Security as a New Dimension in Embedded System Design

Paul Kocher†, Ruby Lee‡*, Gary McGraw¶, Anand Raghunathan§ and Srivaths Ravi§
‡ EE Department, Princeton University, Princeton, NJ
† Cryptography Research, San Francisco, CA
¶ Cigital, Dulles, VA
§ NEC Laboratories America, Princeton, NJ

Abstract

The growing number of instances of breaches in information security in the last few years has created a compelling case for efforts towards secure electronic systems. Embedded systems, which will be ubiquitously used to capture, store, manipulate, and access data of a sensitive nature, pose several unique and interesting security challenges. Security has been the subject of intensive research in the areas of cryptography, computing, and networking. However, security is often mis-construed by embedded system designers as the addition of features, such as specific cryptographic algorithms and security protocols, to the system. In reality, it is an entirely new metric that designers should consider throughout the design process, along with other metrics such as cost, performance, and power.

This paper is intended to introduce embedded system designers and design tool developers to the challenges involved in designing secure embedded systems. We attempt to provide a unified view of embedded system security by first analyzing the typical functional security requirements for embedded systems from an end-user perspective. We then identify the implied challenges for embedded system architects, as well as hardware and software designers (*e.g.*, tamper-resistant embedded system design, processing requirements for security, impact of security on battery life for battery-powered systems, *etc*.). We also survey solution techniques to address these challenges, drawing from both current practice and emerging research, and identify open research problems that will require innovations in embedded system architecture and design methodologies.

Categories and Subject Descriptors

C.0 [**Computer Systems Organization**]: General- *System architectures, Instruction set design*; C.1.0 [**Computer Systems Organization**]: Processor architectures- *General*; C.2.0 [**Computer Systems Organization**]: Computer-Communication Networks- *General, Security and protection*; C.5.3 [**Computer Systems Organization**]: Computer System Implementation- *Microcomputers, Portable devices*; D.0 [**Software**]: General; E.3 [**Data**]: Data encryption- *DES, Public key cryptosystems*

General Terms

Security, Performance, Design, Reliability, Algorithms, Verification

Keywords

Embedded Systems, Security, Cryptography, Security Protocols, Security Processing, Design, Design Methodologies, Architectures, Tamper Resistance, Software Attacks, Viruses, Trusted Computing, Digital Rights Management, Performance, Battery Life

*Prof. Lee's work is supported in part by NSF CCR-0326372 "ITR: Architectures and Design Methodologies for Secure Low-Power Embedded Systems."

DAC 2004, June 7–11, 2004, San Diego, California, USA.

1. INTRODUCTION

Today, security in one form or another is a requirement for an increasing number of embedded systems, ranging from low-end systems such as PDAs, wireless handsets, networked sensors, and smart cards, to high-end systems such as routers, gateways, firewalls, storage servers, and web servers. Technological advances that have spurred the development of these electronic systems have also ushered in seemingly parallel trends in the sophistication of security attacks. It has been observed that the cost of insecurity in electronic systems can be very high. For example, it was estimated that the "I Love You" virus caused nearly one billion dollars in lost revenues worldwide [1]. With an increasing proliferation of such attacks, it is not surprising that a large number of users in the mobile commerce world (nearly 52% of cell phone users and 47% of PDA users, according to a survey by Forrester Research [2]) feel that security is the single largest concern preventing the successful deployment of next-generation mobile services.

With the evolution of the Internet, information and communications security has gained significant attention. For example, various security protocols and standards such as IPSec, SSL, WEP, and WTLS [3, 4], are used for secure communications. While security protocols and the cryptographic algorithms they contain address security considerations from a functional perspective, many embedded systems are constrained by the environments they operate in, and by the resources they possess. For such systems, there are several factors that are moving security considerations from a function-centric perspective into a system architecture (hardware/software) design issue. For example,

- An ever increasing range of attack techniques for breaking security such as software, physical and side-channel attacks require that the embedded system be secure even when it can be logically or physically accessed by malicious entities. Resistance to such attacks can be ensured only if built into the system architecture and implementation.
- The processing capabilities of many embedded systems are easily overwhelmed by the computational demands of security processing, leading to undesirable tradeoffs between security and cost, or security and performance.
- Battery-driven systems and small form-factor devices such as PDAs, cell phones and networked sensors often operate under stringent resource constraints (limited battery, storage and computation capacities). These constraints only worsen when the device is subject to the demands of security.
- Embedded system architectures need to be flexible enough to support the rapid evolution of security mechanisms and standards.
- New security objectives, such as denial of service and digital content protection, require a higher degree of co-operation between security experts and embedded system architects.

This paper will introduce the embedded system designer to the importance of embedded system security, review evolving trends and standards, and illustrate how the security requirements translate into system design challenges. Emerging solutions to address these challenges through a combination of advanced embedded system architectures and design methodologies will be presented.

2. EMBEDDED SYSTEM SECURITY REQUIREMENTS

Embedded systems often provide critical functions that could be sabotaged by malicious entities. Before discussing the common security requirements for embedded systems, it is important to note that there are many entities involved in a typical embedded system manufacturing, supply, and usage chain. Security requirements vary depending on whose perspective we consider. For example, consider a state-of-the-art cell phone that is capable of wireless voice, multimedia, and data communications. Security requirements may vary when considered from the viewpoint of the manufacturer of a component inside the cell phone (*e.g.*, baseband processor), the cell phone manufacturer, the cellular service provider, the content provider, and the end user of the cell phone. The end user's primary concern may be the security of personal data stored and communicated by the cell phone, while the content provider's primary concern may be copy protection of the multimedia content delivered to the cell phone, and the cell phone manufacturer might additionally be concerned with the secrecy of proprietary firmware that resides within the cell phone. For each of these cases, the set of potentially malicious entities can also vary. For example, from the perspective of the content provider, the end user of the cell phone may be an untrusted entity. While this section outlines broad security requirements typical of embedded systems, the security model for each embedded system will dictate the combination of requirements that apply.

Historically, information security was first explored in the context of communications systems. When two entities send or receive sensitive information using public networks or communications channels accessible to potential attackers, they should ideally provide security functions such as *data confidentiality*, *data integrity*, and *peer authentication*. Data confidentiality protects sensitive information from undesired eavesdroppers. Data integrity ensures that the information has not been changed illegitimately. Peer authentication verifies that the information is sent and received by appropriate parties rather than masqueraders. These security functions are required of embedded systems used in a wide range of applications today.

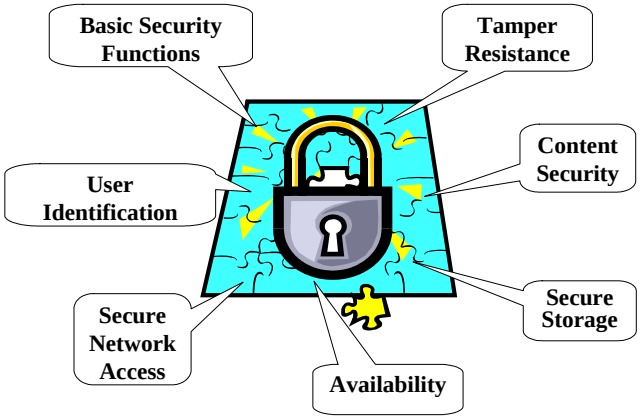

Figure 1: Common security requirements of embedded systems from an end-user perspective

Fig 1 shows some of these security requirements from the perspective of an end-user, where we use the term *basic security functions* to denote the set of confidentiality, integrity and authentication requirements. Very often, access to the embedded system should be restricted to a selected set of authorized users (*user identification*), while access to a network or a service has to be provided only if the device is authorized (*secure network access*). These may use the user or host authentication mechanisms provided by the basic security functions, as well as other mechanisms such as biometrics and access control.

