Security in Embedded Systems: Design Challenges

SRIVATHS RAVI and ANAND RAGHUNATHAN
NEC Laboratories America
PAUL KOCHER
Cryptography Research
and
SUNIL HATTANGADY
Texas Instruments Inc.

Many modern electronic systems—including personal computers, PDAs, cell phones, network routers, smart cards, and networked sensors to name a few—need to access, store, manipulate, or communicate sensitive information, making security a serious concern in their design. Embedded systems, which account for a wide range of products from the electronics, semiconductor, telecommunications, and networking industries, face some of the most demanding security concerns—on the one hand, they are often highly resource constrained, while on the other hand, they frequently need to operate in physically insecure environments.

Security has been the subject of intensive research in the context of general-purpose computing and communications systems. However, security is often misconstrued by embedded system designers as the addition of features, such as specific cryptographic algorithms and security protocols, to the system. In reality, it is *a new dimension* that designers should consider throughout the design process, along with other metrics such as cost, performance, and power.

The challenges unique to embedded systems require new approaches to security covering all aspects of embedded system design from architecture to implementation. Security processing, which refers to the computations that must be performed in a system for the purpose of security, can easily overwhelm the computational capabilities of processors in both low- and high-end embedded systems. This challenge, which we refer to as the "security processing gap," is compounded by increases in the amounts of data manipulated and the data rates that need to be achieved. Equally daunting is the "battery gap" in battery-powered embedded systems, which is caused by the disparity between rapidly increasing energy requirements for secure operation and slow improvements in battery technology. The final challenge is the "assurance gap," which relates to the gap between functional security measures (e.g., security services, protocols, and their constituent cryptographic algorithms) and actual secure implementations. This paper provides an introduction to the challenges involved in secure embedded system design, discusses recent advances in addressing them, and identifies opportunities for future research.

Authors' addresses: S. Ravi and A. Raghunathan, NEC Laboratories America, Princeton, NJ; email: {sravi,anand}@nec-labs.com; P. Kocher, Cryptography Research, San Francisco, CA; email: paul@cryptography.com; Sunil Hattangady, Texas Instruments Inc., Dallas, TX; email: sunil@ti.com.
Permission to make digital or hard copies of part or all of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or direct commercial advantage and that copies show this notice on the first page or initial screen of a display along with the full citation. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, to republish, to post on servers, to redistribute to lists, or to use any component of this work in other works requires prior specific permission and/or a fee. Permissions may be requested from Publications Dept., ACM, Inc., 1515 Broadway, New York, NY 10036 USA, fax: +1 (212) 869-0481, or permissions@acm.org.
© 2004 ACM 1539-9087/04/0800-0461 $5.00

Categories and Subject Descriptors: C.0 [**Computer Systems Organization**]: General—*System architectures*; C.2.0 [**Computer Systems Organization**]: Computer-Communication Networks—*General (Security and protection)*; C.3 [**Computer Systems Organization**]: Special-purpose and application-based systems—*Real-time and Embedded Systems*; C.5.3 [**Computer Systems Organization**]: Computer Systems Implementaion—*VLSI Systems*; D.4.6 [**Software**]: Operating Systems—*Security and Protection*; E.3 [**Data**]: Data encryption; K.6.5 [**Computing Milieux**]: Management of Computing and Information Systems—*Security and Protection*

General Terms: Security, Design

Additional Key Words and Phrases: Embedded systems, security, architecture, hardware design, processing requirements, battery life, security protocols, cryptographic algorithms, encryption, decryption, authentication, security attacks, tamper resistance

1. INTRODUCTION

Today, an increasing number of embedded systems need to deal with security in one form or another—from low-end systems such as wireless handsets, networked sensors, and smart cards, to high-end systems such as network routers, gateways, firewalls, and 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 attacks they face. It has been observed that the cost of insecurity in electronic systems can be very high. A recent computer crime and security survey [Computer Security Institute] from the Computer Security Institute (CSI) and Federal Bureau of Investigation (FBI) revealed that just 223 organizations sampled from various industry sectors had lost hundreds of millions of dollars due to computer crime. Figure 1(a) summarizes the costs from various security attacks including theft of proprietary information, financial fraud, virus attack, and denial of service. Other estimates include a staggering figure of nearly $1 billion in productivity loss due to the "I Love You" virus attack [Counterpane]. With an increasing proliferation of such attacks, it is not surprising that inadequate security is becoming a bottleneck to the adoption of next-generation data applications and services. For example, in the mobile appliance world, a recent survey [ePaynews] revealed that nearly 52% of cell phone users and 47% of PDA users feel that security is the single largest concern preventing the adoption of mobile commerce (see Figure 1(b)).

