECC) made the application of another type of finite fields popular: binary extension fields where elements can be represented as polynomials instead of integers. Binary fields $GF(2^m)$ are considered advantageous for hardware solutions because addition and modular reduction of polynomials are somewhat easier than those of integers.

ECC has the advantage of shorter bitlengths while offering the same level of security (163-bit up to 571-bit). That makes ECC attractive for application in constrained systems like smartcards where chip area is limited and the computational power of microprocessors is sparse. Applications of ECC are digital

* This work origins from the European Commission funded project *USB_CRYPT* established under contract IST-2000-25169 in the Information Society Technologies (IST) Program.

B.S. Kaliski Jr. et al. (Eds.): CHES 2002, LNCS 2523, pp. 500–514, 2003.

signature schemes, encryption schemes, and key agreement schemes [5,6,7]. The Elliptic Curve Digital Signature Standard [8,4] defines prime fields $GF(p)$ and binary fields $GF(2^m)$ as underlying fields for elliptic curves. For a full support of the standard, both type of fields have to be supported. This gives reason to search for hardware architectures that operate in both fields. Such a dual-field arithmetic unit can be realized and most of the hardware resources required for calculations in the prime field $GF(p)$ can be reused for operation in $GF(2^m)$. The cost of such an unified arithmetic unit for $GF(p)$ and $Gf(2^m)$ is only slightly higher than for a mere $GF(p)$-multiplier [15,16].

Former arithmetic units have focused on an efficient implementation of the multiplication and have neglected other operations. This is justified by the observation that the core operation of algorithms like RSA and Diffie-Hellman is modular exponentiation, which is calculated by repeated multiplications. The situation for ECC is slightly different. Although, the performance of ECC is also determined by multiplication, ECC requires besides multiplication and squaring also inversion, addition, and subtraction.

We will present a dual-field arithmetic unit that is capable to calculate all these operations in both types of fields: $GF(p)$ and $GF(2^m)$. The architecture takes low-power design considerations into account and assures a short critical path to enable operation at high clock frequencies. The intended applications of the arithmetic unit are systems with limited silicon area where both types of arithmetic are required and performance is not of utmost importance. The main motivation for the design was to develop an unit that is capable to perform all calculations of the Elliptic Curve Digital Signature Algorithm (ECDSA) and key-agreement protocols defined by the American National Standards Institute (ANSI) [4,5]. Further relevant ECC standards are published by the Institute of Electrical and Electronic Engineers (IEEE) [6], the International Standards Organization (ISO) [7], and the National Institute of Standards and Technology (NIST) [8].

The proposed architecture of the dual-field arithmetic unit focuses on an efficient implementation of operations in the finite fields $GF(p)$ and $GF(2^m)$. Operands are processed at full precision and most operations are executed within a single clock cycle. Multiplication is a multi-cycle operation with bitserial scheduling of the multiplier. Modular reduction is interleaved and uses quotient prediction for operation in $GF(p)$. Intermediate results of $GF(p)$-operations have a redundant number representation which permits to scale the architecture's precision without affecting the maximum clock frequency. The architecture even allows to calculate the Extended Euclidean Algorithm for inverting field elements. The architecture is highly regular and has only a small number of leaf cells—which is a desired property for a full-custom implementation.

The remainder of this article presents related work in §2. In §3 the mathematical background of operations in prime fields and in binary fields is covered. §4 presents the proposed architecture and discusses design considerations. Finally, we present results in §5 and draw conclusions in §6.

2 Related Work

E. Savaş et al. published in 2000 a unified multiplier for $GF(p)$ and $GF(2^m)$ which uses Montgomery multiplication for both fields [15]. Multiplication is done bitserial and the multiplicand is processed in blocks. Arbitrary precision multiplication is possible and precision is only constrained by memory. Their architecture is based on a pipeline of block-sized processing elements. The pipeline can have different configurations to trade area for speed. This approach has the smartness to process arbitrary precision numbers, which comes at the cost of a more complicated architecture that seems to be challenging for a full-custom implementation. Another restraint is the need of the Montgomery algorithm for precomputed constants and the need of transformations.

J. Großschädl's unified multiplier is bitserial too but processes the multiplicand in full precision [16]. Modular reduction is done by an interleaved quotient prediction and a conditional modulus subtraction, which does not require any pre-computations or transformations. Multiplication in $GF(2^m)$ takes m cycles, whereas multiplication in $GF(p)$ takes between $\log_2 p$ and $2\log_2 p$ cycles due to a conditional extra modular reduction cycle. Intermediate results of $GF(p)$-operations are stored in redundant representation because partial-product accumulation is done with carry-save adders. A carry-propagate adder with lower wordsize converts redundant results iteratively into their binary representation. The proposed architecture is simple, requires little hardware resources and has a regular structure that is convenient for a full-custom implementation. The low $GF(p)$-performance is a disadvantage. It is caused by the reduction algorithm and the redundant-to-binary conversion.

J. Goodman et al. presented in [17] a VLSI implementation of a dual-field arithmetic unit. Their so-called Domain-Specific Reconfigurable Cryptographic Processor (DSRCP) is not a mere multiplier for $GF(p)$ and $GF(2^m)$. It can calculate all operations required for elliptic curve cryptography including inversion and comparisons. These operations are executed on an extensive datapath which is controlled by a microcoded control unit. Main components of the datapath are a carry-propagate adder for operation in $GF(p)$ that takes three cycles for an addition and a reconfigurable datapath for operation in $GF(2^m)$. An additional comparator allows comparisons of integers or polynomials. The Montgomery algorithm is used for multiplication in $GF(p)$. Multiplication in $GF(2^m)$ obeys an iterative MSB-first scheme with interleaved modular reduction.

3 Mathematical Background

This section describes the representation of prime field elements and binary field elements and presents operations on these elements.

Prime field elements $A \in GF(p)$ are integers in the set $\{0, 1, \ldots p-1\}$ where p is prime. Binary field elements $A(x) \in GF(2^m)$ are polynomials of degree less than m when a polynomial basis is used to represent the field elements. These polynomials have coefficients in the set $\{0, 1\}$. Both types of field elements can

be represented with bitstrings as shown in (1) and (2). The binary representation of a prime field element needs $n = \lceil \log_2 p \rceil$ bits for storage. Elements of $GF(2^m)$ require m bits to store all coefficients of the polynomial. The memory requirement to store both types of elements is $max(n, m)$ bits.

$$A \in GF(p): \qquad A = \sum_{i=0}^{n-1} a_i 2^i \quad \text{with} \quad n = \lceil \log_2 p \rceil, \quad a_i \in \{0, 1\} \tag{1}$$

$$A(x) \in GF(2^m): \quad A(x) = \sum_{i=0}^{m-1} a_i x^i, \quad a_i \in \{0, 1\} \tag{2}$$

Although elements of both fields are stored uniformly, their field operations differ. Addition of prime field elements is an integer addition with modular reduction, whereas addition of polynomials is done coefficient-wise without the need of modular reduction. Multiplication requires modular reduction in both cases because the result of an integer multiplication as well as the result of a polynomial multiplication could have double the bitlength of their operands. Surprisingly, an almost identical algorithm can calculate multiplication in $GF(p)$ and in $GF(2^m)$ which facilitates an unified hardware approach.

Not all cryptographic algorithms require the inversion of field elements. For instance, the RSA algorithm and the Diffie-Hellman key-exchange are based on exponentiation and require only multiplications and square operations. On the other hand, elliptic-curve cryptography and the digital signature algorithm require inversion too. The inverse of field elements can be calculated by exponentiation using the Fermat theorem or by applying the Extended Euclidean Algorithm (EEA) [1]. The latter has better running time but is more difficult to implement in hardware because it requires inconvenient operations like magnitude-comparisons of integers or bitlength-comparisons of polynomials.

3.1 Addition and Modular Reduction in $GF(p)$

Addition of two integers $A, B \in GF(p)$ is done by calculating the sum $A+B$ with carry propagation. In case, the sum $A + B$ exceeds $p - 1$, a modular reduction is necessary to obtain the result of $A + B \bmod p$ in the range $[0, p - 1]$.

In general, the result of a modular reduction of an integer $I \bmod p$ is the remainder of the integer division $\frac{I}{p}$. The remainder can be calculated using (3).

$$I \bmod p = I - q \cdot p \quad \text{with} \quad q = \left\lfloor \frac{I}{p} \right\rfloor \tag{3}$$

Equation (3) is not very practical because it determines the quotient q by division of large integers. Division can be avoided when the reduction result may exceed the desired interval $[0, p - 1]$. In this case, it possible to estimate a quotient $\hat{q}$ by comparing I with a number N in the magnitude of the modulus p. A good choice is $N = 2^{\lfloor \log_2 p \rfloor}$ where only the most significant bit of p is set. In case p is a generalized Mersenne prime, this estimation is very close.

3.2 Multiplication and Squaring in $GF(p)$

Multiplication $A \cdot B$ is a heavyweight operation compared to addition. The product of two large integers cannot be calculated in a single step. The product is calculated by accumulating partial products $A \cdot b_i$ iteratively—this algorithm is known as the *double-and-add* algorithm. Bitserial multiplication obeys the double-and-add algorithm. It scans all bits b_i of the multiplier B iteratively. If the actual multiplier bit $b_i = 1$, the multiplicand A is accumulated to the intermediate result as done in (4). Two different schemes are possible to scan the multiplier bits: the LSB-first scheme and the MSB-first scheme. The LSB-first scheme starts to scan multiplier bit b_0 and ends with b_{n-1}. The MSB-first scheme operates in the opposite direction.

$$C = A \cdot B = A \cdot \left(\sum_{i=0}^{n-1} b_i 2^i \right) = \sum_{i=0}^{n-1} (A \cdot b_i) 2^i, \quad n = \lceil \log_2 B \rceil \tag{4}$$

Bitserial multiplication can easily be extended to modular multiplication mod p. Extending Equation (4) by an interleaved modular reduction step will reduce the intermediate result in each iteration and yield Algorithm 1. As mentioned above, exact modular reduction would require the calculation of the quotient $q = \lfloor \frac{C}{p} \rfloor$ which involves division. Algorithm 1 avoids division by estimating the quotient $\hat{q}$. The estimation simplifies the algorithm substantially but the result C may exceed the desired range $[0, p-1]$. To obtain a fully reduced result, the modulus p has to be added up to two times.

Algorithm 1 Multiplication in $GF(p)$ with interleaved modular reduction

Input: $A, B \in [0, p-1], \quad 2^{n-1} \leq p < 2^n$
Output: $C = A \cdot B \bmod p$

```
1: C ⇐ 0
2: for i = n − 1 to 0 do
3:     C ⇐ 2 · C + A · b_i
4:     q̂ ⇐ Q_ESTIM(C)
5:     C ⇐ C − q̂ · p
6: end for
7: while C < 0 do
8:     C ⇐ c + p
9: end while
10: return C
```

Squaring is closely related to multiplication because $A^2 \bmod p = A \cdot A \bmod p$. Systems that operate with wordsizes smaller than the bitlength of the operands—like microprocessors—usually have an extra square function to exploit common sub-expressions on wordsize-level. This could save nearly 50 percent of the required wordsize multiplications. For systems that operate on full-length operands there is not such a short cut and squaring is done most efficiently by multiplication.

3.3 Inversion in $GF(p)$

Inversion calculates the multiplicative inverse A^{-1} of an element A. The inverse has the property that $A \cdot A^{-1} \bmod p = 1$. There are two different methods to calculate the inverse. One is based on the theorem of Fermat that states $A^{p-1} \bmod p = 1$ and implies that $A^{p-2} \bmod p = A^{-1} \bmod p$ [1]. Using this theorem, the inverse is calculated by exponentiation which requires about $1.5 \log_2 p$ multiplications—an expensive operation. The other method to calculate the inverse is the Extended Euclidean Algorithm (EEA). It solves the equation $A \cdot X + p \cdot Y = D$ for X, Y and $D = gcd(A, p)$. The EEA's procedure to calculate the inverse is given in Algorithm 2. It is a slightly modified version of the algorithm given in [13].

Algorithm 2 Inversion in $GF(p)$: Extended Euclidean Algorithm (EEA)

Input: $A \in [0, p-1]$, p prime
Output: $A^{-1} \bmod p$

```
1: Y ⇐ A, D ⇐ p, B ⇐ 1, X ⇐ 0
2: while Y ≠ 0 do
3:    while y_0 = 0 do
4:       Y ⇐ Y/2, B ⇐ (B + b_0 p)/2
5:    end while
6:    while d_0 = 0 do
7:       D ⇐ D/2, X ⇐ (X + x_0 p)/2
8:    end while
9:    if Y ≥ D then
10:      Y ⇐ Y − D, B ⇐ (B − X) mod p
11:   else
12:      D ⇐ D − Y, X ⇐ (X − B) mod p
13:   end if
14: end while
15: return X
```

3.4 Addition in $GF(2^m)$

Addition in $GF(2^m)$ is done coefficient-wise as shown in Equation (5).

$$A(x) + B(x) = \sum_{i=0}^{m-1} a_i x^i + \sum_{i=0}^{m-1} b_i x^i = \sum_{i=0}^{m-1} (a_i + b_i) x^i = \sum_{i=0}^{m-1} (a_i \text{ xor } b_i) x^i \quad (5)$$

Coefficients of polynomials are elements of $GF(2) = \mathbb{Z}_2$ and therefore, addition of coefficients is done modulo 2 which corresponds to the Boolean XOR-function. Multiplication of coefficients matches the Boolean AND-function. Subtraction in $GF(2^m)$ is identical with addition because the additive inverse of an element is its identity: $A(x) + A(x) = 0$.

3.5 Multiplication in $GF(2^m)$

Multiplication in $GF(2^m)$ calculates the product of two polynomials and applies modular reduction. Although, polynomial multiplication is completely different from integer multiplication, the resulting algorithm for multiplication in $GF(2^m)$ is very similar to Algorithm 1 for multiplication in $GF(p)$. This property allows building an efficient unified multiplier that supports both fields.

Multiplication in $GF(2^m)$ can also use the double-and-add approach used for multiplication in $GF(p)$ which accumulates partial products as shown in (6). Partial products have to be aligned to the intermediate result which is indicated in (6) by a multiplication by x^i. Multiplication by x^i can easily be computed by shifting the binary representation of the partial product i positions to the left.

$$A(x) \cdot B(x) = A(x) \cdot \left(\sum_{i=0}^{m-1} b_i x^i \right) = \sum_{i=0}^{m-1} \left(A(x) b_i \right) \cdot x^i \tag{6}$$

A modular reduction step after the polynomial multiplication assures that the result is an element of $GF(2^m)$ with an degree less than m. Alternatively, the reduction can be done during the accumulation of partial products as shown in Algorithm 3.

Algorithm 3 Multiplication in $GF(2^m)$ with interleaved modular reduction

Input: $A(x), B(x) \in GF(2^m)$, irreducible polynomial $P(x)$ of degree m
Output: $C(x) = A(x) \cdot B(x) \bmod P(x)$

1: $C(x) \Leftarrow 0$
2: **for** $i = m - 1$ to 0 **do**
3: $\quad C(x) \Leftarrow C(x) \cdot x + A(x) b_i$
4: $\quad C(x) \Leftarrow C(x) + c_m P(x)$
5: **end for**
6: return $C(x)$

The modular reduction $A(x) \bmod P(x)$ in $GF(2^m)$ is done modulo an irreducible polynomial $P(x)$. This operation calculates in principle the remainder of the polynomial division $A(x)/P(x)$. Efficient implementations avoid division by iterated subtraction of the product $P(x) \cdot x^i$. During bitserial multiplication with interleaved modular reduction the intermediate result $C(x)$ can not have higher degree than m. Thus, modular reduction is only necessary when $C(x)$ has degree m. This condition is indicated by $c_m = 1$.

3.6 Squaring in $GF(2^m)$

In contrast to $GF(p)$, squaring in $GF(2^m)$ has lower complexity than multiplication. One reason for this is, that $A(x)^2 \bmod P(x)$ is a linear operation in $GF(2^m)$. Based on this observation one could square efficiently by calculating $A(x)^2 = \sum_{i=0}^{m-1} a_i x^{2i}$. A subsequent modular reduction will yield the desired

result. This method is used in software implementations like [14]. Hardware implementations can exploit this feature when the extension degree m and the irreducible polynomial $P(x)$ are fixed.

3.7 Inversion in $GF(2^m)$

The inverse of an element $A(x) \in GF(2^m)$ can be calculated by the exponentiation $A(x)^{2^m-2}$ mod $P(x)$ or by the Extended Euclidean Algorithm for polynomials (EEA). An improvement of the EEA algorithm is the Modified Almost Inverse Algorithm presented in [14]. Algorithm 4 is a slightly modified version of this. Almost all calculations of the algorithm operate on polynomials but comparisons of polynomials are replaced by integer subtractions and sign testing to avoid additional circuitry in a hardware implementation. In Algorithm 4, polynomials are multiplied by x^{-1} which is a simple shift-right operation.

Algorithm 4 Inversion in $GF(2^m)$: Modified Almost Inverse Algorithm

Input: $0 \neq A(x) \in GF(2^m)$, irreducible polynomial $P(x)$ of degree m
Output: $A(x)^{-1}$ mod $P(x)$

1: $Y(x) \Leftarrow A(x)$, $D(x) \Leftarrow P(x)$, $B(x) \Leftarrow 0$, $X(x) \Leftarrow 1$
2: **loop**
3: **while** $y_0 = 0$ **do**
4: $Y(x) \Leftarrow Y(x) \cdot x^{-1}$, $X(x) \Leftarrow (X(x) + x_0 P(x)) \cdot x^{-1}$
5: **end while**
6: **if** not $(1 - Y < 0)$ **then** {comparison $Y(x) = 1$ by integer subtraction}
7: **return** $X(x)$
8: **end if**
9: **if** $Y - D < 0$ **then** {comparison deg $Y(x) <$ deg $D(x)$ by integer subtraction}
10: $Y(x) \Leftarrow Y(x) + D(x)$, $X(x) \Leftarrow X(x) + B(x)$
11: $D(x) \Leftarrow D(x) + Y(x)$, $B(x) \Leftarrow B(x) + X(x)$
12: **else**
13: $Y(x) \Leftarrow Y(x) + D(x)$, $X(x) \Leftarrow X(x) + B(x)$
14: **end if**
15: **end loop**

4 Architecture

The Elliptic Curve Digital Signature Algorithm (ECDSA) [8] is the target application of the dual-field arithmetic unit. The desired functionality of the unit can be clearly derived from this application. Off course, multiplications in $GF(p)$ and $GF(2^m)$ are the most important functions, but addition and subtraction are required too. In order to calculate the inverse, it is necessary to increment integers, to check whether integer values are negative, and to shift values one position to the left or to the right. Operations like holding the result or clearing the result are obviously useful. The required operations can be summarized in

the following categories: integer arithmetic, modular integer arithmetic, modular polynomial arithmetic, and comparisons.

Functionality is one important aspect of a hardware module. Other quality aspects of a circuit are its size, its speed, and its energy consumption. These factors cannot be optimized independently because they influence each other. Energy efficiency was a prime objective in the design of the arithmetic unit, so the unit was not optimized for lowest gate-count or for a high degree of parallelism. High throughput is achieved by keeping the critical path short to enable operation at high clock frequencies. The architecture is scalable for the maximum bitlength of integers respectively polynomials. Adjusting the bitlength to the requirements of the application (e.g. 192-bit) keeps the gate-count low. It is possible to process smaller integers/polynomials by pre-shifting them in order to align them to the physical dimension of the unit.

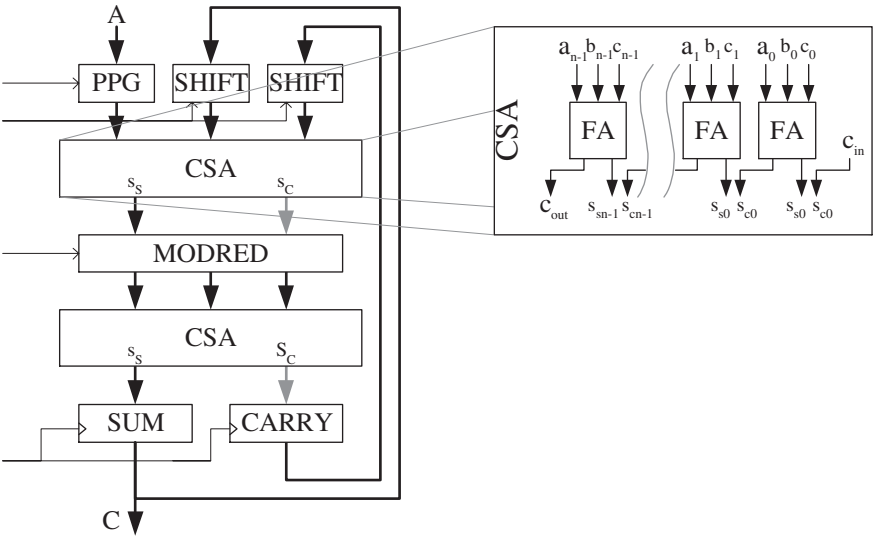


Fig. 1. Architecture of the dual-field arithmetic unit

Figure 1 shows the architecture of the dual-field arithmetic unit. The unit has three major components: a partial-product generator (PPG), a modular reduction unit (MODRED), and a shift unit (SHIFT). During bitserial multiplication, which is done in the MSB-first scheme at full precision, the PPG masks the input A with the actual multiplier bit b_i to generate the partial product $A \cdot b_i$. The partial product is generated the same way for $GF(p)$ and for $GF(2^m)$. Note, that the circuit which serializes the multiplier B is left out in the Figure 1. An adder (CSA) accumulates the partial product to the intermediate result stored in the registers *SUM* and *CARRY*. Prior to this addition the intermediate result is shifted by the *Shift* unit one position to the left to align the last accumulation

result. The *MODRED* unit inserts a modular reduction step by subtracting $\hat{q} \cdot p$ in case of $GF(p)$-operation or $q \cdot P(x)$ in case of $GF(2^m)$-operation. A datapath cycle is finished when the new intermediate result is stored in the registers *SUM* and *CARRY*.

The arithmetic unit uses Carry-Save Adders (CSA) to eliminate carry-propagation delay during $GF(p)$-operation. Carry propagation in conventional adders would cause significant delay. CS-adders prevent this by a redundant representation of the output sum. The redundant sum consists of two integers S_S and S_C, which are stored in the registers SUM and CARRY. The number of storage bits used for this is twice the amount to store the sum in a binary representation. The additional hardware resources are justifiable because they allow to add integers of arbitrary length in constant time. Carry-save adders are based on Formula (7) and are implemented with conventional full-adder cells. The delay of an adder of arbitrary width is equal to the delay of a single full-adder cell.

$$A + B + C = CSA(A, B, C) = S_S + 2S_C \tag{7}$$
$$\text{with} \quad s_{Si} = a_i \text{ xor } b_i \text{ xor } c_i \quad \text{and} \quad s_{Ci} = a_i b_i \text{ or } a_i c_i \text{ or } b_i c_i$$

Carries are only required for addition in $GF(p)$. Addition in $GF(2^m)$ does not use them because polynomials are added coefficient-wise with the Boolean XOR-function. The XOR-function is a sub-function of an CS-adder as Formula (7) reveals: Keeping input $C = 0$, makes the sum-component $S_S = A$ xor B. The carry-component will be $S_C = A$ and B in this case. This property of carry-save adders is used to configure the arithmetic unit for $GF(2^m)$-operation. By forcing the carry-output of both CS-adders to zero, all carries are eliminated and in turn the CS-adders will have the desired functionality of an n-bit XOR-gate.

The input and the output of the dual-field arithmetic unit is restricted to binary n-bit values (input A, output C). Therefore, it is necessary to convert integers stored internally in a redundant representation into their binary representation before output. The conversion of a redundant number into a binary number requires addition with carry propagation. Usually, such a conversion is done with a carry-propagation adder implicating the performance problems mentioned above. Carry-save adders offer another possibility to do the conversion. The observation that the result of an repeated carry-save addition of an redundant number ($S = S_S + 2S_C$) and zero is the binary number S after $\log_2 \log_2 S$ iterations on average, leads to Algorithm 5. Addition, subtraction and multiplication in $GF(p)$ require this conversion before output. Both CS-adders of the arithmetic unit can be used to execute this operation. Hence, the expected running time of an n-bit architecture is halved to $0.5 \log_2 n$ cycles. During conversion, the PPG and MODRED unit have to output 0 in order to keep the third CSA input 0. Conversion is finished when $S_C = 0$. An n-bit NOR-gate reports this condition.

The MODRED unit calculates the correction term for the interleaved modular reduction. It has different functionality for $GF(p)$ and $GF(2^m)$-operations. During $GF(2^m)$-operations, the functionality of MODRED is simple. Whenever the most-significant bit s_{Sm} of the intermediate result S_S is set, MODRED has to output the irreducible polynomial $P(x)$, otherwise 0. The subsequent CS-adder

Algorithm 5 Redundant-to-binary conversion with carry-save adders

Input: redundant number $S = (S_S + 2S_C)$
Output: binary number S

1: **while** $S_C \neq 0$ **do**
2: $\quad (S_S, S_C) \Leftarrow CSA(0, S_S, 2S_C)$
3: **end while**
4: **return** S_S

will execute the reduction by adding the correction term: $S_S \bmod s_{Sm}P(x) = S_S \bmod P(x)$. The reduction works for arbitrary irreducible polynomials and is not restricted to a special kind of polynomials like trinomials or pentanomials.

Modular reduction by a prime integer is more complicated. The quotient is estimated and causes a non-perfect reduction. The intermediate result can exceed the bitlength of the modulus $n = \lceil \log_2 p \rceil$. Therefore, the datapath is chosen to be $n + 2$ bits wide. The quotient estimation works as follows: first the magnitude of the intermediate result is estimated by adding the three highest bits of the redundant intermediate result with carry propagation $\hat{S} = (s_{Sn+1}, s_{Sn}, s_{Sn-1}) + (s_{Cn+1}, s_{Cn}, s_{Cn-1})$. Then the quotient $\hat{q} \in \{-2, -1, 0, 1, 2\}$ is determined by table-lookup. The table entries are chosen such that the desired result of the reduction is in the range $[0, -(p-1)]$. As a consequence of this reduction algorithm the reduced intermediate result $S - \hat{q} \cdot p$ is usually negative and hence the datapath must be capable to handle signed numbers. This reduction scheme works for arbitrary moduli p and is not restricted to generalized Mersenne primes. To ensure that the final result of an operation is fully reduced—or in other words is $\in [0, p-1]$—, the modulus p may have to be added up to two times until the result is positive. Positive results are indicated by a cleared sign bit of S_S and $S_C = 0$. The sign bit is also used for integer comparison, which are based on integer subtractions.

Signed numbers enable the calculation of subtractions. The arithmetic unit can calculate a subtraction $A - B$ by loading A in one cycle and adding $-B$ in the next cycle. $-B$ is calculated by the PPG and a CSA: The PPG generates the one's-complement $\bar{B}$ of B where $\bar{b_i} = \text{not } b_i$ by inverting all bits. The CSA can turn $\bar{B}$ into the two's-complement $-B$ by incrementation. Incrementation is simply achieved by setting the lowest carry bit $s_{C0} = 1$ that is usually 0.

Shift operations are executed in the SHIFT unit. The SHIFT unit can output either 0, its input I, I shift-left 1, or I shift-right 1.

5 Results

The different functionalities of all datapath components can be combined into useful instructions of the whole arithmetic unit. The evolving instructions can be summarized in four categories: load operations, shift operations, addition, and multiplication. Load operations can either load the constants 0, 1 or the values A, $\bar{A}$, or $-A$. The shift operations can shift the stored value one position to the left or to the right. In the addition category are the operations XOR, integer

addition, integer subtraction, incrementation, and integer addition/subtraction with modular reduction.

All the operations listed so far can be executed in one clock cycle. Multiplication takes exactly n clock cycles and can calculate the product of two integers smaller than $2^{n/2}$, or the product of two integers modulo p, or the product of two polynomials mod $P(x)$.

When the datapath is configured for $GF(p)$-operation and the MODRED unit is inactive, the *hold* operation will convert redundant results into their binary representation. On average, 192-bit numbers will be converted in 3.8 cycles, 224-bit numbers in 3.9 cycles, and 256-bit numbers in 4.0 cycles. Control flags indicate whether the result is binary or it is negative. They are always evaluated and are reused for comparisons.

Inversion is a compound operation that has to be controlled from outside. Table 1 lists the estimated number of clock cycles for Algorithm 2 and Algorithm 4. These algorithms are about four times faster than calculating the inverse by exponentiation. The clock-cycle ratio of inversion to multiplication is about 70 for $GF(p)$ and 70 for $GF(2^m)$. This gives reason to avoid inversion when possible and advises to use projective coordinates when implementing elliptic curve cryptography. Table 1 also lists expected running times for a elliptic-curve scalar-multiplication using projective coordinates. All estimates are conservative and include the transformation to affine coordinates.

Table 1. Estimited cycles for inversion and ECC scalar multiplication

$GF(p)$	INV (Alg. 2) [cycles]	ECC proj. [cycles]
192-bit	14,000	720,000
224-bit	16,500	900,000
256-bit	19,400	1,150,000

$GF(2^m)$	INV (Alg 4) [cycles]	ECC proj. [cycles]
163-bit	11,000	490,000
233-bit	16,200	905,000
283-bit	20,700	1,405,000

The dual-field arithmetic unit requires only a few hardware resources. Four n+2-bit register are needed to store the modulus, the multiplier, and the result in redundant representation. The two SHIFT units can be implemented with $2n+4$ 4-to-1 multiplexers. The PPG unit is built of $n + 2$ AND-gates and the same amount of XOR-gates. The MODRED unit has the same complexity as PPG plus $n + 2$ 2-to-1 multiplexers. Two instances of CS-adders require $2n + 4$ full-adder cells and $2n + 4$ AND-gates which eliminate carries during $GF(2^m)$-operation. Table 2 lists the gate count for different bitlengths and gives a rough estimation of the area requirements of a standard-cell implementation on the 0.35 μm CMOS process from Austriamicrosystems. The arithmetic unit is also well suited for a full-custom implementation. The regular part of the datapath is composed of only half a dozen of different gates. It should be of no difficulty to find a bitslice architecture for that part and to design leaf-cells for the gates in an appropriate logic style in order to obtain a sound full-custom layout. The datapath does not

need sophisticated control because most instructions are executed in a single cycle. Only multiplication consumes more cycles. The control unit for bitserial multiplication can be used both for $GF(p)$ and for $GF(2^m)$-operation because the same double-and-add algorithm in the MSB-first scheme is used.

Table 2. Gate count and estimated area on a 0.35 μm CMOS process

Size	AND	XOR	MUX2	MUX4	FA	REG	area on 0,35 μm
163-bit	660	330	165	330	330	660	0.57 mm^2
224-bit	904	452	226	452	452	904	0.78 mm^2
283-bit	1140	570	285	570	570	1140	0.99 mm^2

Most of the design decisions for the dual-field arithmetic unit were guided by low-power considerations. Especially, the design on the algorithmic level and the architectural level of a digital circuit offer promising options to save power [9]. Contrary to low-power measures on logic level, they are difficult to estimate. Therefore, a qualitative reasoning will be given. One design goal was to keep the critical path short. This implicates on one hand a high clock frequency and gives on the other hand the possibility to scale the supply voltage *VDD* of CMOS circuits. Lowering *VDD* is an effective technique to save power as it contributes quadratically to the dynamic power consumption [9]. A lowered supply voltage will also slow down the circuit: VDD can be decreased until the critical path delay reaches the clock period. Short critical paths have another advantage for low-power circuit design: The probability of undesired signal transitions (glitches) is lowered. Glitches will occur more frequently when the combinational logic-depth is high. To ensure a short critical path of the arithmetic unit, CS-adders were chosen. The critical path spans the partial product generator PPG, the modular reduction unit MODRED and two CS-adders.

A low-power driven design decision on the architectural level is the modular reduction unit MODRED. From the functional point of view, it would be possible to omit the MODRED unit and to cover its functionality by an enlarged partial product generator PPG. Thereby, a former single-cycle operation with an interleaved modular reduction would require two clock cycles: One cycle for the operation itself and one for the modular reduction step. This would certainly increase the energy-delay product. Furthermore, such an architecture would have negative impact on the signal activity of the input A of the arithmetic unit: During multiplication, this bus would always change between the multiplicand A and the modulus p or $P(x)$. The insertion of an extra MODRED trades increased area-demands for lower power-consumption and helps to avoid wasteful signal activity.

6 Conclusion

In this article we presented a dual-field arithmetic unit that offers all instructions to implement the elliptic curve digital signature standard over prime fields $GF(p)$ and binary extension fields $GF(2^m)$. Therefore, the unit can calculate shift-operations, increments, and comparisons besides addition and multiplication. These operations enable to calculate the inverse with the extended Euclidean algorithm.

A design objective for the unit was energy efficiency, which yielded a low-power architecture that can be realized on moderate silicon area. The unit requires only little more hardware resources than a mere $GF(p)$-multiplier. The $GF(2^m)$-functionality and some other useful operations come at almost no additional cost. The use of carry-save adders guarantees a short critical path that allows operation at high clock frequencies—independent of the chosen datapath precision. The simplicity of the architecture with its inherent regularity and its limited number of leaf cells makes it well suited for a full-custom implementation.

References

1. A. Menezes, P. van Oorschot, and S. Vanstone, *Handbook of Applied Cryptography*, CRC Press, New York, 1997.
2. R. Lidl and H. Niederreiter, *Introduction to finite fields and their applications*, Cambridge University Press, Cambridge, 1986.
3. A. K. Lenstra and E. R. Verheul, *Selecting Cryptographic Key Sizes*, Journal of Cryptology, vol. 14, pp. 255–293, 2001.
4. ANSI X9.62, *Public Key Cryptography for the Financial Services Industry: The Elliptic Curve Digital Signature Algorithm (ECDSA)*, ANSI standard, 1999.
5. ANSI X9.63, *Public Key Cryptography for the Financial Services Industry: Key Agreement and Key Transport Using Elliptic Curve Cryptography*, ANSI standard, 2001.
6. IEEE P1363, *Standard Specifications for Public-Key Cryptography*, IEEE standard, 2000.
7. ISO / IEC 15946, *Information Technology - Security Techniques - Cryptographic Techniques Based on Elliptic Curves*, Committee Draft (CD), 1999.
8. National Institute of Standards and Technology, *Digital Signature Standard*, FIPS Publication 186–2, Feb. 2000.
9. J. M. Rabaey, M. Pedram (ed.), *Low Power Design Methodologies*, Kluwer Academic Publishers, 1996.
10. E. D. Mastrovito, *VLSI Architectures for Computations in Galois Fields*, PhD thesis, Linköping University, Linköping, Sweden, 1991.
11. C. Paar, *Efficient VLSI Architectures for Bit Parallel Computation in Galois Fields*, PhD thesis, Universität Essen, 1994.
12. Ç. Koç, C.Y. Hung, *A Fast Algorithm for Modular Reduction*, IEE Proceedings: Computers and Digital Techniques 145(4), pp. 265–271, July 1998.
13. M. Brown, D. Hankerson, A. Menezes, *Software Implementation of the NIST Elliptic Curves over Prime Fields*, Proceedings of CT-RSA 2001, LNCS 2020, pp. 250–265, Springer Verlag, 2001.