Another essential security function is the *availability* of the embedded system. In several scenarios, one can expect malicious entities preventing an embedded system from performing the functions it is supposed to, resulting in a degradation of performance or complete denial of service (DoS) to legitimate users.

Embedded system security often requires protecting critical or sensitive information (code or data) throughout its lifetime, including making sure that it is properly erased at the end of its lifetime. *Secure storage* involves securing information in the embedded system's storage devices, external or internal to the system. *Content security* or *Digital Rights Management (DRM)* protects the rights of the digital content used in the system, and is an issue actively pursued by several content providers.

Finally, *tamper resistance* refers to the desire to maintain these security requirements even when the device falls into the hands of malicious parties, and can be physically or logically probed.

3. SECURITY MECHANISMS

This section illustrates some existing security mechanisms for achieving desired security functions.

Basic security functions described above are often achieved using three different classes of cryptographic algorithms – symmetric ciphers, asymmetric ciphers and secure hash algorithms. (For a basic introduction to cryptography, we refer the reader to [3, 4]).

- *Symmetric ciphers* require the sender to use a secret key to encrypt data (the data being encrypted is often referred to as plaintext) and transmit the encrypted data (usually called the ciphertext) to the receiver. On receiving the ciphertext, the receiver then uses the same secret key to decrypt it and regenerate the plaintext. The ciphertext should have the property that it is very hard for a third party to deduce the plaintext, without having access to the secret key. Thus, confidentiality of data is ensured during transmission. Examples of symmetric ciphers include DES, 3DES, AES, and RC4.
- *Secure Hash algorithms* such as MD5 and SHA convert arbitrary messages into unique fixed-length values, thereby providing unique "fingerprints" for messages. Hash functions are often used to construct Message Authentication Codes (MACs), such as HMAC-SHA, which additionally incorporate a key to prevent adversaries who tamper with data from avoiding detection by recomputing hashes.
- *Asymmetric algorithms* (also called public-key algorithms), use a pair of keys: one of the keys locks the data while the other unlocks it. Encryption of a message intended for a given recipient requires only the public key known to the world, but decryption is only possible with the recipient's private key, which the recipient should keep secret. Thus, use of the private key (assuming it is kept secret) provides user or host authentication. Hence, digital signatures are often constructed using public key cryptography and secure hashes. The user can "digitally sign" a message by encrypting a hash of it with his private key; any one can verify this signature by decrypting with the public key.

Asymmetric ciphers (*e.g.*, RSA, Diffie-Hellman, *etc.*) rely on the use of more computationally intensive mathematical functions such as modular exponentiation of large integers, and hence are much slower than symmetric ciphers. Because of this, a symmetric cipher is typically used to encrypt bulk data, while an asymmetric cipher is used to establish (transmit) the secret key for this symmetric cipher across a public network.

Security solutions to meet the various requirements outlined in the previous section typically rely on security mechanisms that use a combination of these cryptographic algorithms in the context of a security protocol. Various security technologies and mechanisms have been designed around these cryptographic algorithms in order to provide specific security services. For example,

- Secure communication protocols (popularly called security protocols) provide ways of ensuring secure communication channels to and from the embedded system. IPSec [5] and SSL [6] are popular examples of security protocols, widely used for Virtual Private Networks (VPNs) and secure web transactions, respectively.
- Digital certificates provide ways of associating identity with an entity, while biometric technologies [7] such as fingerprint recognition and voice recognition aid in end-user identification. Digital signatures, which function as the electronic equivalent of handwritten signatures, can be used to authenticate the source of data as well as verify its identity.
- Digital Rights Management (DRM) protocols such as OpenIPMP [8], ISMA [9] and MOSES [10], provide secure frameworks intended to protect application content against unauthorized use.
- Secure storage and secure execution require that the architecture of the system be tailored for security considerations. Simple examples include the use of dedicated hardware to block illegal accesses to protected areas in the memory [11], authentication of firmware and software that execute on the system, preserving the confidentiality and integrity of code and data associated with a given application or a

process [12], hardware and software techniques to preserve the confidentiality and integrity of data throughout the memory hierarchy [13], and execution of encrypted code in processors to prevent bus probing [14, 15] *etc.*

4. SECURITY ATTACKS AND COUNTERMEASURES

Various attacks on electronic and computing systems have shown that hackers rarely take on the theoretical strength of well-designed cryptographic algorithms. Instead, they rely on exploiting security vulnerabilities in the software and hardware components of the implementation. In this section, we show that unless security is considered throughout the design cycle, embedded system implementation vulnerabilities can easily be exploited to bypass or weaken functional security measures.

4.1 Embedded Software Attacks and Countermeasures

Software in embedded systems is a major source of security vulnerabilities. Three factors, which we call the *Trinity of Trouble* — complexity, extensibility and connectivity — conspire to make managing security risks in software a major challenge.

- **Complexity:** Software is complicated, and will become even more complicated in the near future. More lines of code increases the likelihood of bugs and security vulnerabilities. As embedded systems converge with the Internet and more code is added, embedded system software is clearly becoming more complex. The complexity problem is exacerbated by the use of unsafe programming languages (e.g., C or C++) that do not protect against simple kinds of attacks, such as buffer overflows. For reasons of efficiency, C and C++ are very popular languages for embedded systems. In theory, we could analyze and prove that a small program is free of problems, but this task is impossible for programs of realistic complexity today.
- **Extensibility:** Modern software systems, such as Java and .NET, are built to be extended. An extensible host accepts updates or extensions (mobile code) to incrementally evolve system functionality. Today's operating systems support extensibility through dynamically loadable device drivers and modules. Advanced embedded systems are designed to be extensible (e.g., J2ME, Java Card). Unfortunately, the very nature of extensible systems makes it hard to prevent software vulnerabilities from slipping in as an unwanted extension.
- **Connectivity:** More and more embedded systems are being connected to the Internet. The high degree of connectivity makes it possible for small failures to propagate and cause massive security breaches. Embedded systems with Internet connectivity will only make this problem grow. An attacker no longer needs physical access to a system to launch automated attacks to exploit vulnerable software. The ubiquity of networking means that there are more attacks, more embedded software systems to attack, and greater risks from poor software security practices.

4.1.1 Example of a Software Attack: The Hardware Virus

Software attacks against the operating system kernel, such as those carried out by rootkits, demonstrate the kinds of attack that embedded systems are exposed to [16]. A kernel has full access to the system and can communicate with any part of the address space. This means, among other things, that an attacker can read or write to the BIOS memory on the motherboard or in peripheral hardware.

In earlier times, BIOS memory was stored in ROM or in EPROM chips which could not be updated from software. These older systems require the chips to be replaced or manually erased and rewritten. Since this is not very cost effective, new systems employ EEPROM chips, otherwise known as Flash ROM. Flash ROM can be re-written from software. Modern embedded systems often include Flash ROM.

A given computer can have several megabytes of Flash ROM on various controller cards and the motherboard. These Flash ROM chips are almost never fully utilized and leave lots of room to store backdoor information and viruses. To an attacker, the compelling reason for using these memory spaces is that they are hard to audit and almost never visible to software running on a system. To access such "hardware memory" requires driver level access. Furthermore, this memory is immune to re-booting and system re-installation. If someone suspects a viral infection, restoring the system from tape or backup will not help. The hardware virus has always been, and will remain, one of the best kept secrets of the "black magic" hackers.

