Security as a New Dimension in Embedded System Design

Paul Kocher†, Ruby Lee‡, Gary McGraw¶, Anand Raghunathan§ and Srivaths Ravi§
† Cryptography Research, San Francisco, CA
‡ EE Department, Princeton University, Princeton, NJ
¶ 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, despite these efforts, *security is often mis-construed by designers as the hardware or software implementation of specific cryptographic algorithms and security protocols. 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 and holistic 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

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

Keywords

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

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 mid- and high-end network equipment such as routers, gateways, firewalls, storage 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. Counterpane Internet Security, for example, estimated that the "I Love You" virus caused nearly one billion dollars in lost revenue 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, are used for secure communications in embedded systems. 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.
- 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 neworked sensors often operate under stringent resource constraints (limited battery capac-

ities, limited storage, *etc.*). 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 functionalities and standards.
- New functional security objectives, such as denial of service and digital content protection, require a higher degree of cooperation between security experts and embedded system architects / designers.

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 parties. When they send or receive sensitive or critical information using public networks or communications channels accessible to potential attackers, they should ideally provide basic security functions such as *data confidentiality*, *data integrity*, and *user authentication*. Data confidentiality protects sensitive information from undesired eavesdroppers. Data integrity ensures that the information has not been changed illegitimately. User authentication verifies that the information is sent and received by appropriate parties rather than masqueraders. These are basic security functions are often collectively termed *secure communications* and are required of many embedded systems used in medical, sensing, automotive, financial, military and many other applications.

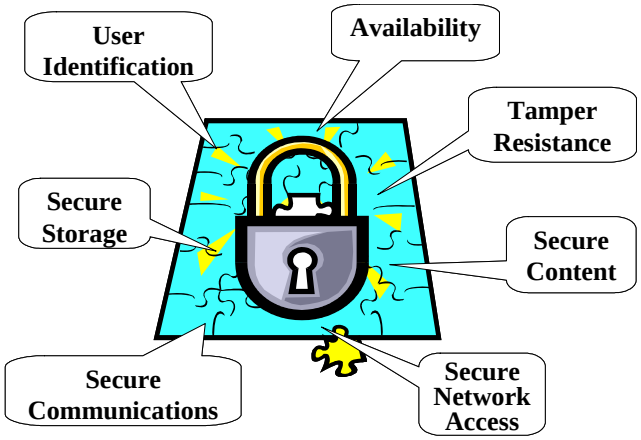

Figure 1: Common security requirements of embedded systems from an end-user perspective

In addition to secure communications, other security requirements have also been mentioned in the context of several embedded systems. Fig 1 shows some of these requirements from the perspective of an end-user. 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*). 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, quality of service, *etc.*. Embedded system *availability* then becomes a critical factor.

Since the mandate of embedded system security often requires protecting sensitive information (code or data) throughout its lifetime, *secure storage* and *content security* become crucial. *Secure storage* involves securing code or data in all system storage devices, external or internal to the system in question. *Content security* protects the rights of the digital content used in the system, and this issue is actively pursued by several content providers.

Finally, an undercurrent behind many of these embedded system security requirements is that we would like the requirements to be met even when the device is physically or logically probed by malicious entities. We refer to the property of the device being "secure" in the face of these threats as *tamper resistance*.