14. D. Hankerson, J. L. Hernandez, A. Menezes, *Software Implementation of Elliptic Curve Cryptography over Binary Fields*, Cryptographic Hardware and Embedded Systems - CHES 2000, LNCS 1965, pp. 1–24, Springer Verlag, Berlin, 2000.
15. E. Savaş, A. Tenca, Ç. Koç, *A Scalable and Unified Multiplier Architecture for Finite Fields $GF(p)$ and $GF(2^m)$*, Cryptographic Hardware and Embedded Systems - CHES 2000, LNCS 1965, pp. 281–296, Springer Verlag, Berlin, 2000.
16. J. Großschädl, *A Bitserial Unified Multiplier Architecture for Finite Fields $GF(p)$ and $GF(2^m)$*, Cryptographic Hardware and Embedded Systems - CHES 2001, LNCS 2162, pp. 206–223, Springer Verlag, 2001.
17. J. Goodman, A. P. Chandrakasan, *An Energy-efficient Reconfigurable Public-Key Cryptography Processor*, IEEE Journal of Solid-State Circuits, pp. 1808–1820, November 2001.

Error Detection in Polynomial Basis Multipliers over Binary Extension Fields

Arash Reyhani-Masoleh[1] and M.A. Hasan[2]

[1] Centre for Applied Cryptographic Research,
Department of Combinatorics and Optimization,
University of Waterloo, Waterloo, Ontario, Canada N2L 3G1.
areyhani@math.uwaterloo.ca
[2] Department of Electrical and Computer Engineering,
University of Waterloo, Waterloo, Ontario, Canada N2L 3G1.
ahasan@ece.uwaterloo.ca

Abstract. In many of cryptographic schemes, the most time consuming basic arithmetic operation is the finite field multiplication and its hardware implementation may require millions of logic gates. It is a complex and costly task to develop such large finite field multipliers which will always yield error free outputs. In this effect, this paper considers fault tolerant multiplication in finite fields. It deals with detection of errors of bit-parallel and bit-serial polynomial basis multipliers over finite fields of characteristic two. Our approach is to partition the multiplier structure into a number of smaller computational units and use the parity prediction technique to detect errors.

Keywords: Finite fields, fault tolerant computing, polynomial basis multiplier, error detection.

1 Introduction

Among the basic arithmetic operations over finite fields $GF(2^m)$, multiplication is the one which has received most attention in the literature [7,4,11,9]. This is mainly because the implementation of a multiplier is much more complex compared to a finite field adder and using multiplication operation repeatedly one can perform other difficult field operations, such as inversion and exponentiation, which are extensively used in cryptographic systems [1,10].

Finite field multiplication is quite different from its counterparts in integer and floating point number systems. For todays cryptographic applications, the field size can be very large and each input of the multiplier can be 160 to 2048 bits long. Such a multiplier may require millions of logic gates and it is a challenging task to implement it free of faults. If one can have a multiplier which is capable of detecting error on-line at the presence of certain faults, cryptographic schemes can be operated more reliably. The importance of eliminating errors in cryptographic computations has been pointed out in some recent articles, for examples [2,5]. The presence of faults in cryptosystems can lead to an active

B.S. Kaliski Jr. et al. (Eds.): CHES 2002, LNCS 2523, pp. 515–528, 2003.

attack and the simplest way to prevent such an attack is to ensure that the computational device verifies the values it computes before sending them out.

In an attempt to detect errors in finite field multipliers, the authors of [3] have considered bit-serial multipliers in $GF(2^m)$ and have presented error detection schemes for four types of multipliers using a parity prediction technique. Their polynomial basis scheme for error detection is applicable to a special class of fields. These fields are defined using irreducible all-one polynomials that are available for certain values of m only. Additionally, when an all-one polynomial is irreducible, the corresponding m is not a prime. This makes many designers to avoid such a value of m and the corresponding irreducible all-one polynomial that define the underlying field for certain cryptosystems, such as those based on elliptic curve cryptography.

In this paper, we consider $GF(2^m)$ multipliers of both bit-parallel and bit-serial types. The polynomial basis is used for representing the field elements. We investigate error detection techniques for such multipliers and develop parity prediction based error detection schemes for both bit-serial and bit-parallel multipliers. The new schemes can be used for any field defining irreducible binary polynomial.

2 Preliminaries

2.1 Multiplication Using Polynomial Basis

Let

$$F(z) = z^m + \sum_{i=0}^{m-1} f_i z^i \tag{1}$$

be a monic irreducible polynomial over $GF(2)$ of degree m, where $f_i \in GF(2)$ for $i = 0, 1, \cdots, m-1$. Let $\alpha \in GF(2^m)$ be a root of $F(z)$, *i.e.*, $F(\alpha) = 0$. Then the set $\{1, \alpha, \alpha^2, \cdots, \alpha^{m-1}\}$ is known as the polynomial (or standard) basis and each element of $GF(2^m)$ can be written with respect to (w.r.t.) this basis, *i.e.,* if A is an element of $GF(2^m)$, then

$$A = \sum_{i=0}^{m-1} a_i \alpha^i,\ a_i \in \{0, 1\}, \tag{2}$$

where a_i's are the coordinates of A w.r.t. polynomial basis (PB). For convenience, these coordinates will be denoted in vector notation as

$$\mathbf{a} = [a_0, a_1, a_2, \cdots a_{m-1}]^T, \tag{3}$$

where T denotes the transposition of a vector.

Let C be the product of any two elements A and B of $GF(2^m)$. Then, C can be represented w.r.t. PB as follows:

$$A \cdot B = A \cdot \sum_{i=0}^{m-1} b_i \alpha^i = \sum_{i=0}^{m-1} b_i \cdot (A\alpha^i),$$

$$C = A \cdot B \mod F(\alpha) = \sum_{i=0}^{m-1} b_i \cdot ((A\alpha^i) \mod F(\alpha)) \tag{4}$$

$$= \sum_{i=0}^{m-1} b_i \cdot X^{(i)}, \tag{5}$$

where

$$X^{(i)} = \alpha \cdot X^{(i-1)} \mod F(\alpha), \quad 1 \le i \le m-1 \tag{6}$$

and

$$X^{(0)} = A.$$

A bit-parallel architecture for $GF(2^m)$ multiplication using (5) is shown in Figure 1. It mainly consists of three types of modules, namely, sum, pass-thru and α modules. The sum module (denoted as a double circle with a plus inside) is to simply add two $GF(2^m)$ elements and it can be realized in hardware using m two-input XOR gates. The pass-thru module (denoted as a double circle with a dot inside) is to multiply a $GF(2^m)$ element by a $GF(2)$ element, *i.e.*, if $X^{(i)} \in GF(2^m)$ and $b_i \in GF(2)$ are two inputs to a pass-thru module, then its output is

$$b_i\, X^{(i)} = \begin{cases} X^{(i)} & \text{if } b_i = 1, \\ 0 & \text{if } b_i = 0. \end{cases}$$

In hardware, each pass-thru module consists of m two-input AND gates.

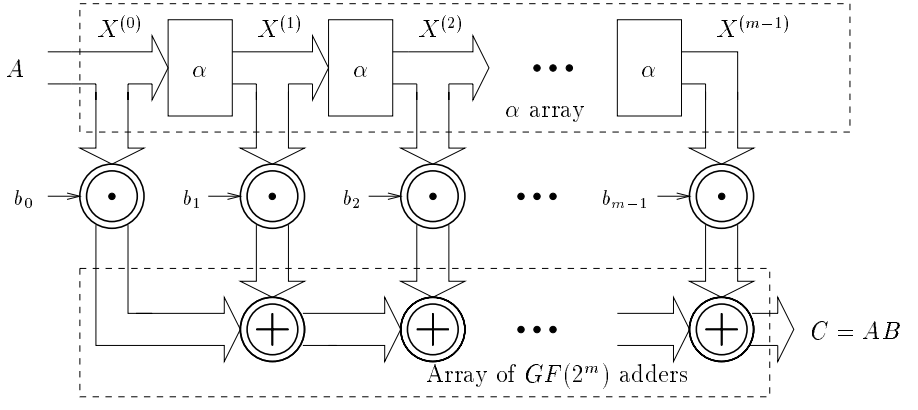


Fig. 1. Multiplication of two elements in $GF(2^m)$

In Figure 1, the third module (*i.e.*, the rectangular shape α module) multiplies its input, which is an element of $GF(2^m)$, by α and reduces the result modulo $F(\alpha)$. Thus, this module is to essentially realize equation (6) in hardware.

Since α is a root of $F(z)$,

$$F(\alpha) = \alpha^m + \sum_{i=0}^{m-1} f_i \alpha^i = 0. \tag{7}$$

Then multiplication of an arbitrary element $A \in GF(2^m)$ by α gives

$$A \cdot \alpha = (\sum_{i=0}^{m-1} a_i \alpha^i) \cdot \alpha = \sum_{i=0}^{m-1} a_i \alpha^{i+1} = \sum_{i=1}^{m-1} a_{i-1} \alpha^i + a_{m-1} \alpha^m. \tag{8}$$

Using (7) and (8), one can write

$$\begin{aligned} X &\triangleq A \cdot \alpha \bmod F(\alpha) \\ &= a_{m-1} \cdot f_0 + \textstyle\sum_{i=1}^{m-1} (a_{i-1} + a_{m-1} \cdot f_i) \alpha^i, \end{aligned} \tag{9}$$

where x_i's are in $GF(2)$ and are the coordinates of X w.r.t. the PB. For any irreducible polynomial over $GF(2)$, $f_0 = 1$. Thus from (9), we write the coordinates of X as

$$x_i = \begin{cases} a_{i-1} + a_{m-1} \cdot f_i & 1 \le i \le m-1, \\ a_{m-1} & i = 0. \end{cases} \tag{10}$$

If ω is the Hamming weight of the irreducible polynomial $F(z)$, then the realization of (10) requires $\omega - 2$ XOR gates, and so does an α module. Thus, unlike the sum and pass-thru modules, the α module has a space (or circuit) complexity which depends on $F(z)$. The space complexity is minimum when $F(z)$ is a trinomial and maximum when $F(z)$ is an all-one-polynomial (AOP).

In vector notations, the coordinates of the $GF(2^m)$ multiplication can be calculated by the well-known formulation [7] as

$$\mathbf{c} = [c_0, c_1, \cdots c_{m-1}]^T = \mathbf{M} \cdot \mathbf{b}, \tag{11}$$

where $\mathbf{M} = [m_{i,j}]$, $m_{i,j} \in GF(2)$, is the $m \times m$ product matrix and $\mathbf{b} = [b_0, b_1, \cdots b_{m-1}]^T$. Note that in Figure 1, the α array generates $m_{i,j}$'s and another part of the multiplier which consists of all pass-thru and sum modules realizes matrix-vector multiplication of (11).

2.2 Error Detection Strategy

In the following sections, we investigate error detection schemes for $GF(2^m)$ multiplication operation that relies on the architecture shown in Figure 1. Towards this effort, the parity prediction method is used. This method is shown in Figure 2 where the CUT (circuit under test) block can be either a complete finite field multiplier or a part of it with A and B as inputs and Y as output, where $A, B, Y \in GF(2^m)$. In this figure, the parity generation (PG) block produces the actual parity of Y, i.e., $p_Y = \sum_{i=0}^{m-1} y_i$, where y_i's are the coordinates of Y w.r.t. the PB. The actual parity p_Y is then compared with the predicted parity $\hat{p}_Y$

using a single XOR gate as shown in the figure. This comparison is monitored by an error indicator flag $\hat{e}_{\text{CUT}}$ where $\hat{e}_{\text{CUT}} = 0$ indicates that no error has been detected and $\hat{e}_{\text{CUT}} = 1$ flags the detection of errors. The parity prediction (PP) block predicts the parity of the output Y using a PP function which depends only on the inputs of the CUT as

$$\hat{p}_Y = \Gamma_{\text{CUT}}(A, B).$$

We assume that the parity of A and B (i.e., p_A and p_B, respectively) are available or they can be reliably pre-computed while loading the coordinates of A and B into the multiplier. We also assume that the PP and PG blocks can be made fault free or any fault in them can be detected using a suitable mechanism since these blocks are simple and/or regular (for example PP can be as simple as an XOR gate and PG is a modulo 2 adder). In the following sections, we derive the function Γ_{CUT} for each of the modules of the multiplier of Figure 1.

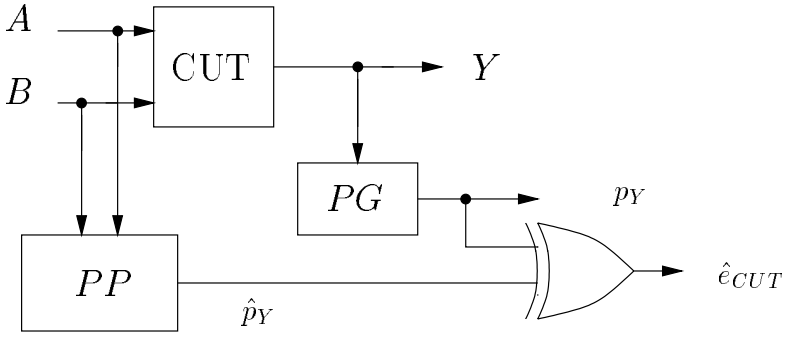


Fig. 2. Error indication of the circuit under test (CUT) using parity prediction method.

For the purpose of this investigation, we consider $GF(2^m)$ multiplier circuit with a single fault. The single fault case provides simplicity in our analysis. Although various types of multiple faults in the multiplier can be detected, we first consider the single fault case and then we show how multiple faults can be detected. This fault is modeled as a stuck-at fault, which appears to be the most common model used for logical faults. For this model, a fault in a logical gate (i.e., XOR, AND, OR, etc.) results in one of its inputs or the output being fixed to either a logic 0 (stuck-at-0, or s-a-0 in short) or a logic 1 (stuck-at-1, or s-a-1), respectively [6].

3 Parity Predictions of Individual Module

In the following, we obtain the parity prediction functions of the modules of the bit-parallel multiplier of Figure 1. PB multipliers (both bit-parallel and bit-serial) that are capable of detecting errors are considered in the next section.

3.1 Parity Prediction in α Module

Let ω be the Hamming weight of the irreducible polynomial $F(z)$. Then, (1) can be written as

$$F(z) = 1 + \sum_{j=1}^{\omega-2} z^{\rho_j} + z^m, \tag{12}$$

where ρ_j's are powers of z in (1) with $f_{\rho_j} = 1$, $1 \le j \le \omega - 2$. Then we have

$$1 \le \rho_1 < \rho_2 < \rho_3 \cdots < \rho_{\omega-2} \le m - 1$$

and (10) can be written as

$$x_i = \begin{cases} a_{\rho_j - 1} + a_{m-1} & i = \rho_j,\ 1 \le j \le \omega - 2, \\ a_{i-1 \bmod m} & \text{otherwise.} \end{cases} \tag{13}$$

Using (13), a circuit diagram for the α module is shown in Figure 3. Note that a stuck-at fault in one of the $(\omega - 2)$ XOR gates of this module causes at most one error at the output.

For $j \in \{\rho_1, \rho_2, \cdots, \rho_{\omega-2}\}$, assume that the j-th gate in Figure 3 is faulty. Then all the output coordinates, except x_j, are error free. If the upper input of the j-th gate is stuck, then the erroneous j-th coordinate is

$$\dot{x}_j = \begin{cases} a_{m-1} \text{ for s-a-0}, \\ \overline{a}_{m-1} \text{ for s-a-1}, \end{cases} \tag{14}$$

where $\overline{x}$ indicates complement of x. On the other hand, if the lower input is stuck, then

$$\dot{x}_j = \begin{cases} a_{j-1} \text{ for s-a-0}, \\ \overline{a}_{j-1} \text{ for s-a-1}. \end{cases} \tag{15}$$

Detection of such errors are discussed below.

Assume that A and X are the input and output of the α module, respectively. Then we have the following lemma.

Lemma 1. *Let $p_A = \sum_{i=0}^{m-1} a_i$ and $p_X = \sum_{i=0}^{m-1} x_i$ be the parity bits of A and X, respectively. Then, the predicted parity of X is*

$$\hat{p}_X = \Gamma_\alpha = p_A + a_{m-1}, \tag{16}$$

where a_{m-1} is the $(m-1)$-th coordinate of input A.

Proof. Using (10), $\hat{p}_X$ can be written as

$$\hat{p}_X = a_{m-1} + \sum_{i=1}^{m-1} (a_{i-1} + a_{m-1} f_i) = a_{m-1} \sum_{i=1}^{m-1} f_i + \sum_{i=0}^{m-1} a_i \,.$$

Since $F(z)$ is irreducible over $GF(2)$, $F(z)$ is not divisible by $z + 1$, and $F(1) = 1$. Then from (1), one obtains $\sum_{i=1}^{m-1} f_i = f_0 = 1$ and the proof is complete.

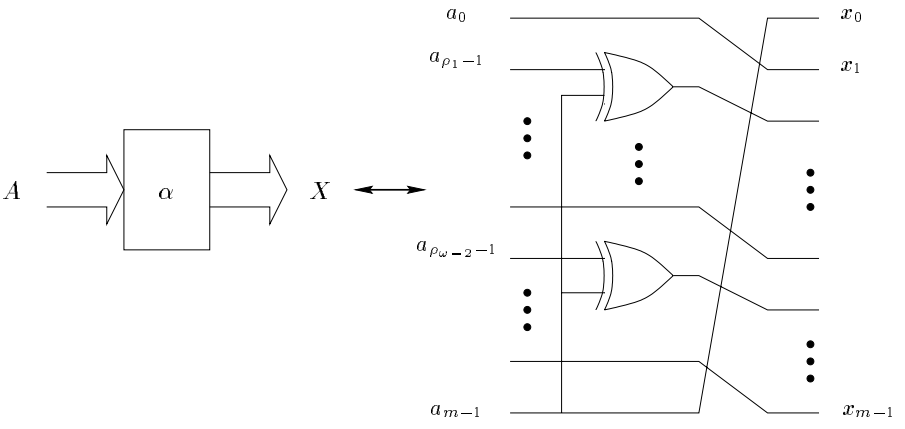


Fig. 3. The original circuit of α module.

Using (16), we can obtain the relation between X and A in the fault free α module as

$$p_X + p_A = a_{m-1}. \tag{17}$$

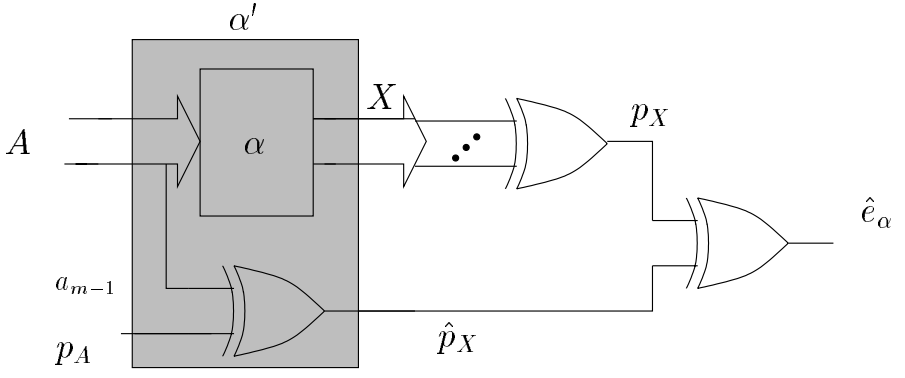


Fig. 4. The circuit for detecting a single fault.

In the α module, a stuck-at fault in one of its gates will result in an output which is different from X and (16) will not hold. Thus, equation (16) can be used for detecting an error in the output of the α module. Circuit for detecting such errors is shown in Figure 4, where $\hat{e}_\alpha = 0$ indicates that no error has been detected and $\hat{e}_\alpha = 1$ flags the detection of an error. Since (16) is over $GF(2)$, the values of $\hat{e}_\alpha$ would detect not only a single error, but also any odd number of errors. With a similar argument, it is clear that even number of errors are not detected by $\hat{e}_\alpha$.

3.2 Parity Predictions of Sum and Pass-Thru Modules

The sum module of Figure 1 is a finite field adder which produces sum of the two elements of $GF(2^m)$ at its output. Let $A = (a_0, a_1, \cdots, a_{m-1})$ and $B = (b_0, b_1, \cdots, b_{m-1})$ be two inputs to this module. Then the output is $D = A+B = (d_0, d_1, \cdots, d_{m-1})$ where $d_i = a_i + b_i$ for $0 \leq i \leq m-1$. The architecture of this module uses m two-input XOR gates. Let $p_A = \sum_{i=0}^{m-1} a_i$ and $p_B = \sum_{i=0}^{m-1} b_i$ be the parity bits of A and B, respectively. Then the parity bit of the output, $p_D = \sum_{i=0}^{m-1} d_i$, is predicted by

$$\hat{p}_D = \Gamma_{sum} = p_A + p_B \tag{18}$$

using one extra XOR gate. Let us denote this $m+1$ XOR gates as the new sum module.

The pass-thru module of Figure 1 multiplies an element $A \in GF(2^m)$ by a single bit $b \in GF(2)$ which can be implemented using m two-input AND gates. Let $G \in GF(2^m)$ be the output of such a module with inputs of A and b. Thus, the output of this module G is zero when $b = 0$ and A when $b = 1$.

Detection of an odd number of errors is accomplished by using a single parity bit similar to the α and sum modules. Let A and $p_A = \sum_{i=0}^{m-1} a_i$ be the input of the pass-thru module and its parity bit respectively. Then, the parity bit of the output $G = bA$ is found as $p_G = \sum_{i=0}^{m-1} g_i$, where $g_i = b \cdot a_i$, $0 \leq i \leq m-1$, are the coordinates of G. Thus, the predicted parity bit of the output can be expressed as

$$\hat{p}_G = \Gamma_{pass} = b \cdot p_A \tag{19}$$

which requires only one AND gate for its implementation. Let us denote the original pass-thru module together with this AND gate as the new pass-thru module similar to the new sum module. These new modules are used in the next section.

4 Error Detections in Polynomial Basis Multipliers

The discussions of the previous section deals with the parity prediction functions of individual modules of the multiplier of Figure 1. Using these parity functions, below we attempt to detect errors in the entire multiplier.

4.1 Bit-Parallel PB Multiplier

Let us generalize (1) for the cascading of j, $1 \leq j \leq m-1$, α modules as follows:

Lemma 2. *As defined earlier $x_{m-1}^{(k)}$ and b_j are the $(m-1)$-th and j-th coordinates of $X^{(j)} = A\alpha^j \mod F(\alpha)$ and B w.r.t. the polynomial basis, respectively. Then*

$$\hat{p}_{X^{(j)}} = p_A + \sum_{k=0}^{j-1} x_{m-1}^{(k)}, \qquad j = 1, 2, \cdots, m-1, \tag{20}$$

Thus, the parity bit of the output of the polynomial basis multiplier can be predicted using the following theorem.

Theorem 1. *Let C be the product of two arbitrary elements A and B of $GF(2^m)$. Let p_A, p_B and p_C be the parity bits of A, B and C respectively. Then,*

$$\hat{p}_C = \sum_{j=0}^{m-1} b_j \hat{p}_{X^{(j)}}, \tag{21}$$

A proof of the above theorem is not included here for lack of space. Note that the theorem is not restricted to any particular irreducible polynomials. When $F(z)$ is an all-one polynomial, the expression for $\hat{p}_C$, which can be obtained from Theorem 1, matches the corresponding result reported in [3].

To detect only one error (in general any odd number of errors) at the output of the multiplier, equation (21) can be realized easily by replacing all the α, pass-thru and sum modules in Figure 1 with the α' module and the new pass-thru and sum modules as shown in Figure 5 (these three new modules are shaded in this figure to distinguish them from the old ones). The bus width of this multiplier is $m+1$. Since the output of any gate of the shaded pass-thru and sum modules in Figure 5 is connected to only one gate, the single stuck fault at any gate of these modules changes only one coordinate of the output of this multiplier. Therefore, a circuit that compares the actual parity p_C with the predicted $\hat{p}_C$, which is shown at the end of the figure, is capable of detecting any single fault in the shaded sum and pass-thru modules of Figure 5. Also, it is clear that any single fault in any XOR gate in the parity generation circuit p_C and the very last XOR gate can be detected by $\hat{e}$. This circuit, however cannot detect a single stuck-at fault in any of the α' modules with the exception of the rightmost α' module, because such a fault is most likely to change more than one bit of the multiplier output. Then, these errors cannot be detected if an even number of output bits are changed due to a single fault in the α' array. To overcome this problem, the following method is proposed.

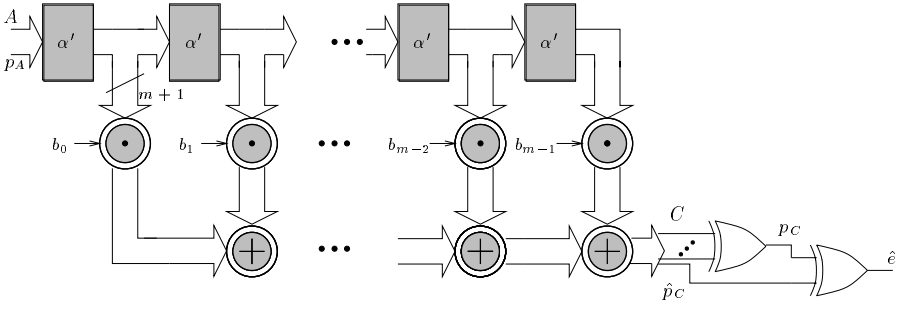


Fig. 5. Multiplication of two elements in $GF(2^m)$ with error detection capability.

For detecting a single fault in the entire multiplier one can change the α array (all α' modules excluding the XOR gates for parity prediction of $\hat{p}_{X^{(j)}}$'s) in such a way so that all $X^{(i)}$'s, $0 \leq i \leq m-1$, are obtained directly from A (instead of $X^{(i-1)}$), i.e., $X^{(i)} = \alpha^i A$ and this is shown in Figure 6. This makes the output of any gate inside the new α array connected to only one gate. In Figure 6, the output of the α^i modules, $X^{(i)}$, $1 \leq i \leq m-1$, are found directly from A. Also, it is noted that the coordinates of $X^{(i)}$'s are obtained using the following matrix equation

$$\mathbf{x}^{(i)} = \mathbf{G}^i \cdot \mathbf{a}, \ 1 \leq i \leq m-1, \tag{22}$$

where $\mathbf{x}^{(i)}$ is a vector whose entries are coordinates of $X^{(i)}$ defined by (6) and

$$\mathbf{G} = \begin{bmatrix} 0 & 0 & \cdots & 0 & 1 \\ 1 & 0 & \cdots & 0 & f_1 \\ 0 & 1 & \cdots & 0 & f_2 \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & \cdots & 1 & f_{m-1} \end{bmatrix}$$

is the α-multiplication matrix. Using (22), the α^i module in Figure 6 is realized with XOR gates according to the $\mathbf{G}^i$ matrix. As a result, a single stuck-at fault at any logic gate in the multiplier, except in the XOR gates for parity prediction of $\hat{p}_{X^{(j)}}$'s, can affect at most one bit of the output so that it is detected using the parity prediction of (21).

Note that $x_{m-1}^{(k)}$ used in (20) is a function of A and can be calculated and then should be realized separately by using Proposition 4.1 of [7] as follows

$$x_{m-1}^{(k)} = a_{m-1-k} + \sum_{t=0}^{k-1} q_{k-1-t,m-1}\, a_{m-1-t}\,, \quad k = 1, \cdots, m-1, \tag{23}$$

where $q_{i,m-1} \in \{0,1\}$, for $i = 0, 1, \cdots, m-2$, is the i-th entry of the last column of the $m-1 \times m$ binary reduction matrix $\mathbf{Q}$ associated with $F(z)$ as follows:

$$\begin{bmatrix} \alpha^m \\ \alpha^{m+1} \\ \vdots \\ \alpha^{2m-2} \end{bmatrix} = \mathbf{Q} \begin{bmatrix} 1 \\ \alpha \\ \vdots \\ \alpha^{m-1} \end{bmatrix} \mod F(\alpha).$$

By substituting (23) into (21), one can realize $\hat{p}_{X^{(j)}}$ as a function of the coordinates of A using XOR gates. Thus any single stuck-at fault in the entire new multiplier results in at most one error and can be detected.

4.2 Bit-Serial PB Multiplier

The PB multiplier of Figure 1 can be realized in a bit-serial fashion as shown in Figure 7. In this figure, both X and Y are m bit registers. Let $X(n)$ and $Y(n)$ be

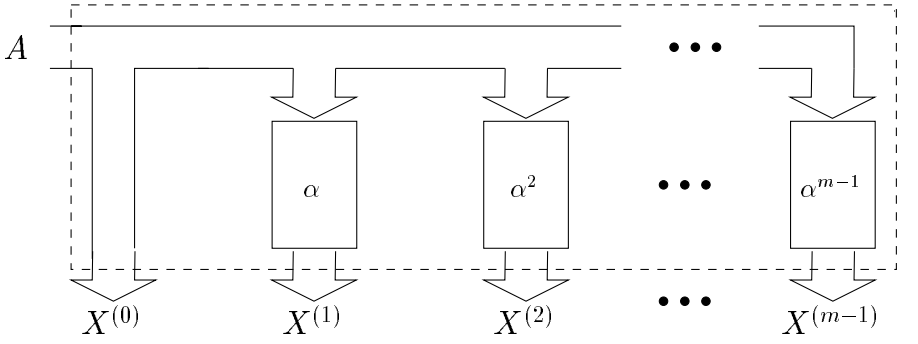


Fig. 6. The architecture of new α array to have a detection capability at the output.

the contents of X and Y registers, respectively, at nth, $1 \leq n \leq m$, clock cycle. Suppose the X register is initialized by A, i.e., $X(0) = A$, then the content of this register at the nth clock cycle is $X(n) = X^{(n)}$, where $X^{(n)} \in GF(2^m)$ is defined in (6). Also, suppose that the register Y is cleared at the initial step, i.e., $Y(0) = 0$. Then one can obtain the content of Y at the first clock cycle as $Y(1) = b_0 A$ and in general at the nth clock cycle as $Y(n) = b_0 A + \sum_{i=1}^{n-1} b_i X(i)$, $1 < n \leq m$. It is easy to verify that after m clock cycle Y contains $C = AB \in GF(2^m)$, i.e., $Y(m) = C$.

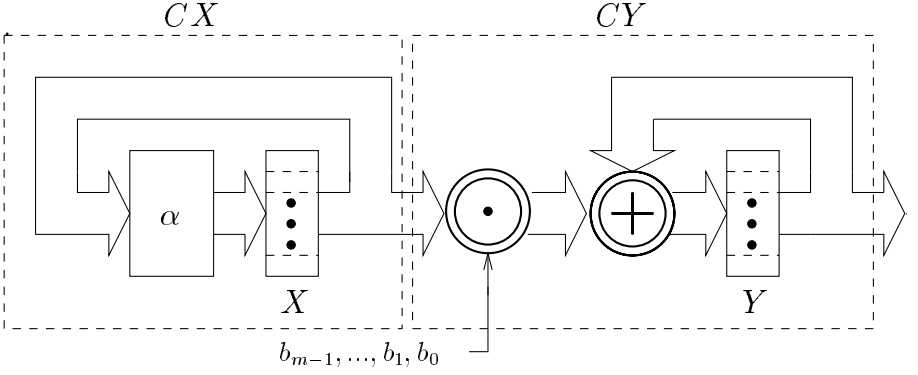


Fig. 7. Bit-serial PB multiplier.

In order to detect errors in the bit-serial multiplier of Figure 7, we check the contents of two registers in every clock cycle. Consider Figure 7 before the triggering of the nth clock cycle when the input and output of the X register are $X(n)$ and $X(n-1)$, respectively and using Lemma 1, we have

$$\hat{p}_{X(n)} = p_{X(n-1)} + x_{m-1}(n-1) = \sum_{i=0}^{m-2} x_i(n-1),$$

where $x_i(n-1) \in GF(2)$ is the ith coordinate of $X(n-1)$. In order to compare $\hat{p}_{X(n)}$ with the actual value of $p_{X(n)}$, we store $\hat{p}_{X(n)}$ into a 1 bit register D_X as shown in Figure 8. Then, after the nth clock cycle, $X(n)$ appears at the output of the X register and the actual value of $p_{X(n)}$ is evaluated and compared with the value of D_X, i.e., $\hat{p}_{X(n)}$ using the last XOR gate of Figure 8. Similar expression can be obtained for the Y register. Since $Y(n) = Y(n-1) + b_{n-1}X(n-1)$, then

$$\hat{p}_{Y(n)} = p_{Y(n-1)} + b_{n-1}p_{X(n-1)},$$

and can be implemented and compared with the actual value of $p_{Y(n)}$ as shown in Figure 8. As a result, after the first clock cycle, both $\hat{e}_{CX}$ and $\hat{e}_{CY}$ should be 0 during the next m clock cycles if there are no single errors.

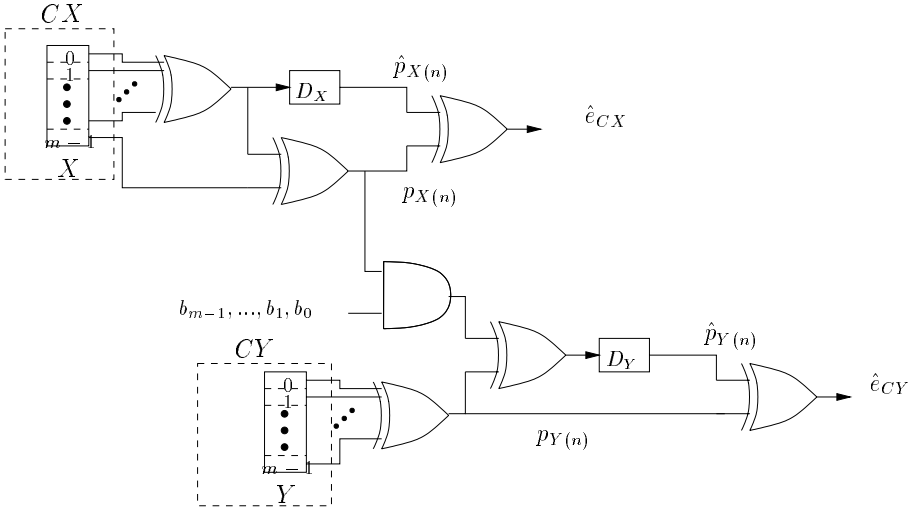


Fig. 8. Detection of errors in the bit-serial PB multiplier.