A simple hardware virus may be designed to impart false data to a system, or to cause the system to ignore certain events. Imagine an anti-aircraft radar that uses an embedded real-time operating system. Within the system are several Flash ROM chips. A virus is installed into one of these chips and it has trusted access to the entire bus. The virus has only one purpose — to cause the radar to ignore certain types of radar signatures.

Viruses have long since been detected "in the wild" that write themselves into the motherboard BIOS memory. In the late 90s, the so-called F00F bug was able to crash a laptop completely. Although the CIH (of Chernobyl) virus was widely popularized in the media, virus code that used the BIOS was published long before the release of CIH.

EEPROM memory is fairly common on many systems. Ethernet cards, video cards, and multimedia peripherals may all contain EEPROM memory. The hardware memory may contain flash firmware or may just be used for data storage. Non-volatile memory chips are found in a variety of hardware devices: TV tuners and remote controls, CD Players, cordless and cellular phones, fax machines, cameras, radios, automotive airbags, anti-lock brakes, odometers, keyless entry systems, printers and copiers, modems, pagers, satellite receivers, barcode readers, point of sale terminals, smart cards, lock boxes, garage door openers, and test and measurement equipment. EEPROM chips remain a prime area for storing subversive code. As more embedded devices become available, the EEPROM based virus will be more applicable and dangerous.

4.1.2 Securing against software attacks

Software is a central and critical aspect of the computer (and embedded system) security problem. Software defects with security ramifications — including implementation bugs such as buffer overflows and design flaws such as inconsistent error handling — promise to be with us for years. All too often, malicious intruders can hack into systems by exploiting software defects [16]. Moreover, Internet-enabled software applications present the most common security risk encountered today, with software's ever-expanding complexity and extensibility adding further fuel to the fire.

Software security's best practices leverage good software engineering practice and involve thinking about security early in the software design life cycle (SDLC), knowing and understanding common threats (including language-based flaws and pitfalls), designing for security, and subjecting all software artifacts to thorough objective risk analyses and testing.

Security is an emergent property of a software system. This is a subtle point often lost on development people who tend to focus on functionality. Obviously, there are security functions in the world, and most modern software includes security features, but adding features such as SSL (for cryptographically protecting communications) does not present a complete solution to the security problem. A security problem is more likely to arise because of a problem in a standard part of the system (e.g., the API to the server) than in some given security feature. This is an important reason why software security must be part of a full lifecycle approach. Just as you cannot test quality into a piece of software, you can not spray paint security features onto a design and expect it to become secure.

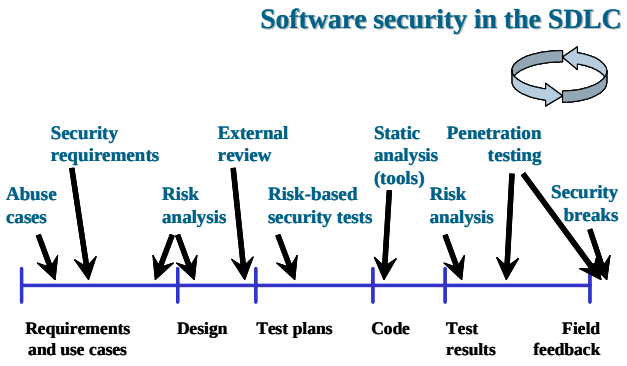

Figure 2: Software security best practices applied to various software artifacts in the Software Design Life Cycle (SDLC)

As practitioners become aware of software security's importance, they are increasingly adopting and evolving a set of best practices to address the problem [17, 18]. There is no substitute for working software security as deeply

into the development process as possible and taking advantage of the engineering lessons software practitioners have learned over the years. Figure 2 specifies one set of best practices and shows how software practitioners can apply them to the various software artifacts produced during software development. Although the artifacts are shown laid out in a linear sequence, most organizations follow an iterative approach, which means that best practices will be cycled through more than once as the software evolves.

Software security best practices apply at various levels:

- **The requirements level:** Security requirements must cover both overt functional security (e.g., the use of applied cryptography) and emergent characteristics.
- **The design and architecture level:** A system must be coherent and present a unified security architecture that takes into account security principles (such as the principle of least privilege) [17].
- **The code level:** Static analysis tools — tools that scan source code for common vulnerabilities — can discover implementation bugs at the code level.

Risks crop up during all stages of the software life cycle, so a constant risk analysis thread, with recurring risk tracking and monitoring activities, is highly recommended.

Security testing must encompass two strategies: testing security functionality with standard functional testing techniques, and risk-based security testing based on attack patterns and threat models. Penetration testing is also useful, especially if an architectural risk analysis is specifically driving the tests.

Operations people should carefully monitor embedded systems being used in the field for security breaks. Attacks will happen, regardless of the strength of design and implementation, so monitoring software behavior, during deployment, is an excellent defense technique.

4.2 Physical and Side-channel Attacks, and Tamper-resistant Hardware

In addition to software attacks, there are *physical* and *side-channel* attacks that exploit the system implementation or its identifying properties to break the security of an embedded system [19, 20, 21, 22, 23, 24, 25, 26]. Historically, many of these attacks have been used to break the security of embedded systems such as smart cards. Physical and side-channel attacks are generally classified into *invasive* and *non-invasive* attacks. Invasive attacks involve getting access to the appliance to observe, manipulate and interfere with the system internals. Since invasive attacks against integrated circuits typically require expensive equipment, they are relatively hard to mount and repeat. Examples of invasive attacks include micro-probing and reverse engineering. Non-invasive attacks, as the name indicates, do not require the device to be opened. While these attacks may require an initial investment of time or creativity, they tend to be cheap and scalable (compared to invasive attacks). There are many forms of non-invasive attacks including timing attacks, power analysis attacks, fault induction techniques, and electromagnetic analysis attacks. In the sections that follow, we describe various physical and side-channel attacks in more detail.

4.2.1 Physical attacks

For an embedded system on a circuit board, physical attacks can be launched by using probes to eavesdrop on inter-component communications. However, for a system-on-chip, sophisticated micro-probing techniques become necessary [21, 22]. The first step in such attacks is de-packaging. De-packaging typically involves removal of the chip package by dissolving the resin covering the silicon using fuming acid. The next step involves layout reconstruction using a systematic combination of microscopy and invasive removal of covering layers. During layout reconstruction, the internals of the chip can be inferred at various granularities. While higher-level architectural structures within the chip such as data and address buses, memory and processor boundaries, *etc.*, can be extracted with little effort, detailed views of lower-level structures such as the instruction decoder and ALU in a processor, ROM cells, *etc.*, can also be obtained. Finally, techniques such as manual microprobing or e-beam microscopy are typically used to observe the values on the buses and interfaces of the components in a de-packaged chip.

Physical attacks at the chip level are relatively hard to use because of their expensive infrastructure requirements (relative to other attacks). However, they can be performed once and then used as precursors to the design of successful non-invasive attacks. For example, layout reconstruction is useful when performing electromagnetic radiation monitoring around selected chip areas. Likewise, the knowledge of ROM contents, such as cryptographic routines and control data, can provide an attacker with information that can assist in the design of a suitable non-invasive attack.