With the evolution of the Internet, information and communications security has gained significant attention [World Wide Web Consortium 1998; U.S. Department of Commerce 1999]. A wide variety of challenging security concerns must be addressed, including data confidentiality and integrity, authentication, privacy, denial of service, nonrepudiation, and digital content protection. Various security protocols and standards such as WEP [IEEE Standard 802.11], WTLS [WAP 2002], IPSec [IPSec], and SSL [SSL] are used today to secure a range of data services and applications. While security protocols and cryptographic algorithms 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 being an afterthought into a mainstream system (hardware/software) design issue.

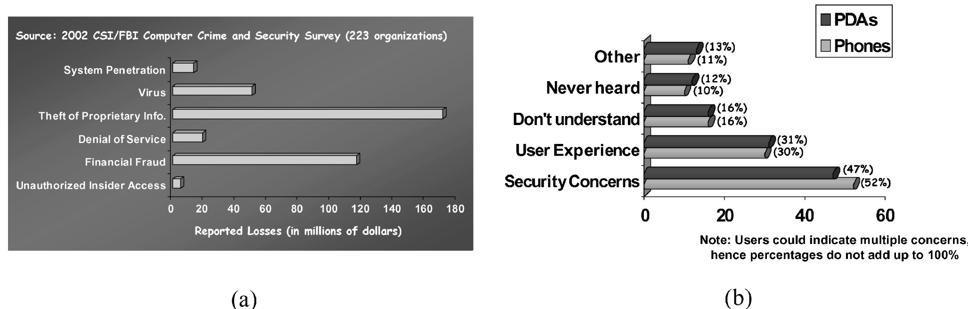


Fig. 1. (a) The cost of insecurity (*source*: [Computer Security Institute]) and (b) factors preventing the adoption of mobile commerce (*source*: [ePaynews]).

For example:

- The processing capabilities of many embedded systems are easily overwhelmed by the computational demands of security processing, leading to failures in sustaining required data rates or number of connections.
- Battery-driven systems and small form-factor devices such as PDAs, cell phones, and networked sensors are often severely resource constrained. It is challenging to implement security in the face of limited battery capacities, limited memory, and so on.
- An ever increasing range of attack techniques for breaking security, such as software, physical, and side-channel attacks, require that the system be secure even when it can be logically or physically accessed by malicious entities. Countermeasures to these attacks need to be built in during system design.

This paper presents an overview of the challenges in the area of secure embedded system design. Section 2 introduces the reader to various security concerns in embedded systems. Section 3 provides a brief overview of basic security concepts. Section 4 describes the design challenges that arise from various embedded system security requirements. Sections 5 and 6 examine some of these challenges in detail. Section 5 analyzes the performance, battery life, and flexibility issues associated with security processing in embedded systems, while Section 6 provides an overview of the various threats possible to an embedded system. Section 7 presents case studies that depict how advanced architectures can be used to address some of these challenges. Section 8 concludes with a brief look ahead into the secure embedded system design roadmap.

2. SECURITY REQUIREMENTS OF EMBEDDED SYSTEMS

Embedded systems often provide critical functions that could be sabotaged by malicious entities. Before discussing the common security requirements of embedded systems, it is important to note that there are many entities involved in a typical embedded system design, manufacturing, and usage chain. Security requirements vary depending on whose perspective we consider.