5 Conclusions and Future Work

In this article, we have considered detection of errors in polynomial basis multipliers. We have used a multiplier structure where a single stuck-at fault causes only odd number of errors at the output. Towards the detection of this type of errors, necessary theoretical results have been presented. Compared to the previously published results [3], the work presented here is quite generic in the sense that it can be applied to any irreducible polynomial defining the field. The

parity prediction method of [3] is only for bit-serial multipliers and based on the prediction of the output parity after the final clock cycle and then comparing it with the actual parity. Although, it reduces the cost of overhead, but its probability of error detection is only about 50% or less. This is because a single fault in their bit-serial multiplier produces multiple errors after m clock cycles and the number of effective errors resulting from the single fault is either odd or even and only the odd number of errors can be detected. The proposed circuit in Figure 8 overcomes this problem. It compares the predicted parity of the storage registers with the actual ones at every clock cycle. Although it costs extra hardware, the probability of error detection of our bit-serial multiplier is about 100%. This result has been verified using a simulation program for a prototype multiplier with $F(z) = z^4 + z + 1$. Using VHDL, we have injected single faults at different nodes of the bit-serial multiplier for all elements of A and B. The probability reaches unity as m increases.

The proposed error detection schemes are not limited to the multiplier architectures discussed in this article. They can be easily extended and applied to other $GF(2^m)$ multipliers. For example, we have considered the bit-serial multiplier introduced by Peterson [7] and have made it capable of detecting single faults. Furthermore, although our discussions have centered around bit-parallel and bit-serial multipliers over $GF(2^m)$, by combining the error detection schemes for serial and parallel multipliers, one can develop an error detection scheme for hybrid multipliers over composite fields [8].

More research is needed to reduce the overhead cost of the proposed multiplier. For example, hardware implementation of the architecture shown in Figure 6 appears to be expensive. Currently we are trying to develop an architecture that can alleviate this problem.

References

1. G. B. Agnew, T. Beth, R. C. Mullin, and S. A. Vanstone. "Arithmetic Operations in $GF(2^m)$". *Journal of Cryptology*, 6:3–13, 1993.
2. D. Boneh, R. A. DeMillo, and R. J. Lipton. "On the Importance of Eliminating Errors in Cryptographic Computations". *Journal of Cryptology*, 14:101–119, 2001.
3. S. Fenn, M. Gossel, M. Benaissa, and D. Taylor. "On-Line Error Detection for Bit-Serial Multipliers in $GF(2^m)$". *Journal of Electronic Testing: Theory and Applications*, 13:29–40, 1998.
4. A. Halbutogullari and C. K. Koc. "Mastrovito Multiplier for General Irreducible Polynomials". *IEEE Transactions on Computers*, 49(5):503–518, May 2000.
5. M. Joye, A. K. Lenstra, and J. J. Quisquater. "Chinese Remaindering Based Cryptosystems in the Presence of Faults". *Journal of Cryptology*, 12:241–245, 1999.
6. P. K. Lala. *Fault Tolerant and Fault Testable Hardware Design.* Prentice Hall, 1985.
7. E. D. Mastrovito. *VLSI Architectures for Computation in Galois Fields.* PhD thesis, Linkoping Univ., Linkoping Sweden, 1991.
8. C. Paar, P. Fleishmann, and P. Soria-Rodriguez. "Fast Arithmetic for Public-Key Algorithms in Galois Fields with Composite Exponents". *IEEE Transactions on Computers*, 48(10):1025–1034, Oct. 1999.

9. A. Reyhani-Masoleh and M. A. Hasan. "A New Efficient Architecture of Mastrovito Multiplier over $GF(2^m)$". In 20^{th} *Biennial Symposium on Communications*, pages 59–63, Kingston, Ontario, Canada, May 2000.
10. H. Wu and M. A. Hasan. "Efficient Exponentiation of a Primitive Root in $GF(2^m)$". *IEEE Transactions on Computers*, 46(2):162–172, Feb. 1997.
11. T. Zhang and K. K. Parhi. "Systematic Design of Original and Modified Mastrovito Multipliers for General Irreducible Polynomials". *IEEE Transactions on Computers*, 50(7):734–748, July 2001.

Hardware Implementation of Finite Fields of Characteristic Three

D. Page and N.P. Smart

Dept. Computer Science,
University of Bristol,
Merchant Venturers Building,
Woodland Road,
Bristol, BS8 1UB,
United Kingdom.
{page,nigel}@cs.bris.ac.uk

Abstract. In this paper we examine a number of ways of implementing characteristic three arithmetic in hardware. While this type of arithmetic is not traditionally used in cryptographic systems, recent advances in Tate and Weil pairing based cryptosystems show that it is potentially valuable. We examine a hardware oriented representation of the field elements, comparing the resulting algorithms for field addition and multiplication operations, and show that characteristic three arithmetic need not significantly under-perform comparable characteristic two alternatives.

1 Introduction

There has been a recent increase in research activity surrounding cryptosystems based on the Tate and Weil pairings. Identity based encryption schemes [6] and signature algorithms [11,16,17] as well as general signature algorithms [7] have been developed and published, all of which utilise pairing based operations. Additionally, extensions to higher genus curves have been fully explored [8]. Pairing based cryptosystems were traditionally thought to be weak when it was shown [13] that the discrete logarithm problem in supersingular curves was reducible to that in a finite field using the Weil pairing. However, this view changed when Joux [12] presented a simple tripartite Diffie-Hellman protocol based on the Weil pairing on supersingular curves which, in part, rekindled interest in this area.

Although there is little discussion about implementation, it was noted by Galbraith [8] that in terms of bandwidth efficiency, it is more efficient to use elliptic curves in characteristic three for systems based on the Weil or Tate pairing. This notion contradicts conventional advice when implementing elliptic curves, which generally suggests using fields of either large prime characteristic or characteristic two. The use of such fields is generally based on the assumption that arithmetic in characteristic three is much slower than the given alternatives and has resulted in a gap in literature surrounding the topic.

B.S. Kaliski Jr. et al. (Eds.): CHES 2002, LNCS 2523, pp. 529–539, 2003.

Since the efficient hardware implementation of elliptic curves arithmetic in characteristic three is potentially of value to the expanding list of systems which use the Weil or Tate pairing, we will fill this gap in this paper. The purpose of this work is to facilitate the use of characteristic three arithmetic in pairing based cryptosystems, and hence reap the advantages of doing so, without imposing the performance overhead which may traditionally be expected. In Section 2 we discuss the Tate pairing and develop some parameters for the comparison of our techniques. We present a way of representing polynomials and performing arithmetic on them in Sections 3 and 4. Finally, we implement these arithmetic operations in field programmable hardware and present the performance results in Section 5.

2 Supersingular Elliptic Curves and the Tate Pairing

We let $\mathbb{G}$ denote a prime order subgroup of an elliptic curve E over the field $\mathbb{F}_q$, which for the moment we assume is a general finite field of arbitrary characteristic. Let the order of $\mathbb{G}$ be denoted by l and define α to be the smallest integer such that

$$l | q^\alpha - 1$$

In practical implementations we will require α to be small and so will usually take E to be a supersingular curve over $\mathbb{F}_q$. Let $\overline{\mathbb{G}}$ denote the group of points of order l of the elliptic curve E over the field $\mathbb{F}_{q^\alpha}$. While the group $\mathbb{G}$ is cyclic of order l, the group $\overline{\mathbb{G}}$ is a product of two cyclic subgroups of order l.

The bandwidth performance of the schemes based on the Weil pairing usually grow with $\log_2 q$ rather than with $\alpha \cdot \log_2 q$, hence it is better to try to minimise q. This leads us to consider fields of characteristic three, since they aid us in minimising the value of q and hence minimising the bandwidth. However, it is unclear as to whether this comes at the expense of a decrease in performance when compared against fields of characteristic two. In this paper we go some way to address this issue in hardware by performing a comparison of the field primitives. A comparison of the actual protocols we leave to a later publication.

In this paper we shall be interested in protocols which make use of the modified Tate pairing given by the map

$$\hat{t} : \mathbb{G} \times \mathbb{G} \to \mathbb{F}_{q^\alpha}^*,$$

which satisfies the following properties

1. Bilinearity:
   - $\hat{t}(P_1 + P_2, Q) = \hat{t}(P_1, Q) \cdot \hat{t}(P_2, Q)$.
   - $\hat{t}(P, Q_1 + Q_2) = \hat{t}(P, Q_1) \cdot \hat{t}(P, Q_2)$.
2. Non-degeneracy: There exists a $P \in \mathbb{G}$ such that $\hat{t}(P, P) \neq 1$.
3. Computable : One can compute $\hat{t}(P, Q)$ in polynomial time.

If we let ϕ denote a "distortion map", or group endomorphism which maps elements in $E[l]$ into linearly independent elements of $E[l]$, then we can define

the modified Tate pairing from the standard Tate pairing $t(P,Q)$ via the use of distortion maps

$$\hat{t}(P,Q) = t(P,\phi(Q))^{(q^\alpha-1)/l}$$

That the Tate pairing is efficiently computable follows from an unpublished, but much referenced, algorithm of Miller [14].

We wish to compute $\hat{t}(P,Q)$ where $P,Q \in \mathbb{G}$. This requires some operations to be performed in $\mathbb{F}_q$ and some to be performed in $\mathbb{F}_{q^\alpha}$, see [5] and [9]. The exact value of α depends on which supersingular curve is chosen. The optimal choices in each characteristic are given by the following table

Field	Curve	α
$\mathbb{F}_{2^p}$	$y^2+y=x^3+x$	4
$\mathbb{F}_{2^p}$	$y^2+y=x^3+x+1$	4
$\mathbb{F}_{3^p}$	$y^2=x^3-x+1$	6
$\mathbb{F}_{3^p}$	$y^2=x^3-x-1$	6
$\mathbb{F}_p$	$y^2=x^3+1$	2
$\mathbb{F}_p$	$y^2=x^3+x$	2

Notice, the value of α is bounded by four in characteristic two, by six in characteristic three and two for curves defined over large prime fields. The underlying security of the system is based both on the computational Diffie-Hellman problem in the subgroup of order l of $E(\mathbb{F}_q)$ (the so called ECDLP security) and on the computational Diffie-Hellman problem in the finite field $\mathbb{F}^*_{q^\alpha}$ (the so called MOV security). Note that the decision Diffie-Hellman problem on supersingular elliptic curves is easy due to the existence of the Weil and Tate pairings, as was first pointed out by Joux [12].

We therefore need to choose, assuming standard current security recommendations,

- $l \approx 2^{160}$
- $q^\alpha \approx 2^{1024}$

If we wish to deploy a system with security roughly equivalent to 1024-bit RSA or 160-bit ECC, then we are led to consider the following parameters in each characteristic

Field	Curve	ECDLP Security	MOV Security
$\mathbb{F}_{3^{97}}$	$y^2=x^3-x+1$	151	922
$\mathbb{F}_{2^{241}}$	$y^2+y=x^3+x+1$	241	964

We shall consider these parameters when describing our implementation of characteristic three arithmetic below, and the corresponding characteristic two implementation with which we compare it.

3 Polynomial Arithmetic Modulo Three

In order to improve on the expected performance of characteristic three arithmetic, we decided to use a novel representation of polynomials [10]. Each set of

polynomial coefficients is held as two values, which we shall denote w_1 and w_2. A given bit in w_1 is set if the corresponding coefficient of the polynomial is equal to one, while if the given bit in w_2 is set then the coefficient of the polynomial is equal to two. If both bits are clear then the coefficient is zero, while the case of both bits set is considered invalid.

Put more simply, w_1 holds the least significant bits of all coefficients in the polynomial while w_2 holds the most significant bits. This method of holding the coefficients is similar to the practice of bit-slicing which is often performed in software. By bit-slicing the high and low bits of each coefficient into separate values, we offer a much more effective way to perform arithmetic as well as a natural representation which is bit oriented in the same way that characteristic two arithmetic is commonly implemented. As an example of this representation, consider the trinomial $x^6 + x + 2$ which can be described as in Figure 1

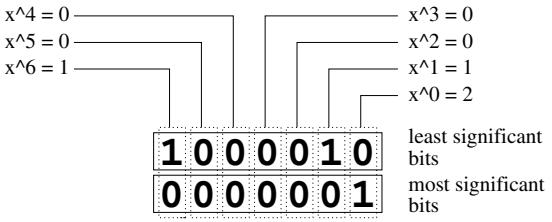


Fig. 1. Bit-sliced Representation

Note that as we are working in hardware and not tied to a word oriented design, where each coefficient occupies a number of bits which roughly equate to the word-size of a processor, this representation is far more compact than other methods. The size of w_1 and w_2 simply grow in length as the degree of the polynomial they represent grows.

3.1 Addition

Addition of polynomials is done on a per-value basis using seven logic operations. Consider the example which adds the polynomial represented by the values (a_1, a_2) to the polynomial (b_1, b_2), producing a result in (r_1, r_2). We can express the addition as a logic diagram, shown in Figure 2, or in the form of a simple pseudo-code program

```
t  = (a1 | b2) ^ (a2 | b1);
r1 = (a2 | b2) ^ t;
r2 = (a1 | b1) ^ t;
```

Note that negation and multiplication by two in this representation are particularly easy operations to implement since

$$2 \cdot (a_1, a_2) = -(a_1, a_2) = (a_2, a_1)$$

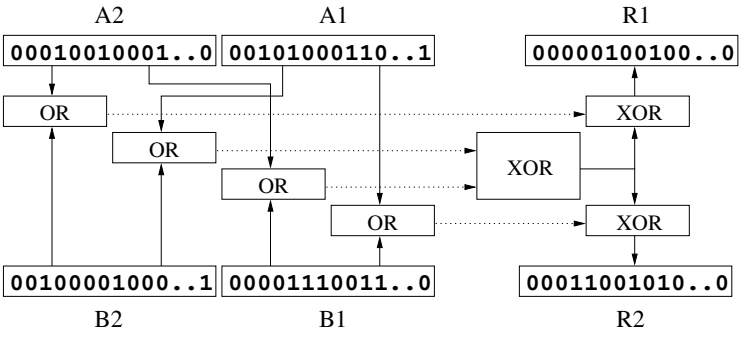


Fig. 2. Addition

3.2 Multiplication

A natural way to multiply elements in this representation is in a bit-serial manner. In this method we take two operands and perform a multiply by repeatedly shifting the multiplier down by one bit position and shifting the multiplicand up by one bit position. The multiplicand is then added or subtracted from the output value, on each iteration, depending on whether the least significant bit of the first or second word of the multiplier is set to one. This is possible due to the identity mentioned above which notes that the double operation is equivalent to the negation operation.

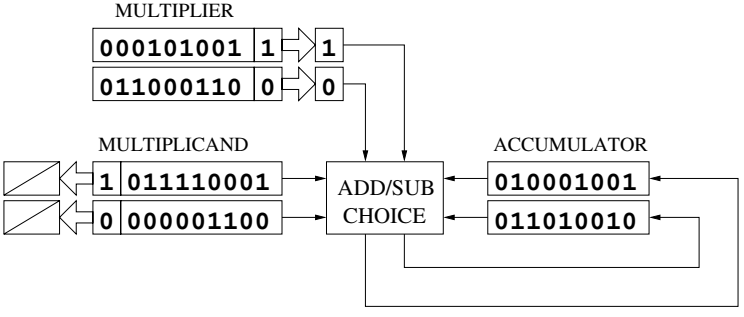


Fig. 3. Multiplication

The advantage of this full bit serial technique is that it requires less intermediate storage and is far more suited to a hardware implementation, using a basic iterated structure and only simple logic elements, i.e. no direct multiplier or adder circuitry is required. However, a major disadvantage of the full bit-serial multiplier is that an analogous cubing operation is only as fast as a general multiply, where as with other representation methods we can perform a more efficient cubing operation than a general multiply.

4 Implementation of Arithmetic in $\mathbb{F}_{3^{6p}}$

When considering pairing based cryptosystems, we are not only required to perform some operations in $\mathbb{F}_{3^p}$ but will also need to compute in the extension $\mathbb{F}_{3^{6p}}$. Since in applications p is a prime greater than five we can use the following representation of the finite field $\mathbb{F}_{3^{6p}}$

$$\mathbb{F}_{3^{6p}} = \mathbb{F}_{3^p}[\theta]/(\theta^6 + \theta + 2)$$

This provides a performance efficient reduction operation for multiplication. For example, consider the multiplication of two polynomials, a and b, in the field $\mathbb{F}_{3^{6p}}$ which we denote

$$a = a_5\theta^5 + a_4\theta^4 + a_3\theta^3 + a_2\theta^2 + a_1\theta + a_0$$

and

$$b = b_5\theta^5 + b_4\theta^4 + b_3\theta^3 + b_2\theta^2 + b_1\theta + b_0$$

Firstly, we multiply the two polynomials using a school-book method to produce a degree ten resulting polynomial r. We can then perform reduction of r, with respect to the irreducible trinomial $\theta^6 + \theta + 2$, using the circuitry as in Figure 4 since the multiplication results in

$$\begin{aligned} a \cdot b = r &= r_{10}\theta^{10} + r_9\theta^9 + \cdots + r_2\theta^2 + r_1\theta + r_0 \\ &= s_5\theta^5 + s_4\theta^4 + s_3\theta^3 + s_2\theta^2 + s_1\theta + s_0 \end{aligned}$$

and we know that $\theta^6 = 2\theta + 1$, so

$$\begin{aligned} s_0 &= r_0 + r_6 \\ s_1 &= r_1 + 2r_6 + r_7 \\ s_2 &= r_2 + 2r_7 + r_8 \\ s_3 &= r_3 + 2r_8 + r_9 \\ s_4 &= r_4 + 2r_9 + r_{10} \\ s_5 &= r_5 + 2r_{10} \end{aligned}$$

Note that we can perform a subtraction operation in place of the double operation because of the characteristic of this field and representation as described in Section 3.

5 Timing of Field Operations

In order to show that arithmetic in $\mathbb{F}_{3^n}$ is suitable, in terms of performance and size, for use in cryptosystems, we implemented a number of algorithms in field-programmable hardware. Our algorithms for addition and multiplication were implemented using version 2.1 of the Celoxica [1] Handel-C [2] hardware

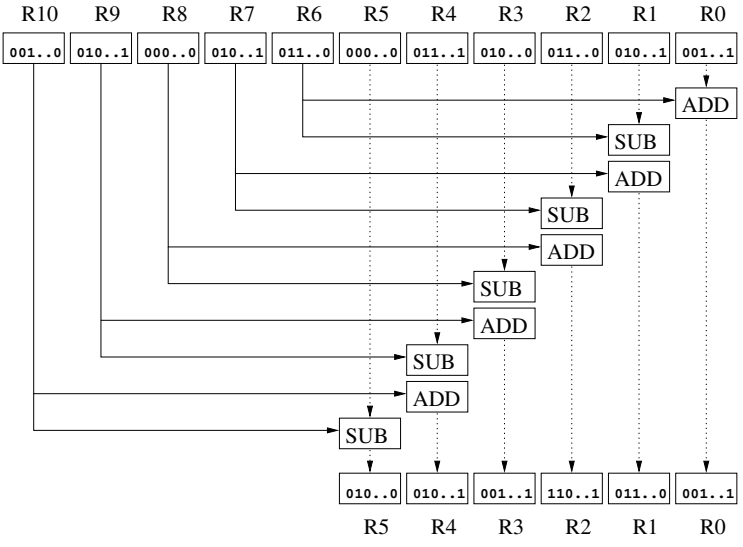


Fig. 4. Reduction Modulo $\theta^6 + \theta + 2$

compilation system and a PCI resident, Xilinx4000XL FPGA based prototyping device [3]. The Handel-C language and compiler tool-chain allowed us to experiment in a familiar high level language, very similar to C, and directly produce hardware implementations from a program in that language. The output of the Handel-C compiler was placed and routed using Xilinx Foundation 3.1i.

All designs communicate input and output data though on-board RAM and use a system clock of 20MHz. We average the results of our timings over 10000 experiments to gain a more representative answer than might otherwise be obtained.

We note that due to our use of a slightly unconventional design process, our results may not be suitable for comparison with, for example, highly optimised VHDL designs. Additionally, we note that we used a somewhat dated version of the Handel-C and Xilinx tool-chains and that more recent versions may offer enhanced optimisation phases which could improve the performance, clock speed and size of our designs. Specifically, we expect to drastically reduce the size of our designs, by using shared arithmetic elements, since the current results are blatantly larger than one might expect. However, we feel that the comparisons offered below are valid in showing both the advantage of our alternative representation and that such arithmetic need not be considered significantly slower than comparable characteristic two alternatives.

In all our experiments, the following notation is used to describe the type of arithmetic being tested

– $\mathbb{F}_{3^{97}} - S$ corresponds to an implementation using the standard software technique of representing each polynomial as an array of 97 integers, where arithmetic is performed using a naive multiplication algorithm.

– $\mathbb{F}_{3^{97}} - B$ refers to our alternative representation using a full bit-serial multiplication method.

The performance for $\mathbb{F}_{2^{241}}$ and $\mathbb{F}_{3^{97}}$ polynomial addition and multiplication, modulo their respective irreducible trinomial, are shown below

Hardware implementation [unoptimised]			
Field	Addition	Multiplication	Slices
$\mathbb{F}_{3^{97}} - S$	$25.29\mu s$	$4393.34\mu s$	2149
$\mathbb{F}_{3^{97}} - B$	$1.20\mu s$	$102.21\mu s$	4136
$\mathbb{F}_{2^{241}}$	$0.80\mu s$	$96.63\mu s$	4920

Notice that addition and multiplication, in our alternative representation of characteristic three, are an order of magnitude faster than the standard $\mathbb{F}_{3^{97}}$ algorithms. Additionally, addition and multiplication are very close to being as fast as arithmetic in $\mathbb{F}_{2^{241}}$.

These addition and multiplication algorithms were implemented with the same basic structure with reduction happening in-place rather than at the end of a multiplication. However, since both the $\mathbb{F}_{3^{97}}$ and $\mathbb{F}_{2^{241}}$ algorithms are bit rather than word oriented, they can easily be accelerated by making size/speed tradeoffs. For example, we can use some extra space to allow reduction to be performed at the end of multiplication and sacrifice further space to add a degree of parallelism to our bit-serial multiplication technique. We also apply additional optimisations which are based on knowledge about how the Handel-C compiler generates hardware for a given input.

By applying these optimisations, we obtain two faster versions of our basic algorithms in both fields

Hardware implementation [optimised]			
Field	Addition	Multiplication	Slices
$\mathbb{F}_{3^{97}}$	$1.15\mu s$	$50.68\mu s$	8733
$\mathbb{F}_{2^{241}}$	$0.70\mu s$	$37.32\mu s$	10139

Since the majority of elliptic curve operations will use these primitives as the basis for more complex operations, the small difference in terms of performance is an important result, it essentially says that characteristic three arithmetic is not necessarily much slower than characteristic two arithmetic.

We can use these optimised addition and multiplication designs as the basis for further algorithms to perform arithmetic in extensions of their respective base fields. We now need to compare arithmetic in $\mathbb{F}_{3^{6 \cdot 97}}$ with arithmetic in $\mathbb{F}_{2^{4 \cdot 241}}$ due to the different values of α in Section 2

Hardware implementation [optimised]			
Field	Addition	Multiplication	Slices
$\mathbb{F}_{3^{6 \cdot 97}}$	$5.90\mu s$	$1843.71\mu s$	10854
$\mathbb{F}_{2^{4 \cdot 241}}$	$3.10\mu s$	$609.04\mu s$	12286

These results show that addition in the two extension fields is roughly equivalent in terms of how long it takes, while using multiplication in $\mathbb{F}_{3^{6\cdot 97}}$ is three times as costly as in $\mathbb{F}_{2^{4\cdot 241}}$. The space required for both implementations is about the same.

Notice that the above implementation used naive arithmetic for performing the extension field multiplication. This was chosen so as to minimise the area of the final hardware solution. Hence, we see that in both cases that if M_b denotes the time needed to perform a base field multiplication and M_e denotes the time needed to perform an extension field multiplication, that

$$M_e \approx n^2 M_b$$

where $n = 6$ in characteristic three and $n = 4$ in characteristic two.

An interesting extension to these results would be to consider the use of Karatsuba multiplication. Although this would lead to a significant increase in area, due to the need to store intermediate results, it could further improve on the arithmetic performance in both fields.

First we deal with the case of even characteristic, where we need to multiply two polynomials of degree three. Using Karatsuba multiplication we can reduce this to three multiplications of polynomials of degree one, plus a little book keeping which we shall ignore. We then multiply the polynomials of degree one, again using Karatsuba, using three base field multiplications. Hence, in characteristic two one expects to obtain

$$M_e \approx 9M_b.$$

In characteristic three we need to multiply two polynomials of degree five over the base field. Using a trivial extension of Karatsuba, which can be found for example in [4] in a similar context, we first apply standard Karatsuba to reduce the problem to the multiplication of three polynomials of degree two. These three products are then computed via performing six base field multiplications each. Hence, in characteristic three one expects to obtain

$$M_e \approx 18M_b.$$

We would therefore expect that a fully optimised version of extension field arithmetic for both characteristics would result in a multiplication algorithm for characteristic three extension fields which is four times slower than the corresponding implementation of characteristic two extension fields. This may not be such a problem in practice as much of the protocols based on the Tate pairing make use of only base field arithmetic, and only the computation of the pairing requires extension field arithmetic. When implementing pairing computations one also attempts to reduce the number of full extension field multiplications that one needs to perform, see [5] and [9] for details.

Finally, to offer further comparison between our techniques, we also implemented them in a software environment. The timings were taken using the same 150MHz Intel PentiumPro equipped FPGA host PC used in the hardware experiments and were compiled using GCC 2.95.1 with all optimisations turned on.

The timings for addition and multiplication in both the base field and extension field are shown below

Software implementation [optimised]		
Field	Addition	Multiplication
$\mathbb{F}_{3^{97}} - S$	$11.89\mu s$	$1013.61\mu s$
$\mathbb{F}_{3^{97}} - B$	$3.98\mu s$	$153.85\mu s$
$\mathbb{F}_{2^{241}}$	$3.31\mu s$	$178.60\mu s$

Software implementation [optimised]		
Field	Addition	Multiplication
$\mathbb{F}_{3^{6 \cdot 97}}$	$8.91\mu s$	$5138.75\mu s$
$\mathbb{F}_{2^{4 \cdot 241}}$	$5.12\mu s$	$3156.86\mu s$

By comparing the results for software and hardware implementation, we can see that in both cases $\mathbb{F}_{3^{97}} - B$ based arithmetic is quicker than a corresponding naive representation. Furthermore, the improvement in the hardware implementation of $\mathbb{F}_{3^{97}} - B$ over $\mathbb{F}_{3^{97}} - S$ is greater than that in software indicating that it is indeed more naturally defined in this medium. Finally, even though our software test environment is far from state of the art, in both cases our hardware implementations significantly out-perform their software equivalents. This is clearly the expected outcome but it is reassuring that even by using an out of date hardware design tool-chain, we were able to produce effective designs using the Handel-C system.

6 Conclusion

We have shown how the use of a novel representation can result in an implementation of characteristic three arithmetic suitable for use in hardware cryptosystems based on the Tate pairing. The use of characteristic three with the Tate pairing is preferred due to the improved bandwidth considerations implied by the security parameters.

Our implementation techniques offer a considerable improvement over the standard techniques based on using a word oriented approach to holding polynomial coefficients. We have also demonstrated that it is possible to implement characteristic three arithmetic which is comparable in performance to a space-equivalent characteristic two alternative. This is a valuable result which allows system designers to benefit from bandwidth reduction without degraded performance.

References

1. Celoxica Technology Overview `http://www.celoxica.com`
2. Celoxica Handel-C Language Overview `http://www.celoxica.com/products/technical_papers/datasheets/DATHNC001_2.pdf`

3. Celoxica Reconfigurable Hardware Development Platform: RC1000 `http://www.celoxica.com/products/technical_papers/datasheets/DATRHD001_2.pdf`
4. D. Bailey and C. Paar. Efficient arithmetic in finite field extensions with application in elliptic curve cryptography. *J. Cryptology*, **14**, 153–176, 2001.
5. P.S.L.M. Barreto, H.Y. Kim and M. Scott. Efficient algorithms for pairing-based cryptosystems. To appear *Advances in Cryptology - CRYPTO 2002*, Springer LNCS 2442, 2002.
6. D. Boneh and M. Franklin. Identity-based encryption from the Weil pairing. In *Advances in Cryptology - CRYPTO 2001*, Springer-Verlag LNCS 2139, 213–229, 2001.
7. D. Boneh, B Lynn and H. Shacham. Short signatures from the Weil pairing. In *Advances in Cryptology - ASIACRYPT 2001*, Springer-Verlag LNCS 2248, 514–532, 2001.
8. S.D. Galbraith. Supersingular curves in cryptography. In *Advances in Cryptology - ASIACRYPT 2001*, Springer-Verlag LNCS 2248, 495–513, 2001.
9. S.D. Galbraith, K. Harrison and D. Soldera. Implementing the Tate pairing. *Algorithmic Number Theory Symposium, ANTS-V*, Springer-Verlag LNCS 2369, 324–337, 2002.
10. K. Harrison, D. Page and N.P. Smart. Software implementation of finite fields in characteristic three. Preprint, 2002.
11. F. Hess. Efficient Identity based Signature Schemes based on Pairings To appear *Selected Areas in Cryptography 2002*.
12. A. Joux. A one round protocol for tripartite Diffie-Hellman. In *Algorithmic Number Theory Symposium, ANTS-IV*, Springer-Verlag LNCS 1838, 385–394, 2000.
13. A.J. Menezes, T. Okamoto and S. Vanstone. Reducing elliptic curve logarithms to logarithms in a finite field. *IEEE Trans. Info. Th.*, **39**, 1639–1646, 1993.
14. V. Miller. Short programs for functions on curves. Unpublished manuscript, 1986.
15. P.L. Montgomery. Modular multiplication without trial division. *Math. Comp.*, **44**, 519–521, 1985.
16. K. Paterson. ID-based Signatures from Pairings on Elliptic Curves. Preprint 2002.
17. R. Sakai, K. Ohgishi and M. Kasahara. Cryptosystems based on pairing. In *SCIS 2000*, 2000.

Preventing Differential Analysis in GLV Elliptic Curve Scalar Multiplication

Mathieu Ciet*, Jean-Jacques Quisquater, and Francesco Sica*

UCL Crypto Group
Place du Levant, 3. B-1348 Louvain-la-Neuve. Belgium
{ciet, jjq, sica}@dice.ucl.ac.be – http://www.dice.ucl.ac.be/crypto

Abstract. In [2], Gallant, Lambert and Vanstone proposed a very efficient algorithm to compute $Q = kP$ on elliptic curves having non-trivial efficiently computable endomorphisms. Cryptographic protocols are sensitive to implementations, indeed as shown in [6,7] information about the secret can be revealed analysing external leakage of the support, typically a smart card. Several software countermeasures have been proposed to protect the secret. However, speed computation is needed for practical use. In this paper, we propose a method to protect scalar multiplication on elliptic curves against Differential Analysis, that benefits from the speed of the Gallant, Lambert and Vanstone method. It can be viewed as a two-dimensional analogue of Coron's method [1] of randomising the exponent k. We propose two variants of this method (one linear and one affine), the second one slightly more effective, whereas the first one offers "two in one", combining point-blinding and exponent randomisation, which have hitherto been dealt separately. For instance, for at most a mere 37.5% (resp. 25%) computation speed loss on elliptic curves over fields with 160 (resp. 240) bits the computation of kP can take on 2^{40} different consumption patterns.

Keywords. *Public key cryptography, differential power analysis, elliptic curve cryptosystem, fast computation.*

1 Introduction

Since the paper of Kocher [6] on the timing attack in 1996 and the Kocher, Jaffe and Jun paper Differential Power Analysis (DPA) [7] in 1999, it is well known that non careful implementations can leak and that it is possible to recover the secret key using the information on which access is possible. For the particular case of elliptic curves, different methods has been proposed to prevent these attacks as [1,4,5,8,9]. Independently Gallant Lambert and Vanstone, proposed in [2] a new principle of computation using efficient endomorphism on certain elliptic curves. In the past, Solinas also proposed to use such endomorphisms but his method could only be applied for elliptic curve defined over binary fields, the

* Supported by the European Commission through the IST Programme under Contract IST-1999-12324, http://cryptonessie.org/.

B.S. Kaliski Jr. et al. (Eds.): CHES 2002, LNCS 2523, pp. 540–550, 2003.

endomorphism considered being the Frobenius. However, the motivation of the Gallant-Lambert-Vanstone (GLV for short) and Solinas methods are not exactly the same, in the first case "reduction" of secret multiplier is obtained and in the second new decomposition is the basis of speedup computation.

In this paper we propose a specific method that benefits from fast endomorphisms, allowing its use in the case of smart card or parallel implementation. The speedup obtained by using a particular endomorphism is not helpful in solving the discrete logarithm problem. Hence the curves on which our methods apply are not cryptographically weaker than a generic curve. The methods presented here do not immunise elliptic curve cryptosystems against simple power analysis. It is possible to combine several methods to prevent simple and differential power analysis, such as randomisation of the private multiplier or blinding the point (see [1]) or elliptic curve isomorphisms, field isomorphisms or extension fields (see [5]). Differential analysis is not only a theoretical attack but can be applied in practice to recover the secret key, analysing leakage.

This paper is organised as follows. In a first part, a countermeasure against differential analysis using randomisation is reviewed. Afterwards, the GLV method based on efficient endomorphisms to speed up computation is explained. Then our method to prevent elliptic curve cryptosystems from differential analysis is proposed. It is based on random sublattices, and after a theoretical discussion distinguishing two cases a practical application is given. At the same time considerations of extra computation time is taken into account and analysis of real prevention is discussed, by counting the number of different possible representations of the same multiplier k. Before concluding, an affine generalisation is introduced which from the implementation perspective is quite interesting because of the small modifications needed compared to the original GLV method. For all methods presented here, no extra routine are necessary to be implemented.

2 Differential Analysis and Previous Work on Randomised Endomorphisms

In [7], Kocher, Jaffe and Jun introduced the differential power analysis (DPA). In practice, a cryptosystem is not a black box, it can reveal a part of information about the secret. DPA consists in using side-channel information about the state of the machine which computes, typically a smart card. Differential power analysis uses power consumption and analyses such data statistically. We refer the reader to [1,5] for a description in case of elliptic curve cryptography.

Let us just mention that we consider two generic types of countermeasures to protect the computation of a multiple kP of P lying on an elliptic curve.

- blinding the base point: replace P by a random $\widetilde{P}$ such that $kP = k\widetilde{P}$,
- randomised secret exponent: replace k by a random $\tilde{k}$ such that $kP = \tilde{k}P$.

