Universal Exponentiation Algorithm
– A First Step Towards *Provable* SPA-resistance –

[Published in Ç.K. Koç, D. Naccache, and C. Paar, Eds., *Cryptographic Hardware and Embedded Systems – CHES 2001*, vol. 2162 of *Lecture Notes in Computer Science*, pp. 300–308, Springer-Verlag, 2001.]

Christophe Clavier and Marc Joye

Gemplus Card International, Card Security Group
Parc d'Activités de Gémenos, B.P. 100, 13881 Gémenos, France
{`christophe.clavier, marc.joye`}`@gemplus.com`
`http://www.geocities.com/MarcJoye/`

Abstract. Very few countermeasures are known to protect an exponentiation against simple side-channel analyses. Moreover, all of them are heuristic.
This paper presents a universal exponentiation algorithm. By tying the exponent to a corresponding addition chain, our algorithm can virtually execute any exponentiation method.
Our aim is to transfer the security of the exponentiation method being implemented to the exponent itself. As a result, we hopefully tend to reconcile the provable security notions of modern cryptography with real-world implementations of exponentiation-based cryptosystems.

Keywords. Implementation, exponentiation, RSA cryptosystem, discrete logarithm, side-channel attacks, simple power analysis (SPA), provable security, addition chains, smart cards.

1 Introduction

The security of a cryptosystem is evaluated as the latter's ability to resist attacks in a given adversarial model. It is very challenging to guess the strategy the adversary will follow in an attempt to break the system. So, the only assumptions made by modern cryptography refer to the *computational abilities* of the adversary [6]. Loosely speaking, a cryptosystem is then said *secure* if there is no polynomial-time adversary able to gain more "useful" information than a honest user by deviating from the "prescribed" behavior.

In [9, 11], Kocher *et al.* launched a new class of attacks: the so-called *side-channel attacks*. In such a scenario, an adversary monitors some side-channel information (e.g., power consumption) during the execution of a crypto-algorithm and thereby may foil the security of the corresponding "provably secure" cryptosystem. So what does provable security mean? The security is usually proven by reduction: one shows that the only way to break the cryptosystem is to break the underlying cryptographic primitive (e.g., the RSA function). Since this is assumed to be computationally infeasible, the cryptosystem is declared secure. A side-channel attack does *not* violate this assumption, it just considers other directions to break the cryptographic primitive. Consequently, we stress that the notions of provable security, or more exactly *provable computational security*, are very useful and must be part of the analysis of any cryptosystem.

Unfortunately, there is no counterpart to side-channel attacks. Defining a security model for this class of attacks seems unrealistic since we do not see how to limit the power of the adversary. The best we can hope to prove is the security relative to one particular attack.

This paper focuses on modular exponentiation (e.g., the RSA function or the discrete logarithm function) as a cryptographic primitive. Using a representation with addition chains, we "transfer" the security of the exponentiation method actually implemented in the exponent itself (which is the secret data). The resulting algorithm, which we call *universal exponentiation algorithm*, works with virtually all exponentiation methods. It simply reads triplets of values $(\gamma(i) : \alpha(i), \beta(i))$, meaning that the content of register $R[\alpha(i)]$ must be multiplied by the content of register $R[\beta(i)]$ and that the result must be written into register $R[\gamma(i)]$. We provide in this way a kind of reduction. Instead of carefully analyzing a specific exponentiation method, the implementor simply verifies that the *atomic* operation $R[\gamma(i)] \leftarrow R[\alpha(i)] \cdot R[\beta(i)]$ does not leak any "useful" information through a given side-channel attack. This methodology is reminiscent of the traditional security proofs. In the traditional case, the security of a cryptographic primitive is conjectured (e.g., inverting the RSA function is infeasible) whereas in our case the security of an atomic operation is assessed through experiments (e.g., I cannot "break" a multiplication by SPA). The main difference is that the security assumption is scrutinized by fewer people and hence is more controversial.

The rest of this paper is organized as follows. The next section recalls the definition of an addition chain. Based on it, we then present our universal exponentiation algorithm. In Section 3, we discuss the merits of our approach from a security viewpoint. Section 4 suggests some modifications to our basic algorithm. Finally, we conclude in Section 5.