4.2.2 Timing analysis

For many devices, even computing the correct result does not ensure security. In 1996, Kocher [27] showed how keys could be determined by analyzing small variations in the time required to perform cryptographic computations. The attack involves making predictions about secret values (such as individual key bits), and then using statistical techniques to test the prediction by determining whether the target device's behavior is correlated to the behavior expected by the prediction.

To understand the attack, consider a computation that begins with a known input and includes a sequence of steps, where each step mixes in one key bit and takes a non-constant amount of time. For example, for a given input, two operations are possible for the first step, depending on whether the key bit is zero or one.

For the attack, the adversary observes a set of inputs and notes the approximate total time required for a target device to process each. Next, the adversary measures the correlation between the measured times and the estimated time for the first step assuming the first step mixes in a zero bit. A second correlation is computed using estimates with a bit value of one. The estimates using the bit value actually used by the target device should show the strongest correlation to the observed times. The attack can then be repeated for subsequent bits.

What makes the attack interesting is that "obvious" countermeasures often do not work. For example, quantizing the total time (*e.g.*, delaying to make the total computation take an exact multiple of 10ms) or adding random delays increases the number of measurements required, but does not prevent the attack. Obviously, making all computations take exactly the same amount of time would eliminate the attack, but few programs running on modern microprocessors operate in exactly constant time. Writing constant-time code (particularly in high-level languages) can be tricky.

Fortunately, there are techniques that can reliably prevent timing attacks in many systems. For example, message blinding can be used with RSA and other public key cryptosystems to prevent adversaries from correlating input and output values with timing measurements. For further information about timing attacks, see [27].

4.2.3 Power analysis

Timing channels are not the only way that devices "leak" information. For example, the operating current drawn by a hardware device is correlated to computations it is performing. In most integrated circuits, the major contributors to power consumption are the gates themselves and the parasitic capacitance of the internal wiring. Power consumption increases if more state transitions occur, or if transitions are occurring predominantly at gates with greater size or capacitive load. There are two main categories of power analysis attacks, namely, simple power analysis (SPA) and differential power analysis (DPA).

SPA attacks rely on the observation that in some systems, the power profile of cryptographic computations can be directly used to interpret the cryptographic key used. For example, SPA analysis can be used to break RSA implementations by revealing differences between the multiplication and squaring operations performed during modular exponentiation. In many cases, SPA attacks have also been used to augment or simplify brute-force attacks. For example, it has been shown in [28] that the brute-force search space for one software DES implementation on an 8-bit processor with 7 bytes of key data can be reduced to 2^{40} keys from 2^{56} keys with the help of SPA.

DPA attacks use statistical techniques to determine secret keys from complex, noisy power consumption measurements. For a typical attack, an adversary repeatedly samples the target device's power consumption through each of several thousand cryptographic computations. These power traces are collected using high-speed analog-to-digital converters, such as those found in digital storage oscilloscopes.

After the data collection, the adversary makes a hypothesis about the key. For example, if the target algorithm is DES (see [3] for a detailed description of DES), a typical prediction might be that the 6 key bits entering S-box 4 are equal to '011010'. If correct, an assertion of this form allows the adversary to compute four bits entering the second round of the DES computation. If the assertion is incorrect, however, an effort to predict any of these bits will be wrong roughly half the time.

For any of the four predicted bits, the power traces are divided into two subsets: one where the predicted bit value is 0, and one set where the predicted value is 1. Next, an average trace is computed for each subset, where the nth sample in each average trace is the average of the nth samples in all

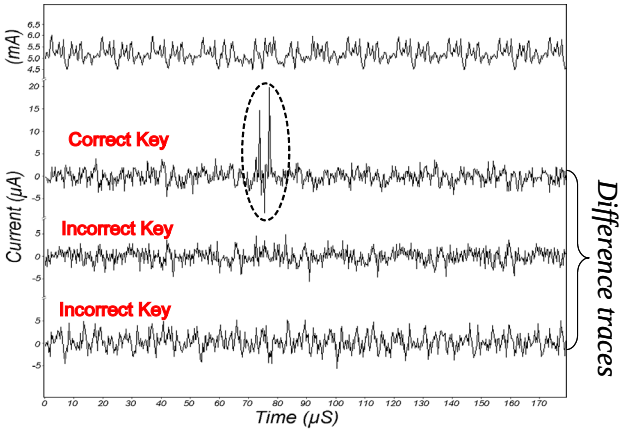

Figure 3: Power consumption traces generated during a DPA attack on a DES implementation

traces in the subset. Finally, the adversary computes the difference of the average traces (called *difference traces*).

If the original hypothesis is incorrect, the criteria used to create the subsets will be approximately random. Any randomly-chosen subset of a sufficiently-large data set will have the same average as the main set. As a result, the difference will be effectively zero at all points, and the adversary repeats the process with a new guess.

If the hypothesis is correct, however, choice of the subsets will be correlated to the actual computation. In particular, the second-round bit will have been '0' in all traces in one subset and '1' in the other. When this bit is actually being manipulated, its value will have a small effect on the power consumption, which will appear as a statistically-significant deviation from zero in the difference trace. Figure 3 shows an actual power consumption profile and the difference traces when both correct and incorrect keys are guessed.

DPA allows adversaries to pull extremely small "signals" from extremely noisy data, often without even knowing the design of the target system. These attacks are of particular concern for devices such as smart cards that must protect secret keys while operating in hostile environments. Countermeasures that reduce the quality of the measurements (such as running other circuits simultaneously) only increase the number of samples the adversary needs to collect, and do not prevent the attack. For further information about DPA, see [29].

Attacks such as DPA that involve many aspects of system design (hardware, software, cryptography, *etc.*) pose additional challenges for embedded system engineering projects because security risks may be concealed by layers of abstraction. Countermeasures used to mitigate these attacks are also frequently mathematically rigorous, non-intuitive, and require patent licensing [30]. As a result, projects requiring effective tamper resistance, particularly when used for securing payments, audiovisual content, and other high-risk data, remain expensive and challenging.

4.2.4 *Fault induction*

Hardware security devices depend on more than correct software. If the hardware ever fails to make correct computations, security can be jeopardized.

For example, almost any computation error can compromise RSA implementations using the Chinese Remainder Theorem (CRT). The computation involves two major subcomputations, one that computes the result modulo p and the other modulo q, where p and q are the factors of the RSA public modulus n. If, for example, the mod p computation result is incorrect, the final answer will be incorrect modulo p, but correct modulo q. Thus, the difference between the correct answer and the computed answer will be an exact multiple of q, allowing the adversary to find q by computing the greatest common denominator (GCD) of this difference and n.

To deter this specific attack, RSA implementations can check their answers by performing a public key operation on the result and verifying that it regenerates the original message. Unfortunately, error detection techniques for symmetric algorithms are not nearly as elegant, and there are many other kinds of error attacks. As a result, many cryptographic devices include an assortment of glitch sensors and other features designed to detect conditions likely to cause computation errors. For further discussion of this topic, see [31].

4.2.5 *Electromagnetic Analysis*

Electromagnetic analysis attacks (EMA) have been well documented since the eighties, when it was shown in [32] that electromagnetic radiation from a video display unit can be used to reconstruct its screen contents. Since then, these attacks have only grown in sophistication [33]. The basic premise of many of these attacks is that they attempt to measure the electromagnetic radiation emitted by a device to reveal sensitive information. Successful deployment of these attacks against a single chip would require intimate knowledge of its layout, so as to isolate the region around which electromagnetic radiation measurements must be performed. Like power analysis attacks, two classes of EMA attacks, namely, simple EMA (SEMA) and differential EMA (DEMA) attacks have been proposed [34, 35].