In all currently proposed countermeasures, only one consists of a modified efficient scalar multiplication technique based on randomised endomorphisms.

Joye and Tymen first presented in [5, Section 5] a randomised endomorphism to prevent DPA for elliptic curve implementation, based on previous work of Solinas [12]. Let us consider a Koblitz curve E defined over $\mathbb{F}_{2^n}$ with $y^2 + xy = x^3 + ax^2 + 1$ as equation, and $a \in \{0, 1\}$. Letting $\tau : (x, y) \mapsto (x^2, y^2)$ the Frobenius endomorphism, which satisfies the equation $u^2 - (-1)^{1-a}u + 2 = 0$. The ring $\mathbb{Z}[\tau]$ is an euclidian domain with the norm $\mathbf{N}(.)$ defined as $\mathbf{N}(r + \tau s) = r^2 + (-1)^{1-a}rs + 2s^2$. The key point of their countermeasure with randomised endomorphisms [5, Section 5] is to use the following property. Let $\rho \in \mathbb{Z}[\tau]$. If $k_1 \equiv k_2 \pmod{\rho(\tau^n - 1)}$, then $\forall P \in E$, $k_1 P = k_2 P$. Thus, they choose randomly $\rho \in \mathbb{Z}[\tau]$ such that $\mathbf{N}(\rho) < 2^{40}$, then compute $\kappa' = k \pmod{\rho(\tau^n - 1)}$, they decompose $\kappa' = \sum_i k'_i \tau^i$ using NAF algorithm [3], and compute $Q = kP$ as $\sum_i k'_i \tau^i(P)$.

This method only applies in characteristic two and for ABC curves. We propose hereafter a new method valid in any characteristic. After recalling the GLV method which constitutes the base of our algorithms in the next section, we develop a first variant of our DPA-resistant algorithm in two versions (cases 1 and 2). These variants combine the two countermeasure types expressed above into a unique algorithm. Thereafter, Section 7 will explore another variant of a DPA-resistant algorithm based on the GLV method, which achieves better performance over similar proved security. Here we only randomise the exponent k. On the other hand the implementation code contains only slight modifications of its deterministic version.

3 The Gallant-Lambert-Vanstone Computation Method

In this part, we briefly summarize the GLV computation method [2]. Let E be an elliptic curve defined over a finite field $\mathbb{F}_q$ and P be a point of this curve with order a large prime n (say $\#E(\mathbb{F}_q)/n \leq 4$). Let us consider Φ a non-trivial endomorphism of E defined over $\mathbb{F}_q$ and $X^2 + rX + s$ its characteristic polynomial.

By the Hasse bound, since n is large, $\Phi(P) = \lambda P$ for some $\lambda \in [1, n-1]$. Indeed, there is only one copy of $\mathbb{Z}/n$ inside $E(\mathbb{F}_q)$ and $\Phi(P)$ has also order dividing n. We can easily exclude the case where $\lambda = 0$ which is exceptional (for instance in the examples we have $n \nmid s$, by the Hasse bound). In all cases, λ is obtained as a root of $X^2 + rX + s$ modulo n. The GLV algorithm decomposes k as $k \equiv k_1 + k_2\lambda \pmod{n}$ where $k_i = O(\sqrt{n})$ for $i = 1, 2$. We quickly describe this construction. Let $f : \mathbb{Z} \times \mathbb{Z} \to \mathbb{Z}/n$ denote the homomorphism defined by $(i, j) \mapsto (i + j\lambda) \pmod{n}$. The goal is to find a small vector u such that $f(u) = k$. As $f((k, 0)) = k$, the problem is reduced to finding two linearly independent vectors v_1 and v_2 of small length (say $O(\sqrt{n})$) such that $f(v_1) = f(v_2) = 0$ and to decompose $(k, 0)$ in this basis with coefficients in $\mathbb{Q}$ and then rounding off $(k, 0)$ to the nearest vector v which is a linear combination of v_1 and v_2 with coefficient in $\mathbb{Z}$. Finally, u is chosen as $u = (k, 0) - v$. The problem of finding v_1 and v_2 is solved in [2] using the extended Euclidean algorithm (see also [10,11] for alternative methods).

Finally kP is computed more efficiently than previous existing methods by calculating first $\Phi(P)$, decomposing $k \equiv k_1 + k_2\lambda \pmod{n}$ with $\max(|k_1|, |k_2|) = O(\sqrt{n})$, and computing $k_1 P + k_2 \Phi(P)$ using elliptic Straus-Shamir multiplication described in [3] or in [13].

4 Elliptic Scalar Multiplication Using Sub-lattices and the GLV Method

4.1 Retrieving New v_1 and v_2

Let Φ be a non trivial endomorphism defined over $\mathbb{F}_q$ such that $\Phi^2 + r\Phi + s = 0$, as in the GLV method. We want to describe a way to mask scalar multiplication using a GLV decomposition so that it becomes immune to DPA. The idea is to use a random sub-lattice $\mathcal{L} = A(\mathbb{Z} \times \mathbb{Z})$, or rather a random matrix A with coefficients in some range (here $[0, R]$ for an given integer R, to be chosen later). We will denote $\Delta = \det A$. The linear map A will often be viewed as a matrix $A = \begin{pmatrix} \alpha & \beta \\ \gamma & \delta \end{pmatrix}$ with respect to the canonical basis. Later, we will also translate $\mathbb{Z} \times \mathbb{Z} \cong \mathbb{Z}[\Phi]$ via the isomorphism which sends the canonical basis to $\{1, \Phi\}$. The matrix A will be therefore also viewed as a linear transformation of $\mathbb{Z}[\Phi]$, sending $\{1, \Phi\}$ respectively to $\{\Phi_0, \Phi_1\}$.

Consider the GLV homomorphism:

$$\begin{aligned} f \colon \mathbb{Z} \times \mathbb{Z} &\to \mathbb{Z}/n \\ (i, j) &\mapsto i + \lambda j \pmod{n} \ . \end{aligned}$$

For a sublattice $\mathcal{L} = A(\mathbb{Z} \times \mathbb{Z}) \subset \mathbb{Z} \times \mathbb{Z}$, we denote $v_1^{(\mathcal{L})}$ and $v_2^{(\mathcal{L})}$ two linearly independent vectors in the kernel of $(\mathbb{Z} \times \mathbb{Z})|_{\mathcal{L}} \to \mathbb{Z}/n$ which is the restriction of f to $\mathcal{L}$. We also require that $v_1^{(\mathcal{L})}$ and $v_2^{(\mathcal{L})}$ have rectangle norm $O(\sqrt{n})$.[1]

Such vectors can be computed from the traditional GLV vectors v_1 and v_2. Indeed, note that the index of $\mathcal{L}$ inside $\mathbb{Z} \times \mathbb{Z}$ is

$$(\mathbb{Z} \times \mathbb{Z} : \mathcal{L}) = |\Delta| \ ,$$

and this is also the order of $(\mathbb{Z} \times \mathbb{Z})/\mathcal{L}$. Therefore we may set[2]

$$v_i^{(\mathcal{L})} = \Delta v_i \in (\ker f) \cap \mathcal{L} \ . \tag{1}$$

[1] The rectangle norm of (x, y) is by definition $\max(|x|, |y|)$. We denote it by $|(x, y)|$.

[2] The vectors $v_i^{(\mathcal{L})}$ generate a sublattice of index $|\Delta|$ inside $(\ker f) \cap \mathcal{L}$. In practice (see Appendix A) we use different vectors v_i' than those defined by (3). However we can only prove a bound on the length of the GLV decomposition using these vectors, so in the future performance loss could still be lowered theoretically.

4.2 Decomposition of k

We want to write $kP = k_1'\Phi_0(P) + k_2'\Phi_1(P)$ for some $k_1', k_2' = O(\sqrt{n})$. We use the same strategy as in the original GLV method. Indeed as before there exist $\lambda_0, \lambda_1 \in [0, n-1]$ such that $\Phi_0(P) = \lambda_0 P$ and $\Phi_1(P) = \lambda_1 P$. We are aiming therefore at a decomposition of the form

$$k \equiv k_1'\lambda_0 + k_2'\lambda_1 \pmod{n}, \quad \text{with } k_1', k_2' = O(\sqrt{n}) \ . \tag{2}$$

Note that $\lambda_0 \equiv \alpha + \beta\lambda \pmod{n}$ and $\lambda_1 \equiv \gamma + \delta\lambda \pmod{n}$. Let us consider the modified GLV map

$$\begin{aligned} f' \colon \mathbb{Z} \times \mathbb{Z} &\to \mathbb{Z}/n \\ \begin{pmatrix} i \\ j \end{pmatrix} &\mapsto i\lambda_0 + j\lambda_1 \pmod{n} \ . \end{aligned}$$

Observe that f' is a linear map which can be written in matrix form as

$$\begin{pmatrix} i \\ j \end{pmatrix} \mapsto (i,j)\begin{pmatrix} \lambda_0 \\ \lambda_1 \end{pmatrix} = (i,j)A\begin{pmatrix} 1 \\ \lambda \end{pmatrix} \pmod{n} \ .$$

Hence, denoting by A^{T} the transpose of A, we can write $f' = f \circ \ell$, with

$$\begin{aligned} \ell \colon \mathbb{Z} \times \mathbb{Z} &\to \mathbb{Z} \times \mathbb{Z} \\ \begin{pmatrix} i \\ j \end{pmatrix} &\mapsto \begin{pmatrix} i' \\ j' \end{pmatrix} = A^{\mathrm{T}}\begin{pmatrix} i \\ j \end{pmatrix} \ . \end{aligned}$$

Let $\mathcal{L}' = \ell(\mathbb{Z} \times \mathbb{Z})$, $\mathcal{K}' = \ker f'$ and $\mathcal{K} = \ker f$. Note that Δ is also the index of $\mathcal{L}'$ inside $\mathbb{Z} \times \mathbb{Z}$. Therefore two short vectors v_1' and v_2' of $\mathcal{K}'$ are given by

$$\begin{aligned} v_i' &= \pm\big(A^{\mathrm{T}}\big)^{-1} v_i^{(\mathcal{L}')} \quad i = 1, 2 \\ &= \hat{A}^{\mathrm{T}} v_i \quad \text{by (1)} \ , \end{aligned} \tag{3}$$

where $\hat{A} = \begin{pmatrix} \delta & -\gamma \\ -\beta & \alpha \end{pmatrix}$ is the adjoint matrix of A.

Since $|v_i| \leq \sqrt{1 + |r| + s}\sqrt{n}$ by [11, Theorem 1], if $\alpha, \beta, \gamma, \delta \in [0, R]$ we get

$$|v_i'| \leq 2R\,|v_i| \leq 2R\sqrt{1 + |r| + s}\sqrt{n} \ . \tag{4}$$

To find k_1' and k_2' satisfying (2), we proceed as in the original GLV method with a small difference. We need first to find a vector w such that $f'(w) = k \pmod{n}$. In the original method, we could take $w = (k, 0)$. Here we define $\lambda_0^{-1} \in [0, n-1]$ such that $\lambda_0\lambda_0^{-1} \equiv 1 \pmod{n}$ (we have $\lambda_0 \not\equiv 0 \pmod{n}$, see the discussion following (7)). Finding λ_0^{-1} amounts to an application of the extended Euclidean algorithm to n, λ_0. Then we have $f'(k\lambda_0^{-1}, 0) = k \pmod{n}$. From this point everything else is identical to the original method by replacing $(k, 0)$ with $(k\lambda_0^{-1}, 0)$. At the end we get that

$$\max(|k_1'|, |k_2'|) \leq 2R \max(|k_1|, |k_2|) \leq 2R\sqrt{1 + |r| + s}\sqrt{n} \ . \tag{5}$$

4.3 The Case of Small Determinant

The bound of (5) can be considerably sharpened when Δ is small. We shall not dwell on all cases, but shall consider only the case $\Delta = 1$. Let us draw some generic observations from the last section. Indeed from the relation $f' = f \circ \ell$ it is immediate to see that $\ell(\mathcal{K}') \subset \mathcal{K}$. On the other hand clearly

$$\hat{A}^{\mathrm{T}}(\mathcal{K}) \subset \mathcal{K}' \ ,$$

which implies

$$\Delta\mathcal{K} \subset A^{\mathrm{T}}\mathcal{K}' \subset \mathcal{K} \ .$$

Hence if $\Delta = 1$ one gets equalities in all the above inclusions. In particular, $\mathcal{K}' = \mathcal{K}$ and this implies that one may take $(v_1', v_2') = (v_1, v_2)$ in the GLV sublattice decomposition, so that in (5) the parameter R is no longer there.

In the appendix, we give full details of the modified GLV algorithms. In the following, we measure the protection offered by this method against DPA and its performance slow-down.

5 On the Protection Offered by the Randomised GLV Method against DPA

We have seen in the previous section how the choice of a random A with coefficients in $[0, R]$ affects the length of the vector $u = (k_1, k_2)$. A more detailed performance analysis of the analogue of the GLV algorithm in this case will be carried out in the next section. Here we will evaluate the number of different decompositions of kP offered by different choices of A with coefficients less than R.

In order to evaluate this quantity, we note that the number of matrices A such that $\Delta \neq 0$ is around R^4 and the number of those with $\Delta = \pm 1$ is around R^2.

The starting point of the analysis is the remark that DPA as described in Section 2 cannot work if the GLV decomposition is randomised as before, because the power consumption pattern is also randomised for a single exponent k. This will be quantified by the analysis of (6) and (7) below.

We are concerned with two points, both of which add entropy and enhance the security of our system. Let A be the previous matrix and $B = \begin{pmatrix} \epsilon & \zeta \\ \eta & \theta \end{pmatrix} \neq A$ another such matrix, with coefficients in $[0, R]$. Call

$$\begin{pmatrix} \Psi_0 \\ \Psi_1 \end{pmatrix} = B \begin{pmatrix} 1 \\ \Phi \end{pmatrix} \ .$$

We want to avoid a situation where

$$kP = k_1'\Phi_0(P) + k_2'\Phi_1(P) = \tilde{k}_1'\Psi_0(P) + \tilde{k}_2'\Psi_1(P),$$

$$k_i' = \tilde{k}_i' \quad \text{and} \tag{6}$$

$$\Phi_i(P) = \Psi_i(P) \quad i = 1, 2 \ . \tag{7}$$

Let us analyse first (7). This condition is equivalent to

$$\alpha + \beta\lambda + c_1 n = \epsilon + \zeta\lambda \quad \text{and} \quad \gamma + \delta\lambda + c_2 n = \eta + \theta\lambda \ , \tag{8}$$

with $c_1, c_2 \in \mathbb{Z}$. We want to show that one must have $c_1 = c_2 = 0$ as soon as for instance

$$R < \sqrt{n}/\sqrt{1 + |r| + s} \ . \tag{9}$$

We analyse the left-hand inequality, the other one being analogous. One has

$$(\alpha - \epsilon) + (\beta - \zeta)\lambda \equiv 0 \pmod{n} \ ,$$

so that $(\alpha - \epsilon, \beta - \zeta) \in \ker f$. But for $(x, y) \in \ker f - \{(0, 0)\}$ it is known by the proof of [11, Theorem 1] that

$$\max(|x|, |y|) > \sqrt{n}/\sqrt{1 + |r| + s} > R \ ,$$

by (9), hence this forces $(x, y) = (0, 0)$, thus proving our claim. Therefore, given A, there is no other B satisfying (6) and (7).

Case 1: $\Delta \neq 0$. Since the total number of matrices A is of order R^4, the probability that two decompositions of kP match is less than $1/R^4$.

Case 2: $\Delta = \pm 1$. In this case the total number of A's is around R^2, so the probability that two decompositions of kP match is less than $1/R^2$.

6 On the Additional Computation Cost of the Randomised GLV Method

We now measure the extra computation cost of this method with respect to the plain GLV method. Three parts of extra computation can be distinguished:

1. computation of $\Phi_0(P)$, $\Phi_1(P)$,
2. computation of v_1' and v_2',
3. computation of $kP = k_1'\lambda_0 P + k_2'\lambda_1 P$ with respect to the original $kP = k_1 P + k_2 \lambda P$.

We analyse these points.

The decomposition of k into $k_1'\lambda_0 + k_2'\lambda_1$ through the GLV method can be applied as described at the end of Section 4.2. The increase in computation is to invert λ_0 modulo n, and it can be neglected in comparison with the global computation of the elliptic curve scalar multiplication.

The computation of v_1' and v_2' by (3) is also fast and can be neglected. This step does not apply in Case 2 ($\Delta = \pm 1$).

Secondly, the elliptic Straus-Shamir method of Solinas [13] can be used to compute $\Phi_0(P)$, $\Phi_1(P)$ and $k_1'\Phi_0(P) + k_2'\Phi_1(P)$. It is known that its average

computation cost $C(l)$ for $l = \max(\log_2 k'_1, \log_2 k'_2)$ is l doublings and $l/2$ curve additions to obtain $k'_1\Phi_0(P) + k'_2\Phi_1(P)$. Therefore, using (5), the cost of computing $k'_1\Phi_0(P) + k'_2\Phi_1(P)$ is augmented by

$$\begin{cases} 300\,\dfrac{C(\log_2 R)}{C\,(\log_2 \sqrt{n})}\% \approx \dfrac{300\log_2 R}{\log_2 \sqrt{n}}\% & \text{in Case 1,} \\ 200\,\dfrac{C(\log_2 R)}{C\,(\log_2 \sqrt{n})}\% \approx \dfrac{200\log_2 R}{\log_2 \sqrt{n}}\% & \text{in Case 2.} \end{cases} \tag{10}$$

since $\log_2 k_i \approx (\log n)/2$. By letting

$$R < n^{1/16} \qquad \text{which implies (9)}\ , \tag{11}$$

one gets for instance that the augmentation cost is less than 37.5%. This is the case, for instance, if n has 160 bits and $R = 2^{10}$.

In Case 2, one has to double the size of R to achieve the same security, but performance is better than Case 1 for the same R, hence the 50% increase in computation cost.

7 An Affine Generalisation

One can in fact easily generalise our ideas to an affine setting, where instead of masking the GLV decomposition with a linear map $x \mapsto Ax$, one uses an affine map $x \mapsto Ax+\rho$. We present a special case of this affine method, in which $A = \mathrm{Id}$ and only ρ is randomised. It can be implemented with very small modifications of the code. The idea is to randomise the part "**Finding** v" of [2].

The vector $(k, 0)$ breaks down as $(k, 0) = \beta_1 v_1 + \beta_2 v_2$, with $\beta_1, \beta_2 \in \mathbb{Q}$ using [2, Lemma 1]. Let $R > 0$, $\rho_1, \rho_2 \in [0, R]$ two random integers and $b'_i = \lceil \beta_i \rfloor - \rho_i$ for $i \in \{1, 2\}$, where $\lceil \cdot \rfloor$ means the nearest integer. The vector $u' = (k'_1, k'_2)$, constructed as $u' = (k, 0) - v'$ where $v' = b'_1 v_1 + b'_2 v_2$, has norm at most $c \max(|v_1|, |v_2|)$ with $c \leq 2R + 1$. Furthermore, by construction we have $kP = k'_1 P + k'_2 \Phi(P)$.

With our method, we have a very easy control between the additional running time and the expected security, indeed for a typical field elliptic curve with a 160-bit number n, $|v_1|$ and $|v_2|$ are 80-bit numbers and we have a 25% increase in computation compared to the plain GLV method when $R = 2^{20}$.

On the other hand, security here can easily be justified, since the difference between the modified vector u' and the original one u is $\rho_1 v_1 + \rho_2 v_2$, which is a different vector for each choice of ρ_1, ρ_2. Hence the probability that (6) holds is R^{-2}.

8 Conclusion

In this paper, we propose to modify the Gallant-Lambert-Vanstone computation method to prevent differential analysis, in a way which benefits from the computational speedup of the method. The class of curves where this computation

is possible is the same as for the GLV method. Our method can be viewed as a two-dimensional generalisation of Coron's [1].

We have distinguished two cases of possible linear randomisations. In the second one the class of random matrices is more restricted and security is slightly reduced, however, this class offers the advantage of not having to redo the precomputation stage to find the vectors v_i' which we take to be the original v_i of the plain GLV method.

Thus, for instance, a single computation of kP can assume a randomly chosen consumption pattern, the probability that two of these matching being less than 2^{-40} independently of the chosen elliptic curve. With this security threshold, on a 160-bit elliptic curve, performance is only at most 37.5% slower than using the original GLV method (50% when using unimodular matrices).

In these two cases, we mask both the exponent k and the points P, $\Phi(P)$.

These considerations can be carried through to affine randomisations, where we presented the simplest example with collision probability 2^{-40} and 25% additional running time on a 160-bit elliptic curve. This variant can be considered as a generalisation of Coron's first countermeasure of randomising the private exponent [1, Section 5].

Acknowledgements. The authors would like to thank Marc Joye for useful comments on the preliminary versions of this paper, and François Koeune for answering some questions on side-channel cryptanalysis. We are also grateful to the anonymous referees for their valuable remarks.

References

1. J-S. Coron. Resistance against Differential Power Analysis for Elliptic Curve Cryptosystems. In Ç. K. Koç and C.Paar, editors, *Advances in Cryptology - Proceedings of CHES 1999*, volume 1717 of *Lecture Note in Computer Science*, pages 292–302. Springer, 1999.
2. R. P. Gallant, J. L. Lambert, and S. A. Vanstone. Faster Point Multiplication on Elliptic Curves with Efficient Endomorphisms. In J. Kilian, editor, *Advances in Cryptology - Proceedings of CRYPTO 2001*, volume 2139 of *Lecture Notes in Computer Science*, pages 190–200. Springer, 2001.
3. D. M. Gordon. A Survey of Fast Exponentiation Methods. *Journal of Algorithms*, 27(1):129–146, 1998.
4. M. Joye and J-J. Quisquater. Hessian Elliptic Curves and Side-Channel Attacks. In Ç. K. Koç, D. Naccache, and C. Paar, editors, *Advances in Cryptology - Proceedings CHES 2001*, volume 2162 of *Lecture Notes in Computer Science*, pages 402–410. Springer, 2001.
5. M. Joye and C. Tymen. Protections against Differential Analysis for Elliptic Curve Cryptography -An Algebraic Approach-. In Ç. K. Koç, D. Naccache, and C. Paar, editors, *Advances in Cryptology - Proceedings CHES 2001*, volume 2162 of *Lecture Notes in Computer Science*, pages 377–390. Springer, 2001.
6. P. C. Kocher. Timing Attacks on Implementations of Diffie-Hellman, RSA, DSS, and Other Systems. In N. Koblitz, editor, *Advances in Cryptology - Proceedings of CRYPTO 1996*, volume 1109 of *Lecture Notes in Computer Science*, pages 104–113. Springer, 1996.

7. P. C. Kocher, J. Jaffe, and B. Jun. Differential Power Analysis. In M. Wiener, editor, *Advances in Cryptology - Proceedings of CRYPTO 1999*, volume 1666 of *Lecture Notes in Computer Science*, pages 388–397. Springer, 1999.
8. P.-Y. Liardet and N.P. Smart. Preventing SPA/DPA in ECC Systems using the Jacobi Form. In Ç. K. Koç, D. Naccache, and C. Paar, editors, *Advances in Cryptography - Proceedings CHES 2001*, volume 2162 of *Lecture Notes in Computer Science*, pages 391–401. Springer, 2001.
9. B. Möller. Securing Elliptic Curve Point Multiplication against Side-Channel Attacks. In G.I. Davida and Y. Frankel, editors, *Advances in Cryptology - Proceedings of ISC 2001*, volume 2200 of *Lecture Note in Computer Science*, pages 324–334. Springer, 2001.
10. Y-H. Park, S. Jeong, C. Kim, and J. Lim. An Alternate Decomposition of an Integer for Faster Point Multiplication on Certain Elliptic Curves. In D. Naccache and P. Paillier, editors, *Advances in Cryptology - Proceedings of PKC 2002*, volume 2274 of *Lecture Notes in Computer Science*, pages 323–334. Springer, 2002.
11. F. Sica, M. Ciet, and J-J. Quisquater. Analysis of the Gallant-Lambert-Vanstone Method based on Efficient Endomorphisms: Elliptic and Hyperelliptic Curves. In H. Heys and K. Nyberg, editors, *Advances in Cryptology - Proceedings of SAC 2002*, Lecture Notes in Computer Science. Springer, 2002. To appear.
12. J. A. Solinas. An Improved Algorithm for Arithmetic on a Family of Elliptic Curves. In Burton S. Kaliski Jr., editor, *Advances in Cryptology - Proceedings of CRYPTO 1997*, volume 1294 of *Lecture Notes in Computer Science*, pages 357–371. Springer, 1997.
13. J.A. Solinas. Low-Weight Binary Representations for Pairs of Integers. Technical Report CORR 2001-41, CACR, Available at: `www.cacr.math.uwaterloo.ca/techreports/2001/corr2001-41.ps`, 2001.

A Algorithm for Elliptic Scalar Multiplication Using Sub-lattices

1. Randomly choose $\alpha, \beta, \gamma, \delta \in [1, R]$ such that $\alpha\delta - \beta\gamma \neq 0$ in Case 1 (resp. $= \pm 1$ in Case 2).
2. Compute $\Phi_0(P)$ and $\Phi_1(P)$ with Solinas' algorithm as
$$\begin{pmatrix} \Phi_0(P) \\ \Phi_1(P) \end{pmatrix} = \begin{pmatrix} \alpha & \beta \\ \gamma & \delta \end{pmatrix} \begin{pmatrix} 1 \\ \Phi(P) \end{pmatrix} .$$
3. Compute $\lambda_0^{-1} \in [0, n-1]$ such that $\lambda_0\lambda_0^{-1} \equiv 1 \pmod{n}$ with an application of the extended Euclidean algorithm.
4. Compute $\lambda' = \lambda_1\lambda_0^{-1} \pmod{n}$.
5. Find v_1' and v_2' in $\mathbb{Z}\times\mathbb{Z}$ by applying the original GLV algorithm, replacing λ by λ' (resp. $v_i' = v_i$ in Case 2).
6. Express $(k\lambda_0^{-1}, 0) = \beta_1 v_1' + \beta_2 v_2'$, where $\beta_i \in \mathbb{Q}$.
7. Let $b_i = \lceil \beta_i \rfloor$ and $v' = b_1 v_1' + b_2 v_2'$.
8. Compute $(k_1', k_2') = (k\lambda_0^{-1}, 0) - v'$.
9. Compute kP as $k_1'\Phi_0(P) + k_2'\Phi_1(P)$ with the elliptic Straus-Shamir (Solinas) algorithm.

B Algorithm for Affine Generalisation

1. Randomly choose $\rho_1, \rho_2 \in [1, R]$.
2. Decompose k as $(k, 0) = \beta_1 v_1 + \beta_2 v_2$, with $\beta_1, \beta_2 \in \mathbb{Q}$ using [2, Lemma 1].
3. Let $b'_i = \lceil \beta_i \rfloor - \rho_i$ for $i \in \{1, 2\}$, where $\lceil \cdot \rfloor$ means the nearest integer.
4. Let $u' = (k, 0) - v'$ where $v' = b'_1 v_1 + b'_2 v_2$, and $(k'_1, k'_2) = u'$.
5. Compute kP as $kP = k'_1 P + k'_2 \Phi(P)$, with the elliptic Straus-Shamir (Solinas) algorithm

Randomized Signed-Scalar Multiplication of ECC to Resist Power Attacks

Jae Cheol Ha[1] and Sang Jae Moon[2]

[1] Division of Information Science, Korea Nazarene Univ., Cheonan, Choongnam, 330-718, Korea
jcha@kornu.ac.kr
[2] School of Electronic and Electrical Engineering Kyungpook National Univ., Taegu, 702-701, Korea
sjmoon@knu.ac.kr

Abstract. Recently it has been shown that smart cards as cryptographic devices are vulnerable to power attacks if they have no defence against them. Randomization on ECC scalar multiplication is one of the fundamental concepts in methods of defence against side-channel attacks. In this paper by using the randomization concept together with the NAF recoding algorithm, we propose an efficient countermeasure for ECCs against power attacks. The countermeasure provides a randomized signed-scalar representation at every scalar multiplication to resist DPA. To protect against SPA it additionally employs a simple SPA-immune addition-subtraction multiplication algorithm. Our analysis shows that it needs no additional computation load compared to the ordinary binary scalar multiplication, where the average number of doublings plus additions for a bit length n is $1.5n+O(1)$.

Keywords: Elliptic curve cryptosystems, Side-channel attack, Power analysis attack, SPA, DPA, Non-adjacent form.

1 Introduction

The use of elliptic curve cryptosystems(ECC) was first proposed in 1985 by Miller [1] and Koblitz[2]. ECCs can use much smaller sizes of key bits, typically around 160 bits which provides the same security level as a 1024 bit RSA. In addition ECCs have better performance in computation speed than other multiplicative groups of RSA and ElGamal type encryptions at the same security level. These advantages make ECCs more attractive for cryptographic implementation on smart cards because limited memory and computation capability are available on them. Unfortunately cryptographic systems on smart cards are much vulnerable to side-channel attacks[3,4,5,6,7,8,9,10,11,12,13,14,15] such as fault attacks, timing attacks and power attacks. Since power attacks are known to be the most practical and powerful especially to cryptosystems on smart cards, in this paper we discuss in building a countermeasure against power attacks.

B.S. Kaliski Jr. et al. (Eds.): CHES 2002, LNCS 2523, pp. 551–563, 2003.

Kocher et al.[7] originally presented power attacks of the simple and differential power analyses(referred to as SPA and DPA, respectively). In SPA, a single power signal trace of a cryptographic execution is measured and analyzed to classify point doublings or additions over the power trace. On the other hand, a DPA attacker usually measures hundreds of power signal traces and divides them into two groups by using a classification criterion, and makes a subtraction between the two averaged values. Since the averaging can reduce noisy components and result in the amplification of small power differences occurred in the execution of an algorithm, DPA is in general more powerful than SPA. Coron[8] introduced three key concepts to build countermeasures against power attacks; randomization of the private exponent, blinding the point and randomized projective coordinates.

More recently Oswald and Aigner[9] randomized the binary algorithm itself to resist power attacks. They inserted a random decision in the process of building the addition-subtraction chain which had been originally utilized for speeding up the ordinary binary scalar multiplication of an elliptic curve point[16]. The speeding-up chain method was first introduced in 1990 by Morain and Olivos[17].

The contribution of this paper is to propose a simple and powerful countermeasure for ECC scalar multiplication against power attacks. It uses the randomization concept together with non-adjacent form(NAF) algorithm[18,19] to change an ordinary binary multiplication representation to a form of signed-scalar one. The proposal provides an differently randomized signed-scalar representation at every scalar multiplication to resist DPA. To defeat SPA it employs a simple SPA-immune addition-subtraction multiplication algorithm. In addition the new countermeasure seems to be able to make timing attacks very difficult to work because every execution time of scalar multiplication depends on every different signed-scalar representation. The analysis shows that our proposal needs no additional computation load compared to the ordinary binary scalar multiplication, where the average number of doublings plus additions for a large n is $1.5n + O(1)$.

This paper is organized as follows. The ECC ordinary scalar multiplication and power attacks are briefly described in section 2. We explain the non-adjacent form(NAF) recoding algorithm in section 3, and finally present the new countermeasure with analyses and comparisons.

2 Elliptic Curve Cryptosystems and Power Attacks

An elliptic curve is a set of points (x, y) which are solutions of a bivariate cubic equation over a field K. An equation of the form

$$y^2 + a_1xy + a_3y = x^3 + a_2x^2 + a_4x + a_6$$

where $a_i \in K$, defines an elliptic curve over K. As an example, if char $K \neq 2$ and char $K \neq 3$, the above equation can be transformed to

$$y^2 = x^3 + ax + b$$

with $a, b \in K$. This curve has one point O at infinity, which is the identity element of the group.

Let $P = (x_1, y_1) \neq O$ be a point, inverse of P is $-P = (x_1, -y_1)$. Let $Q = (x_2, y_2) \neq O$ be a second point with $Q \neq -P$, the sum $P + Q = (x_3, y_3)$ can be calculated as

$$x_3 = \lambda^2 - x_1 - x_2$$

$$y_3 = \lambda(x_1 - x_3) - y_1$$

with

$$\lambda = \begin{cases} \frac{y_2 - y_1}{x_2 - x_1}\ , & if \quad P \neq Q \\ \frac{3x_1^2 + a_4}{2y_1}\ , & if \quad P = Q. \end{cases}$$

To subtract the point $P = (x, y)$, one adds the point $-P$.

2.1 Binary Scalar Multiplication

The operation of adding a point P to itself k times is called scalar multiplication by k and denoted $Q = kP$. We usually make use of the binary algorithm for the computation of the scalar multiplication $Q = kP$. The binary algorithm is described in the following figure 1. The binary algorithm is the analogue of the square-and-multiply method for exponentiation. For validity and explanation see [16].

`Binary algorithm`
`Input:` A point P, an n-bit integer $k = \sum_{i=0}^{n-1} k_i\, 2^i$, $k_i \in \{0, 1\}$
`Output:` $Q = kP$

1.1 $Q = O$
1.2 for $i = n - 1$ to 0 by -1 do {
1.3 $Q = 2Q$
1.4 if ($k_i == 1$) then $Q = Q + P$ }
1.5 Return Q

Fig. 1. Binary scalar multiplication algorithm

If the occurrence of 1's of a scalar integer k is assumed $\frac{1}{2}$ in probability, the number of doublings in the algorithm is equal to n, and the average number of additions $\frac{n}{2}$, where n is the number of bits of k.

2.2 Power Attacks

Power attacks of SPA and DPA were introduced in [7]. An SPA consists in observing the power consumption of one single execution of a cryptographic algorithm. In ECC computing of $Q = dP$ as shown in figure 1, it might be

possible to distinguish a point addition from a measured power signal trace, because the step 1.4 of figure 1 is conducted over the period of $k_i = 1$. Figure 2 shows a countermeasure[8] to the SPA, where the instructions conducted during a cryptographic algorithm do not depend on the data being processed.

Notice that the algorithm of figure 2 no longer defends against DPA. The step of 2.5 in the figure 2 has a small different amount of power consumption depending on whether the bit is 1 or 0. Even though it is very hard to find such a small difference by SPA, DPA is a powerful technique that exploits secret information by statistical analyzing power consumption.