2 Universal Exponentiation Algorithm

2.1 Addition chains

We start by a brief introduction to addition chains. For further details, we refer the reader to [8].

Definition 1. *An* addition chain *for a positive integer d is a sequence $\mathcal{C}(d) = \{d^{(0)}, d^{(1)}, \ldots, d^{(\ell)}\}$ satisfying*

1. $d^{(0)} = 1$, $d^{(\ell)} = d$, *and*
2. *for all* $1 \le i \le \ell$, *there exist* $j(i), k(i) < i$ *such that* $d^{(i)} = d^{(j(i))} + d^{(k(i))}$.

Integer ℓ defines the *length* of chain $\mathcal{C}$. An addition chain is called a *star-chain* if for all $1 \le i \le \ell$ there exists $k(i) < i$ such that $d^{(i)} = d^{(i-1)} + d^{(k(i))}$.

A slightly more general notion is that of addition-subtraction chains.

Definition 2. *An* addition-subtraction chain *for an integer d is a sequence $\mathcal{C}(d) = \{d^{(0)}, d^{(1)}, \dots, d^{(\ell)}\}$ satisfying*

1. *$d^{(0)} = 1$, $d^{(\ell)} = d$, and*
2. *for all $1 \le i \le \ell$ there exist $j(i), k(i) < i$ such that $d^{(i)} = \pm d^{(j(i))} \pm d^{(k(i))}$.*

2.2 A universal algorithm

Let $\mathcal{C}(d) = \{d^{(0)}, d^{(1)}, \dots, d^{(\ell)}\}$ be an addition chain for exponent d. So for all $1 \le i \le \ell$, we have $d^{(i)} = d^{(j(i))} + d^{(k(i))}$. This provides an easy means to evaluate $y = x^d$: For $i = 1$ to ℓ compute

$$x^{d^{(i)}} = x^{d^{(j(i))}} \cdot x^{d^{(k(i))}}$$

and then set $y = x^{d^{(\ell)}}$. So, from an addition chain of length ℓ, ℓ multiplications are required to compute y.

Example 1. An addition chain for 5 is $\mathcal{C}(5) = \{1, 2, 3, 5\}$ and so $x^1 = x$, $x^2 = x^1 \cdot x^1$, $x^3 = x^2 \cdot x^1$, and finally $x^5 = x^3 \cdot x^2$.

At step i, $x^{d^{(i)}}$ is evaluated as $x^{d^{(i)}} = x^{d^{(j(i))}} \cdot x^{d^{(k(i))}}$. Assuming that $x^{d^{(j(i))}}$ and $x^{d^{(k(i))}}$ respectively belong to registers $R[\alpha(i)]$ and $R[\beta(i)]$ and that the result, $x^{d^{(i)}}$, is written in register $R[\gamma(i)]$, exponent d can be represented by the *register sequence*

$$\Gamma(d) = \big\{\big(\gamma(i) : \alpha(i), \beta(i)\big)\big\}_{1 \le i \le \ell}\,, \tag{1}$$

meaning that $R[\gamma(i)] = R[\alpha(i)] \cdot R[\beta(i)]$. (By convention, the value $d = 1$ is represented by $\Gamma(1) = \emptyset$.)

From this, we obtain the following exponentiation algorithm (for $d > 1$):

```
Input: $x, \Gamma(d)$
Output: $y = x^d$
$R[\alpha(1)] \leftarrow x$; $R[\beta(1)] \leftarrow x$
for $i = 1$ to $\ell$ do
    $R[\gamma(i)] \leftarrow R[\alpha(i)] \cdot R[\beta(i)]$
return $R[\gamma(\ell)]$
```

Algorithm 1. Universal exponentiation algorithm.

Note that $R[\alpha(1)]$ and $R[\beta(1)]$ are initialized to x because the second item of each addition chain is always $d^{(1)} = 2$. Note also that one may have $\alpha(1) = \beta(1)$.