For example, let us consider a state-of-the-art cellular handset that is capable of wireless voice, multimedia, and data communications. Figure 2 illustrates

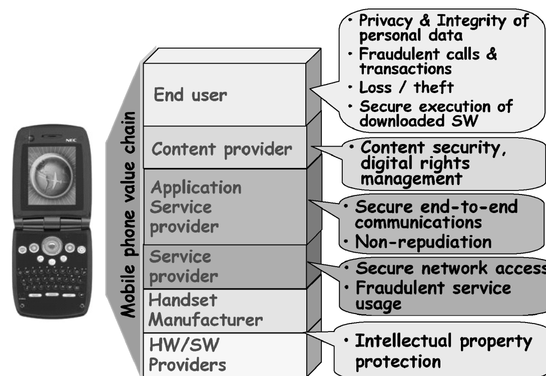


Fig. 2. Security requirements for a cell phone.

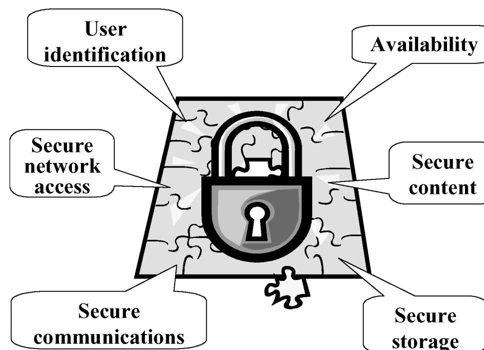


Fig. 3. Common security requirements of embedded systems.

security requirements from the viewpoint of the provider of HW/SW components inside the cell phone (e.g., baseband processor, operating system), the cell phone manufacturer, the cellular service provider, the application service provider (e.g., mobile banking service), the content provider (e.g., music or video), and the end user of the cell phone. The end user's primary concerns may include 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 untrusted (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.

Figure 3 lists the typical security requirements seen across a wide range of embedded systems, which are described as follows:

- *User identification* refers to the process of validating users before allowing them to use the system.

- *Secure network access* provides a network connection or service access only if the device is authorized.
- *Secure communications* functions include authenticating communicating peers, ensuring confidentiality and integrity of communicated data, preventing repudiation of a communication transaction, and protecting the identity of communicating entities.
- *Secure storage* mandates confidentiality and integrity of sensitive information stored in the system.
- *Content security* enforces the usage restrictions of the digital content stored or accessed by the system.
- *Availability* ensures that the system can perform its intended function and service legitimate users at all times, without being disrupted by denial-of-service attacks.

3. BASIC SECURITY CONCEPTS

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. Broadly, cryptographic algorithms can be classified into three classes—symmetric ciphers, asymmetric ciphers, and hashing algorithms, which are briefly described below (for a detailed introduction to cryptography, we refer the reader to Stallings [1998] and Schneier [1996]).

- *Symmetric ciphers* require the sender and receiver to use the same secret key to encrypt and decrypt data. They are typically used for ensuring confidentiality of data, and can be chosen from two classes—*block* and *stream* ciphers. Block ciphers operate on similar-sized blocks of plaintext (original data) and ciphertext (encrypted data). Examples of block ciphers include DES, 3DES, AES, and so on. Stream ciphers such as RC4 convert a plaintext to ciphertext one bit (or byte) at a time. For both classes of symmetric ciphers, encryption or decryption then proceeds through a repeated sequence (rounds) of mathematical computations. For example, block ciphers such as 3DES, IDEA, and AES use operations such as permutations and substitutions.
- *Asymmetric ciphers* (also called public-key algorithms), on the other hand, use a private (secret) key for decryption, and a related public (nonsecret) key for encryption or verification. They are typically used in security protocols for verifying certificates that identify communicating entities, generating and verifying digital signatures, and for exchanging symmetric cipher keys. These algorithms rely on the use of computationally intensive mathematical functions, such as modular exponentiation, for encryption and decryption. RSA, Diffie–Hellman, and so on are examples of asymmetric ciphers.
- *Hashing algorithms* such as MD5 and SHA provide ways of mapping messages (with or without a key) into a fixed-length value, thereby providing "signatures" for messages. Thus, integrity checks can be performed on communicated messages by (a) having the sender "securely sending" the actual hash value of a message along with the message itself, (b) allowing the receiver to

compute the hash value of the received message, and (c) comparing the two signatures to verify message integrity.