SPA resistant Binary algorithm
Input: A point P, an n-bit integer $k = \sum_{i=0}^{n-1} k_i\, 2^i$, $k_i \in \{0, 1\}$
Output: $Q[0] = kP$

2.1 $Q[0] = O$
2.2 for $i = n - 1$ to 0 by -1 do {
2.3 $Q[0] = 2Q[0]$
2.4 $Q[1] = Q[0] + P$
2.5 $Q[0] = Q[k_i]$ }
2.6 Return $Q[0]$

Fig. 2. Binary scalar multiplication algorithm immune to SPA

In order to be resistant to DPA, some countermeasures have been proposed[8, 9,10,11,12,13,14,15]. Three countermeasures for ECC were first suggested by Coron[8] : randomization of the private exponent, blinding the point P and randomized projective coordinates. Specific countermeasures on a Koblitz curve[12], on a Montgomery-form elliptic curve[13], on a Jacobi-form elliptic curve[14], and on a Hessian-form elliptic curve[15] were also proposed.

More recently, Oswald-Aigner[9] have proposed randomized addition- subtraction chains for an elliptic curve scalar multiplication as a countermeasure against power attacks. It randomizes the binary algorithm by using addition-subtraction chains which had been proposed by Morain-Olivos[17].

Our difference from Oswald-Aigner's work is to use a randomization on the NAF algorithm to resist DPA, and employ a simple SPA-immune scheme. Even though the idea seems somewhat straightforward, the result turns out to be a simple and powerful countermeasure against power attacks. Moreover it needs no additional computation load compared to the ordinary binary algorithm, while the randomized addition-subtraction chains proposed by Oswald-Algner needs approximately 9% more additions than the ordinary binary algorithm.

3 Non-adjacent Form(NAF) Recoding Algorithm

Since subtraction has the same load as addition in the elliptic curve group, the NAF representation in the form of addition-subtraction chain can reduce the number of point operations in ECCs compared to the ordinary binary representation. A minimum discussion about the NAF to describe the randomized signed-scalar representation will be given here. For details see [16,18,19,20,21].

Consider an integer representation of the form $d = \sum_{i=0}^{n} d_i\, 2^i$, $d_i \in \{\bar{1}, 0, 1\}$ where $\bar{1} = -1$. We call it a binary signed-digit representation. A non-adjacent form (NAF) has the lowest weight among all signed-digit representations of a given k. Notice that every integer k has each unique NAF. The NAF recoding number d of scalar k can be constructed by the NAF recoding algorithm of Reitwiesner[18] given in table 1. First the auxiliary carry variable c_0 is set to 0. Reitwiesner's algorithm computes d starting from the LSB of k and scanning two bits at a time to the left. The i-th NAF recorded digit d_i and $(i+1)$-th value of the auxiliary binary carry c_{i+1} for $i = 0, 1, 2, \cdots, n$ are successively produced using table 1[19].

Even though the number of bits of k is equal to n, the number of bits in the d can be $n+1$, As an example, when $k = (11110)$ in the binary form, we compute the NAF recoding number d as follows.

$$k = (11110) = 2^4 + 2^3 + 2^2 + 2^1 = 30$$

$$d = (1000\bar{1}0) = 2^5 - 2^1 = 30$$

Given a NAF recoding number d, the addition-subtraction scalar multiplication algorithm is given in figure 3. The number of doubling operations required can be at most 1 more than that of the ordinary binary algorithm. On the other hand, the number of subsequent additions or subtractions is simply equal to the number of non-zero bits of the NAF recoding number d. The average number of additions (or subtractions) for the bit length n can be reduced $\frac{n}{3}$ [20,21].

Table 1. NAF recoding method

Input			Output	
k_{i+1}	k_i	c_i	c_{i+1}	d_i
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	0
1	0	1	1	$\bar{1}$
1	1	0	1	$\bar{1}$
1	1	1	1	0

`Addition-Subtraction algorithm`
`Input:` A point P, an $(n+1)$-bit integer $d = \sum_{i=0}^{n} d_i 2^i$, $d_i \in \{\bar{1}, 0, 1\}$
`Output:` $Q = dP$

3.1 $Q = O$
3.2 for $i = n$ to 0 by -1 do {
3.3 $Q = 2Q$
3.4 if ($d_i == 1$) then $Q = Q + P$
3.5 if ($d_i == \bar{1}$) then $Q = Q - P$ }
3.6 Return Q

Fig. 3. Addition-subtraction scalar multiplication algorithm

4 The New Countermeasure Based on Randomized Signed-Scalar Representation

4.1 Randomized Signed-Scalar Representation

To prevent DPA, we intend to randomize the ECC scalar multiplication procedure. The randomization results in a signed-scalar representation which is not in the form of NAF. The randomized signed-scalar recoding algorithm can be built by employing the concept used in the NAF recoding algorithm as follows.

Note that we use auxiliary carry c_{i+1} in the NAF recoding algorithm. The carry c_{i+1} means an $(i+1)$-th carry with $c_0 = 0$, and d_i is an i-th NAF digit. Therefore, the concatenated $c_{i+1}d_i$ has the value of $c_{i+1}2^1 + d_i 2^0$. Therefore, we can consider that the representation $c_{i+1}d_i = 01$ has another identical representation, i.e. $c_{i+1}d_i = 1\bar{1}$, or the reverse.

To insert randomness in the NAF recoding algorithm, we generate a random number $r = (r_{n-1}r_{n-2}\cdots r_0)$ which is an n-bits integer. In our countermeasure algorithm, the random recorded digit d_i and next value of the auxiliary binary variable c_{i+1} for $i = 0, 1, 2, \cdots, n$ can be sequentially generated as shown in table 2.

As an example, if we have $k_{i+1}k_ic_i = 001$ and random bit $r_i = 0$, we take NAF recoding $c_{i+1}d_i = 01$. For $r_i = 1$, then we can choose another recoding $c_{i+1}d_i = 1\bar{1}$. However, two different choices have the same value. In all cases except when $k_{i+1}k_ic_i = 001$, $k_{i+1}k_ic_i = 010$, $k_{i+1}k_ic_i = 101$ or $k_{i+1}k_ic_i = 110$ the random signed-scalar recoding method is always adapted independent of r_i.

As a practical example, when $k = (111011110)$ in binary form, we compute a random recoding number d with $r = (101010011)$.
$k = (111011110) = 2^8 + 2^7 + 2^6 + 2^4 + 2^3 + 2^2 + 2^1 = 478$
$c = (1111111000), \quad r = (101010011)$
$d = (1000\bar{1}00\bar{1}10) = 2^9 - 2^5 - 2^2 + 2^1 = 478$
With a random number $r = (110101001)$, we compute another recoding number d as follows.
$c = (1110111100), \quad r = (110101001)$
$d = (100\bar{1}1000\bar{1}0) = 2^9 - 2^6 + 2^5 - 2^1 = 478$

Table 2. Random singed-scalar recoding method

Input				Output		
k_{i+1}	k_i	c_i	r_i	c_{i+1}	d_i	Remarks
0	0	0	0	0	0	NAF
0	0	0	1	0	0	NAF
0	0	1	0	0	1	NAF
0	0	1	1	1	$\bar{1}$	AF
0	1	0	0	0	1	NAF
0	1	0	1	1	$\bar{1}$	AF
0	1	1	0	1	0	NAF
0	1	1	1	1	0	NAF
1	0	0	0	0	0	NAF
1	0	0	1	0	0	NAF
1	0	1	0	1	$\bar{1}$	NAF
1	0	1	1	0	1	AF
1	1	0	0	1	$\bar{1}$	NAF
1	1	0	1	0	1	AF
1	1	1	0	1	0	NAF
1	1	1	1	1	0	NAF

The hardware or software implementation of randomized signed-scalar representation is not difficult. In hardware, it additionally needs a random number generator and a 4-input/2-output logic circuit to implement the operations of table 2. This computational load is negligible.

4.2 Analysis and Comparisons

It is possible to find the probability of each symbol in the new random recoding algorithm for a given value of k. We assume that an n-bit binary number k is uniformly distributed in the range $[0,\ 2^n-1]$. Thus each bit of k can be generated a value of zero or one with equal probability, i.e. $P(k_i = 0) = P(k_i = 1) = \frac{1}{2}$ for all i. It is also supposed that the each bit probability of zero or one for a random number r is 1/2. The random signed-scalar numbers produced by the new algorithm can be considered a finite Markov chain model which is a similar analysis method to that employed in [19]. In this paper, the state variables are taken to be the quadruplets (k_{i+1}, k_i, c_i, r_i). There are 16 states for 4-bit combinations of input as given in table 3.

For example, consider input state s_2 which represents $(k_{i+1}, k_i, c_i, r_i) = (0, 0, 1, 0)$. The output (c_{i+1}, d_i) can be calculated as (0, 1) using the table 3 and the next state is $(k_{i+2}, k_{i+1}, c_{i+1}, r_{i+1}) = (k_{i+2}, 0, 0, r_{i+1})$. By assuming $P(k_{i+2} = 0) = P(k_{i+2} = 1) = \frac{1}{2}$ and $P(r_{i+1} = 0) = P(r_{i+1} = 1) = \frac{1}{2}$, there are 4 transitions with equal probability from state $s_2 = (0, 0, 1, 0)$ to the states $s_0 = (0, 0, 0, 0)$, $s_1 = (0, 0, 0, 1)$, $s_8 = (1, 0, 0, 0)$ and $s_9 = (1, 0, 0, 1)$.

Table 3. State transition table for the random singed-scalar recoding algorithm

State		Output	Next state			
			(k_{i+2}, r_{i+1})			
s_i	(k_{i+1}, k_i, c_i, r_i)	(c_{i+1}, d_i)	(0,0)	(0,1)	(1,0)	(1,1)
s_0	(0,0,0,0)	(0,0)	s_0	s_1	s_8	s_9
s_1	(0,0,0,1)	(0,0)	s_0	s_1	s_8	s_9
s_2	(0,0,1,0)	(0,1)	s_0	s_1	s_8	s_9
s_3	(0,0,1,1)	$(1,\bar{1})$	s_2	s_3	s_{10}	s_{11}
s_4	(0,1,0,0)	(0,1)	s_0	s_1	s_8	s_9
s_5	(0,1,0,1)	$(1,\bar{1})$	s_2	s_3	s_{10}	s_{11}
s_6	(0,1,1,0)	(1,0)	s_2	s_3	s_{10}	s_{11}
s_7	(0,1,1,1)	(1,0)	s_2	s_3	s_{10}	s_{11}
s_8	(1,0,0,0)	(0,0)	s_4	s_5	s_{12}	s_{13}
s_9	(1,0,0,1)	(0,0)	s_4	s_5	s_{12}	s_{13}
s_{10}	(1,0,1,0)	$(1,\bar{1})$	s_6	s_7	s_{14}	s_{15}
s_{11}	(1,0,1,1)	(0,1)	s_4	s_5	s_{12}	s_{13}
s_{12}	(1,1,0,0)	$(1,\bar{1})$	s_6	s_7	s_{14}	s_{15}
s_{13}	(1,1,0,1)	(0,1)	s_4	s_5	s_{12}	s_{13}
s_{14}	(1,1,1,0)	(1,0)	s_6	s_7	s_{14}	s_{15}
s_{15}	(1,1,1,1)	(1,0)	s_6	s_7	s_{14}	s_{15}

Let T_{ij} be the probability of moving from state s_i to state s_j. From the above example we find that $T_{20} = T_{21} = T_{28} = T_{29} = \frac{1}{4}$ and $T_{0j} = 0$ for $j = 2, 3, 4, 5, 6, 7, 10, 11, 12, 13, 14, 15$ using table 3. By computing probabilities T_{ij} for all i and j, we can draw the one-step transition probability matrix of the chain as follows.

$$T = \begin{pmatrix}
\frac{1}{4} & \frac{1}{4} & 0 & 0 & 0 & 0 & 0 & 0 & \frac{1}{4} & \frac{1}{4} & 0 & 0 & 0 & 0 & 0 & 0 \\
\frac{1}{4} & \frac{1}{4} & 0 & 0 & 0 & 0 & 0 & 0 & \frac{1}{4} & \frac{1}{4} & 0 & 0 & 0 & 0 & 0 & 0 \\
\frac{1}{4} & \frac{1}{4} & 0 & 0 & 0 & 0 & 0 & 0 & \frac{1}{4} & \frac{1}{4} & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & \frac{1}{4} & \frac{1}{4} & 0 & 0 & 0 & 0 & 0 & 0 & \frac{1}{4} & \frac{1}{4} & 0 & 0 & 0 & 0 \\
\frac{1}{4} & \frac{1}{4} & 0 & 0 & 0 & 0 & 0 & 0 & \frac{1}{4} & \frac{1}{4} & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & \frac{1}{4} & \frac{1}{4} & 0 & 0 & 0 & 0 & 0 & 0 & \frac{1}{4} & \frac{1}{4} & 0 & 0 & 0 & 0 \\
0 & 0 & \frac{1}{4} & \frac{1}{4} & 0 & 0 & 0 & 0 & 0 & 0 & \frac{1}{4} & \frac{1}{4} & 0 & 0 & 0 & 0 \\
0 & 0 & \frac{1}{4} & \frac{1}{4} & 0 & 0 & 0 & 0 & 0 & 0 & \frac{1}{4} & \frac{1}{4} & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & \frac{1}{4} & \frac{1}{4} & 0 & 0 & 0 & 0 & 0 & 0 & \frac{1}{4} & \frac{1}{4} & 0 & 0 \\
0 & 0 & 0 & 0 & \frac{1}{4} & \frac{1}{4} & 0 & 0 & 0 & 0 & 0 & 0 & \frac{1}{4} & \frac{1}{4} & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & \frac{1}{4} & \frac{1}{4} & 0 & 0 & 0 & 0 & 0 & 0 & \frac{1}{4} & \frac{1}{4} \\
0 & 0 & 0 & 0 & \frac{1}{4} & \frac{1}{4} & 0 & 0 & 0 & 0 & 0 & 0 & \frac{1}{4} & \frac{1}{4} & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & \frac{1}{4} & \frac{1}{4} & 0 & 0 & 0 & 0 & 0 & 0 & \frac{1}{4} & \frac{1}{4} \\
0 & 0 & 0 & 0 & \frac{1}{4} & \frac{1}{4} & 0 & 0 & 0 & 0 & 0 & 0 & \frac{1}{4} & \frac{1}{4} & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & \frac{1}{4} & \frac{1}{4} & 0 & 0 & 0 & 0 & 0 & 0 & \frac{1}{4} & \frac{1}{4} \\
0 & 0 & 0 & 0 & 0 & 0 & \frac{1}{4} & \frac{1}{4} & 0 & 0 & 0 & 0 & 0 & 0 & \frac{1}{4} & \frac{1}{4}
\end{pmatrix}$$

Let π_i denote the limiting probability of state s_i. It can be found by solving linear equations $\pi T = \pi$ with $\pi_0 + \pi_1 + \cdots + \pi_{15} = 1$. The solutions of these equations are $\pi_i = \frac{1}{16}$ for $0 \leq i \leq 15$. Therefore the probability of each digit in a random signed-scalar number d is found by summing the limiting probability π_i of the state according to each output. As an example, the states for $d_i = 0$ are 8 states for $i = 0, 1, 6, 7, 8, 9, 14, 15$. By summing each limiting probability we get

$$P(d_i = 0) = \pi_0 + \pi_1 + \pi_6 + \pi_7 + \pi_8 + \pi_9 + \pi_{14} + \pi_{15} = \frac{1}{2},$$

$$P(d_i = 1) = \pi_2 + \pi_4 + \pi_{11} + \pi_{13} = \frac{1}{4},$$

$$P(d_i = \bar{1}) = \pi_3 + \pi_5 + \pi_{10} + \pi_{12} = \frac{1}{4}.$$

It is interesting to note that the average number of non-zero digits in the randomized signed-scalar number d is equal to $\frac{n}{2}$. Therefore the average number of additions (subtractions) required by the randomized signed-scalar recoding algorithm is $\frac{n}{2}$, which is the same as in the ordinary binary algorithm.

We consider now some possible power attacks. In DPA, suppose that the difference between doubling and addition (subtraction) operations is not distinguishable with a single power measurement. Every time the scalar multiplication is performed, it traces a different computational path due to randomization of the scalar such as that performed by our countermeasure algorithm. Therefore it makes the DPA attacks infeasible. In addition, the intermediate values to be attacked are computed at different times, or sometimes not even calculated. This make the DPA bias signal useless.

In the SPA case, it is assumed that the distinction between doubling and addition (subtraction) is visible with one power consumption measurement. It would be used to identify the correct secret key as in an ordinary binary representation. However, attackers can not distinguish addition from subtraction in our algorithm, which makes it difficult to identify the correct key.

Any possible weakness from the viewpoint of SPA can totally destroy DPA immunity countermeasure. This clearly shows the importance of developing a good SPA countermeasure. We here present a countermeasure against SPA, which is the SPA-immune addition-subtraction scalar algorithm given in figure 4. The proposed method is modified using the SPA resistant binary algorithm described in figure 2. The scheme in figure 4 also makes the power consumption independent of the secret digits. Consequently, the SPA countermeasure adopted the random signed-scalar recoding method makes the power attacks infeasible since random exponent d changes at each new operation of the scalar multiplication.

At this point we briefly compare the efficiency of the new algorithm with other algorithms that exist. The comparison of the number of operations is shown in table 4, which includes major operations(additions and doublings) without data copying or selection. Firstly, we compare our algorithm with the unprotected ordinary binary algorithm as each faces power attacks. The expected numbers

```
SPA resistant Addition-Subtraction algorithm
Input:  A point P, an (n+1)-bit integer d = Σ_{i=0}^{n} d_i 2^i, d_i ∈ {1̄, 0, 1}
Output: Q[0] = dP
```

4.1 $Q[0] = O$
4.2 $P[0] = P,\ P[1] = P,\ P[\bar{1}] = -P$
4.3 for $i = n$ to 0 by -1 do {
4.4 $\quad Q[0] = 2Q[0]$
4.5 $\quad Q[1] = Q[0] + P[d_i]$
4.6 $\quad Q[\bar{1}] = Q[1]$
4.7 $\quad Q[0] = Q[d_i]$ }
4.8 Return $Q[0]$

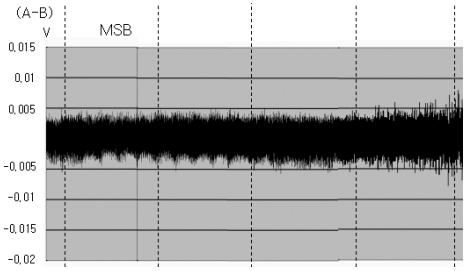
Fig. 4. SPA-immune addition-subtraction multiplication algorithm

of additions and doublings for an unprotected ordinary algorithm are $\frac{n}{2}$ and n respectively. As mentioned above, the NAF algorithm can reduce the additions (subtractions) to $\frac{n}{3}$ from $\frac{n}{2}$ of a ordinary binary algorithm. Our algorithm to resist DPA required $\frac{n}{2}$ additions and $(n+1)$ doublings which takes no extra time over using the unprotected ordinary binary algorithm. If this algorithm is compared with the unprotected NAF algorithm, it would be somewhat slower according to the above analysis. It is clear that the protected algorithms for SPA using the ordinary binary method required n additions and n doublings. Our countermeasure algorithm against DPA and SPA required almost exactly the same computation load.

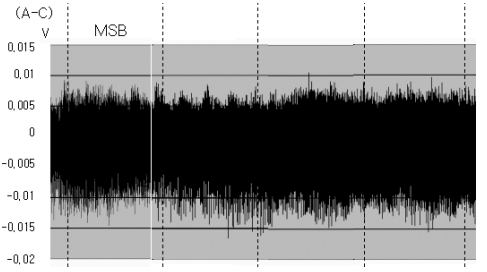
Table 4. Comparison of expected operations

Algorithm	Additions	Doublings	References
Unprotected ordinary binary	$\frac{n}{2}$	n	Fig. 1
Unprotected NAF	$\frac{n}{3}$	$n+1$	Fig. 3, Table 1
Protected ordinary binary against SPA	n	n	Fig. 2
Protected our algorithm against DPA	$\frac{n}{2}$	$n+1$	Fig. 3, Table 2
Protected our algorithm against DPA + SPA	$n+1$	$n+1$	Fig. 4, Table 2

Coren also proposed a modified binary scalar multiplication algorithm to resist SPA and three countermeasures against DPA[8]. We especially compare our countermeasure with the Coren's first DPA solution adopted SPA immuune algorithm as shown in figure 2. In his countermeasure, one needs to store an additional parameter the number of points $\#E(K)$, which is often not desirable. The main difference with our countermeasure is the bitlength of randomized key. If one selects a random number r of size m bits, in practice $m = 20$ bits, then his solution to resist DPA and SPA may increase computational operations up to $(n+m)$ additions and $(n+m)$ doublings.



(a) Difference between the correct one {1, 0, 0, X,···} and other scalar {1, 0, 1, X,···}



(b) Difference between the correct one {1, 0, 0, X,···} and the randomized signed-scalar representation

Fig. 5. The MESD attack with averaging over 300 traces for each scalar multiplication

4.3 Experimental Result

We experimentally applied the new countermeasure against a multiple-exponent, single-data (MESD) attack[11] for a simple test. The experimental result is shown in figure 5, where we assumed the attack was on the third digit {0} from the correct NAF scalar digits {1, 0, 0,···}. Since the step 4.4 and 4.5 of the multiplication algorithm procedure in figure 4 make a difference in power consumption, the averaged power peaks occur at the bit period right after the wrong guessing bit. The difference trace of (a) in figure 5 shows a lot of peaks over the period right after the third bit, where (a) is the difference between the averaged trace of the correct one {1, 0, 0, X,···} and a different representation {1, 0, 1, X,···}, and "don't care" is denoted by X. The occurrence of peaks implies that our guess is wrong in third bit. Therefore, the MESD attack is successful.

If we use the countermeasure, it generates a differently randomized signed-scalar representation at every multiplication execution. Due to the randomization, there are no peak appearances distinguishable between two power traces. It can make DPA including MESD infeasible. This is shown in (b) of figure 5.

5 Conclusion

The countermeasure provides a differently randomized signed-scalar representation at every multiplication execution so that it makes DPA infeasible. In addition it uses an addition-subtraction multiplication algorithm to interleave dummy operations to protect against SPA. It also seems to be able to defeat timing attacks because every execution time of a scalar multiplication changes according to every differently randomized signed-scalar representation. The structure of the countermeasure was analyzed using a finite Markov chain model. The result shows that it needs no additional computation load compared to the ordinary binary scalar multiplication, where the average number of doublings plus additions for a bit length n is $1.5n + O(1)$.

Acknowledgments. We would like to thank JongRok Kim and SungHyun Kim at the Samsung Electronics Corporation for their providing with experimental devices and helpful discussions. We are also grateful to the anonymous referees for their comments and suggestions.

References

1. V. S. Miller, "Use of elliptic curve in cryptography," In *Advances in Cryptology – CRYPTO '85*, LNCS 218, pp. 417–426, Springer-Verlag, 1986.
2. N. Koblitz, *Elliptic curve cryptosystems,* In Mathematics of Computation, vol. 48, pp. 203–209, 1987.
3. R. Anderson and M. Kuhn, "Low cost attacks on tamper resistant devices," In *Pre-proceedings of Security Protocol Workshop*, pp. 125–136, Springer-Verlag, April 1997.
4. Bell Press Release, "New threat model breaks crypto codes," Sept. 1996, available at URL `<http://www.bellcore.com/PRESS/ADVSRY96/facts.html>`.
5. D. Boneh, R. A. DeMillo, and R. J. Lipton, "On the importance of checking cryptographic protocols for faults," In *Advances in Cryptology – EUROCRYPT '97*, LNCS 1233, pp. 37–51, Springer-Verlag, 1997.
6. F. Bao, R. H. Deng, Y. Han, A. Jeng, A. D. Narasimbalu, and T. Ngair, "Breaking public key cryptosystems on tamper resistant devices in the presence of transient faults," In *Pre-proceedings of Security Protocol Workshop*, pp. 115–124, Springer-Verlag, April 1997.
7. P. Kocher, J. Jaffe and B. Jun, "Differential power analysis," In *Advances in Cryptology – CRYPTO '99*, LNCS 1666, pp. 388–397, Springer-Verlag, 1999.
8. J. S. Coron, "Resistance against differential power analysis for elliptic curve cryptosystems," In *Cryptographic Hardware and Embedded Systems – CHES '99*, LNCS 1717, pp. 292–302, Springer-Verlag, 1999.
9. E. Oswald and M. Aigner, "Randomized addition-subtraction chains as a countermeasure against power attacks," In *Cryptographic Hardware and Embedded Systems – CHES '01*, LNCS 2162, pp. 39–50, Springer-Verlag, 2001.
10. B. Möller, "Securing elliptic curve point multiplication against side-channel attacks," In *Information Security Conference – ISC'01*, LNCS 2200, pp. 324–334, 2001.

11. T.S. Messerges, E.A. Dabbish, and R.H. Sloan, "Power analysis attacks of modular exponentiation in smartcards," In *Cryptographic Hardware and Embedded Systems – CHES '99*, LNCS 1717, pp. 144–157, Springer-Verlag, 1999.
12. M. A. Hasan, "Power analysis attacks and algorithmic approaches to their countermeasures for Koblitz curve cryptosystems," In *Cryptographic Hardware and Embedded Systems – CHES '00*, LNCS 1965, pp. 93–108, Springer-Verlag, 2000.
13. K. Okeya and K. Sakurai, "Power analysis breaks elliptic curve cryptosystems even secure against the timing attack," In *Advances in Cryptology – INDOCRYPT '00*, LNCS 1977, pp. 178–190, Springer-Verlag, 2000.
14. P. Y. Liardet and N. P. Smart, "Preventing SPA/DPA in ECC systems using the Jacobi form," In *Cryptographic Hardware and Embedded Systems – CHES '01*, LNCS 2162, pp. 391–401, Springer-Verlag, 2001.
15. M. Joye and J. J. Quisquater, "Hessian elliptic curves and side-channel attacks," In *Cryptographic Hardware and Embedded Systems – CHES '01*, LNCS 2162, pp. 402–410, Springer-Verlag, 2001.
16. D. E. Knuth, *The art of computer programming*, vol 2: Seminumerical Algorithms, Reading, MA: Addison-Wesley, 2nd Edition, 1981.
17. F. Morain and J. Olivos, "Speeding up the computation on an elliptic curve using addition-subtraction chains," *Inform Theory Appl.*, vol. 24, pp. 531–543, 1990.
18. G. W. Reitwiesner, *Binary arithmetic*, Advances in Computers, 1:231–308, 1960.
19. O. Eğecioğlu and Ç. K. Koç, "Exponentiation using canonical recoding," *Theoretical Computer Science*, vol. 129, no. 2, pp. 407–417, 1994.
20. O. Eğecioğlu and Ç. K. Koç, "Fast modular exponentiation," In E. Arikan, editor, *Communication, Control, and Signal Processing : Proceedings of 1990 Bilkent International Conference on New Trends in Communication, Control, and Signal Processing*, pp. 188–194, Bilkent Univ. Ankara, Turkey, July 1990.
21. C. N. Zhang, "An improved binary algorithm for RSA," *Computer Math. Applic.*, vol. 25, no. 6, pp. 15–24, 1993.

Fast Multi-scalar Multiplication Methods on Elliptic Curves with Precomputation Strategy Using Montgomery Trick

Katsuyuki Okeya[1] and Kouichi Sakurai[2]

[1] Hitachi, Ltd., Systems Development Laboratory,
292, Yoshida-cho, Totsuka-ku, Yokohama, 244-0817, Japan
ka-okeya@sdl.hitachi.co.jp
[2] Kyushu University,
Graduate School of Information Science and Electrical Engineering, 6-10-1, Hakozaki,
Higashi-ku, Fukuoka, 812-8581, Japan
sakurai@csce.kyushu-u.ac.jp

Abstract. Our development of efficient methods for the precomputation of multi-scalar multiplication for elliptic curve cryptosystems (ECCs) is presented. Multi-scalar multiplication is required in many forms of ECC, including schemes for the verification of ECDSA signatures. The simultaneous method is one known method for fast multi-scalar multiplication. The method has two stages: a precomputation stage and an evaluation stage. Points for use in the evaluation stage are computed in the precomputation stage. The actual multi-scalar multiplication is carried out on the basis of the precomputed points in the evaluation stage. In the evaluation stage of the simultaneous method, we are able to quickly compute the points of the multi-scalar multiple because few additions are required. On the other hand, if we use a large window width, we have to compute an enormous number of points in the precomputation stage. Hence, we have to compute an abundance of inversions, which carries a high computational cost. The result is that a large amount of time is required by the precomputation stage. This is the well-known draw-back of the simultaneous method. In our proposed method, we apply the Montgomery trick to reduce the number of inversions required with a width window w from $O(2^{2w})$ to $O(w)$. In addition, our proposed method computes uP and vQ for any u, v, then compute $uP + vQ$, where P, Q are elliptic points. This procedure enables us to remove points that will not be used later from the process of precomputation. Without our proposed method, an algorithm to compute precomputation table would have to be changed dependently on unused points. Compared with the method without Montgomery trick, our proposed method is 3.6 times faster than the conventional simultaneous method, i.e., than in the absence of the Montgomery trick. Moreover, the optimal window width for our proposed method is 3, whereas the corresponding width for conventional simultaneous methods is 2.

Keywords: *elliptic curve cryptography, multi-scalar multiplication, precomputation, Montgomery trick, simultaneous*

B.S. Kaliski Jr. et al. (Eds.): CHES 2002, LNCS 2523, pp. 564–578, 2003.

1 Introduction

During the world-wide deployment of the electronic-signature law, the infrastructure for digital signatures is spreading. To support the expected rapid growth in the scale of electronic commerce, efficient implementation of public key cryptosystems such as digital signatures is becoming more and more important. Several digital signature schemes have been developed, studied, and to some extent, applied; these include RSA [RSA78] and DSA [DSA] schemes. The ECDSA signature scheme [ANSI,IEEEp1363,SEC-1], which is based on elliptic curve cryptography [Kob87,Mil86], is particularly noteworthy, because it provides high levels of security with short keys. In this article, we will propose an elliptic multi-scalar multiplication method that includes an efficient form of precomputation. This method provides a faster way to carry out the multi-scalar multiplication that is required in such elliptic curve cryptosystems as the signature verification procedure of an ECDSA signature scheme.

1.1 Elliptic Curve Operations

Some elliptic curve cryptosystems such as signature generation of ECDSA signature scheme need operations of scalar multiplication. In many cases, scalar multiplication is dominant in the overall time taken in computation. Several methods for the fast computation of scalar multiplication have been proposed; these include methods base on the use of more efficient coordinate systems (such as projective coordinates [CC87] and Jacobian coordinates [CC87,CMO98]), on the use of precomputation tables (such as the window method [Knu81] and comb method [LL94]), on the subtraction of points (such as NAF [MO90]), and on the non-use of the y-coordinate (such as the Montgomery-form elliptic curve [Mon87]), and others.

On the other hand, verifying an ECDSA signature requires an operation in which the point $kP+lQ$ is computed from the elliptic points P, Q and the integers k, l. This operation is referred to as multi-scalar multiplication. While two rounds of scalar multiplication are conventionally used to obtain kP and lQ, certain methods operate by the direct computation of the multi-scalar multiple, i.e., by the simultaneous computation of $kP + lQ$. The simultaneous method [Elg85, HHM00,BHLM01] and interleaving exponentiation [Möl01] are two examples of methods that operate in this way.

Multi-scalar multiplication has other applications. A method of scalar multiplication in which endomorphisms [GLV01] are applied for faster computation has recently been proposed. This method requires multi-scalar multiplication. Thus, fast methods of multi-scalar multiplication are desirable for use in elliptic curve cryptosystems.

1.2 Our Contributions

We propose a simultaneous method of multi-scalar multiplication in which Montgomery trick [Coh93] is applied to obtain an efficient form of precomputation.

A simultaneous method for calculating multi-scalar multiple consists of two stages: a stage of precomputation and a stage of evaluation [Möl01]. Points for use in multi-scalar multiplication in the evaluation stage are computed in the precomputation stage. The additions calculated in the precomputation stage are in affine coordinates, since the points have to be in this form for the fast computation of elliptic addition in the evaluation stage. However, each addition of points in affine coordinates requires a finite field operation of inversion, and inversion carries a high computational cost. In particular, a given increase in the number of points to be precomputed leads to a much greater increase in the computational cost of precomputation.

The main contribution of our method is that Montgomery trick is used in the precomputation of additions of points in affine coordinates. Montgomery trick [Coh93] provides a way of inverting n elements of a finite field with a single inversion operation and $3(n-1)$ multiplications rather than with n inversion operations. A further advantage of Montgomery trick is that no more memory is consumed in computation than with the straightforward method. Montgomery trick thus reduces the number of inversion operations required in plural additions of points in affine coordinates, while taking up no more memory. This reduction allows us to compute $O(w)$ inversions instead of $O(2^{2w})$ inversions for a window width w. As a result, precomputation according to our proposed method is 3.6 times faster than with an otherwise equivalent method in which Montgomery trick is not applied.

Another known example of the use of Montgomery trick in fast computation is in precomputation for elliptic scalar multiplication by the window method [CMO98]. While uP is computed in precomputation for scalar multiplication, $uP + vQ$ is precomputed in multi-scalar multiplication, so the relation between the procedures of computation is complicated. This is particularly, in cases where some pairs (u, v) need not be computed and in the case of NAF (Non Adjacent Form) pairs $(\pm u, \pm v)$. In our proposed method, uP and vQ are computed first, and this is followed by the computation of $uP + vQ$. Following this procedure simplifies the relation between the procedures of computation.

This simplification adds a further improvement to that which we obtain by using Montgomery trick. As the window width is increased, increasingly large numbers of precomputed points go unused in the evaluation stage. We eliminate the computation of such points in the precomputation stage. The simplification allows us to discard such points without modifying the procedure of computation for the remainder of the points of precomputation. Without our proposed method, the algorithm for computing the precomputation table would have to be changed according to the unused points. Eliminating the computation of the unused points achieves further speedup and further reduces the consumption of memory. The respective effects are estimated as a 20% speedup and a reduction of about ten points in the precomputation stage for the simultaneous sliding window NAF method with a window width w of 3 and 160-bit scalars. Moreover, the optimum window width for our proposed method is 3, whereas the corresponding width for conventional simultaneous methods is 2.

As well as the simultaneous sliding window NAF method, our proposed method is adaptable to simultaneous methods in general, including the NAF method [Aki01], the interleaving exponentiation method [Möl01], and so on.

The remainder of this article is organized as follows: Section 2 explains multi-scalar multiplication. Section 3 is a review of conventional simultaneous methods of scalar multiplication. Section 4 outlines the efficient method of precomputation in which Montgomery trick is applied. Section 5 gives a comparison of the method and conventional methods in terms of computational cost. We confirm that the proposed method is faster than an otherwise equivalent method in which Montgomery trick is not applied.