For star chains, we have $d^{(i)} = d^{(i-1)} + d^{(k(i))}$. Therefore pairs are sufficient to represent d: $\alpha(i) = \gamma(i-1)$ for all $1 \le i \le \ell$ and can be omitted from the representation. Hence, we have the *star register sequence*

$$\Gamma^*(d) = \left\{\left(\gamma(i) : \beta(i)\right)\right\}_{1 \le i \le \ell} \ . \tag{2}$$

The corresponding exponentiation algorithm is:

```
Input: $x, \Gamma^*(d)$
Output: $y = x^d$
$R[\gamma(0)] \leftarrow x$; $R[\beta(1)] \leftarrow x$
for $i = 1$ to $\ell$ do
  $R[\gamma(i)] \leftarrow R[\gamma(i-1)] \cdot R[\beta(i)]$
return $R[\gamma(\ell)]$
```

Algorithm 2. Universal star exponentiation algorithm.

3 Towards Provable SPA-resistance

The ultimate goal of smart-card manufacturers is a proof that their implementations are resistant to side-channel analysis. In this paper, we adopt the methodology of modern cryptography towards this goal.

Take for example the encryption scheme RSA-OAEP [3]. The minimal security requirement for an encryption scheme is *one-wayness* (OW). This captures the property that an adversary cannot recover the whole plaintext from a given ciphertext. In some cases, partial information about a plaintext may have disastrous consequences. This notion is captured by *semantic security* or the equivalent notion of *indistinguishability* [7]. Basically, indistinguishability means that the only strategy for an adversary to distinguish between the encryptions of any two plaintexts is to guess at random. The strongest attacks one can imagine (at the protocol level) are the so-called *adaptive chosen-ciphertext attacks* (CCA2). Those attacks consider an active adversary who can obtain the decryption of any ciphertext of her/his choice. From the pair of adversarial goal (IND) and adversarial model (CCA2), we derive the security notion of IND-CCA2. In an IND-CCA2 scenario, an adversary has access to a decryption oracle. S/he first outputs a pair of plaintexts m_0 and m_1. Then, given a challenge ciphertext c_b which is either the encryption of m_0 or m_1, the adversary has to guess with a probability non-negligibly better than 1/2 if c_b encrypts m_0 or m_1. The attack is called adaptive, if after receiving the challenge c_b, the adversary may still obtain decryptions of chosen ciphertexts, the only restriction being not to probe on c_b.

In [3], Bellare and Rogaway remarkably proved that if an adversary is able to break the IND-CCA2 security of RSA-OAEP then the same adversary is able to break the OW security of the RSA function, that is, to compute an e^{th} root modulo a large composite number $N = pq$ (where typically p and q are 512-bit primes). Since the latter is assumed infeasible, RSA-OAEP is declared provably secure. We note that their proof only holds in the *random oracle model* [2], i.e., an ideal world where hash functions behave like random functions. To summarize, the security of RSA-OAEP is proven by

1. identifying the security goal and the adversarial model (i.e., IND-CCA2);
2. defining the working hypotheses (i.e., random oracle model);
3. exhibiting a reduction (i.e., breaking the IND-CCA2 of RSA-OAEP $\Rightarrow$ breaking the OW of the RSA function);
4. assuming that the reduced problem is intractable (i.e., inverting RSA is infeasible);
5. deducing the security notion (i.e., IND-CCA2 security of RSA-OAEP in the random oracle model).

The security of RSA-OAEP is at the protocol level. To break the IND-CCA2 security, the adversary has a black-box access to a decryption oracle: s/he knows the input and obtains the corresponding output. In the case of side-channel attacks, the adversary is more powerful: s/he gets access to some internal states of the computation.

So, by monitoring the power consumption of an RSA exponentiation, an attacker is even sometimes able to recover the secret decryption exponent d used in the computation of $y = x^d \bmod N$ and the OW assumption of the RSA function is no longer valid. Suppose for example that the RSA function is naively implemented with the square-and-multiply method. As shown in the next figure, the exponent can then be recovered very easily: a lower consumption level corresponds to a squaring and a higher consumption level corresponds to a multiplication.