5. EMBEDDED PROCESSING ARCHITECTURES FOR SECURITY

In the past, embedded systems tended to perform one or a few fixed functions. The trend is for embedded systems to perform multiple functions and also to provide the ability to download new software to implement new or updated applications in the field, rather than only in the more controlled environment of the factory. While this certainly increases the flexibility and useful lifetime of an embedded system, it poses new challenges in terms of the increased likelihood of attacks by malicious parties. An embedded system should ideally provide required security functions, implement them efficiently and also defend against attacks by malicious parties. We discuss these below, especially in the context of the additional challenges faced by resource-constrained embedded systems in an environment of ubiquitous networking and pervasive computing.

Figure 4 illustrates the architectural design space for secure embedded processing systems. Different macro-architecture models are listed in the first row, and described further below. These include embedded general-purpose processor (EP) *vs.* application-specific instruction set processor (ASIP) *vs.* EP with custom hardware accelerators connected to the processor bus, *etc.*). The second row details instruction-set architecture and micro-architecture choices for tuning the base processor where appropriate. The third row articulates security processing features that must be chosen or designed. For example, choosing the functionality to be implemented by custom instructions, hardware accelerators or general-purpose instruction primitives. The fourth row involves selection of attack-resistant features in the embedded processor and embedded system design. These protect against both software attacks and physical attacks, as illustrated in section 4. This may include an enhanced memory management unit to manage a secure memory space, process isolation architecture, additional redundant circuitry for thwarting power analysis attacks, and fault detection circuitry.

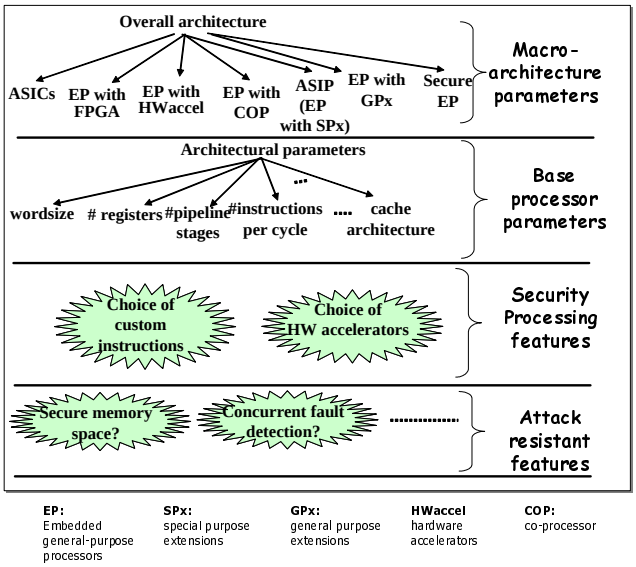

Figure 4: Architectural design space for secure information processing

5.1 Security Processing Architectures

We now describe an evolution of macro-architecture models that have been proposed to provide required security functions. Basic security functions, like confidentiality, integrity and authentication, can be implemented with appropriate security protocols and cryptographic algorithms. The latter include symmetric ciphers, asymmetric ciphers and secure hashing algorithms, as discussed in section 3. However, such ciphers can be rather compute intensive and power hungry, which is a challenge for resource-constrained embedded systems.

A hardware-only approach uses ASICs (Application Specific Integrated Circuits) to implement a given cryptography algorithm in hardware. This "hardwired algorithm" approach using fixed function ASICs can be very cost-effective when one or only a few ciphers are needed, and the ASICs are produced in large volume. However, it is much less effective in terms of cost and flexibility, when a variety of ciphers are desired to support multiple security protocols, emerging standards, and inter-operability with many devices.

A software-only approach using a typical embedded general-purpose processor (EP) core for performing security protocol and cryptography processing at network link speeds faces both a *processing gap* and a *battery gap*. The processing gap refers to the fact that the computational requirements of standard security protocols utilizing cryptography are significantly higher than the processor capabilities available in embedded systems such as wireless handheld devices [36, 37]. For example, the total processing requirements for software implementations of SSL (with 3DES used for encryption and decryption, and SHA for message authentication) executing on an iPAQ handheld (235 MIPS StrongARM processor) was shown in [36] to be around 651.3 MIPS, at a link speed of 10 Mbps (current and emerging data rates for wireless LAN technologies are in the range of 2-60 Mbps). The battery gap refers to the significant additional energy consumption required by security processing on battery-powered wireless handhelds [38, 39]. For example, when a device such as Symbol PPT2800 PocketPC is encrypting transmitted data, a considerable part (nearly 21%) of the overall energy consumption is spent on security processing [38].

Several hybrid hardware-software approaches have been proposed to efficiently implement security functions. One uses a general-purpose embedded processor core with hardware accelerators (chips or cores) for the most performance critical steps. For example, since most of the time consumed in executing a public key algorithm such as RSA is in performing modular exponentiation, accelerator chips typically provide hardware speedup for modular multiplication. However, other ciphers may not be accelerated at all by this approach while still incurring the cost of the accelerator chips (or cores in System-On-Chip designs). The use of FPGAs (Field Programmable Gate Arrays) allows some reconfiguration to support different ciphers. However, this hardware flexibility may not always meet other goals in cost, energy consumption and performance simultaneously.

Another approach is to tightly integrate such acceleration hardware with the processor core itself and invoke it with custom instructions. This is denoted the ASIP (Application Specific Instruction Processor) approach. For example, embedded processors such as Xtensa [40] are equipped with tools that allow a designer to extend the basic instruction set of a processor with a set of application-specific or custom instructions. A typical example is to implement one round of a symmetric cipher such as DES (Data Encryption Standard) in hardware and invoke this with a custom instruction. This can provide very significant acceleration to DES with excellent energy consumption, but no performance or energy improvement at all for other ciphers. For certain embedded systems, this is not only adequate, but often the most cost-effective solution. For example, MOSES [41] is an ASIP that uses a custom, low-overhead instruction set for a selected set of symmetric, asymmetric, and hashing algorithms.

In general-purpose processors, new instructions have also been proposed to accelerate a given set of existing symmetric ciphers [42, 43]. While these new instructions typically perform one or a few operations rather than an entire round of a block cipher, they are still fairly specific to the block ciphers considered. Hence, we denote these architectures "GPP with SPx", where SPx stands for Special-Purpose extensions. They are similar, in concept, to ASIPs.

The next step is to consider a general-purpose processor architecture with general-purpose extensions, denoted the "EP with GPx" approach. This allows only new instructions that are indeed "general-purpose" primitives, i.e., versatile operation primitives that could potentially be used to compose more complicated functions in many algorithms and different applications. For example, on studying the design of symmetric-key block ciphers, arbitrary bit permutations have been identified as the only fundamental operation useful in the design of block ciphers that is very slow in existing processors [44, 45]. Such bit-level permutation operations can quickly achieve diffusion in block ciphers, a technique identified by Shannon [46] as fundamental to the design of block ciphers. Several different bit permutation instructions were recently proposed [44, 47, 48, 49] that allow any one of the n! permutations of the n bits in a register (or block to be enciphered) to be achieved in at most O(log(n)) instructions. This has been further reduced to O(1) cycles in both general-purpose [50] and ASIP [45] architectures. This is much faster than the O(n) instructions or cycles needed using existing instructions available in processors. The versatility of these bit permutation instruction primitives has also been demonstrated, for example, in the radical acceleration of the sorting of lists of bytes [51].