3. FUNCTIONAL SECURITY MEASURES

Several functional security primitives have been proposed in the context of network security. These include various cryptographic algorithms used for encrypting and decrypting data, and for checking the integrity of data. Most cryptographic algorithms fall into one of three classes – symmetric ciphers, asymmetric ciphers and hashing 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 or privacy of data is ensured during transmission. Examples of symmetric ciphers include DES, 3DES, AES, and RC4. Most symmetric ciphers are constructed from computationally light-weight operations such as permutations, substitutions, *etc.* Thus, they are well suited for securing bulk data transfers.
- *Hashing algorithms* such as MD5 and SHA convert arbitrary messages into unique fixed-length values, thereby providing unique "thumbprints" 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), on the other hand, typically use a private (secret) key for decryption, and a related public (non-secret) key for encryption. Encryption requires only the public key, which is not sufficient for decryption. Digital signatures are also constructed using public key cryptography and hashes. Asymmetric algorithms (*e.g.*, RSA, Diffie-Hellman, *etc.*) rely on the use of more computationally intensive mathematical functions such as modular exponentiation for encryption and decryption. Therefore, they are often used for security functions complementary to secure bulk data transfers such as exchanging symmetric cipher keys.

Security solutions to meet the various security requirements outlined in the previous section typically rely on security mechanisms that use a combination of the aforementioned cryptographic primitives in a specific manner (*i.e.*, security protocols). 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 authentication. 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], MPEG [9], ISMA [10] and MOSES [11], 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 bus monitor logic to block illegal accesses to protected areas in the memory [12], authentication of firmware that executes on the system, application isolation to preserve the privacy and integrity of code and data assoicated with a given application or a process [13], HW/SW techniques to preserve the privacy and integrity of data throughout the memory hierarchy [14], execution of encrypted code in processors to prevent bus probing [15, 16] *etc.*

4. DESIGNING SECURE EMBEDDED SYSTEM IMPLEMENTATIONS

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

4.1 Embedded Software Attacks and Countermeasures

Why is software so difficult to control? Three factors, which we call the *Trinity of Trouble* — complexity, extensibility and connectivity — conspire to make managing risks in software a major challenge.

- **Complexity:** Software is complicated, and will become even more complicated in the near future. The equation is simple: more lines of code equals more bugs. As embedded systems converge with the Internet and more code is added, they are 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. In theory, we could analyze and prove that a small program was free of problems, but this task is impossible for even the simplest desktop systems today. For reasons of efficiency, C and C++ are very popular languages for embedded systems.

- **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, and so on). 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 practice.

4.1.1 An Example SW Attack: The Hardware Virus

Software attacks against the kernel, such as those carried out by rootkits, demonstrate the kinds of attack that embedded systems are exposed to [17]. 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/write to the BIOS memory on the motherboard or in peripheral hardware.

In the "old days", 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. Of course this is not very cost effective, so 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 floating around on various controller cards and the motherboard. These flash-ROM chips are almost never fully utilized and this leaves us tremendous amounts of room to store backdoor information and viruses. The compelling thing about using these memory spaces is that they are hard to audit and almost never visible to software running on a system. To access the hardware memory requires driver level access. Furthermore, this memory is immune against reboots and system re-installation.

One key advantage of a hardware virus is that it will survive a reboot and a 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 the VX-Works OS. Within the system are several flash RAM 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 the firmware may just be used for data storage. Nonvolatile 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, antilock 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 SW attacks

A central and critical aspect of the computer security problem is a software 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 [17]. 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 life cycle, 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-issue part of the system (say, 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 can not test quality into a piece of software, you can not spray paint security features onto a design and expect it to become secure.

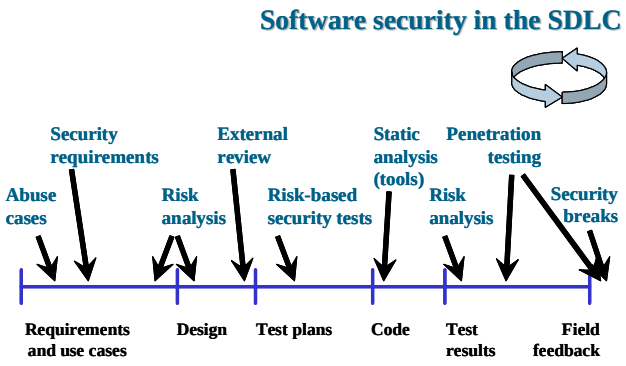

Figure 2: Software security best practices applied to various software artifacts

As practitioners become aware of software security's importance, they are increasingly adopting and evolving a set of best practices to address the problem [18, 19]. There's 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 laid out according to a traditional waterfall model in this picture, most organizations follow an iterative approach today, 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 (say, 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) [18].
- **The code level:** Static analysis tools — tools that scan source code for common vulnerabilities — can discover implementation bugs at the code level.