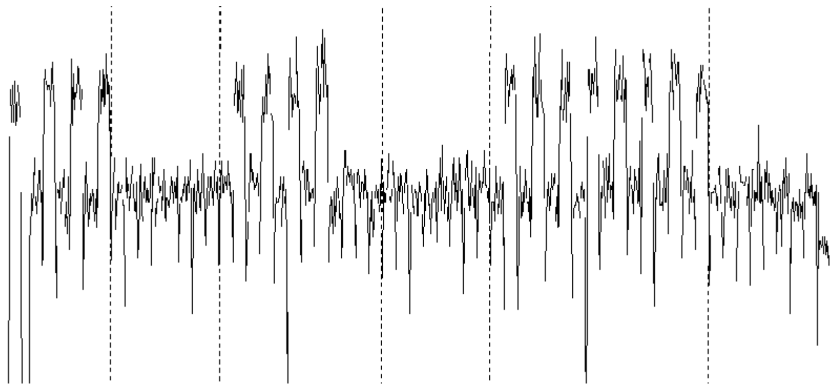

Fig. 1. Power trace of a square-and-multiply exponentiation.

In our simplified model, we consider the fundamental security goal of *unbreakability* (UB). A cryptosystem is said unbreakable if it is infeasible to recover the secret key. This kind of attack is usually referred to as a total breaking. We also consider an attacker who has access to some side-channel information. Depending on the side-channel information and the way it is treated, we define several adversarial models. In the *simple power analysis* (SPA) model, an attacker acquires the power trace of a single execution of the crypto-algorithm. From this, we derive the security notion of UB-SPA. Likewise, one can define the UB-DPA (DPA stands for *differential power analysis* [11]) and so on; one can also consider other security goals and derive security notions like OW-SPA or IND-SPA. It is worth noting here that, contrary to modern cryptography, the definition of an adversarial model is not *absolute*: in a CCA2 attack, an adversary obtains the plaintext corresponding to a chosen ciphertext whereas in an attack like a SPA, the "quality" of the returned information depends on the acquisition tools among other things.

Concentrating on the exponentiation function and more particularly on the RSA function, one can show that if an adversary is able to break the UB-SPA security of the universal exponentiation algorithm, s/he is also able to invert the RSA function. (We note that the main threat for an RSA exponentiation is the SPA; for DPA, efficient counter-measures are known.) In order to break the UB-SPA, an adversary must be able to gain some secret information from the basic operation $R[\gamma(i)] \leftarrow R[\alpha(i)] \cdot R[\beta(i)]$ by SPA, that is, s/he must be able to, at least, differentiate among the triplets $(\gamma(i) : \alpha(i), \beta(i))$ and to recover *all* their values to break the UB property. Assuming that the latter is infeasible (this can be verified experimentally), one has strong evidence[1] that the universal exponentiation algorithm resists to SPA. As a conclusion, if RSA-OAEP is implemented with the universal exponentiation algorithm, we have strong evidence that it resists SPA attacks. Note here that the security is assessed at the implementation level.

From a security viewpoint, the advantage of our method is evident. It reduces the problem of scrutinizing *any* exponentiation algorithm to that of the simpler operation $R[\gamma(i)] \leftarrow R[\alpha(i)] \cdot R[\beta(i)]$. This makes the job of the implementor a lot easier since s/he has a better knowledge of the sensitive parts of her/his algorithm. Moreover, the security passes from a macroscopic level (a software exponentiation) to a microscopic level (a hardware multiplication). Finally, the analysis must be done once for all and remains valid whatever the exponentiation algorithm underlying a given Γ-representation.

Remark 1. In some ways, to relax the assumption the universal exponentiation algorithm is UB-SPA, one can always randomly add dummy operations at the expense of a longer running time (e.g., to add to a Γ representation, a triplet that does not affect the final result). One can also exploit the property that $R[\gamma(i)] \leftarrow R[\alpha(i)] \cdot R[\beta(i)]$ and $R[\gamma(i)] \leftarrow R[\beta(i)] \cdot R[\alpha(i)]$ both lead to the same

[1] In contrast with modern cryptography, we cannot say that we have a proof of security because as aforementioned this depends on the quality of the experiments.

result. Another solution consists to randomly permute the order of the registers and their values during the course of the exponentiation.