Security solutions to meet the various security requirements outlined in Section 2 typically rely on the aforementioned cryptographic primitives, or on security mechanisms that use a combination of these primitives in a specific manner (e.g., security protocols). Various security technologies and mechanisms have been designed around these cryptographic algorithms in order to provide specific security services. For example:

- Security protocols provide ways of ensuring secure communication channels to and from the embedded system. IPSec [IPSec] and SSL [SSL] are popular examples of security protocols, widely used for Virtual Private Networks (VPNs) and secure web transactions, respectively (we will be examining security protocols in greater detail in Section 5).
- Digital certificates provide ways of associating identity with an entity, while biometric technologies [Reid 2003] 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 [OpenIPMP], MPEG [MPEG], ISMA [ISMA], and MOSES [MOSES], provide secure frameworks for protecting 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 hardware to monitor bus transactions and block illegal accesses to protected areas in the memory [Discretix], authentication of firmware that executes on the system, application isolation to preserve the privacy and integrity of code and data associated with a given application or process [Lie et al. 2000], HW/SW techniques to preserve the privacy and integrity of data throughout the memory hierarchy [Suh et al. 2003], execution of encrypted code [Best 1981; Kuhn 1997], and so on.

4. SECURE EMBEDDED SYSTEM DESIGN CHALLENGES

Designers of a large and increasing number of embedded systems need to support various security solutions in order to deal with one or more of the security requirements described earlier. These requirements present significant bottlenecks during the embedded system design process, which are briefly described below:

- *Processing Gap*: Existing embedded system architectures are not capable of keeping up with the computational demands of security processing, due to increasing data rates and complexity of security protocols. These shortcomings are most felt in systems that need to process very high data rates or a large number of transactions (e.g., network routers, firewalls, and web servers), and in systems with modest processing and memory resources (e.g., PDAs,

wireless handsets, and smartcards). In this paper, we will examine the two sides of the processing gap issue (requirements and availability) and study various solutions proposed to address this mismatch.

- *Battery Gap*: The energy consumption overheads of supporting security on battery-constrained embedded systems are very high. Slow growth rates in battery capacities (5–8% per year) are easily outpaced by the increasing energy requirements of security processing, leading to a battery gap. Various studies [Carman et al. 2000; Perrig et al. 2002; Potlapally et al. 2003] show that the widening battery gap would require designers to make energy-aware design choices (such as optimized security protocols, custom security hardware, and so on) for security.
- *Flexibility*: An embedded system is often required to execute multiple and diverse security protocols and standards in order to support (i) multiple security objectives (e.g., secure communications, DRM, and so on), (ii) interoperability in different environments (e.g., a handset that needs to work in both 3G cellular and wireless LAN environments), and (iii) security processing in different layers of the network protocol stack (e.g., a wireless LAN enabled PDA that needs to connect to a virtual private network, and support secure web browsing may need to execute WEP, IPSec, and SSL). Furthermore, with security protocols being constantly targeted by hackers, it is not surprising that they keep continuously evolving (see also Section 5.4). It is, therefore, desirable to allow the security architecture to be flexible (programmable) enough to adapt easily to changing requirements. However, flexibility may also make it more difficult to gain assurance of a design's security.
- *Tamper Resistance*: Attacks due to malicious software such as viruses and trojan horses are the most common threats to any embedded system that is capable of executing downloaded applications [Howard and LeBlanc 2002; Hoglund and McGraw 2004; Ravi et al. 2004]. These attacks can exploit vulnerabilities in the operating system (OS) or application software, procure access to system internals, and disrupt its normal functioning. Because these attacks manipulate sensitive data or processes (integrity attacks), disclose confidential information (privacy attacks), and/or deny access to system resources (availability attacks), it is necessary to develop and deploy various HW/SW countermeasures against these attacks.

  In many embedded systems such as smartcards, new and sophisticated attack techniques, such as bus probing, timing analysis, fault induction, power analysis, electromagnetic analysis, and so on, have been demonstrated to be successful in easily breaking their security [Ravi et al. 2004; Anderson and Kuhn 1996, 1997; Kommerling and Kuhn 1999; Rankl and Effing; Hess et al. 2000; Quisquater and Samyde 2002; Kelsey et al. 1998]. Tamper resistance measures must, therefore, secure the system implementation when it is subject to various physical and side-channel attacks.