Note that 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 fielded systems during use for security breaks. Simply put, attacks will happen, regardless of the strength of design and implementation, so monitoring software behavior is an excellent defensive technique.

4.2 Tamper-resistant Hardware

There are a wide range of attacks (different from software attacks) that exploit the system implementation and/or identifying properties of the implementation to break the security of an embedded system. These are called *physical* and *side-channel* attacks [20, 21, 22, 23, 24, 25, 26, 27]. 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 such as timing attacks, fault induction techniques, power and electromagnetic analysis based attacks, *etc.* In the sections that follow, we will be examining some of these 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 microprobing techniques become necessary [22, 23]. 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, one of us (Paul Kocher) 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), 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 don't 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 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/output values with timing measurements. For further information about timing attacks, see [28].

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 losses due to the parasitic capacitance of the internal wiring. Power consumption increases if more state transitions occur and if transitions are occurring predominately 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 [29] that the brute-force search space for one SW 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

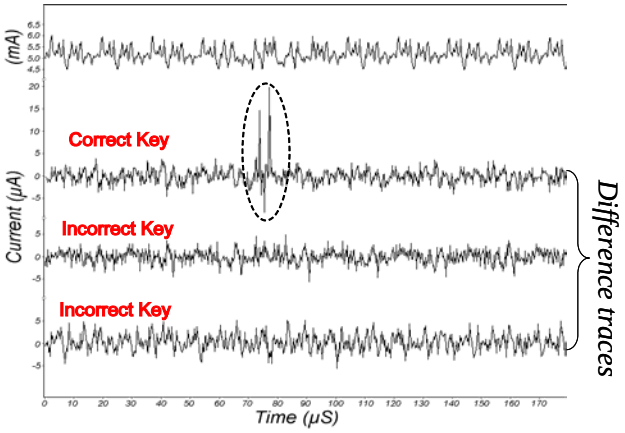

Figure 3: Power consumption traces generated during a DPA attack on a DES implementation

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. Typically, these power traces are collected using high-speed A/D 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 the Data Encryption Standard (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 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 [30].

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 [31]. 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 [32].

4.2.5 Electromagnetic Analysis

Electromagnetic analysis attacks (EMA) have been well documented since the eighties, when it was shown in [33] 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 [34]. 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 [35, 36].

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 in 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 basic 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.

5.1 Security-Aware Processing Architectures

Most functional security measures are designed using appropriate cryptographic algorithms like symmetric ciphers, asymmetric ciphers and hasing schemes. However, several studies have revealed that such ciphers can be rather compute-intensive and power hungry, which is a challenge for resource constrained embedded systems.

The computational requirements of standard security protocols such as SSL tend to be significantly higher than the processor capabilities available in embedded systems such as wireless handheld devices [37, 38]. Data presented in [37] reveal that the total processing requirements for software implementations of SSL executing on an iPAQ handheld (235 MIPS StrongARM processor) is around 651.3 MIPS, when the protocol uses 3DES for encryption/decryption and SHA for message authentication, at 10 Mbps (current and emerging data rates for wireless LAN technologies are in the range of 2-60 Mbps).

The energy consumption characteristics of security processing on battery-powered wireless handhelds have also been studied recently [39, 40]. When a device such as Symbol PPT2800 PocketPC is securely transmitting data, data from [39]) shows that a considerable part (nearly 21%) of the overall energy consumption is spent on security processing.