In addition to simplifying the security analysis, our universal exponentiation algorithm has the following features:

- it is *simple*: its implementation is straightforward and so programming errors are likely avoided;
- it is *flexible*: owing to the genericity of the Γ-representation, it can virtually execute all exponentiation algorithms;
- it is *fast*: contrary to the protected square-and-multiply method (a.k.a. square-and-multiply-always method) which requires $2 \log_2 d$ multiplications for computing $y = x^d$, our algorithm may require as few as $1.25 \log_2 d$ multiplications (cf. § 4.1);
- it is *economic*: if the exponentiation algorithm underlying a Γ-representation happens to be flawed, it is enough to correct the Γ-representation: a complete re-programming is unnecessary.

The last property is especially interesting for a smart-card implementation. The program code is usually stored in ROM memory via an expensive process called masking and the secret key (e.g., the RSA decryption exponent d) is stored in EEPROM memory at the personalization stage. So in case of secret leakage or mis-programming, one has just to change or correct the Γ-representation of the secret exponent.

4 Practical Considerations

If we want to realize a smart-card implementation of the proposed algorithms (Algorithms 1 and 2), we face some constraints. A smart-card has a limited number of registers and so we need a way to produce Γ-representations with a predetermined number of registers. Moreover, a Γ-representation with fewer registers requires fewer memory for its storage. Another difficulty may occur when the secret exponent is generated outside the card by a third party because it is given in its binary representation.

In this section, we suggest two different approaches that alleviate the above limitations.

4.1 On-line generation

A straightforward solution is to produce a Γ-representation on-line, i.e., by the smart-card itself. Several good heuristics are known for producing relatively short addition chains. In [12], Walter suggests the following method to compute $y = x^d$ (see also [4]).

Define $d_0 = d$, $x_0 = x$, and $y_0 = 1$. Next, at each step, write $d_i = m_i\, d_{i+1} + r_i$ for appropriately chosen values for (m_i, r_i). Hence, letting $x_{i+1} = {x_i}^{m_i}$ and

${y_{i+1}} = {x_i}^{r_i} y_i$, we get

$$\begin{aligned} y = {x_0}^{d_0}\, y_0 &= ({x_0}^{m_0})^{d_1}\, ({x_0}^{r_0}\, y_0) \\ &= {x_1}^{d_1}\, y_1 = ({x_1}^{m_1})^{d_2}\, ({x_1}^{r_1}\, y_1) \\ &= {x_2}^{d_2}\, y_2 = ({x_2}^{m_2})^{d_3}\, ({x_2}^{r_2}\, y_2) \\ &= {x_3}^{d_3}\, y_3 = \cdots \end{aligned}$$

The idea behind Walter's method is to find pairs (m_i, r_i) so that the evaluations of both ${x_i}^{m_i}$ and ${x_i}^{r_i}$ are inexpensive. This is the case when r_i lies in the addition chain used to evaluate ${x_i}^{m_i}$.

Such a method is very well suited to a smart-card implementation. It is easy to implement and the corresponding register sequence, $\Gamma(d)$, requires only one more register than the standard square-and-multiply method. Furthermore, the average length of $\Gamma(d)$ is only $1.25 \log_2 d$, with a very small deviation. See [12] for details.

Note that the computation of $\Gamma(d)$ must be performed in a secured environment since its disclosure reveals the value of secret exponent d. For example, this can performed at the personalization of the card.

4.2 Exponent splitting

The second solution we propose relies on the simple observation that

$$x^d = x^a \cdot x^{d-a} \tag{3}$$

for some a. The idea of splitting the data was already abstracted in [5] as a general countermeasure against differential power analysis attacks. We note that the values of *both* a and $(d-a)$ are required to recover the value of d. In other words, only one exponentiation, x^a or x^{d-a}, needs to be secured.

Given a register sequence for a, $\Gamma(a)$ or $\Gamma^*(a)$, we can compute $y' = x^a$ and $d' = d - a$, and so $x^d = y' \cdot x^{d'}$. There are two possible alternatives. The first one is, for a given a, to store a *chosen* (and thus fixed) register sequence, $\Gamma(a)$, during the personalization of the card. (In this case a star representation, $\Gamma^*(a)$, may be preferred since it requires fewer memory.) The advantage of this approach is that this imposes the underlying methods for computing $y' = x^a$ and $d' = d - a$.