2 Multi-scalar Multiplication

Let P and Q be elliptic points, and k and l be integers. In multi-scalar multiplication, an elliptic point $kP+lQ$ is computed from points P and Q and integers k and l. Multi-scalar multiplication is widely used in such elliptic curve cryptosystems as the procedure for signature verification in the ECDSA signature [ANSI, IEEEp1363,SEC-1], and EC-MQV [IEEEp1363,SEC-1] schemes. Furthermore, multi-scalar multiplication has other applications. Recently, a method of scalar multiplication [GLV01] in which endomorphisms are used for fast computation has been proposed. This method of scalar multiplication involves multi-scalar multiplication. In most cases where multi-scalar multiplication is applied, the process is dominant in determining the overall computational cost.

Methods for the computation of multi-scalar multiples can be classified into two types. In methods of one type, independent computation of the scalar multiples kP and lQ is followed by addition $(kP)+(lQ)$. In methods of the other type, the multi-scalar multiple $kP + lQ$ is computed in one stage, without separate computation of kP and lQ. An example of the former type is a method in which kP is computed by a comb method [LL94] and lQ is computed by a window method; these steps are followed by computation of $kP + lQ$. This approach has been applied in the signature verification process of the ECDSA, where we can assume that the point P is fixed; we can thus use a fixed-base comb method to compute kP and a window method to compute lQ, then compute the addition of the two. Examples of the latter type are examples of simultaneous methods. This article is mainly concerned with such examples.

3 Simultaneous Scalar Multiplication

A simultaneous method for scalar multiplication has two stages; one of precomputation and the other of evaluation. All of the elliptic points which will be required in the evaluation stage are computed and stored in a table during the precomputation stage. In the evaluation stage, the multi-scalar multiplied point is computed by using the table which was prepared in the precomputation stage.

Concrete algorithms for the precomputation stage of conventional simultaneous methods of scalar multiplication are often omitted; for example, this step

will be given as "Compute points of precomputation". In the remainder of this section, we describe some actual algorithms for the precomputation stage and review simultaneous methods of scalar multiplication.

Hereafter, we assume that p is a prime and $\mathbf{F}_p$ is the prime field of characteristic p, and we limit our discussion to those elliptic curves which are defined over the finite field $\mathbf{F}_p$. The argument applies to all elliptic curves defined over finite fields of characteristic 2 and those over optimal extension fields (OEF) [BP98]. In that case, while our proposed method may be fast, it is unfortunately, not particularly fast. This is because there are fast methods for inversion in such fields [HHM00,BP99]. For further details on elliptic curve cryptography, see [BSS99,Eng99,Men93,Sil86].

We choose the simultaneous sliding window NAF method from among the simultaneous methods, and now explain this method. However, the method we propose below is adaptable to the other simultaneous methods. The NAF and sliding window tricks are applied in the simultaneous sliding window NAF method. The original simultaneous method is known as Shamir's trick [Elg85, HHM00,BHLM01]. We assume that $w \geq 2$ applies to the window width w in the simultaneous sliding window NAF method.

Precomputation Stage. In the precomputation stage of the simultaneous sliding window NAF method, we compute elliptic points $uP + vQ$ for all $u, v \in [-f(w), f(w)]$ such that $u \neq 0 \pmod 2$ or $v \neq 0 \pmod 2$, where $f(w)$ is the integer which is given by $f(w) = \frac{2^{w+2}-(-1)^w-3}{6}$. Here, the condition $u \neq 0 \pmod 2$ or $v \neq 0 \pmod 2$ is needed because the least significant bits of u and v which are equal to 0 are not used in the evaluation stage when we apply the sliding window technique.

The relation $-R = (x, -y)$ holds for any elliptic point $R = (x, y)$. Thus, once we have computed $uP + vQ$ and $-uP + vQ$, we are easily able to get the points $-uP - vQ$ and $uP - vQ$ without significantly adding to computational costs, by simply substituting $-y$ for the y-coordinate y.

The formulae for the addition of points in affine coordinates are as follows: $x_3 = (\frac{y_2-y_1}{x_2-x_1})^2 - x_1 - x_2$, $y_3 = (\frac{y_2-y_1}{x_2-x_1})(x_1 - x_3) - y_1$, where $P_1 = (x_1, y_1)$, $P_2 = (x_2, y_2)$, $P_3 = (x_3, y_3)$ and $P_3 = P_1 + P_2$. Hence, the addition formulae require an inversion, $(x_2 - x_1)^{-1}$. Since the points uP and $-uP$ have the same x-coordinates, the inverse of the addition $uP + vQ$ is the same as that of $-uP + vQ$.

Algorithm 1 : The precomputation stage in the simultaneous sliding window NAF method
INPUT : Elliptic points P and Q and the window width w
OUTPUT : The precomputation table $\{uP + vQ | u, v \in [-f(w), f(w)]$ s.t. $u \neq 0 \pmod 2$ or $v \neq 0 \pmod 2\}$

1. for $u = 2$ to $f(w)$ do
   1.1. $uP \leftarrow (u-1)P + P$.
2. for $u = 1$ to $f(w)$ do
   2.1. $-uP \leftarrow -(uP)$.
3. for $v = 2$ to $f(w)$ do

3.1. $vQ \leftarrow (v-1)Q + Q$.
4. $PreComp = \{(u,v) \in [1, f(w)] \times [1, f(w)] | u \neq 0 \pmod 2$ or $v \neq 0 \pmod 2\}$.
5. for any $(u,v) \in PreComp$ do
   5.1. $uP + vQ \leftarrow (uP) + (vQ)$, $-uP + vQ \leftarrow (-uP) + (vQ)$.
6. for $v = 1$ to $f(w)$ do
   6.1. $-vQ \leftarrow -(vQ)$.
7. for any $(u,v) \in PreComp$ do
   7.1. $-uP - vQ \leftarrow -(uP + vQ)$, $uP - vQ \leftarrow -(-uP + vQ)$.

Evaluation Stage. In the evaluation stage, we use the table prepared in the stage to compute the multi-scalar multiple $kP + lQ$.

Algorithm 2 : The evaluation stage in the simultaneous sliding window NAF method
INPUT : t-bit integers k and l, elliptic points P and Q, the window width w, and a precomputation table $\{uP + vQ | u, v \in [-f(w), f(w)]$ s.t. $u \neq 0 \pmod 2$ or $v \neq 0 \pmod 2\}$
OUTPUT : The multi-scalar multiple $kP + lQ$

1. Write $k = (k_{t-1}, k_{t-2}, \cdots, k_0)$ and $l = (l_{t-1}, l_{t-2}, \cdots, l_0)$, where each k_i and l_i is an NAF bit.
2. $R \leftarrow \mathcal{O}$, $i \leftarrow t-1$
3. while $i \geq 0$ do
   3.1. if $k_i = 0$, $l_i = 0$ then $R \leftarrow 2R$, $i \leftarrow i-1$
   else do
   3.1.1 $j \leftarrow \max\{i - w + 1, 0\}$.
   3.1.2 while $k_j = 0$, $l_j = 0$ do
   3.1.2.1 $j \leftarrow j + 1$.
   3.1.3 $k' \leftarrow (k_i, k_{i-1}, \cdots, k_j)$, $l' \leftarrow (l_i, l_{i-1}, \cdots, l_j)$.
   3.1.4 $R \leftarrow 2^{i-j+1}R + (k'P + l'Q)$.
   3.1.5 $i \leftarrow j - 1$.
4. Output R.

For fast computation, the precomputation stage uses the addition formulae of $A \leftarrow A + A$ and the doubling formulae of $A \leftarrow A$, and the evaluation stage uses the addition formulae of $J^m \leftarrow J + A$, the doubling formulae of $J \leftarrow J^m$ for doubling prior to addition, and the doubling formulae of $J^m \leftarrow J^m$ for doubling prior to doubling, where A, J and J^m indicate affine coordinates, Jacobian coordinates, and modified Jacobian coordinates [CMO98], respectively.

Other examples of simultaneous methods of scalar multiplication are the interleaving exponentiation method [Möl01] and the method of simultaneous scalar multiplication [Aki01] on a Montgomery-form elliptic curve [Mon87].

4 Proposed Precomputation Stage

We explain our proposed method in this section, that is, the method of computation for use in the precomputation stage in which Montgomery trick is applied to obtain efficiency. Our proposed method reduces the number of inversions required and thus provides a quick way of preparing the precomputation table.

Given a window width of w, precomputation for a simultaneous method requires the computation of $O(2^{2w})$. Thus, a large window width requires the computation of an enormous number of points. Moreover, the computation of points in affine coordinates[1] requires the computation of inversion, which carries a high computational cost. Therefore, we have to reduce the number of inversion operations to obtain faster computation in the precomputation stage.

For faster computation in our proposed method, Montgomery trick is applied in computing plural inverses with a single inversion operation rather than with plural inversion operations. Montgomery trick has been applied for fast computation in, for example, precomputation for elliptic scalar multiplication with the window method [CMO98]. While precomputation for scalar multiplication is required to provide $\{uP|u \in [0, 2^w - 1]\}$, precomputation for multi-scalar multiplication is required to provide $\{uP + vQ|u, v \in [0, 2^w - 1]\}$. In the case of scalar multiplication, u is the only variable we need to consider, so we compute $\{2P\}, \{3P, 4P\}, \{5P, 6P, 7P, 8P\}, \cdots$. However, we have two variables in the case of multi-scalar multiplication, namely u and v, so the relation between procedures of computation is complicated. This is particular so in cases where some pairs (u, v) need not be computed and in cases of NAF pairs $(\pm u, \pm v)$. This is because the flow of the algorithm used to compute the precomputation table may have to change according to the presence of such unused points. In our proposed method, $\{uP|u \in [0, 2^w - 1]\}$ and $\{vQ|v \in [0, 2^w - 1]\}$ are computed first, and these are followed by computation of $\{uP + vQ|(u, v) \in [1, 2^w - 1] \times [1, 2^w - 1]\}$. This procedure simplifies the relation between procedures of computation.

In the rest of this section, after reviewing Montgomery trick, we go on to describe our proposed method of precomputation in which we apply the trick.

4.1 Montgomery Trick

Given n elements $a_1, a_2, \cdots, a_n$ from a finite field $\mathbf{F}_p$, Montgomery trick[2] may be used to compute their inverses $b_1, b_2, \cdots, b_n$ in the following way [Coh93]:

Algorithm 3 : Montgomery trick
INPUT : $a_1, a_2, \cdots, a_n$
OUTPUT : Inverses $b_1, b_2, \cdots, b_n$ of $a_1, a_2, \cdots, a_n$

[1] If the precomputation table is not represented in affine coordinates, computation of the multi-scalar multiples in the evaluation stage will be slow.

[2] The algorithm of Montgomery trick which is given in [Coh93] is for integer factorization. We have modified algorithm 3 for use as the precomputation stage of simultaneous scalar multiplication by removing the part that carries out integer factorization.

1. $c_1 \leftarrow a_1$.
2. for $i = 2$ to n do
   2.1. $c_i \leftarrow c_{i-1}a_i$.
3. $u \leftarrow c_n^{-1}$.
4. for $i = n$ down to 2 do
   4.1. $b_i \leftarrow c_{i-1}u$.
   4.2. $u \leftarrow ua_i$.
5. $b_1 \leftarrow u$.

The computational cost of Montgomery trick is $3(n-1)M + I$, where M and I respectively denote operations of multiplication and inversion in $\mathbf{F}_p$.

Lemma 1. *For n elements, we apply Montgomery trick m times, and compute n inverses. The computational cost of this is then $3(n-m)M + mI$.*

Proof. Assume that we separate n into $n_1 + n_2 + \cdots + n_m$. The computational cost is then

$$\sum_{j=1}^{m}(3(n_j - 1)M + I) = \left(3\sum_{j=1}^{m} n_j M\right) - 3mM + mI$$
$$= 3(n-m)M + mI.$$

□

Remark 1. Lemma 1 shows that dividing n elements into m groups and applying Montgomery trick to each group leads to a computational cost of $3(n-m)M + mI$, which is independent of the division into m groups. This implies that the computational cost of the precomputation stage is solely dependent on number of times Montgomery trick is applied in the precomputation stage.

4.2 Simultaneous Sliding Window NAF Method

We describe a fast algorithm, in which Montgomery trick is applied, for the precomputation stage of the simultaneous sliding window NAF method. For a point R, x_R and y_R respectively denote the x- and y-coordinates.

Algorithm 4 : Applying Montgomery trick in a precomputation stage for the simultaneous sliding window NAF method
INPUT : Elliptic points P and Q and window width w
OUTPUT : The precomputation table $\{uP + vQ | u, v \in [-f(w), f(w)]$ s.t. $u \neq 0 \pmod 2$ or $v \neq 0 \pmod 2\}$

1. for $i = 1$ to $w - 1$ do
   1.1. Compute points $2^{i-1}P + jP, 2^{i-1}Q + jQ$ for any $j \in [1, 2^{i-1}]$:
      1.1.1 Use Montgomery trick to compute inverses of $(x_{jP} - x_{2^{i-1}P})$, $(x_{jQ} - x_{2^{i-1}Q})$, $(2y_{2^{i-1}P})$ and $(2y_{2^{i-1}Q})$ for any $j \in [1, 2^{i-1} - 1]$.

1.1.2 Compute points $(2^{i-1}+j)P, (2^{i-1}+j)Q$ using $(x_{jP} - x_{2^{i-1}P})^{-1}$, $(x_{jQ} - x_{2^{i-1}Q})^{-1}$, $(2y_{2^{i-1}P})^{-1}$ and $(2y_{2^{i-1}Q})^{-1}$.

2. Compute points $2^{w-1}P + jP, 2^{w-1}Q + jQ$ for any $j \in [1, f(w) - 2^{w-1}]$:
   2.1. Use Montgomery trick to compute inverses of $(x_{jP} - x_{2^{w-1}P}), (x_{jQ} - x_{2^{w-1}Q})$ for any $j \in [1, f(w) - 2^{w-1}]$.
   2.2. Compute points $(2^{w-1}+j)P, (2^{w-1}+j)Q$ using $(x_{jP} - x_{2^{w-1}P})^{-1}, (x_{jQ} - x_{2^{w-1}Q})^{-1}$.
3. for $u = 1$ to $f(w)$ do
   3.1. $-uP \leftarrow -(uP)$.
4. $PreComp = \{(u, v) \in [1, f(w)] \times [1, f(w)] | u \neq 0 \pmod 2$ or $v \neq 0 \pmod 2\}$.
5. Compute points $uP + vQ, -uP + vQ$ for any $(u, v) \in PreComp$:
   5.1. Use Montgomery trick to compute inverses of $(x_{vQ} - x_{uP})$ for any $(u, v) \in PreComp$.
   5.2. Compute points $uP + vQ$,$-uP + vQ$ using $(x_{vQ} - x_{uP})^{-1}$.
6. for $v = 1$ to $f(w)$ do
   6.1. $-vQ \leftarrow -(vQ)$.
7. for any $(u, v) \in PreComp$ do
   7.1. $-uP - vQ \leftarrow -(uP + vQ)$, $uP - vQ \leftarrow -(-uP + vQ)$.

Remark 2. The use of a coordinate system other than affine coordinates leads to greatly increased computational costs. This is because the computational cost of the precomputation stage with affine coordinates is $5M + S$ per point due to the application of Montgomery trick, whereas the equivalent cost with Jacobian coordinates ($J \leftarrow J + A$) is $8M + 3S$ per point; the former approach thus provides a faster way to compute elliptic addition, where S denotes an operation of squaring in $\mathbf{F}_p$.

4.3 Further Improvement: Reducing the Number of Points in Precomputation

We now propose a further improvement to the precomputation stage. As was mentioned above, while a larger window width requires the computation of larger numbers of points in precomputation, it also leads to fewer additions in the evaluation stage. Hence, a relatively large window width requires the precomputation of points which are not actually used in the evaluation stage. For example, in the case of the simultaneous method with a window width w of 3, while 63 points are precomputed, only 46 additions are computed, on average in the evaluation stage for 160-bit scalars. Thus, about 17 points are unnecessarily precomputed. Therefore, we thus use a trick to avoid the precomputation of points that will not actually be used. This provides us with a further increase in speed along with reduced memory consumption, since fewer points have to be stored. For the simultaneous sliding window NAF method with a window width w of 3 and 160-bit scalars, the effects are estimated as a 20% speedup and the elimination of about ten points in the precomputation stage.

Remark 3. This improvement is only achieved within the framework of our proposed method. For example, improvement in this way is not applicable to the following method. $P, Q \to 2P, P+Q, 2Q \to 3P, 4P, 2P+Q, 3P+Q, P+2Q, 2P+2Q, 3Q, P+3Q, 4Q \to \cdots$. If $P+Q$ is an unused point which is eliminated from the computation, then $3P+Q$ is not computable in the third phase, since $(P+Q)+2P$ is the only available way to compute $3P+Q$. Thus, unused points require modification of the algorithm.

Next, we show a concrete algorithm which determines precomputation points that are used in the evaluation stage.

Algorithm : An algorithm which determines precomputation points that are used in the evaluation stage of the simultaneous sliding window NAF method
INPUT : t-bit integers k, l, a window width w.
OUTPUT : Pairs of integers (k', l') that are used in the evaluation stage.

1. Write $k = (k_{t-1}, k_{t-2}, \cdots, k_0)$ and $l = (l_{t-1}, l_{t-2}, \cdots, l_0)$, where each k_i and l_i is an NAF bit.
2. $i \leftarrow t-1$, $num \leftarrow 0$.
3. while $i \geq 0$ do
   3.1. if $k_i = 0$, $l_i = 0$ then $i \leftarrow i-1$
   else do
   3.1.1 $j \leftarrow \max\{i-w+1, 0\}$.
   3.1.2 while $k_j = 0$, $l_j = 0$ do
   3.1.2.1 $j \leftarrow j+1$.
   3.1.3 $k' \leftarrow (k_i, k_{i-1}, \cdots, k_j)$, $l' \leftarrow (l_i, l_{i-1}, \cdots, l_j)$.
   3.1.4 for $j = 1$ to num
   3.1.4.1 If $T[j] = (k', l')$ then go to Step 3.1.6.
   3.1.5 $num \leftarrow num+1$, $T[num] \leftarrow (k', l')$.
   3.1.6 $i \leftarrow j-1$.
4. Output T.

Algorithm 4, the proposed precomputation stage, computes points $u_jP + v_jQ$ for $T[j] = (u_j, v_j)$ for $j = 1, \cdots, num$ in Step 5.

5 Computational Cost and Comparison

5.1 Precomputation Stage

Here, we start by estimating the computational cost of Algorithm 1, which is the conventional method of precomputation. Assume that $w > 1$. Doubling is the operation at both Steps 1.1 and 3.1 if u or v is equal to 2. If not, both steps are additions. The two elliptic additions in Step 5.1 only require a single 1 inversion, because they have a common inverse. The number of iterations of Step 5 is $\#PreComp = f(w)^2 - \left(\lfloor \frac{f(w)}{2} \rfloor\right)^2$. That is, $(2\#PreComp + 2f(w) - 4)$ elliptic additions and 2 elliptic doublings are computed by Algorithm 1. Meanwhile, the

number of inversions is reduced by $(\#PreComp)$. Thus, the computational cost is

$$\begin{aligned}&(2\#PreComp + 2f(w) - 4)\,(2M + S + I) + 2(2M + 2S + I) - (\#PreComp)\,I\\ =\ &(4\#PreComp + 4f(w) - 4)\,M + (2\#PreComp + 2f(w))\,S\\ &+ (\#PreComp + 2f(w) - 2)\,I.\end{aligned}$$

In the case of $w = 1$, the computational cost is $4M + 2S + I$.

Next, we estimate the computational cost of Algorithm 4, our proposed method. Assume that $w > 2$. In our proposed method, $(2\#PreComp + 2f(w) - 2w)$ elliptic additions and $(2w - 2)$ elliptic doublings are computed. Meanwhile, the number of inversions is reduced by $(\#PreComp)$ and Montgomery trick is applied $(w + 1)$ times in computing $(\#PreComp + 2f(w) - 2)$ inverses. Using Lemma 1, we obtain the following results for computational cost.

$$\begin{aligned}&(2\#PreComp + 2f(w) - 2w)(2M + S + I)\\ &+(2w - 2)(2M + 2S + I) - (\#PreComp)I\\ =\ &(4\#PreComp + 4f(w) - 4)M + (2\#PreComp + 2f(w) + 2w - 4)S\\ &+(\#PreComp + 2f(w) - 2)I\\ \to\ &(4\#PreComp + 4f(w) - 4)M + (2\#PreComp + 2f(w) + 2w - 4)S\\ &+3\,(\#PreComp + 2f(w) - 2 - (w + 1))\,M + (w + 1)I\\ =\ &(7\#PreComp + 10f(w) - 3w - 13)M\\ &+(2\#PreComp + 2f(w) + 2w - 4)S + (w + 1)I.\end{aligned}$$

Here, $\#PreComp = f(w)^2 - \left(\lfloor \frac{f(w)}{2} \rfloor\right)^2$. In the case of $w = 2$, the computational cost is $25M + 10S + 2I$.

5.2 Evaluation Stage

$\mathcal{A}D, DA$ and $\mathcal{D}D$ respectively denote addition prior to doubling, doubling prior to addition and doubling prior to doubling in the evaluation stage. We use the following coordinate systems for fast computation: $\mathcal{A}D : J + A \to J^m$ $(9M + 5S)$, $\mathcal{D}A : J^m \to J$ $(3M + 4S)$, $\mathcal{D}D : J^m \to J^m$ $(4M + 4S)$.

The white space of $(0, 0)$ between two consecutive windows has expected length of $\frac{4/9}{1-(4/9)} (= 0.8)$, since the probability that a NAF bit is equal to 0 is $\frac{2}{3}$. As a result, an average of $\frac{t-w}{w+0.8}$ additions have to be computed in the multi-scalar multiplication. Thus, the computational cost[3]is

$$\frac{t - w}{w + 0.8}(\mathcal{A}D + DA) + \left(t - w - \frac{t - w}{w + 0.8}\right)\mathcal{D}D$$

[3] Since 0 and non-zero bits are not randomly distributed, we need an estimate of the number of computations in which the bit dependence is considered. The smaller the window width, the larger the error in the estimate. In the case of $w = 2$, the computational cost for a 160-bit scalar is about $1869.5M$.

$$= \frac{t-w}{w+0.8}(\mathcal{A}D + DA + (w-0.2)\mathcal{D}D)$$
$$= \frac{t-w}{w+0.8}((4w+11.2)M + (4w+8.2)S).$$

Table 1. Computational cost for point multiplication $kP + lQ$

Method	Precomputation	Evaluation	Total	Points
	Proposed method			Stored
Simultaneous ($w = 1$)	$2.8M + I$	$2575.8M$	$2608.6M$	3
	$2.8M + I$		$2608.6M$	
Simultaneous ($w = 2$)	$38.0M + 13I$	$2039.2M$	$2467.2M$	15
	$68.0M + 3I$		$2197.2M$	
Simultaneous ($w = 3$)	$172.4M + 61I$	$1770.9M$	$3773.3M$	63
	$345.0M + 4I$		$2235.9M$	
Simultaneous NAF ($w = 1$)	$5.6M + I$	$2204.8M$	$2240.4M$	8
	$5.6M + I$		$2240.4M$	
Simultaneous NAF ($w = 2$)	$29.6M + 6I$	$1910.4M$	$2120.0M$	24
	$41.6M + 2I$		$2012.0M$	
Simultaneous NAF ($w = 3$)	$164.0M + 33I$	$1725.0M$	$2879.0M$	120
	$252.6M + 4I$		$2097.6M$	
Simul. slid. window NAF($w = 2$)	$24.0M + 5I$	$1869.5M$	$2043.5M$	16
	$33.0M + 2I$		$1962.5M$	
Simul. slid. window NAF($w = 3$)	$141.6M + 29I$	$1626.2M$	$2637.8M$	96
	$159.2M + 4I$		$1905.4M$	(33.6)
Interleaving ($w = 4$)	$35.2M + 12I$	$1740.3M$	$2135.5M$	20
	$52.4M + 4I$		$1912.7M$	
Interleaving ($w = 5$)	$68.8M + 24I$	$1642.4M$	$2431.2M$	44
	$119.0M + 5I$		$1911.4M$	

"Precomputation", "Proposed method" and "Evaluation" indicate the computational cost of the conventional precomputation method, our proposed method of precomputation, and the evaluation stage, respectively "Total" means the overall computational cost of the precomputation and evaluation stages. "Points Stored" means the number of points stored in the precomputed table. This number determines the consumption of memory. We assume that $t = 160$, $S = 0.8M$, and $I = 30M$, where t is the number of bits in the input integers k and l.

5.3 Comparison

Firstly, we compare the computational cost of the respective precomputation stages. The computational cost of Algorithm 1 is $16M + 10S + 5I$ for $w = 2$ and $100M + 52S + 29I$ for $w = 3$, while the corresponding figures for Algorithm 4, our proposed method, are $25M + 10S + 2I$ and $175M + 54S + 4I$[4], respectively.

[4] When we take the elimination of unused points into consideration, this computational cost is about $159.2M+4I$ and the number of the points stored in the precomputation table is about 33.6.

Table 2. Computational cost for point multiplication $kP + lQ$, with P fixed

Method	Precomputation	Evaluation	Total
	Proposed method		
Simultaneous ($w = 1$)	$2.8M + I$	$2575.8M$	$2608.6M$
	$2.8M + I$		$2608.6M$
Simultaneous ($w = 2$)	$31.6M + 11I$	$2039.2M$	$2400.8M$
	$58.6M + 2I$		$2157.8M$
Simultaneous ($w = 3$)	$155.0M + 55I$	$1770.9M$	$3575.9M$
	$311.6M + 3I$		$2172.5M$
Simultaneous NAF ($w = 1$)	$5.6M + I$	$2204.8M$	$2240.4M$
	$5.6M + I$		$2240.4M$
Simultaneous NAF ($w = 2$)	$26.0M + 5I$	$1910.4M$	$2086.4M$
	$35.0M + 2I$		$2005.4M$
Simultaneous NAF ($w = 3$)	$152.0M + 29I$	$1725.0M$	$2747.0M$
	$230.0M + 3I$		$2045.0M$
Simul. slid. window NAF($w = 2$)	$20.4M + 4I$	$1869.5M$	$2009.9M$
	$26.4M + 2I$		$1955.9M$
Simul. slid. window NAF($w = 3$)	$129.6M + 25I$	$1626.2M$	$2505.8M$
	$137.4M + 3I$		$1853.6M$
Interleaving ($w = 4$)	$14.8M + 5I$	$1740.3M$	$1905.1M$
	$31.0M + 4I$		$1891.3M$
Interleaving ($w = 5$)	$31.6M + 11I$	$1642.4M$	$2004.0M$
	$63.6M + 5I$		$1856.0M$

"Precomputation", "Proposed method" and "Evaluation" indicate the computational cost of the conventional precomputation method, our proposed method of precomputation, and the evaluation stage, respectively. "Total" means the overall computational cost of the precomputation and evaluation stages. We assume that $t = 160$, $S = 0.8M$ and $I = 30M$, where t is the number of bits in the input integers k and l.

Secondly, we compare the respective results for total computational cost. Assume that $t = 160$, where t is the number of bits in the input integers k and l. In the actual implementation [LH00], $S/M = 0.8$ and $I/M = 30$ are assumed[5]. The computational cost of Algorithm 2 is $1869.5M$ for $w = 2$ and $1626.2M$ for $w = 3$, respectively. Thus, the total computational cost with Algorithm 1 is $2043.5M$ for $w = 2$ and $2637.8M$ for $w = 3$, and the corresponding figures with Algorithm 4 are $1962.5M$ and $1905.4M$, respectively. The comparative results on computational cost that we have covered thus far are covered in Table 1.

We see from Table 1, the simultaneous sliding window NAF method is the fastest in terms of total computational cost. In the precomputation stage for the simultaneous sliding window NAF method, Algorithm 4, our proposed method, is 1.9 times faster than Algorithm 1 when $w = 2$, and 3.6 times faster when $w = 3$. The best choice of window width for the simultaneous sliding window

[5] If the ratio I/M is larger than 30, the proposed method is much more efficient than the conventional method. If not, the proposed method is not so efficient.

NAF method with our proposed method is 3, whereas the best width with the conventional method is 2.

On the other hand, in the multi-scalar multiplication $kP+lQ$ of the ECDSA scheme's signature-verification procedure, the point P is assumed to be fixed. Thus, the computation of points uP for $u \in [0, 2^w - 1]$ or $[0, f(w)]$ in advance of the precomputation stage removes the cost of computing these points from the precomputation stage. Moreover, the simultaneous computation of vQ and $uP + vQ$ reduces the number of applications of Montgomery trick by one, that is, the number of inversions is reduced by one. In a case where P is fixed, we obtain Table 2 in the same way[6].

Acknowledgements. The authors would like to thank the anonymous referees for their useful comments.

References

[Aki01] Akishita, T., *Fast Simultaneous Scalar Multiplication on Elliptic Curve with Montgomery Form*, Eighth Annual Workshop on Selected Area in Cryptography (SAC2001), (2001).

[ANSI] ANSI X9.62, Public Key Cryptography for the Financial Services Industry, *The Elliptic Curve Digital Signature Algorithm (ECDSA)*, (1998).

[BHLM01] Brown, M., Hankerson, D., López, J., Menezes, A., *Software Implementation of the NIST Elliptic Curves over Prime Fields*, Topics in Cryptology - CT-RSA 2001, LNCS2020, (2001), 250-265.

[BP98] Bailey, D.V., Paar, C., *Optimal Extension Fields for Fast Arithmetic in Public-Key Algorithms*, Advances in Cryptology - CRYPTO'98, LNCS1462, (1998), 472-485.

[BP99] Bailey, D.V., Paar, C., *Inversion in Optimal Extension Fields*, Conference on The Mathematics of Public Key Cryptography, (1999).

[BSS99] Blake, I.F., Seroussi, G., Smart, N.P., *Elliptic Curves in Cryptography*, Cambridge University Press,(1999).

[CC87] Chunnovsky, D., Chundovsky, G., *Sequences of numbers generated by addition in formal groups and new primality and factoring tests*, Advances in Applied Mathematics, 7 (1987), 385-434.

[CMO98] Cohen, H., Miyaji, A., Ono, T., *Efficient Elliptic Curve Exponentiation Using Mixed Coordinates*, Advances in Cryptology - ASIACRYPT '98, LNCS1514, (1998), 51-65.

[Coh93] Cohen, H., *A course in computational algebraic number theory*, GTM138, Springer-Verlag, New York, (1993).

[DSA] National Institute of Standards and Technology (NIST), *Digital Signature Standard (DSS)*, FIPS PUB 186-2, (2000).

[6] The combination of a fixed-base comb method and a sliding window method might provide good efficiency for the ECDSA verification. The fixed-base comb method ($w = 8$) and the sliding window method ($w = 4$) using the technique of Montgomery trick is an optimal choice in terms of speed and memory consumption. The computational cost with 160-bit scalars is $1862.6M$. Thus, the simultaneous sliding window NAF ($w = 3$) is faster.

[Elg85] ElGamal, T., *A public-key cryptosystem and a signature scheme based on discrete logarithm*, IEEE Transactions on Information Theory 31 (1985), 469-472.

[Eng99] Enge, A., *Elliptic Curves and their applications to Cryptography*, Kluwer Academic publishers,(1999).

[GLV01] Gallant, R.P., Lambert, R.J., Vanstone, S.A., *Faster Point Multiplication on Elliptic Curves with Efficient Endomorphisms*, Advances in Cryptology - CRYPTO 2001, LNCS2139, (2001), 190-200.

[HHM00] Hankerson, D., Hernandez, J.L., Menezes, A., *Software Implementation of Elliptic Curve Cryptography over Binary Fields*, Cryptographic Hardware and Embedded Systems (CHES2000), LNCS1965, (2000), 1-24.

[Kob87] Koblitz, N., *Elliptic curve cryptosystems*, Math. Comp. 48, (1987), 203-209.

[Knu81] Knuth, D.E., *The Art of Computer Programming, 2 - Semi-numerical Algorithms*, Addison-Wesley, 2nd edition, (1981).

[IEEEp1363] IEEE P1363 Standard Specifications for Public Key Cryptography (1999). Available at http://grouper.ieee.org/groups/1363/

[LH00] Lim, C.H., Hwang, H.S., *Fast implementation of Elliptic Curve Arithmetic in $GF(p^m)$*, Proc. PKC'00 LNCS1751, (2000), 405-421.

[LL94] Lim, C.H., Lee, P.J., *More Flexible Exponentiation with Precomputation*, Advances in Cryptology - CRYPTO '94, LNCS839, (1994), 95-107.

[Men93] Menezes, A.J., *Elliptic Curve Public Key Cryptosystems*, Kluwer Academic Publishers, (1993).

[Mil86] Miller, V.S., *Use of elliptic curves in cryptography*, Advances in Cryptology - CRYPTO '85, LNCS218,(1986),417-426.

[MO90] Morain, F., Olivos, J., *Speeding up the computations on an elliptic curve using addition-subtraction chains*, Theoretical Informatics and Applications 24 No.6, (1990), 531-544.

[Möl01] Möller, B., *Algorithm for multi-exponentiation*, Eighth Annual Workshop on Selected Area in Cryptography (SAC2001), (2001), 179-194.

[Mon87] Montgomery, P.L., *Speeding the Pollard and Elliptic Curve Methods of Factorizations*, Math. Comp. 48, (1987), 243-264

[RSA78] Rivest, R.L., Shamir, A., Adleman, L., *A Method for Obtaining Digital Signatures and Public-Key Cryptosystems*, Communications of the ACM, Vol.21, No.2, (1978), 120-126.

[SEC-1] Standards for Efficient Cryptography, *Elliptic Curve Cryptography Ver.1.0*, (2000), Available at http://www.secg.org/secg_docs.htm

[Sil86] Silverman, J.H., The Arithmetic of Elliptic Curves, GMT106, Springer-Verlag, New York, (1986).

Experience Using a Low-Cost FPGA Design to Crack DES Keys

Richard Clayton and Mike Bond

University of Cambridge, Computer Laboratory, Gates Building,
JJ Thompson Avenue, Cambridge CB3 0FD, United Kingdom
{richard.clayton, mike.bond}@cl.cam.ac.uk

Abstract. This paper describes the authors' experiences attacking the IBM 4758 CCA, used in retail banking to protect the ATM infrastructure. One of the authors had previously proposed a theoretical attack to extract DES keys from the system, but it failed to take account of real-world banking security practice. We developed a practical scheme that collected the necessary data in a single 10-minute session. Risk of discovery by intrusion detection systems made it necessary to complete the key "cracking" part of the attack within a few days, so a hardware DES cracker was implemented on a US$995 off-the-shelf FPGA development board. This gave a 20-fold increase in key testing speed over the use of a standard 800 MHz PC. The attack was not only successful in its aims, but also shed new light on the protocol vulnerabilities being exploited. In addition, the FPGA development led to a fresh way of demonstrating the non-randomness of some of the DES S-boxes and indicated when pipelining can be a more effective technique than replication of processing blocks. The wide range of insights we obtained demonstrates that there can be significant value in implementing attacks "for real".