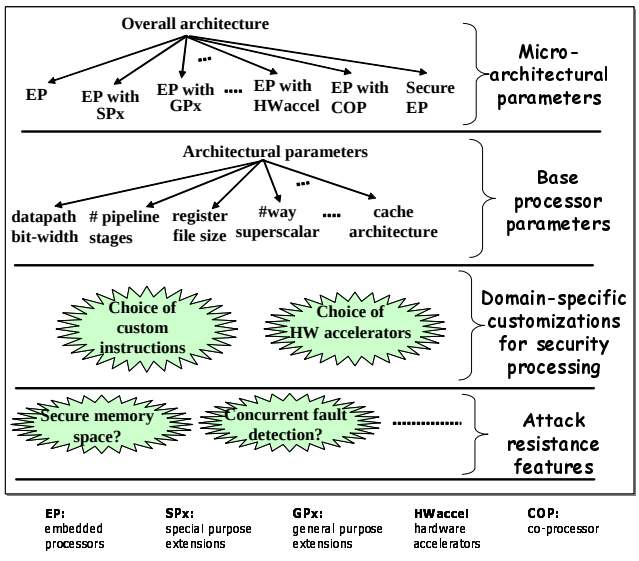

Figure 4: Architectural design space for security-aware processing

While efficiency in performance and energy consumption must be reflected in a security processing architecture, the architecture must also be capable of detecting and responding to various kinds of attacks that are possible due to different architectural (HW or SW) vulnerabilities. Keeping these considerations in mind, we evolved the architectural design space for *security-aware processing* as illustrated in Figure 4. The design decisions to be taken include:

- The macro-architectural style (embedded general-purpose processor (EP) *vs.* application-specific instruction set processor (ASIP) *vs.* EP with custom hardware accelerators connected to the processor bus, *etc.*)
- Tuning the architecture of the base processor by optimizing parameters such as data path bit-width, number of pipeline stages, register file size, *etc.*
- Choosing the functionality to be implemented as custom instructions or hardware accelerators.
- Selecting the attack-resistance features (from the range of available mechanisms) to be included in the architecture, *e.g.*, additional redundant circuitry for power attack resistance, an enhanced memory management unit to manage a secure memory space, fault detection circuitry, *etc.*

The following sections will detail the evolution of various technologies for both efficient security processing and attack-resistant architectures.

5.1.1 Security Processing Architectures

Past embedded architecture approaches have hardwired algorithms in ASICs (Application Specific Integrated Circuits). While fixed function ASIC approaches are suitable when one or only a few ciphers are needed, they are less desirable when a variety of ciphers are desired to support multiple security protocols, emerging standards and interoperability with many devices. The use of FPGAs (Field Programmable Gate Arrays) allows some reconfiguration of ciphers, but often does not meet cost, energy consumption and performance goals simultaneously. Another approach has been to use a general-purpose processor core with accelerator chips 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.

Another approach is to tightly integrate such acceleration hardware with the processor core itself and invoke it with custom instructions. For example, embedded processors such as Xtensa [41] 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 [42] is a proposal for a custom, low-overhead instruction set for a set of symmetric, asymmetric and hashing algorithms.

In the general-purpose processor arena, new instructions have also been proposed to accelerate the class of existing symmetric ciphers [43, 44]. These new instructions typically perform a smaller function rather than an entire round of a block cipher. However, they are still fairly specific to the block ciphers considered. An alternative approach has been to consider the design of future as well as existing ciphers, leading to the addition of only versatile, "general-purpose" instruction primitives to a processor. These should accelerate not only existing ciphers but also enable the design of future ciphers that can achieve the same levels of security, with shorter computation time while consuming less energy. For example, bit permutation instructions are identified [45, 46] as the only o perations in many block ciphers that are very slow on existing processors. Therefore, novel bit permutation instructions have recently been proposed [47] that allow any arbitrary permutation of the n bits in a register (or block to be enciphered) to be achieved in at most log(n) instructions, much faster than the O(n) instructions or cycles needed with the instructions available in existing processors. Such bit-level permutation operations can quickly achieve diffusion in block ciphers, a technique identified by Shannon [48] as fundamental to the design of block ciphers.