General-purpose instruction primitives for asymmetric (public-key) ciphers are very different from those used in symmetric ciphers. They tend to be focussed on accelerating a large number of modular multiplication operations on large operands, *e.g.*, 1024 bit numbers, in order to implement the main underlying operation – modular exponentiation, in algorithms such as RSA, Diffie-Hellman and El Gamal. Newer public-key algorithms such as Elliptic Curve Cryptography (ECC) [52] focus on computationally faster operations and smaller operands; ECC on binary fields perform polynomial multiplication rather than integer multiplication, using much shorter operands to achieve equivalent levels of security. For example, 163 bit ECC keys are thought to provide security equivalent to 1024 bit RSA keys. On the implementation side, the trend has been to implement dual-field multipliers [53] that can efficiently perform both integer and binary multiplications on the same multiplier circuit.

Recently, new instruction-set architectures have been proposed for tiny general-purpose processors with general-purpose extensions to accelerate both symmetric ciphers and asymmetric ciphers. For example, PAX [54] is a minimalist RISC (Reduced Instruction Set Computer) processor with a few additional (but simple to implement) instructions that accelerate both ECC public key operations and block cipher operations, and can achieve the link speeds of current and proposed wireless networks at low MHz rates. This approach is quite promising for embedded processors in smart cards, sensors and hand-held devices, where flexibility is required with small form factors and very low energy consumption.

The above macro architectural models often overlap and blend from one to another. The most effective and efficient custom instructions for ASIPs, or ISA primitives for "EP with GPx" architectures, are by no means resolved. More research is needed. Also, interaction between cryptography algorithm designers and processor designers is encouraged [55]. The last three rows of Figure 4 illustrate detailed architectural decisions that must be made to fully define the secure embedded processing architecture.

Security processing also involves security protocol processing, although a large portion of this is taken by the actual cryptography processing. New work is in progress to minimize the overhead in processing, data buffering and memory round-trips incurred by secure network communications protocols, such as IPSEC and SSL. While network processors have been designed to accelerate the processing of the traditional network protocol stack, "security protocol engines" have been proposed to accelerate the security protocol processing, *e.g.*, the IPSEC (Internet Protocol SECurity) portion of the IP network protocol processing [56, 57, 58].

5.2 Attack-Resistant Architectures

The security processing architectures discussed in the previous section implement basic security functions like confidentiality, integrity and authentication, but do not provide protection from software or physical attacks by malicious entities. Furthermore, architectural features that accelerate cryptography and security protocols do not protect against Denial-of-Service (DoS) attacks. A secure embedded system should also incorporate appropriate attack-resistant features.

Several attempts have been made to provide protection from attacks, but because of the scope of this problem, comprehensive solutions are still elusive and an open area of research. Rather, solutions have been proposed for specific applications, *e.g.*, Digital Rights Management (DRM) to try to mitigate software, music or movie piracy, or for specific attacks, *e.g.*, password theft. Application-specific solutions, *e.g.*, DRM for DVD players, may be the most widely deployed ones for embedded systems in the near future.

In Digital Rights Management, the ability to distribute essentially perfect copies of a piece of valuable software or multimedia file through the Internet or wireless networks poses a daunting challenge to large, established content owners and distributors. Here, the "attacker" may actually be the owner of the embedded system. For example, the owner or user of an entertainment device may try to extract, copy, and redistribute copies of music, movies

or software. Some commercial initiatives, such as Palladium by Microsoft and TCPA (Trusted Computing Platform Alliance), may have initially been motivated by the DRM interests of large content providers. These initiatives have now grown to encompass broader security requirements and have also been renamed Next Generation Secure Computing Base (NGSCB) [59, 60] and Trusted Computing Group (TCG) [61], respectively.

These two initiatives are based on the assumption that compatibility with existing non-secure operating systems (*e.g.*, Windows) and legacy application software must be preserved. This basic assumption may have led to the conceptual architectural model of a separate, parallel secure domain that co-exists and is protected from both non-secure applications programs and operating systems. The idea is to put a "brick wall" inside the processor, isolating both secure computations and memory, and protecting them from corruption, or even observation, by non-secure entities. This is like a mini firewall internal to the processor, and is achieved with a variety of new features in both software and hardware.

Key security objectives in NGSCB include strong process isolation, sealed memory, platform attestation, and a secure path to the user. Strong process isolation provides protected execution, implemented with mechanisms that include a new privilege domain as indicated by a new mode bit or privilege level (distinct from existing supervisor and user privilege levels). New instructions are used to enter and exit this secure mode or domain, with secure saving and restoring of contexts. Sealed memory involves tying some sensitive information to a given platform and software context, using encryption and hashing techniques. Such sealed memory can only be unsealed with the correct key in the correct software and hardware environment. Platform attestation is a method for a given computing device to try to assure a remote server that it is running appropriate software on acceptable hardware, with respect to certain security safeguards, *e.g.*, a corporate file server might only allow connections from computers that are in approved configurations. A secure path to the user allows a user to invoke a secure program in the trusted domain, without intervention of the existing non-secure operating system. This includes a secure path from the keyboard and a secure path to the display, *i.e.*, secure paths to certain input-output devices. LaGrande [62] is Intel's codename for new hardware features (in the microprocessor, chipset, buses, memory system, and input-output devices) that implement these and similar concepts.

ARM has also recently proposed a smaller set of features for secure processor cores (called TrustZone technology [63]) targeted at the embedded processor and System-On-Chip (SOC) markets. These protect against a set of attacks including password theft on login. MIPS has also proposed the security architecture SmartMIPS [64] for smartcards, which includes a set of Instruction Set Architecture (ISA) extensions for accelerating cryptographic functions, and also memory management functions to support secure programming and secure operating systems. None of these proposed security features for commercial microprocessors or cores are available for widespread deployment yet.

In academic research, architectural features for computer security were investigated about thirty years ago. However, in the last decade or two, possibly coincident with the wide approval for RISC processors as the architecture for high-performance processors, very little research or education in computer architecture has been devoted to security issues. Recently, this has changed. Some recent papers propose techniques to provide memory integrity through encryption and hashing [12, 13], and processor techniques to prevent machine hijacking and hostile code insertion by detecting return address corruption during buffer overflow attacks [65].

6. DESIGN METHODOLOGY AND TOOL REQUIREMENTS

As security emerges as a mainstream design issue, addressing some of the challenges outlined previously will require the support of appropriate design tools and methodologies. In this section, we briefly describe our vision for developments in this area.

Compared to an embedded system's functionality and other design metrics (*e.g.*, area, performance, power), security is currently specified by system architects in a vague and imprecise manner. Security experts are often the only people in a design team who have a complete understanding of the security requirements. This is a problem, since different aspects of the embedded system design process can impact security. Hence, design methodologies for secure embedded systems will have to start with techniques to specify security requirements in a way that can be easily communicated to the design team, and evaluated throughout the design cycle. Any attempt to specify security requirements needs to address the "level" of security desired, *e.g.*, what level of tamper resistance should be incorporated in the system. Security standards, such as the FIPS security requirements for cryptographic modules [66], and the Common Criteria for information technology security evaluation [67] could provide some initial guidelines in this direction, although they tend to be quite cumbersome and difficult to understand for the average designer.

Techniques for formal or semi-formal specifications of security requirements can enable the development of tools that validate and verify whether these requirements are met, at various stages in the design process. For example, formal verification techniques have been used to detect bugs in security protocol implementations [68, 69].