1 Introduction

The IBM 4758 is a "cryptoprocessor" or "security module" – a tamper-resistant coprocessor that runs software providing cryptographic and security related services. Its API is designed to protect the confidentiality and integrity of data while still permitting access according to a configurable usage policy. Cryptoprocessors are commonly used in financial environments to protect ATM (cash machine) infrastructures, process customer Personal Identification Numbers (PINs), and secure transaction streams between banks. Other applications of cryptoprocessors include the protection of credit dispensing networks for prepayment electricity meters, and governing access to keys at certification authorities.

In [2] one of the authors of this paper described a number of flaws in the Common Cryptographic Architecture (CCA) – the default financial software for the IBM 4758. We set out to implement an actual attack on a 4758 in a simulated banking environment, combining these flaws to extract valuable key material. In particular, we wished to demonstrate the extraction of a "PIN derivation key", a Triple DES (3DES) key which can be used to calculate a customer's PIN from

B.S. Kaliski Jr. et al. (Eds.): CHES 2002, LNCS 2523, pp. 579–592, 2003.

the account number embossed on the front of their card. By creating fake cards with real account numbers, an attacker could then use the calculated PINs to plunder ATMs of their choice, anywhere in the country.

During this process, it became clear that there is far more to implementing a practical attack than meets the eye, and we had to make substantial modifications to the scheme and create new technology in order to get the job done. As a direct result of this work, IBM have released a new version of the CCA [12] containing multiple modifications to the API to defeat each technique used, and so the attack we describe can no longer be mounted.

Section 2 summarises the theory behind the building blocks of the attack, and section 3 describes why the banking environment and procedural controls make successful application of these building blocks a difficult task. Section 4 describes the extensions made to satisfy the requirements from section 3 – in particular, the attack was restructured to collect all necessary data within a single, very quick, period of access to the 4758. Section 5 covers the design and implementation of an FPGA based DES cracker to provide the necessary brute force attack performance. This provided some unexpected insights into DES and key cracker design generally. Section 6 presents brief results from test runs, and finally, conclusions are drawn in section 7.

2 Attacks on the IBM 4758 CCA

In his earlier paper [2], Bond identifies a number of weaknesses in the CCA, which he termed "building blocks".

The CCA keys are typically DES or 3DES keys, and are stored by encryption under a master key. When keys are to be transferred between banks, they are encrypted with a *key-encrypting key* (KEK) instead of the master key. As a KEK is the highest level of key shared between banks, there is no option but to transfer the KEK itself in plaintext. The standard procedure is to split it into three parts using XOR, and transfer each part with a separate courier. At the destination, three "security officers" enter the key parts into the cryptoprocessor, which recombines them into the original key. This procedure leads to a significant security problem: although the security officers must all collude to discover the key value, just one of them could modify the final key at will by changing the value of their single key part. In fact, unknown key parts can be generated which simulate the presence of other officers and so the control that a single corrupt security officer has over their own key part is enough to allow the CCA software to be attacked.

The CCA software does not place restrictions on key generation, so it is easy to create a large number of unknown DES keys. A particular *test pattern* can be encrypted under each of these keys to create a set of *test vectors*. A brute force attack can then be used to attack all of the unknown keys in parallel. To determine a single key by brute force might take decades to complete, but as there are multiple targets the expected time to determine one of the key values becomes only a few days. Therefore there is a trade-off between the time spent

on key generation and the time (and memory) spent on the brute force activity, which can be characterised as a "meet-in-the-middle" attack.

Bond also described a key binding problem. The CCA uses the common "two key" mode of 3DES, where keys consist of two halves, each a single DES key. The mode consists of encryption with the first half, decryption with the second and then encryption again with the first half. So-called "replicate keys" can be generated with both halves identical. In this case, two of the DES operations cancel out, making the key in effect a single DES key, and therefore suitable for inter-working with legacy systems. However, the CCA permits halves to be swapped at will between different keys. This binding problem means that if two replicate keys can be discovered, their halves can be swapped to create a full (non-replicate) 3DES key whose value is known.

Bond went on to combine these building blocks into several hypothetical attacks that were capable of compromising all the exportable keys in the device. However, the assembly of building blocks was only demonstrative, and his paper stopped short of actually describing attack code that could be deployed. Further investigation has made it clear that although the basic theory was correct, the security procedures in a banking environment would put extra requirements upon the attack that are not easy to fulfil.

3 Banking Security

To deploy any attack on a real-world bank, an attacker must circumvent a wide range of bank procedures that protect against fraud. These include:

- Dual control
- Functional separation
- Double-entry book-keeping
- Regular audits of security procedures
- Analysis of mainframe audit trails
- Compulsory uninterrupted holiday periods for staff

Before we show how to defeat the dual control on the security module using Bond's techniques, it is instructive to consider why the attacker does not target the bank mainframe. If an attacker had control over this he could simply increase his bank balance, creating money from nowhere. But bank procedures are specifically designed to ensure that even with top-level access, covering up the evidence of such a change is very difficult, and sophisticated balancing checks would report an inconsistency.

To determine how the fraud was done, the internal auditors would consult the audit trails. Practically every action that takes place in a bank leaves a logging record behind, from international fund transfers right down to the times that staff enter and exit rooms. Given the size and complexity of these auditing systems, cleaning every record to remove details of unauthorised activity is a mammoth task. The need for redundancy in the face of failure means that many of the mainframe logs will be append-only files kept at multiple remote, physically

secured, sites. There may be further trails to clean on other external machines and the attacker will only be able to guess whether the cover-up work is complete.

As an alternative to attacking the mainframe, security modules could be targeted. They do have much better physical security than other bank systems (for instance, the IBM 4758 is validated to FIPS 140-1 Level 4, which is the highest commercial evaluation level attainable). However, access to their software API is poorly audited, and the conditions under which to raise an alarm are badly understood. For instance, in order to steal PINs, the attacker need only breach the confidentiality of data rather than damage its integrity. The data flowing out of security modules is encrypted and the programmers creating the audit procedures may not fully understand the consequences of access to this "unreadable" data and fail to record all of the relevant events.

Bond's attack has the potential to defeat the dual controls on the CCA software within the 4758 and steal PIN derivation keys (or encryption keys for randomly chosen PINs), and this unauthorised activity would be likely to go unnoticed. But to manufacture false cards for use with stolen PINs, access to the mainframe database is needed to retrieve account information. If the attacker chooses the right access point, he could passively observe genuine database accesses, or could camouflage his requests by mixing them in with other traffic. It is definitely *possible* to collect this information stealthily, but there is always a risk that a particular sequence of events will be flagged as unusual and a detailed manual inspection triggered. The sooner the fraud is complete the better.

Similar time constraints apply to attacks on "Bills of Lading" systems, which are also protected by security modules. Here, the assets might be the multi-million pound cargo of an oil tanker that is in transit at sea for a month. If a corrupt insider can defeat the security module and sell the same oil twice, he will want the maximum possible time before the deception is detected.

Thus, in all attack scenarios the risk of early detection and the weight of evidence remaining must be assessed. The attacker needs to buy as much time as possible in which to launder money and assume a new identity.

Unfortunately, naïve application of Bond's "meet-in-the-middle" approach to key extraction from the CCA does not make for a promising attack. It requires multiple DES keys to be discovered, with each discovery providing data for the next stage of the attack. The source data has to be collected in three separate sessions of unauthorised access to the security module, with the cracking intervals between sessions lasting from a week to a month, depending upon the computing power available to the attacker. This exposes the attacker to considerable risk of detection and if one of the earlier sessions triggers an investigation it gives the authorities the opportunity to catch him "red-handed".

We therefore set out to optimise the key extraction attack with two main goals in mind:

- Collect all the data required to complete the attack in one session lasting under half an hour – fast enough to perform during a lunch-break.
- Minimise the number of meet-in-the-middle attacks required, and implement the brute-force search cheaply and quickly.

4 Optimisation of the Attack Code

4.1 The Original Attack

Straightforward assembly of Bond's building blocks results in a three-stage attack:

(1) Test Pattern Generation: Discover a normal data encryption key to use as a test pattern for attacking an exporter key. This is necessary because exporter keys are only permitted to encrypt other keys, not chosen values. The method is to encrypt a test pattern of binary zeroes using a set of randomly generated data keys, and then to use the meet-in-the-middle attack to discover the value of one of these data keys.

(2) Exporter Key Harvesting: Use the known data key from stage (1) as a test pattern to generate a second set of test vectors for a meet-in-the-middle attack that reveals two *double-length replicate exporter keys* (replicate keys have both halves the same, thus acting like single DES keys). Once this stage is complete, the values of two of the keys in the set will be known.

(3) Valuable Data Export: Retrieve the valuable key material (e.g. PIN derivation keys). This requires a known double-length exporter key, as the CCA will not export a 3DES key encrypted under a single DES exporter key, for obvious security reasons. Here, the key-binding flaw in the CCA software is used to swap the halves of two known replicate keys from stage (2) in order to make a double-length key with unique halves. This full 3DES key can then be used for the export process.

4.2 The Optimised Attack

In order to perform the attack in a single access session, the second set of test vectors had to be generated immediately after the first. However, it was not possible to know in advance which data key from the set would be discovered by the search, in order to use it as a test pattern. Generating a second set of test vectors for every possible data key would work in principle, but the number of operations the security module would have to perform would be exponentially increased, and at the maximum transaction rate (roughly 300 per second) would take ten days of unauthorised access.

So the first stage of the online attack had to yield the value of a particular data key that was chosen in advance, which could then be used as the test pattern for the second stage. The solution was to create a "related key set" using the `Key_Part_Import` command as described in Bond's paper. From the discovery of any single key, the values of all of the rest can be calculated. This related key set was made by generating an unknown data key part and XORing it with 2^{14} different known values (the integers $0 \ldots 16383$ were used). Any one of the keys

could then immediately be used for the second stage of the attack, even though its actual value would only be discovered later on.

The second stage was to export this single data key under a set of double-length replicate exporter keys and to use a meet-in-the-middle attack on the results. Two keys needed to be discovered so that their halves could be swapped to create a non-replicate exporter key. Once again the same problem arose in that it would be impossible to tell in advance which two keys would be discovered, and so the valuable key material could not be exported until after the cracking was complete. Generating a set of related exporter keys again solved the problem. Discovering just one replicate key now gave access to the *entire* set. Thus a double-length exporter with unique halves could be produced prior to the cracking activity by swapping the halves of any two of the related keys.

Implementation of this second stage of the process revealed an interesting and well-hidden flaw in the `Key_Part_Import` command. Although the concept of binding flaws had already been identified in the encrypted key tokens, it was also present in `Key_Part_Import`. It was possible to subvert the creation of a double-length replicate key so as to create a uniquely halved double-length key by the simple action of XORing in a new part with differing halves. This second instance of the flaw would have been missed had the theory not actually been implemented "for real". From the point of view of the system maintainer, this demonstrates the well-known principle that when generic weaknesses have been identified in an API, equally generic solutions should be sought, and patching individual parts of the transaction set is unlikely to solve all of the problems.

Finally, the new double-length exporter key made from the unknown replicate key part from stage two was used to export the valuable key material.

Although the attack still has three conceptual stages, there is no dependency on knowing the actual values of keys during the period of access to the 4758, so the stages can be run in a single session and the cracking effort done in retrospect.

5 Optimising the Key Search with an FPGA

5.1 Cracking Performance

Bond's paper proposed using a home PC for the DES key cracking, reflecting the resources available to a real-world attacker. However, experimentally cracking a single key showed that a typical 800 MHz machine would take about 20 days to crack one key out of a set of 2^{16}, this being the maximum number of encrypted results that it is realistic to consider producing during a "lunch-break-long" period of access to the CCA software. The cost of getting "no questions asked" access to multiple PCs in parallel is substantial, so a faster method was desirable in order to reduce the risk of the bank spotting the unauthorised access to the 4758 before the attack was complete.

DES was designed to work well with the integrated circuits of the mid-1970s and it has proved to be difficult to create high-speed software implementations

on contemporary processor architectures. Hardware solutions are known to be many orders of magnitude faster than software crackers running on general purpose PCs. We therefore investigated the capabilities of modern FPGA systems. High-level hardware design languages such as Verilog allow them to be programmed by relative amateurs, so this was not stepping outside of the attack scenario. We became particularly interested in Altera's "Excalibur" NIOS evaluation board [1], which is an off-the-shelf, ready-to-run, no-soldering-required system that comes complete with all the tools necessary to develop systems such as a DES cracker. Altera's generosity meant that we got our board free; other attackers would be able to purchase it for US$995.

5.2 How the DES Cracker Works

The basic idea of a brute force "DES cracker" is to try all possible keys in turn and stop when one is found that will correctly decrypt a given value into its plaintext. Sometimes, the plaintext that is to be matched is known, as in this case, and sometimes the correct decryption can only be determined statistically or through an absence of unacceptable values (for example, in the RSA decryption challenges posed in the late 1990s [16], the decrypted output needed to resemble English text).

This cracker design actually works the other way round; it takes an initial plaintext value and encrypts it under incrementing key values until the encrypted output matches one of the values being sought. The design runs at 33.33 MHz, testing one key per clock cycle, which is rather slow for cracking DES keys – and it would take, with average luck, 34.6 years to crack a single key. However, the attack method allows many keys to be attacked in parallel and because they are interrelated it does not matter which one is discovered first.

The design was made capable of cracking up to 16384 keys in parallel (i.e. it simultaneously checks against the results of encrypting the plaintext with 16384 different DES keys). The particular Excalibur board being used imposed the 16384 limitation; if more memory had been available then the attack could have proceeded more quickly. The actual comparison was done in parallel by creating a simple hash of the encrypted values (by XORing together groups of 4 or 5 bits of the value) and then looking in that memory location to determine if an exact match had occurred. Clearly, there is a possibility that some of the encrypted values obtained from the 4758 would need to be stored in identical memory locations. We just discarded these clashes and collected rather more than 16384 values to ensure that the comparison memory would be reasonably full.

As already indicated, 69.2 years are necessary to try all possible keys and therefore guarantee a result. However, probabilistic estimates can be made of the likely running time. These estimates are valid because so many keys are being searched for, and because DES can be viewed as creating essentially random encrypted values (approximating a random function being a property of good crypto algorithms). Over a full search, the average time to find the next key can be calculated, by simple division, to be about 37 hours. However, it is more useful to consider the time to find the first key and model the system using a

Poisson distribution. The probability that the first r attempts will all fail is $e^{-\lambda r}$ where λ is the probability any given attempt matches, which if checking against 16384 keys will be: $2^{14}/2^{56} = 2^{-42}$. At 33.33 MHz with average luck ($p = 0.5$), the first key will be found within 25.4 hours. With bad luck ($p = 0.001$, i.e. all except one run in a thousand) the first key will be found within 10.5 days.

As already indicated, the attack requires two cracking runs, so one would hope to complete it in just over 2 days. In practice, the various keys we searched for were found in runs taking between 5 and 37 hours, which is well in accordance with prediction.

5.3 Implementation Overview

The DES cracker was implemented on the Altera Excalibur NIOS Development board [1]. This board contains an APEX EP20K200EFC484-2X FPGA chip which contains 8320 Lookup Tables (LUTs) – equivalent to approximately 200,000 logic gates. The FPGA was programmed with a DES cracking design written in Verilog alongside of which, within the FPGA, was placed a 16-bit NIOS processor. The NIOS is an Altera developed RISC design which can be easily integrated with custom circuitry. The NIOS processor runs a simple program (written in GNU C and loaded into some local RAM on the FPGA) which looks after a serial link. The test vectors for the DES crack are loaded into the comparison memory via the serial link, and when cracking results are obtained they are returned over the same link. Although the NIOS could have been replaced by a purely hardware design, there was a considerable saving in complexity and development time by being able to use the pre-constructed building blocks of a processor, a UART and some interfacing PIOs. Fig. 1 shows the general arrangement:

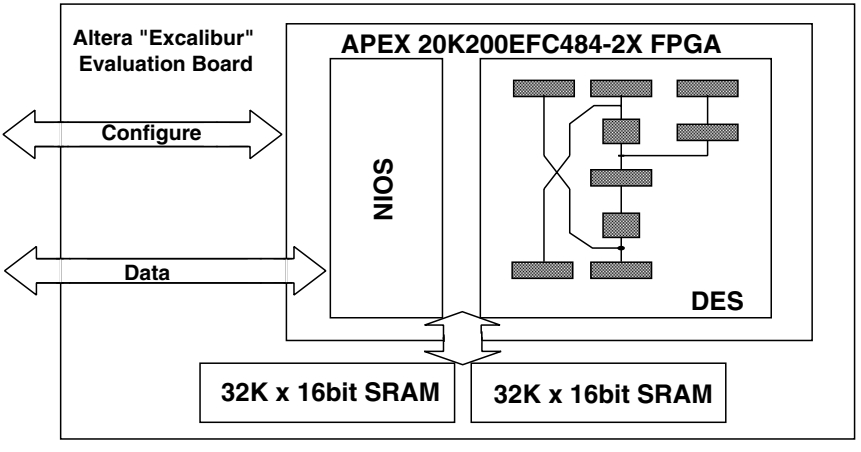


Fig. 1. DES cracker design

A pipelined version of the DES algorithm was used with the same input data value being encrypted by a succession of key values. At each clock interval, the intermediate left/right results of each DES stage are clocked into the next set of registers to act as inputs for the next stage of the encryption. Therefore, after an initial start-up period of 16 clocks, results appear from the end of pipeline at the clock rate of 33.33 MHz.

The key value must remain available for use by every stage of the algorithm, however we avoided the need to provide registers to pipeline its value from stage to stage. Instead, we used a Linear Feedback Shift Register (LFSR), which has been extended beyond its 56-bit value so that as it shifts, the extra bits serve to keep a record of older values of the key. This extended register is then statically connected, in an appropriate manner, to provide the key for the various pipeline stages. This space-saving technique was previously used by Hamer and Chow in their *Transmogrifier* DES cracker design [8]. The use of the LFSR had the further benefit of searching key space in a pseudo-random manner, so the 4758 programs were able to use densely packed sets of key values.

A tedious complication was that the Altera board has a limited amount of RAM as standard, just two 32K × 16-bit SRAMs. These could be arranged to form a single 32K × 32-bit memory, but it was still necessary for the 64-bit comparison to be done in two halves. If the first half matches (as will happen every few seconds) then the pipeline must be suspended for a moment and the second half of the value checked.

This can be seen on the logic analyser picture in Fig. 2 below. The regular signal on the third trace is the clock. The second signal down shows a 32-bit match is occurring. This causes a STOP of the pipeline (top signal) and access to an odd numbered address value (bottom signal). The other signals are some of the data and address lines.

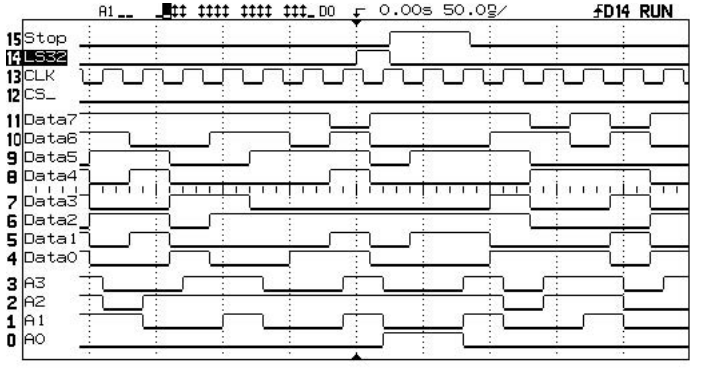


Fig. 2. The DES cracker actually running

5.4 Implementation of the DES S-Boxes

Although most of the cracker design was straightforward, the implementation of the DES S-boxes is of some interest. There are eight of these, each taking six bits of input and providing a 4-bit result. The S-boxes provide the non-linear component within the DES algorithm and they are defined in FIPS-46 [15] as tables of values that appear to be completely random.

The simplest way to implement the S-boxes would be as 128 read-only memories (8 for each of the 16 pipeline stages). Unfortunately, although the particular Altera FPGA architecture being used can be programmed to provide ROMs, only 52 were available on the particular chip being used. Therefore, the S-boxes had to be created from logic components. Hamer and Chow [8] (who used the same FPGA architecture) observed that one could create the 6-bit LUTs needed for the S-box bits from six 4-bit LUTs as shown in Fig. 3:

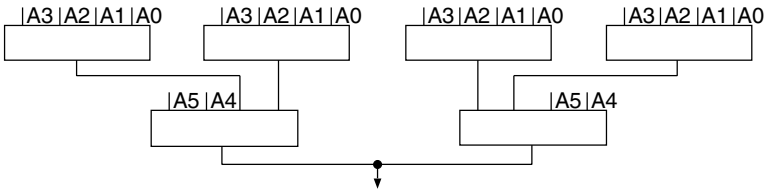


Fig. 3. Using six 4-input LUTs to create one DES S-box bit

The final OR of the results is achieved "for free" by the FPGA circuitry. This gives a LUT count for each S-box of 24, giving an overall usage of 3072 LUTs for the whole design. This is over a third of the entire chip (8320 LUTs).

Because multi-level logic minimisation is complex, some optimal solutions may be missed. In order to ensure that the logic synthesis program would use the Hamer/Chow scheme, we wrote the Verilog for the S-boxes as follows:

```
wire [5:0] A = {address[5], address[0], address[4:1]};
reg [3:0] row0, row1, row2, row3;
always @(A)
begin
    case (A[3:0])
         0: begin row0 = 14; row1 = 0; row2 = 4; row3 = 15; end
       ... etc etc
        15: begin row0 =  7; row1 = 8; row2 = 0; row3 = 13; end
    endcase
    case ({A[5],A[4]})
         0: result = row0;
         1: result = row1;
         2: result = row2;
         3: result = row3;
    endcase
end
```

However, when making S-box 4, the logic minimisation process managed to save a couple of LUTs. This was of considerable interest because the design was clearly going to be quite a tight fit into the FPGA, so it was investigated further.

As can be seen by inspecting the code, the use of A0...A3 in the first stage and A4...A5 in the second stage is essentially arbitrary. The same result is obtained using another selection of 4 and then 2 bits by suitable alteration of the A[i] in the case statements. All of the 720 possible arrangements were tried, for each of the eight S-boxes. The result was that several other S-boxes were found to exhibit small amounts of non-randomness:

S-box	1	2	3	4	5	6	7	8
LUTs	24	**23**	**23**	**16**	24	24	**23**	**22**

Thus at each pipeline stage, 13 LUTs can be saved (almost 7% of the total).

This was not an entirely surprising result, although this is a new way of finding it. It has long been known that the DES S-boxes do have some internal structure [9,17,3] and in the 1970s this led some people to conclude that there were back doors into the DES system, especially since the NSA were said to have been involved in the S-box designs.

5.5 Pipelining vs. Looping Designs

Key-cracking machines can be constructed at two extremes of system architecture. The cryptographic primitive can be arranged in a loop with a counter, as would be usual in a software implementation, or the loop can be "unrolled" to create a pipelined design. Between these two extremes, one can create hybrids where a few stages are pipelined and a lower maximum value of loop counter is used. No matter what the architecture, provided there is room within the FPGA, it is possible to add further instances of the basic design in parallel so as to provide a performance increase. It is interesting to consider which of these architectures is in fact optimal.

Experiments showed that a "loop" architecture duplicated 16 times, along with the logic to select results from each loop unit in turn, occupied 11,662 LUTs. In contrast, a fully pipelined architecture occupied 8,303 LUTs. Exactly the same performance might be expected from both designs; there is a start-up delay of 16 clocks and then they deliver one encrypted value on every subsequent clock. Therefore, it might seem that the pipeline design is to be preferred.

However, if one's FPGA is not large enough to hold the design (and it appears to be a fundamental rule of systems design that one inevitably runs out of gates or pins) then the pipelined architecture will not be implementable since it is "all or nothing". In contrast one can remove loop units to produce a cut-down design that delivers, for example, 10 results per 16 clocks. The saving is approximately 540 LUTs per loop unit (logic minimisation effects mean that the exact saving can vary). This might make the loop architecture preferable.

Historically, designs were always of the loop variety [5,4] because until relatively recently it was an achievement to get the whole of a single loop unit into

a chip. By 1993, chips were larger and Wiener [18] proposed a pipelined design. However, although he used an LFSR to avoid the difficulty of a "ripple carry" across a 56-bit counter, he did not use the Hamer/Chow insight into how this could be used to avoid pipelining the key values. Kaps and Paar investigated several different FPGA designs [14], though their interest was in determining how pipelining or partial loop unrolling affected maximum clock speed. In the current work, the limiting speed was the external SRAM, so there was no benefit in making the cracker design run faster since all of the designs generated results faster than they could be compared against the set of encrypted values.

Speed and size are not the only constraints. The first machine to be actually constructed, the Electronic Freedom Foundation's (EFF) design [6], used a multiple loop unit design. A pipelined scheme was considered, but was rejected as being more complex and hence more risky for a project that needed to work first time [7].

One must conclude that there is no easy solution here. The optimal design approach seems to be to try the pipelined design first. If that does not fit into a particular chip then it will be necessary to discard the work done thus far and create a design that crams in as many loop units as possible.

6 Results

Although many paper designs for DES cracking machines were proposed in the 1970s [5], 1980s [4] and 1990s [18], no publicly known machines were actually constructed until the Electronic Freedom Foundation built *Deep Crack* in 1998 [6]. This was an ASIC gate-array design, since this was the cheapest way of building it. Other work has been done before and since on FPGA based cracker designs such as [14,8] and most of these designs appear to have been synthesised and tested. However, the current work appears to be the first FPGA DES cracker design in the open literature (and only the second actual system after the EFF machine) that has actually found a key "in anger". Of course this achievement could only be done so quickly and for such a low cost because of the "meet-in-the-middle" nature of the problem we tackled.

The full attack described in this paper was run on two occasions in 2001 at the full rate of 33.33 MHz (approx. 2^{25} keys/second). In both cases the expected running time of 50.8 hours (based on average luck in locating a key) was comfortably beaten and so it would have been possible to start using the PIN derivation keys well before unauthorised access to the 4758 would have been detected.

Date	Start	Finish	Duration	Key value found
Aug 31	19:35	17:47	22 h 12 min	`#3E0C7010C60C9EE8`
Sep 1	18:11	23:08	4 h 57 min	`#5E6696F6B4F28A3A`

Oct 9	17:01	11:13	19 h 12 min	`#3EEA4C4CC78A460E`
Oct 10	18:17	06:54	12 h 37 min	`#B357466EDF7C1C0B`

We communicated our results to IBM. In early November 2001 they issued a warning to CCA users [11] cautioning them against enabling various functionality that the attacks depended upon. In February 2002 they issued a new version of the CCA software [12] with some substantial amendments that addressed all the issues raised by our attacks and those discussed by Bond in his earlier paper.

7 Conclusions

We have shown that the practical implementation of a theoretical attack is a worthwhile activity. Our research revealed aspects of both the system attacked and the attack method itself that would have been difficult to spot in any other way.

At the hardware design level we showed that pipelined implementations of DES could be made considerably smaller than designs using multiple looping units. We also found a new way of demonstrating that the DES S-boxes are not as random as they might at first appear.

At the conceptual level, we discovered a second specific instance of the generic key-binding flaw discussed in Bond's original paper. This highlights the risks of patching individual parts of a system to deal with security problems. Generic solutions must be sought for generic problems.

The specification-level faults that have been exploited in this attack have turned out to be just part of the story. Although we devoted some of our effort into reducing the effective strength of the CCA's 3DES implementation to that of single DES, IBM's analysis of our attack uncovered an implementation-level fault that made this whole stage unnecessary [13]. The CCA code was failing to prevent export of a double-length key under a double-length replicate key, despite the specifications stating that this would not be allowed.

In the future we must expect to see attacks that combine exploitation of both specification mistakes and faults in implementing the specification. It is hard to see how existing analysis practices at either the specification or the implementation level can hope to spot this type of hybrid. Making serious attempts to actually implement otherwise theoretical attacks may be our only handle on this problem.

Acknowledgements. We would like to acknowledge the generosity of Altera in providing the FPGA board used in this project for free. We'd also like to thank Ross Anderson and Simon Moore for their helpful comments and encouragement throughout. Mike Bond was able to conduct the research thanks to the funding received from the UK Engineering and Physical Research Council (EPSRC) and Marconi plc.

References

1. Altera Inc.: Excalibur Development Kit, featuring NIOS.
`http://www.altera.com/products/devkits/altera/kit-nios.html`

2. M. Bond: Attacks on Cryptoprocessor Transaction Sets. Proc. Workshop Cryptographic Hardware and Embedded Systems (CHES 2001), LNCS 2162, Springer-Verlag, pp 220–234 (2001)
3. E. F. Brickell, J. H. Moore and M. R. Purtill: Structure in the S-boxes of the DES (extended abstract). In A. M. Odlyzko (ed.), Advances in Cryptology – CRYPTO'86, LNCS 263, Springer-Verlag, pp 3–8 (1987)
4. M. Davio, Y. Desmedt, J. Goubert, F. Hoornaert and J. Quisquater: Efficient hardware and software implementations for the DES. In G. R. Blakley and D. Chaum (ed.), Advances in Cryptology – CRYPTO'84, LNCS 196, Springer-Verlag, pp 144–146 (1985)
5. W. Diffie and M. E. Hellman: Exhaustive Cryptanalysis of the NBS Data Encryption Standard. IEEE Computer 10(6), pp 74–84 (1977)
6. Electronic Frontier Foundation: Cracking DES : Secrets of Encryption Research, Wiretap Politics & Chip Design. O'Reilly. (May 1998)
7. J. Gilmore: Personal communication. (17 Nov 2001)
8. I. Hamer and P. Chow: DES Cracking on the Transmogrifier 2a. Cryptographic Hardware and Embedded Systems, LNCS 1717, Springer-Verlag, pp 13–24 (1999)
9. M. E. Hellman, R. Merkle, R. Schroppel, L. Washington, W. Diffie, S. Pohlig and P. Schweitzer: Results of an Initial Attempt to Cryptanalyze the NBS Data Encryption Standard. Information Systems Laboratory SEL 76-042, Stanford University (Sep 9 1976)
10. IBM Inc.: IBM 4758 PCI Cryptographic Coprocessor CCA Basic Services Reference and Guide for the IBM 4758-001, Release 1.31. IBM, Armonk, N.Y. (1999) `ftp://www6.software.ibm.com/software/cryptocards/bscsvc02.pdf`
11. IBM Inc.: Update on CCA DES Key-Management. (Nov 2001) `http://www-3.ibm.com/security/cryptocards/html/ccaupdate.shtml`
12. IBM Inc.: CCA Version 2.41. (5 Feb 2002) `http://www-3.ibm.com/security/cryptocards/html/release241.shtml`
13. IBM Inc.: Version history of CCA Version 2.41, IBM 4758 PCI Cryptographic Coprocessor CCA Basic Services Reference and Guide for the IBM 4758-002. IBM, pg xv (Feb 2002)
14. J. Kaps and C. Paar: Fast DES Implementation for FPGAs and its Application to a Universal Key-Search Machine. Selected Areas in Cryptography, pp 234–247 (1998)
15. National Bureau of Standards: Data Encryption Standard. Federal Information Processing Standard (FIPS), Publication 46, US Department of Commerce (Jan 1977)
16. RSA Security Inc.: Cryptographic Challenges. `http://www.rsasecurity.com/rsalabs/challenges/index.html`
17. A. Shamir: On the security of DES. In Hugh C. Williams (ed.), Advances in Cryptology – CRYPTO'85, LNCS 218, Springer-Verlag, pp 280–281 (1986)
18. M. Wiener: Efficient DES Key Search. TR-244, School of Computer Science, Carleton University (May 1994)

A Time-Memory Tradeoff Using Distinguished Points: New Analysis & FPGA Results

Francois-Xavier Standaert, Gael Rouvroy, Jean-Jacques Quisquater, and Jean-Didier Legat

UCL Crypto Group,
Laboratoire de Microelectronique, Universite Catholique de Louvain,
Place du Levant, 3, B-1348 Louvain-La-Neuve, Belgium
{standaert,rouvroy,quisquater,legat}@dice.ucl.ac.be

Abstract. In 1980, Martin Hellman [1] introduced the concept of cryptanalytic time-memory tradeoffs, which allows the cryptanalysis of any N key symmetric cryptosystem in $O(N^{\frac{2}{3}})$ operations with $O(N^{\frac{2}{3}})$ storage, provided a precomputation of $O(N)$ is performed beforehand. This procedure is well known but did not lead to realistic implementations. This paper considers a cryptanalytic time-memory tradeoff using distinguished points, a method referenced to Rivest [2]. The algorithm proposed decreases the expected number of memory accesses with sensible modifications of the other parameters and allows much more realistic implementations of fast key search machines. We present a detailed analysis of the algorithm and solve theoretical open problems of previous models. We also propose efficient mask functions in terms of hardware cost and probability of success. These results were experimentally confirmed and we used a purpose-built FPGA design to perform realistic tradeoffs against DES. The resulting online attack is feasible on a single PC and we recover a 40-bit key in about 10 seconds.

1 Introduction

Generally speaking, a block cipher allows to encrypt a n-bit text using a k-bit key to produce a n-bit ciphertext. Let $q = \lceil \frac{k}{n} \rceil$. If q plaintext/ciphertext pairs are known, with a high probability, the key can be determined by exhaustive key search, but it usually requires a too long processing time. Another possibility is a chosen plaintext attack using a precomputation table where an attacker precomputes the encryptions of q chosen plaintexts under all possible keys and stores the corresponding ciphertext/key pairs, but it usually requires a too large memory. The aim of a time-memory tradeoff is to mount an attack of which the online processing complexity is lower than an exhaustive key search and the memory complexity is lower than a precomputation table.
In [3,4,5], Borst et Al. propose a theoretical analysis of the time-memory tradeoff using distinguished points. They conclude that of theoretical interest remains the problem of determining two parameters: the expected number of chains and average chain length after sort. In this paper, we present a theoretical analysis

B.S. Kaliski Jr. et al. (Eds.): CHES 2002, LNCS 2523, pp. 593–609, 2003.

of a time-memory tradeoff using distinguished points and evaluate the different parameters introduced by this variant of Hellman's method. We discuss the complexity of the attack as well as the resulting probability of success, isolate the different phenomenons involved in a tradeoff using distinguished points and evaluate their practical influence. Precisely, we propose approximations to solve the open problem of [5] and correct the probability of success to correspond with practical implementations. The resulting analysis was confirmed by experimental results and allowed to mount realistic attacks against the block cipher DES.
The paper is organized as follows. In section 2, some basic schemes are given to help the understanding of the tradeoff. Formal definitions and algorithms are in section 3 and 4. Section 5 identifies the critical situations due to the use of distinguished points in the tradeoff. Section 6 proposes efficient mask functions in terms of success rate and hardware cost. The main contribution of this paper lies in section 7 and 8. We evaluate the different parameters of the tradeoff and compare the resulting theory with experimental results. Conclusions are in section 9.