Public key algorithms require operations very different from those used in symmetric ciphers. Widely used algorithms like RSA, Diffie-Helman and El-Gamal require huge numbers of modular muliplication operations on large operands, *e.g.*, 1024 bit numbers, in order to implement their main underlying operation – modular exponentiation. Here, the trend is not only to design large fast multipliers but also to find newer mathematical basis for public key algorithms that are easier to compute and use smaller operands, such as Elliptic Curve Cryptography (ECC) [49]. ECC on binary fields perform polynomial multiplication rather than integer multiplicia-tion, 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 [50] that can efficiently perform both integer and binary multiplications on the same multipler circuit.

Very recently, work on tiny processors that accelerate both symmetric ciphers and public key algorithms, as well as secure hashes, have been proposed. For example, PAX [51] 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 ciphers, and can achieve the link speeds of current and proposed wireless networks at low MHz rates. These are quite promising as embedded processors in smart cards, sensors and hand-held devices that require flexibility with small form factors and low energy cinsumption.

Moving beyond cryptography acceleration, work is in progress to accelerate secure communications protocols, such as IPSEC and SSL, as well. 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.

5.1.2 Secure or Attack-Resistant Architectures

The security processing architectures discussed in the previous section are by no means *secure* since they do not provide any protection from attacks by malicious entities. Several attempts have been made to provide such protection from attacks, but because of the scope of this problem, comprehensive solutions are still an 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 DRM, 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 in possession of the embedded system. For example, the owner or user of an entertainment device may try to extract 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) [52, 53] and Trusted Computing Group (TCG) [54], respectively.

These two initiatives are based on the assumption that compatibility with existing non-secure operating systems (*e.g.*, Windows) and legacy applications software must be preserved. This basic assumption led to the 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 sav-

ing 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 [55] is Intel's codename for new hardware features (in the microprocessor, chip-set, 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 [56]) 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 [57] for smartcards, which not only includes a set of Instruction Set Architecture (ISA) extensions for accelerating cryptographic functions but also the memory management functions necessary 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 proposed 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 [13, 14], and processor techniques to prevent machine hijacking and hostile code insertion by detecting return address corruption during buffer overflow attacks [58].

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 [59], and the Common Criteria for information technology security evaluation [60] 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 these whether 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 [61, 62].

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 attendant tradeoffs in terms of cost, performance, and power consumption, would be invaluable in helping embedded system architects understanding and making better design choices. For example, system architects would like to understand the performance and power impact of the 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 mitigate the performance and power consumption impact of security features. For example, hardware synthesis (and software compilation) techniques to automatically ensure that minimize the dependence of power and execution time on sensitive data could help ensure design embedded systems that are highly tamper-resistant to side channel attacks by construction. Some initial efforts along these directions are described in [42, 63, 64, 65].

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 pervasive deployment and research. Although historically, various security issues have been investigated in the context of network security and cryptography, 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 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" in every sense of the word. 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.

Acknowledgments: We acknowledge all brand or product names that are trademarks or registered trademarks of their respective owners.

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] *Moving Picture Experts Group*. `http://mpeg.telecomitalialab.com`.

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

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

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

[13] 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.

[14] 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.

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

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

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

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

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

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

[21] 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.

[22] 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.

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

[24] 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.

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

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

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

[28] 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.

[29] 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.

[30] 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.

[31] *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`.

[32] 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.

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

[34] 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.

[35] 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.

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

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

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

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

[40] 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.

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

[42] 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.

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

[44] 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.

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

[46] 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.

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

[49] 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.

[50] 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.

[51] 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.

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

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

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

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

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

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

[58] 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.

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

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

[61] 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.

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

[63] 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.

[64] 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.

[65] 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.