Time-to-market pressures in the semiconductor and embedded system industries lead to design processes that are increasingly based on the re-use of components from various sources. It will be particularly challenging to maintain security requirements in the face of these trends. It is very difficult, if not impossible, to guarantee the security of a system when the underlying components are untrusted. Furthermore, even the composition of individually secure components can expose unexpected security bugs due to their interaction.

During embedded system architecture design, techniques to map security requirements to alternative solutions, and to explore the associated tradeoffs in terms of cost, performance, and power consumption, would be invaluable in helping embedded system architects understand and make better design choices. For example, system architects would like to understand the performance and power impact of the processing architecture used to perform security processing, and the tamper-resistance schemes used.

During the hardware and software implementation processes, opportunities abound to improve the tamper resistance of the embedded system, as well as to mitigate the performance and power consumption impact of security features. For example, hardware synthesis (and software compilation) techniques to automatically minimize the dependence of power and execution time on sensitive data could help ensure that embedded system designs are highly tamper-resistant to side channel attacks by construction.

Some initial efforts towards design methodologies to support security are described in [41, 70, 71, 72].

In summary, as security becomes a requirement for a wide range of embedded systems, design tools and methodologies will play a critical role in empowering designers (who are not necessarily security experts) to address the design challenges described in this paper.

7. CONCLUSIONS

Today, secure embedded system design remains a field in its infancy in terms of research and pervasive deployment. Although historically, various security issues have been investigated in the context of cryptography, network security and computer security, the challenges imposed by the process of securing emerging environments or networks of embedded systems compel us to take a fresh look at the problem. The good news is that unlike the problem of providing security in cyberspace (where the scope is very large), securing the application-limited world of embedded systems is more likely to succeed in the near term. However, the constrained resources of embedded devices pose significant new challenges to achieving the desired levels of security.

We believe that a combination of advances in architectures and design methodologies would enable us to scale the next frontier of embedded system design, wherein, embedded systems will be "secure" to the extent required by the application and the environment. To realize this goal, we should look beyond the basic security functions of an embedded system and provide defenses against broad classes of attacks — all without compromising performance, area, energy consumption, cost and usability.

8. REFERENCES

[1] *Counterpane Internet Security, Inc.* `http://www.counterpane.com`.

[2] *ePaynews - Mobile Commerce Statistics.* `http://www.epaynews.com/statistics/mcommstats.html`.

[3] W. Stallings, *Cryptography and Network Security: Principles and Practice*. Prentice Hall, 1998.

[4] B. Schneier, *Applied Cryptography: Protocols, Algorithms and Source Code in C*. John Wiley and Sons, 1996.

[5] *IPSec Working Group*. `http://www.ietf.org/html.charters/ipsec-charter.html`.

[6] *SSL 3.0 Specification*. `http://wp.netscape.com/eng/ssl3/`.

[7] *Biometrics and Network Security*. Prentice Hall PTR, 2003.

[8] *OpenIPMP*. `http://www.openipmp.org`.

[9] *Internet Streaming Media Alliance*. `http:/www.isma.tv/home`.

[10] *MPEG Open Security for Embedded Systems (MOSES)*. `http://www.crl.co.uk/projects/moses/`.

[11] *Discretix Technologies Ltd.* (http://www.discretix.com).

[12] D. Lie, C. A. Thekkath, M. Mitchell, P. Lincoln, D. Boneh, J. C. Mitchell, and M. Horowitz, "Architectural support for copy and tamper resistant software," in *Proc. ACM Architectural Support for Programming Languages and Operating Systems (ASPLOS)*, pp. 168–177, 2000.

[13] G. E. Suh, D. Clarke, B. Gassend, M. van Dijk, and S. Devadas, "AEGIS: Architecture for Tamper-Evident and Tamper-Resistant Processing," in *Proc. Intl Conf. Supercomputing (ICS '03)*, pp. 160–171, June 2003.

[14] R. M. Best, *Crypto Microprocessor for Executing Enciphered Programs*. U.S. patent 4,278,837, July 1981.

[15] M. Kuhn, *The TrustNo 1 Cryptoprocessor Concept*. CS555 Report, Purdue University (http://www.cl.cam.ac.uk/~mgk25/), Apr. 1997.

[16] G. Hoglund and G. McGraw, *Exploiting Software: How to Break Code (*http://www.exploitingsoftware.com*)*. Addison-Wesley, 2004.

[17] J. Viega and G. McGraw, *Building Secure Software (*http://www.buildingsecuresoftware.com*)*. Addison-Wesley, 2001.

[18] G. McGraw, "Software Security," *IEEE Security & Privacy*, vol. 2, pp. 80–83, March–April 2004.

[19] R. Anderson and M. Kuhn, "Tamper resistance - a cautionary note," 1996.

[20] R. Anderson and M. Kuhn, "Low cost attacks on tamper resistant devices," in *IWSP: Intl. Wkshp. on Security Protocols, Lecture Notes on Computer Science*, pp. 125–136, 1997.

[21] O. Kommerling and M. G. Kuhn, "Design principles for tamper-resistant smartcard processors," in *Proc. USENIX Wkshp. on Smartcard Technology (Smartcard '99)*, pp. 9–20, May 1999.

[22] *Smart Card Handbook*. John Wiley and Sons.

[23] E. Hess, N. Janssen, B. Meyer, and T. Schutze, "Information Leakage Attacks Against Smart Card Implementations of Cryptographic Algorithms and Countermeasures," in *Proc. EUROSMART Security Conference*, pp. 55–64, June 2000.

[24] J. J. Quisquater and D. Samyde, "Side channel cryptanalysis," in *Proc. of the SECI*, pp. 179–184, 2002.

[25] J. Kelsey, B. Schneier, D. Wagner, and C. Hall, "Side Channel Cryptanalysis of Product Ciphers," in *Proc. ESORICS'98*, pp. 97–110, Sept. 1998.

[26] S. Ravi, A. Raghunathan, and S. Chakradhar, "Tamper Resistance Mechanisms for Secure Embedded Systems," in *Proc. Int. Conf. VLSI Design*, Jan. 2004.

[27] P. C. Kocher, "Timing attacks on implementations of Diffie-Hellman, RSA, DSS, and other systems," *Advances in Cryptology – CRYPTO'96, Springer-Verlag Lecture Notes in Computer Science*, vol. 1109, pp. 104–113, 1996.

[28] T. S. Messerges, E. A. Dabbish, and R. H. Sloan, "Examining Smart-Card Security under the Threat of Power Analysis Attacks," *IEEE Trans. Comput.*, vol. 51, pp. 541–552, May 2002.

[29] P. Kocher, J. Jaffe, and B. Jun, "Differential Power Analysis," *Advances in Cryptology – CRYPTO'99, Springer-Verlag Lecture Notes in Computer Science*, vol. 1666, pp. 388–397, 1999.

[30] *U.S. Patents Nos. 6,278,783; 6,289,455; 6,298,442; 6,304,658; 6,327,661; 6,381,699; 6,510,518; 6,539,092; 6,640,305; and 6,654,884.* http://www.cryptography.com/technology/dpa/licensing.html.

[31] D. Boneh, R. DeMillo, and R. Lipton, "On the importance of eliminating errors in cryptographic computations," *Cryptology*, vol. 14, no. 2, pp. 101–119, 2001.

[32] W. van Eck, "Electromagnetic radiation from video display units: an eavesdropping risk?," *Computers and Security*, vol. 4, no. 4, pp. 269–286, 1985.