2 Basic Scheme

The time-memory tradeoff method for breaking ciphers is composed of a precomputation task and an online attack. We briefly introduce these steps with two intuitive schemes:

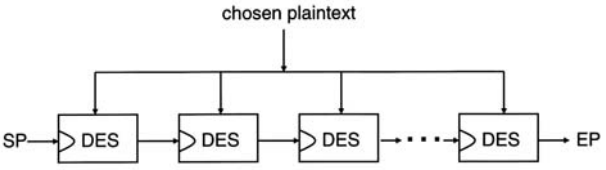


Fig. 1. Precomputation task

1. A chain is formed by a number l of encryptions using a chosen plaintext and l different keys. A defined property holds for the first and last keys and we call them distinguished points. During the precomputation, we compute a number of chains and store start points, end points and the corresponding chain length in a table.
2. Let the chosen plaintext be encrypted with a secret key. During the attack, we can use the resulting ciphertext as a key and start a chain until we find a distinguished point. Then, we check if this end point is in our table, take the corresponding start point and restart chaining until we find the ciphertext again. The secret key is its predecessor in the computed chain.

This basic scheme illustrates that the success rate of the attack depends on how well the computed chains "cover" the key space. In the next sections, we

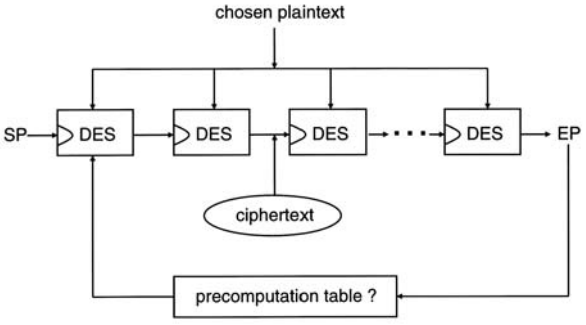


Fig. 2. Online attack

develop these two schemes and present effective algorithms for precomputation and online attack.

3 Definitions

Let E : $\{0,1\}^n \times \{0,1\}^k \rightarrow \{0,1\}^n$ be a block cipher with block length n and key length k. The encryption of one block is written as:

$$C = E_K(P) \tag{1}$$

Where $C \in \{0,1\}^n$, $K \in \{0,1\}^k$ and $P \in \{0,1\}^n$ denote the ciphertext, the secret key and the plaintext.
We define two functions. The first one just mixes its arguments and rejects z bits to reach the key size $k = n - z$.

$$g : \{0,1\}^n \rightarrow \{0,1\}^k. \tag{2}$$

We call g a mask function. There are many possibilities to define g. Earlier papers proposed to use permutations. In section 6, we suggest efficient mask functions in terms of implementation cost and probability of success.
We also define a function $f : \{0,1\}^k \rightarrow \{0,1\}^k$

$$f(K) = g(E_K(P)) \tag{3}$$

Finally, for a random start point $SP \in \{0,1\}^k$, we define a chain $K_0, K_1, K_2, ..., K_t$ of length t as

$$K_0 = SP \tag{4}$$

$$K_i = f(K_{i-1}) = f^i(K_0) \tag{5}$$

In the tradeoff, only the start point SP and end point $EP = K_t$ are stored.

Definition of a DP-property: Let $\{0,1\}^k$ be the key space and $d \in \{1,2,3,...,k-1\}$. Then DP-d is a DP-property of order d if there is an easily checked property which holds for 2^{k-d} different elements of $\{0,1\}^k$. In our application, having d bits locked to a fixed value, say 0, is a DP-property of order d.

Definition of a distinguished point: Let $K \in \{0,1\}^k$ and $d \in \{1,2,3,...,k-1\}$. Then K is a distinguished point (DP) of order d if the DP-property defined beforehand holds for K. Note that using this definition of distinguished point, we do not need to store the fixed bits and reduce the memory requirements of the tradeoff.

4 Algorithms

The algorithm proposed requires to choose a DP-property of order d and a maximum chain length t. We precompute r tables by choosing r different mask functions. For each mask function m different start points (which are distinguished) will be randomly chosen. For each start point a chain will be computed until a DP is encountered or until the chain length is $t+1$. Only start points iterating to a DP in less than t iterations will be stored with the corresponding chain length, the others will be discarded. Moreover, if the same DP is an end point for different chains, then only the chain of maximal length will be stored. This involves a lower memory complexity than Hellman's tradeoff.

Precomputation algorithm: Generate r tables with (SP,EP,l)-triples, sorted on EP.

1. Choose a DP-property of order d.
2. Choose r different mask functions g_i, $i = 1, 2, ..., r$. It defines r different f functions: $f_i = g_i(E_K(P))$, $i = 1, 2, ..., r$.
3. Choose the maximum chain length t.
4. For $i = 1$ to r
   a) Choose m random start points $SP_1^{(i)}, SP_2^{(i)}, ..., SP_m^{(i)}$.
   b) For $j = 1$ to m, $l = 1$ to t
      i. Compute $f_i^l(SP_j^{(i)})$.
      ii. If $f_i^l(SP_j^{(i)})$ is a DP then store the triple $(SP_j^{(i)}, EP_j^{(i)} = f_i^l(SP_j^{(i)}), l)$ and take next j.
      iii. If $l > t$ "forget" $SP_j^{(i)}$ and take next j.
   c) Sort triples on end points. If several end points are identical, only store the triple with the largest l.
   d) Store the maximum l for each table: l_{max}^i.

For the search algorithm, a table only has to be accessed when a DP is encountered during an iteration which allows efficient implementations of the online attack. Moreover, if the encountered DP is not in the table, then one will not find the target key by iterating further. Hence the current search can skip the rest of this table.

Search algorithm: Given $C = E_K(P)$ find K.

1. For $i = 1$ to r
   a) Look up l^i_{max}.
   b) $Y = g_i(C)$.
   c) For $j = 1$ to l^i_{max}
      i. If Y is a DP then
         A. If Y in table i, then
            – Take the corresponding $SP^{(i)}$ and length l in the table.
            – If $j < l$
              • Compute predecessor $\tilde{K} = f_i^{l-1-j}(SP_l^{(j)})$.
              • If $C = E_{\tilde{K}}(P)$ then $K = \tilde{K}$: STOP.
              • If $C \neq E_{\tilde{K}}(P)$, take next i.
         B. Else take next i.
      ii. Set $Y = f(Y)$.

5 Overlap Situations

In the tradeoff method described, one tries to store information about as many different keys as possible by taking as long chains as possible. Consequently, the very short chains increase the memory complexity of the tradeoff and could be rejected. However, different critical overlap situations can appear and add constraints to the tradeoff.

1. A chain can cycle. This is the case where we find i and j with $i \neq j$ and $K_{i+1} = K_{j+1}$.
2. Two chains computed with the same mask function can merge. This is the case where two different start points have the same image. This means that from the moment of the merge until the end of at least one chain, both chains contain the same keys.
3. A chain can collide with another chain computed with a different mask function. This is the situation where two chains computed with different mask functions have some common points between SP and EP. It means that some keys are stored several times, which is not efficient.

As suggested by the precomputation algorithm, we dealt with cycles by choosing an adequate maximum chain length t and the mergers were rejected after the precomputation by keeping the longest of two merging chains. Consequently, the effectiveness of the tradeoff highly depends on the choice of its parameters. In the section 7, we evaluate the different parameters of the tradeoff as well as the resulting complexity and probability of success of both precomputation and online attack.

6 Efficient Mask Functions

In previous papers about time-memory tradeoffs, mask functions were implemented as permutations. We propose efficient mask functions in terms of collisions and implementation cost.

Basically, the problem when using a permutation is the possibility that two mask functions have some common keys after a collision. For example, if the mask function corresponds to a permutation of two bits, the common length after a collision is obviously $\frac{1}{2} + \frac{1}{4} + \frac{1}{8} + ...$
Moreover, if a large number of mask functions is needed (which is practically probable), their implementation becomes fastidious (specially in hardware designs that are particularly efficient in time-memory tradeoffs). We suggest the following definition of efficient mask functions. Let $m_i(x)$ and $m_j(x)$ be two mask functions:

$$m_i(x) : \{0,1\}^n \rightarrow \{0,1\}^n$$
$$m_j(x) : \{0,1\}^n \rightarrow \{0,1\}^n$$

Let $n(x) = m_i(x) - m_j(x)$. Efficient mask functions are such that for every $i \neq j$, we have

$$Ker(n(x)) = \oslash^1. \tag{6}$$

We define mask functions as:

$$m_i(x) : \{0,1\}^n \rightarrow \{0,1\}^n : x \rightarrow m_i(x) = x \oplus i \tag{7}$$

Where $\oplus$ denotes the bitwise XOR operator. These mask functions fulfill condition 6 and are specially easy to implement in hardware or software.

7 Theoretical Analysis

As mentioned in section 5, the effectiveness of the tradeoff highly depends on the choice of its parameters. Consequently, their correct prediction before implementation is crucial. Although existing papers evaluate the processing complexity, memory complexity and success rate of the tradeoff and mention that length of chains, number of chains and number of mask functions play a crucial role in the performance of the tradeoff, there exist no precise indications about how to choose these parameters. Actually, only [3,4,5] focus on the distinguished points variant but their analysis is not complete and some problems remained open. In the next section, we propose a theoretical model for cryptanalytic time-memory tradeoffs using distinguished points that takes into account the new situations due to distinguished points. Approximations are proposed to solve the open problem of [5].

7.1 Probability to Reach a Distinguished Point

The main modification caused by the introduction of distinguished points is the variable chain length. Consequently, we computed the probability to reach a distinguished point in less than l iterations. In the following computation, we guess that the maximum chain length is such that no cycle could appear. Let

[1] $Ker(n(x)) = \{x \in \{0,1\}^n | n(x) = 0\}$

$P_1(l)$ be the probability that a DP is reached in less than ($\leq$) l iterations. Let $P_2(l)$ be the probability that no DP is reached in less than l iterations. We have $P_1 = 1 - P_2$ and we can easily compute $P_2(l)$:

$$P_2(l) = \prod_{i=0}^{l-1}(1 - \frac{2^{k-d}}{2^k - i}) \tag{8}$$

An approximate expression can be obtained knowing that $i \ll 2^k$, by fixing i to $\frac{l-1}{2}$:

$$P_2(l) \simeq (1 - \frac{2^{k-d}}{2^k - \frac{l-1}{2}})^l \tag{9}$$

Finally, we have

$$P_1(l) = 1 - \prod_{i=0}^{l-1}(1 - \frac{2^{k-d}}{2^k - i}) \tag{10}$$

$$P_1(l) \simeq 1 - (1 - \frac{2^{k-d}}{2^k - \frac{l-1}{2}})^l \tag{11}$$

Obviously, the probability to reach a distinguished point in exactly l iterations can be derived from $P_1(l)$ - $P_1(l-1)$ if we guess that $P_1(0) = 0$. Important information about the efficient areas of computation can be observed when representing the amount $l \times P$(A DP is reached in exactly l iterations) as shown by Figure 3 for the DP-10 property.

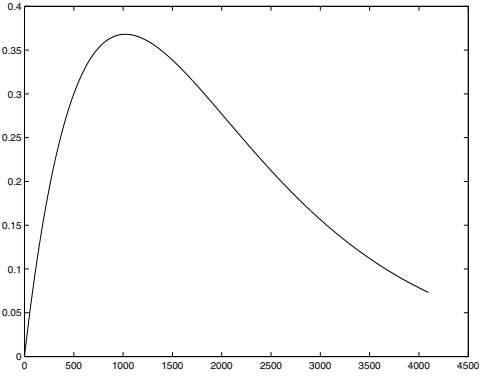


Fig. 3. X = l, Y=$l \times P$(A DP is reached in exactly l iterations)

7.2 Compute the Average Chain Length β

Figure 3 suggests that practical precomputations should possibly be performed for an efficient interval of chain lengths. Therefore, we evaluate the average chain length in a region between lengths t_{min} and t_{max}:

$$\beta = \frac{\sum_{l=t_{min}}^{t_{max}} l.P(DP.in.exactly.l.iterations)}{\sum_{l=t_{min}}^{t_{max}} P(DP.in.exactly.l.iterations)} \tag{12}$$

The denominator is easy to estimate and corresponds to the quotient between the number of chains included in region $[t_{min}, t_{max}]$ and the total number of chains. We call it cover and denote it γ:

$$\gamma = \sum_{l=t_{min}}^{t_{max}} P(DP.in.exactly.l.iterations) = P_1(t_{max}) - P_1(t_{min} - 1) \tag{13}$$

Numerator can be estimated with the next formula:

$$\begin{aligned} &\sum_{l=t_{min}}^{t_{max}} l.P(DP.in.exactly.l.iterations) \\ &= \sum_{l=t_{min}}^{t_{max}} l.(\prod_{i=0}^{l-2}(1 - \frac{2^{k-d}}{2^k - i}) - (\prod_{i=0}^{l-1}(1 - \frac{2^{k-d}}{2^k - i})) \\ &\simeq \sum_{l=t_{min}}^{t_{max}} l.((1 - \frac{2^{k-d}}{2^k - \frac{t}{2}})^{l-2} - (1 - \frac{2^{k-d}}{2^k - \frac{t}{2}})^{l-1}) \end{aligned} \tag{14}$$

Where $t = \frac{t_{max}+t_{min}}{2}$. We can rewrite equation 14 in a simpler form:

$$\sum_{l=t_{min}}^{t_{max}} l.((1-x)^{l-2} - (1-x)^{l-1}) \tag{15}$$

Where $x = \frac{2^{k-d}}{2^k - \frac{t}{2}}$. Hence

$$\begin{aligned} &\sum_{l=t_{min}}^{t_{max}} l.((1-x)^{l-2} - (1-x)^{l-1}) \\ &= t_{min}.(1-x)^{t_{min}-2} - t_{max}.(1-x)^{t_{max}-1} + \sum_{l=t_{min}-1}^{t_{max}-2} (1-x)^l \\ &= (1-x)^{t_{min}-2}.(t_{min} + \frac{1-x}{x}) - (1-x)^{t_{max}-1}.(t_{max} + \frac{1}{x}) \end{aligned} \tag{16}$$

Finally, the average chain length is:

$$\beta \simeq \frac{(1-x)^{t_{min}-2}.(t_{min} + \frac{1-x}{x}) - (1-x)^{t_{max}-1}.(t_{max} + \frac{1}{x})}{\gamma} \tag{17}$$

We designed some experiments in order to confirm this analysis and computed chains using a block cipher DES with a reduced key of 40 bits. First, we evaluated the influence of the DP-property:

DP-property	Length region (log_2)	Experimental β (log_2)	Theoretical β (log_2)
DP-11	9-13	11.2140	11.2140
DP-12	10-14	12.2137	12.2139
DP-13	12-14	13.0965	13.0966
DP-14	13-15	14.0967	14.0966
DP-15	11-18	15.0771	15.0836

Then, we observed the influence of the chain lengths:

DP-property	Length region (log_2)	Experimental β (log_2)	Theoretical β (log_2)
DP-13	10-13	11.9790	11.9987
DP-13	12-14	13.0965	13.0966
DP-13	13-16	14.0107	13.9955

7.3 Previous Proposals for the Success Rate

In this section, we consider the success rate when using one table generated with one mask function and denote it SR. The probability of success when using several mask functions is evaluated later and we denote it PS. For all the following computations, the function f is modelled as a random function mapping the key set onto itself if the key K is randomly chosen. Actually, earlier evaluations of the success rate, [1,7], do not consider the distinguished point variant and SR was always estimated in the following way: first the probability $P(K_{ij}$ is new) that a newly generated key K_{ij} is different from all keys generated previously is evaluated by:

$$P(K_{ij}.is.new) \geq (1 - \frac{it}{N})^{j+1} \tag{18}$$

Where i is the number of chains already computed, j the length of current chain, t the fixed chain length and $N = 2^k$ is the size of the key space. Then a lower bound of the success rate is evaluated:

$$SR \geq \frac{1}{N} \sum_{i=1}^{m} \sum_{j=1}^{t} (1 - \frac{it}{N})^{j+1} \tag{19}$$

Where m is the number of chains computed. Using $e^{-x} \simeq 1 - x$, we have the following approximation:

$$\begin{aligned} (1 - \frac{it}{N}) &\simeq e^{\frac{-it}{N}} \\ (1 - \frac{it}{N})^{j+1} &\simeq e^{\frac{-ijt}{N}} \end{aligned} \tag{20}$$

Equation 20 indicates that for a fixed value of N, there is not much to be gained by increasing m and t beyond the point at which $mt^2 = N$. Because when

$e^{\frac{-ijt}{N}} \simeq e^{\frac{-mt^2}{N}}$ and $mt^2 \gg N$, most terms will be small. Finally, we can evaluate the success rate:

$$\begin{aligned} SR &\geq \frac{1}{N} \sum_{i=1}^{m} \sum_{j=1}^{t} (1 - \frac{it}{N})^{j+1} \\ &\simeq \frac{1}{t} \sum_{i=1}^{m} \frac{1 - e^{\frac{-it^2}{N}}}{\frac{it}{N}} \frac{t}{N} \\ &\simeq \frac{1}{t} \int_0^{\frac{mt}{N}} \frac{1 - e^{-tx}}{x} dx \\ &\simeq h(u) \frac{mt}{N} \end{aligned} \tag{21}$$

Where $u = \frac{mt^2}{N}$ and $h(u) = \frac{1}{u} \int_0^u \frac{1-e^{-x}}{x} dx$. The function $h(u)$ denotes a lower bound of coverage. The next table evaluates $h(u)$ for different values of u and illustrates that $h(u)$ rapidly decreases after u exceeds 1.

u	$h(u)$	u	$h(u)$	u	$h(u)$
2^{-4}	0.99	2^{-1}	0.89	2^2	0.49
2^{-3}	0.97	2^0	0.80	2^3	0.33
2^{-2}	0.94	2^1	0.66	2^4	0.21

However, in case of a tradeoff using distinguished points, equation 19 is not correct anymore when the merger problem appears. Indeed, the keys are stored in terms of a number of chains and the relevant probability is the probability to find a new chain, not a new key. The practical consequence is that the success rate (21) does not correctly take the mergers into account. This point was neglected in [5] and in the next section, we propose other approximations and show that the possibility to carry on computing chains after $mt^2 = N$ has to be considered if the objective is to get the fastest online attack.

7.4 A Prediction of the Mergers

Let $s(j)$ be a storage function denoting the number of keys stored after sort and rejection of mergers and $j = \gamma m$ be the number of chains computed in region $[t_{min}, t_{max}]$ (γ is the cover defined in section 7.2 and m is the number of start points considered). Let $p(j)$ be the probability that a new chain is found after a storage $s(j)$. We have:

$$s(j) = s(j-1) + \beta \times p(j-1) \tag{22}$$

$$p(j) = \prod_{l=0}^{\beta-1} \frac{2^k - s(j) - l}{2^k} \tag{23}$$

Because $l \ll 2^k$, e can reduce this system to a non-linear difference equation:

$$s(j+1) = s(j) + \beta \times \prod_{l=0}^{\beta-1} 1 - \frac{s(j)}{2^k} - \frac{l}{2^k}$$

$$s(j+1) \simeq s(j) + \beta \times (1 - \frac{s(j)}{2^k})^\beta \tag{24}$$

First proposal for $s(j)$: We computed a lower bound for $s(j)$ by using the linear approximation: $(1 - \frac{s(j)}{2^k})^\beta = (1 - \beta \times \frac{s(j)}{2^k} + O(s(j)^2))$ and solving the resulting linear recurrence:

$$s(j+1) \simeq (1 - \frac{\beta^2}{2^k}) \times s(j) + \beta \tag{25}$$

Which has solution:

$$s(j) \simeq (s(0) - \frac{2^k}{\beta}) \times (1 - \frac{\beta^2}{2^k})^j + \frac{2^k}{\beta} \tag{26}$$

The function $(1 - \frac{s(j)}{2^k})^\beta$ represents the probability that a new chain is found. If we define the saturation as the moment when we have $s(j+1) = s(j)$ corresponding to a probability zero that a new chain is found, Figure 4 illustrates that our linear approximation involves a too fast saturation. As a consequence, we can derive a lower bound for the success rate: $SR \leq \frac{s(\gamma m)}{2^k}$.

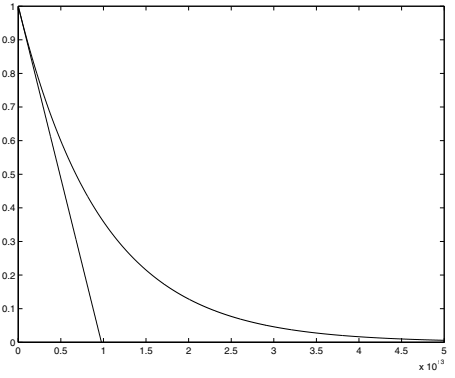


Fig. 4. Linear approximation of $(1-x)^k$ with $k = 2^{10}$

Comparing this result with precedent conclusions about the success rate, we observe that the amount $s(\gamma m)$ is defined as a number of keys stored and β is the average chain length. This means that $\frac{s(\gamma m)}{\beta}$ is a number of chains like m. Therefore the condition suggested by Hellman: $mt^2 = N$ is similar to the condition of

saturation when considering a linear approximation of the probability to find a new chain: $s(\gamma m) = \frac{2^k}{\beta}$ is equivalent to $\frac{s(\gamma m)}{\beta} = \frac{2^k}{\beta^2}$. This last point suggests that the decision to stop computations as soon as $mt^2 = N$ is not always optimal. For example if the objective is to get the fastest online attack, we will try to minimize the number of mask functions which involves the largest number of chains stored for every mask function.

Second proposal for $s(j)$: An improved approximation of the storage function is based on the convergence of Euler's methods described in [8]. The following equation:

$$\frac{s_{n+1} - s_n}{1} = \beta \times (1 - \frac{s_n}{2^k})^\beta \tag{27}$$

Can be approximated by

$$s'(j) = f(s, j) \tag{28}$$

Where $f(s,j) = \beta \times (1 - \frac{s(j)}{2^k})^\beta$ if $|\frac{df}{ds}| < L$ and $|s''(j)| < Y$ (L and Y are fixed constants). Because $s(j)$ is a storage function, it is obvious that $s''(j) < 0$ and $lim(s''(j)) = 0$. As $\frac{df}{ds} = \frac{-\beta^2}{2^k}(1 - \frac{s}{2^k})^{\beta-1}$ and $s \in [0, 2^k]$, the second condition is also fulfilled. Therefore, we solved the following differential equation:

$$\frac{s'(j)}{\beta} = (1 - \frac{s(j)}{2^k})^\beta \tag{29}$$

Using $t(j) := (1 - \frac{s(j)}{2^k})$ and $t'(j) := \frac{-s'(j)}{2^k}$, equation 29 is equivalent to:

$$\frac{-2^k}{\beta} \times t'(j) = t(j)^\beta \tag{30}$$

Which has an exact solution. Finally, we found

$$s(j) = 2^k \times \left(1 - (\frac{2^k}{-\beta j + \beta^2 j + K})^{\frac{1}{\beta - 1}}\right) \tag{31}$$

With $K = 2^k \times (1 - \frac{s(0)}{2^k})^{1-\beta}$ and the discretization error is estimated by:

$$e_{n+1} - e_n = \frac{df}{ds}(j, s) \times e_n + \frac{1}{2}s''(j) \tag{32}$$

These computations are in accordance with experimental results presented in section 8. We can conclude that:

1. The success rate evaluated in precedent papers is not directly applicable to tradeoffs using distinguished points. Mergers were not correctly taken into account.
2. The decision to stop precomputations at $mt^2 = N$ is not always optimal, depending on the objective: optimal precomputation time or optimal attack time.

7.5 A Prediction of the Average Chain Length after Sort

An important consequence of the mergers is the possible modification of the average chain length after sort. Intuitively, this modification depends on the number of chains rejected and the choice of t_{min} and t_{max}. Looking at equation 23, we can observe that the probability that a chain merges is increasing with the length of this chain. Consequently, the average length of rejected chains is larger than the initial average chain length predicted by equation 17.
Practically, the rejection process is such that if two chains merge, only the longest one is stored. Therefore, a possible prediction of the modification of the average chain length could be achieved by computing the mergers prediction for every possible chain length, using initial conditions for the storage:

$$\beta_{mod} = \frac{\sum_{t_{min}}^{t_{max}} l.n_l}{\sum_{t_{min}}^{t_{max}} n_l} \tag{33}$$

Where n_l denotes the number of chains of length l after sort and is evaluated using the storage function of section 7.4. As the evaluation of $t_{max} - t_{min}$ difference equations is fastidious, we can divide the chain lengths in a number of sets. Let m be the number of chains computed, with average length β and cover γ. Let p be the number of sets used to evaluate β_{mod} (we chose $p = 4$). According to section 7.2, we evaluate β and γ for each set and denote them β_i and γ_i (good approximations need $\gamma_i \simeq \gamma_j$ for every i, j). If m is the total number of chains computed, the number of chains computed in each region, say N_i is:

$$N_i = m \times \gamma_i \tag{34}$$

Then we solve for $i = p$ to 1:

$$s_i(j+1) \simeq s_i(j) + \beta_i \times (1 - \frac{s_i(j)}{2^k})^{\beta_i} \tag{35}$$

With the initial condition $s_p(0) = 0$ and $s_{i-1}(0) = s_i(N_i)$. From equation (26) or (31), we can derive the quantities $s_i(N_i)$ and $s_1(N_1)$ denotes the final number of keys stored. Finally, we compute the number of chains in each region: $c_i(N_i) = \frac{s_i(N_i) - s_i(0)}{\beta_i}$ and approximate the modified average length of chains as:

$$\beta_{mod} \simeq \frac{\sum_{i=1}^{p} c_i(N_i) \times \beta_i}{\sum_{i=1}^{p} c_i(N_i)} \tag{36}$$

7.6 Prediction of Collisions and Final Probability of Success

According to previous sections, the expected success rate using only one mask function is:

$$SR \simeq \frac{s(\gamma m)}{2^k} \tag{37}$$

If we use r different mask functions, the resulting probability of success is:

$$PS(r) = 1 - (1 - SR)^r \tag{38}$$

7.7 Memory Complexity

Storage is needed for r tables. Each table contains about $\frac{s(\gamma m)}{\beta_{mod}}$ chains. These are in the form of triples and if we denote by e the actual size of an entry in the table, using the result of the previous section for r, the memory complexity C_{mem} can be expressed as:

$$C_{mem} \simeq e \times \frac{s(\gamma m)}{\beta_{mod}} \times r \tag{39}$$

7.8 Precomputation Complexity

During the precomputation, we iterate for each point until either a distinguished point is reached or t_{max} iterations have been made. If a distinguished point is reached, then on average β iterations are computed. If a distinguished point is not reached, then t_{max} iterations are computed. We define the expected number of iterations for one chain as δ:

$$\delta = t_{max} \times (1 - P_1(t_{max})) + \beta \times P_1(t_{max}) \tag{40}$$

As iterations are made on $r \times m$ start points, the precomputation complexity C_{prec} can be estimated by:

$$C_{prec} \simeq r \times m \times \delta \tag{41}$$

7.9 Processing Complexity

In a tradeoff using distinguished points, a table only has to be accessed when a DP is encountered during an iteration which means that for every mask function, only one chain has to be computed. Moreover, if the encountered DP is not in the table, then one will not find the target key by iterating further. Therefore, if r is the number of tables generated and β_{mod} is the average length of chains after sort, the processing complexity C_{proc} can be lower bounded by:

$$C_{proc} \leq r \times \beta_{mod} \tag{42}$$

Note that the possible reduction of the average chain length improves C_{proc} and the resulting online attack is faster. This last point strengthens the assumption that, for a fixed DP-property, computations beyond $mt^2 = N$ should be considered if the fastest online attack is to be reached.

8 Practical Experiments

To confirm our analysis, we compare some theoretical predictions with experimental results. All our precomputations were carried out on a VIRTEX1000 FPGA board developed by the Microelectronics Laboratory at UCL. The board is composed of a control FPGA (FLEX10K) and a VIRTEX1000 FPGA associated with several processors (ARM, PIC) and fast access memories. The description of the hardware/software co-design used to perform the tradeoff and the complete experimental results can be found in [6].

Practically, we implemented two tradeoffs:

1. A first one against DES with a 40-bit key where we optimized the online attack. In this way, we illustrated a situation where condition $mt^2 = N$ is not optimal. The 40-bit DES is obtained from DES by fixing 16 key bits to arbitrary values.
2. A second one against DES with a 56-bit key where the precomputation task was critical. Therefore, we only implemented one mask function in order to evaluate the mergers and average chain length.

Both experiments confirmed our theoretical estimations. It is important to notice that experimental storage values were counted in terms of chains and therefore $\frac{s(\gamma m)}{\beta_{mod}}$ is the most significant data to compare. β and β_{mod} were evaluated on a sample of the results.

8.1 DES-40

Precomputation task: Table 1 summarizes our experimental results and theoretical predictions (in a log_2 scale) with a DP-property DP-11 and chain lengths $\in [2^9 - 2^{13}]$. It confirms our analysis to be a correct prediction of the tradeoff.

Table 1. Experimental results - Theoretical predictions

m	β	β_{mod}	$s(\gamma m)$	$\frac{s(\gamma m)}{\beta_{mod}}$
23.4219	11.2140	10.9721	30.4181	19.4460

m	β	β_{mod}	$s(\gamma m)$	$\frac{s(\gamma m)}{\beta_{mod}}$
21	11.2140	11.0400	29.7609	18.7209
22	11.2140	10.9653	30.0504	19.0851
23	11.2140	10.8847	30.2713	19.3866
24	11.2140	10.8040	30.4447	19.6407

Remark that $mt^2 = N$ would mean to limit the precomputations to $m = 2^{17.5738}$ which would not lead to an optimal online attack. As the precomputation complexity was easily reachable using FPGA's, we maximized the storage $s(\gamma m)$. Moreover the diminution of the average chain length also improves the online attack efficiency.

Online attack: As the success rate of one single table is $\frac{2^{30.4181}}{2^{40}}$, we can derive the final probability of success in terms of a number of mask functions r. Experimentally, we used 2^{10} mask functions and observed a probability of success of 72% which is to compare with the 73.74% theoretically predicted. The online attack[2] was performed on a single PC[3]. Thanks to the optimized C_{proc}, we recovered a key in about 10 seconds. An exhaustive key search on the same PC would have taken about 50 days.

[2] First presented at the rump session of CRYPTO2001
[3] 18Gbytes ROM/256Mbytes RAM/350MHz

8.2 DES-56

Precomputation task: For this experiment, we experimentally observed the storage function $s(\gamma m)$ at different levels of precomputation. Table 2 summarizes our experimental results and theoretical experiments (in a log_2 scale) with a DP-property DP-18 and chain lengths $\in [2^0 - 2^{30}]$. Due to the critical pre-

Table 2. Experimental results - Theoretical predictions

m	β	β_{mod}	$s(\gamma m)$	$\frac{s(\gamma m)}{\beta_{mod}}$
20	18	17.8284	37.3781	19.5497
21	18	17.8261	38.1309	20.3048
22	18	17.6150	38.6140	20.9990
23	18	17.2983	38.9408	21.6425

m	β	β_{mod}	$s(m)$	$\frac{s(\gamma m)}{\beta_{mod}}$
20	18	17.8310	37.3022	19.4712
21	18	17.7242	37.8599	20.1357
22	18	17.5819	38.2685	20.6866
23	18	17.4104	38.5461	21.1357

computation task, this experiment was not likely to be optimized in terms of processing complexity. Anyway, it confirms our analysis to be a correct prediction of the tradeoff. Online attacks against DES-56 offer a variety of compromises between time and memory. The average chain lengths and number of mask functions needed to reach high success rates ($\beta = 2^{18}$ and $r \simeq 2^{18}$ in our example) will make FPGA's the relevant tools to mount efficient online attacks. They offer the high encryption rates and reconfigurability needed for cryptanalytic applications.

9 Conclusion

Confirmed by experimental results, we propose a new theoretical analysis of cryptanalytic time-memory tradeoffs using distinguished points and underline particularities of this variant of Hellman's proposal. The model allows the prediction of both precomputation and online attack parameters: the complexity of the tradeoff is evaluated as well as its resulting probability of success. Predictions of earlier papers are modified in order to correctly take the mergers into account and we suggest situations where our theoretical predictions induce practical improvements.

We implemented the tradeoff against DES with a 40-bit key and recovered a key in 10 seconds, with a success rate of 72%, using one PC. The exhaustive search of the key on the same PC would have taken about 50 days. In parallel, practical experiments against DES with a 56-bit key confirmed the effectiveness of FPGA's in cryptanalytic applications. We used FPGA's to perform precomputation tasks but in case of tradeoffs implemented against ciphers with large keys, an FPGA-based implementation of the online attack would be very efficient and reasonably expensive compared with software ones.

References

1. M.Hellman, *A Cryptanalytic Time-Memory Tradeoff*, IEEE transactions on Information Theory, Vol 26, 1980, pp.401-406.
2. D.Denning, *Cryptography and Data Security*, p.100, Addison-Wesley, 1982, Out of Print.
3. J.Borst, B.Preneel and J.Vandewalle, *On the Time-Memory Tradeoff Between exhaustive key search and table precomputation*, Proc. of the 19th Symposium in Information Theory in the Benelux, WIC, 1998, pp.111-118.
4. J.Borst, B.Preneel and J.Vandewalle, *A Time-Memory Tradeoff using Distinguished Points*, Technical report ESAT-COSIC Report 98-1, Departement of Electrical Engineering, Katholieke Universiteit Leuven, 1998.
5. J.Borst, *Block Ciphers: Design, Analysis and Side-Channel Analysis*, Chapter 3: A Time-Memory Tradeoff attack, Phd Thesis, Departement of Electrical Engineering, Katholieke Universiteit Leuven, 2001.
6. J.J.Quisquater, F.X.Standaert, G.Rouvroy, J.P.David, J.D.Legat, *A Cryptanalytic Time-Memory Tradeoff: First FPGA Implementation*, To appear in the Proceedings of FPL 2002 (The Field Programmable Logic Conference) .
7. K.Kusuda and T.Matsumoto, *Optimization of Time-Memory Tradeoff Cryptanalysis and its Applications to DES, FEAL-32 and Skipjack*, IEICE Transactions on Fundamentals of Electronics, Communications and Computer Science, EP79-A, 1996, pp.35-48.
8. S.D.Conte, C.de Boor, *Elementary of Numerical Analysis*, Chapter 8.4, Mc Graw-Hill, 1981.
9. Amos Fiat and Moni Naor, *Rigorous Time/Space Tradeoffs for Inverting Functions*, SIAM J. Computing 29(3), 1999 pp. 790-803.