Another alternative consists in *randomly* computing a register sequence, $\Gamma(a)$ or $\Gamma^*(a)$, for a "on the fly". The advantages of this second approach are twofold. First, no register sequence needs to be stored in non-volatile memory and so this results in some memory savings. Second, the methods for evaluating $y' = x^a$ and $d' = d - a$ differ at each execution. Independently, this randomization also helps to prevent differential attacks like the DPA.

5 Conclusion

In this paper, we presented an universal exponentiation algorithm. Through the notion of register sequence, $\Gamma(d) = \{(\gamma(i) : \alpha(i), \beta(i))\}_{1 \le i \le \ell}$, built from addition chains, we explained how this helps to protect an exponentiation-based cryptosystem against simple side-channel attacks like SPA. Assuming that a more atomic operation (i.e., the multiplication of registers $R[\gamma(i)] \leftarrow R[\alpha(i)] \cdot R[\beta(i)]$) does not leak secret information, we "proved" the security of our implementation. There is no secret at all involved in our universal exponentiation algorithm: the secret exponent d is intimately tied to $\Gamma(d)$ and recovering the value of d supposes the recovery of the whole sequence $\Gamma(d)$, which is a contradiction. Furthermore, our algorithm can be trivially implemented and it greatly simplifies the security analysis since the critical (i.e., sensitive) parts are better understood.

As a final conclusion, we hope that this first step towards provable security of real-world implementations will be a motivating starting-point for further research in this very important subject.

Acknowledgements

The authors are grateful to Jacques Fournier, Karine Gandolfi and Florence Quès for some comments.

References

1. Mihir Bellare, Anand Desai, David Pointcheval, and Phillip Rogaway. Relations among notions of security for public-key encryption schemes. Full paper (30 pages), February 1999. An extended abstract appears in H. Krawczyk, ed., Advances in Cryptology – CRYPTO '98, volume 1462 of Lecture Notes in Computer Science, pages 26–45, Springer-Verlag, 1998.
2. Mihir Bellare and Phillip Rogaway. Random oracles are practical: A paradigm for designing efficient protocols. In *First ACM Conference on Computer and Communications Security*, pages 62–73. ACM Press, 1993.
3. Mihir Bellare and Phillip Rogaway. Optimal asymmetric encryption. In A. De Santis, editor, *Advances in Cryptology – EUROCRYPT '94*, volume 950 of *Lecture Notes in Computer Science*, pages 92–111. Springer-Verlag, 1995.
4. F. Bergeron, J. Berstel, S. Brlek, and C. Duboc. Addition chains using continued fractions. *Journal of Algorithms*, 10(3):403–412, September 1989.
5. Suresh Chari, Charanjit S. Jutla, Josyula R. Rao, and Pankaj Rohatgi. Towards sound approaches to counteract power-analysis attacks. In M. Wiener, editor, *Advances in Cryptology – CRYPTO '99*, volume 1666 of *Lecture Notes in Computer Science*, pages 398–412. Springer-Verlag, 1999.
6. Oded Goldreich. On the foundations of modern cryptography. In B. Kaliski, editor, *Advances in Cryptology – CRYPTO '97*, volume 1294 of *Lecture Notes in Computer Science*, pages 46–74. Springer-Verlag, 1997.
7. Shafi Goldwasser and Silvio Micali. Probabilistic encryption. *Journal of Computer and System Sciences*, 28:270–299, 1984.
8. Donald E. Knuth. *The art of computer programming/Seminumerical algorithms*, volume 2. Addison-Wesley, 2nd edition, 1981.
9. Paul 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.

10. Paul Kocher. Secure modular exponentiation with leak minimization for smart cards and other cryptosystems. International patent WO 99/67909, March 1998.
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. Colin D. Walter. Exponentiation using division chains. *IEEE Transactions on Computers*, 47(7):757–765, July 1998.
13. Yacov Yacobi. Exponentiating faster with addition chains. In *Advances in Cryptology – EUROCRYPT '90*, volume 473 of *Lecture Notes in Computer Science*, pages 222–229. Springer-Verlag, 1991.