[33] M. G. Kuhn and R. Anderson, "Soft Tempest: Hidden Data Transmission Using Electromagnetic Emanations," in *Proc. Int. Wkshp. on Information Hiding (IH '98)*, pp. 124–142, Apr. 1998.

[34] J. J. Quisquater and D. Samyde, "ElectroMagnetic Analysis (EMA): Measures and Counter-Measures for Smart Cards," *Lecture Notes in Computer Science (Smartcard Programming and Security)*, vol. 2140, pp. 200–210, 2001.

[35] K. Gandolfi, C. Mourtel, and F. Olivier, "Electromagnetic analysis: Concrete results," *Cryptographic Hardware and Embedded Systems*, pp. 251–261, 2001.

[36] S. Ravi, A. Raghunathan, and N. Potlapally, "Securing wireless data: System architecture challenges," in *Proc. Intl. Symp. System Synthesis*, pp. 195–200, October 2002.

[37] D. Boneh and N. Daswani, "Experimenting with electronic commerce on the PalmPilot," in *Proc. Financial Cryptography*, pp. 1–16, Feb. 1999.

[38] R. Karri and P. Mishra, "Minimizing Energy Consumption of Secure Wireless Session with QOS constraints," in *Proc. Int. Conf. Communications*, pp. 2053–2057, 2002.

[39] N. Potlapally, S. Ravi, A. Raghunathan, and N. K. Jha, "Analyzing the energy consumption of security protocols," in *Proc. Int. Symp. Low Power Electronics & Design*, pp. 30–35, Aug. 2003.

[40] *Xtensa application specific microprocessor solutions - Overview handbook*. Tensilica Inc. (http://www.tensilica.com), 2001.

[41] S. Ravi, A. Raghunathan, N. Potlapally, and M. Sankaradass, "System design methodologies for a wireless security processing platform," in *Proc. ACM/IEEE Design Automation Conf.*, pp. 777–782, June 2002.

[42] J. Burke, J. McDonald, and T. Austin, "Architectural support for fast symmetric-key cryptography," in *Proc. Intl. Conf. ASPLOS*, pp. 178–189, Nov. 2000.

[43] L. Wu, C. Weaver, and T. Austin, "Cryptomaniac: A Fast Flexible Architecture for Secure Communication," in *Proc. Int. Symp. Computer Architecture*, pp. 110–119, June 2001.

[44] R. B. Lee, Z. Shi, and X. Yang, "Efficient permutations for fast software cryptography," *IEEE Micro*, vol. 21, pp. 56–69, Dec. 2001.

[45] Z. Shi, X. Yang, and R. B. Lee, "Arbitrary bit permutations in one or two cycles," in *Proc. Int. Conf on Application-Specific Systems, Architectures and Processors*, pp. 237–247, June 2003.

[46] C. E. Shannon, "Communication theory of secrecy systems," *Bell System Tech. Journal*, vol. 28, pp. 656–715, October 1949.

[47] Z. Shi and R. Lee, "Bit Permutation Instructions for Accelerating Software Cryptography," in *Proc. IEEE Intl. Conf. Application-specific Systems, Architectures and Processors*, pp. 138–148, 2000.

[48] X. Yang and R. B. Lee, "Fast subword permutation instructions using omega and flip network stages," in *Proc. Int. Conf. Computer Design*, pp. 15–22, Sept. 2000.

[49] J. P. McGregor and R. B. Lee, "Architectural enhancements for fast subword permutations with repetitions in cryptographic applications," in *Proc. Int. Conf. Computer Design*, pp. 453–461, Sept. 2001.

[50] R. B. Lee, Z. Shi, and X. Yang, "How a processor can permute n bits in O(1) cycles," in *Proc. Hot Chips 14 – A Symposium on High Performance Chips*, Aug. 2002.

[51] Z. Shi, *Bit Permutation Instructions: Architecture, Implementation and Cryptographic Properties*. PhD thesis, Princeton University, 2004.

[52] K. Araki, T. Satoh, and S. Miura, "Overview of Elliptic Curve Cryptography," *Springer-Verlag Lecture Notes in Computer Science*, vol. 1431, pp. 29–48, 1998.

[53] E. Savas, A. F. Tenca, and C. K. Koc, "A Scalable and Unified Multiplier Architecture for Finite Fields GF(p) and GF(2n)," *Springer-Verlag Lecture Notes in Computer Science*, vol. 1965, pp. 277–292, 2000.

[54] A. M. Fiskiran and R. B. Lee, *PAX: A Datapath-Scalable Minimalist Cryptographic Processor for Mobile Environments (in Embedded Cryptographic Hardware: Design and Security)*. Nova Science Publishers (to be published), 2004.

[55] R. B. Lee, R. L. Rivest, M. J. B. Robshaw, Z. J. Shi, and Y. L. Yin, "Permutation operations in cipher design," in *Proc. Intl. Conf. on Information Technology (ITCC)s*, Apr. 2004.

[56] *HIFN Inc.* http://www.hifn.com.

[57] Corrent Inc. http://www.corrent.com.

[58] *Broadcom Corporation, BCM5840 Gigabit Security Processor*. http://www.broadcom.com.

[59] *Next-Generation Secure Computing Base (NGSCB)*. Microsoft Inc. (http://www.microsoft.com/resources/ngscb/productinfo.mspx).

[60] P. N. Glaskowsky, *Microsoft Details Secure PC Plans*. Microprocessor Report, In-stat/MDR, June 2003.

[61] *Trusted Computing Group*. (https://www.trustedcomputinggroup.org/home).

[62] *LaGrande Technology for Safer Computing*. Intel Inc. (http://www.intel.com/technology/security).

[63] R. York, *A New Foundation for CPU Systems Security*. ARM Limited (http://www.arm.com/armtech/TrustZone?OpenDocument), 2003.

[64] *SmartMIPS*. http://www.mips.com.

[65] J. P. McGregor, D. K. Karig, Z. Shi, and R. B. Lee, "A Processor Architecture Defense against Buffer Overflow Attacks," in *Proc. Int. Conf. on Information Technology: Research and Education (ITRE)*, pp. 243–250, Aug. 2003.

[66] *Security Requirements for Cryptographic Modules (FIPS PUB 140-2)*. http://csrc.nist.gov/publications/fips/fips140-2/fips1402.pdf.

[67] *Common Criteria for Information Technology Security*. http://csrc.nist.gov/cc.

[68] E. M. Clarke, S. Jha, and W. Marrero, "Using state space exploration and a natural deduction style message derivation engine to verify security protocols," in *Proc. IFIP Working Conf. on Programming Concepts and Methods*, 1998.

[69] G. Lowe, "Towards a completeness result for model checking of security protocols," in *Proc. 11th Computer Security Foundations Wkshp.*, 1998.

[70] N. Potlapally, S. Ravi, A. Raghunathan, and G. Lakshminarayana, "Algorithm exploration for efficient public-key security processing on wireless handsets," in *Proc. Design, Automation, and Test in Europe (DATE) Designers Forum*, pp. 42–46, Mar. 2002.

[71] L. Benini, A. Macii, E. Macii, E. Omerbegovic, F. Pro, and M. Poncino, "Energy-aware design techniques for differential power analysis protection," in *Proc. Design Automation Conf.*, pp. 36–41, June 2003.

[72] H. Saputra, N. Vijaykrishnan, M. Kandemir, M. J. Irwin, R. Brooks, S. Kim, and W. Zhang, "Masking the Energy Behavior of DES Encryption," pp. 84–89, Mar. 2003.