  Later in this paper (see Section 6), we will discuss some examples of embedded system attacks and related countermeasures.
- *Assurance Gap*: It is well known that truly reliable systems are much more difficult to build than those that merely work most of the time. Reliable

systems must be able to handle the wide range of situations that may occur by chance. Secure systems face an even greater challenge: they must continue to operate reliably despite attacks from intelligent adversaries who intentionally seek out undesirable failure modes. As systems become more complicated, there are inevitably more possible failure modes that need to be addressed. Increases in embedded system complexity are making it more and more difficult for embedded system designers to be confident that they have not overlooked a serious weakness.

- *Cost*: One of the fundamental factors that influence the security architecture of an embedded system is cost. To understand the implications of a security-related design choice on the overall system cost, consider the decision of incorporating physical security mechanisms in a single-chip cryptographic module. The Federal Information Processing Standard (FIPS 140-2) [FIPS] specifies four increasing levels of physical (as well as other) security requirements that can be satisfied by a secure system. Security Level 1 requires minimum physical protection, Level 2 requires the addition of tamper-evident mechanisms such as a seal or enclosure, while Level 3 specifies stronger detection and response mechanisms. Finally, Level 4 mandates environmental failure protection and testing (EFP and EFT), as well as highly rigorous design processes. Thus, we can choose to provide increasing levels of security using increasingly advanced measures, albeit at higher system costs, design effort, and design time. It is the designer's responsibility to balance the security requirements of an embedded system against the cost of implementing the corresponding security measures.

5. SECURITY PROCESSING REQUIREMENTS AND ARCHITECTURES

Security processing refers to the computations that must be performed in a system for the purpose of security. In this section, we will analyze the challenges imposed by security processing on embedded system design in greater detail, using the popular Secure Sockets Layer (SSL) protocol as an example. Section 5.1 examines the working of the SSL protocol. Section 5.2 analyzes the workload imposed by SSL on a typical embedded processor. Section 5.3 further examines how various embedded processors fare against the workload imposed by security processing and provides a quantitative analysis of the processing gap in both low- and high-end embedded systems. Section 5.4 explores the flexibility concerns arising from the evolutionary nature of security protocols, while Section 5.5 presents the impact of security processing on battery life in battery-powered embedded systems. Finally, Section 5.6 presents a taxonomy of security processing architectures that can address various aspects of the outlined problems.

5.1 Anatomy of a Security Protocol

The cryptographic primitives described in Section 3 are used to provide the basic services offered by most security protocols: encryption, peer authentication, and integrity protection for data exchanged over the underlying unprotected networks. In this section, we will examine the functioning of a popular

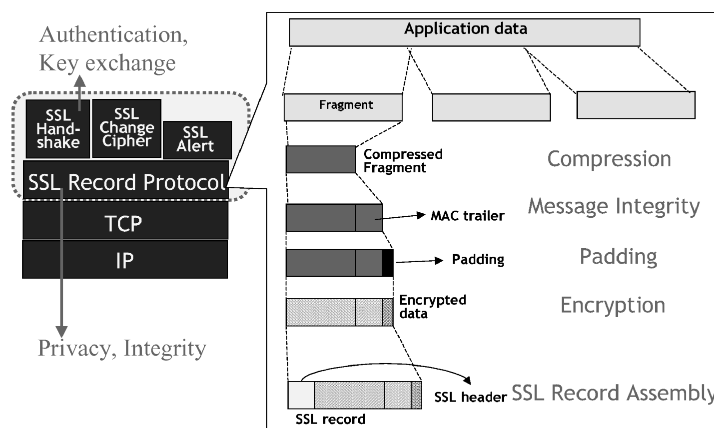


Fig. 4. The SSL protocol, with an expanded view of the SSL record protocol.

security protocol SSL [SSL], which is widely used for secure connection-oriented transactions.

The SSL protocol is typically layered on top of the transport layer of the network protocol stack, and is either embedded in the protocol suite or is integrated with applications such as web browsers. The SSL protocol itself consists of two main layers as shown in Figure 4. The SSL record protocol, which forms the first layer, provides the basic services of confidentiality and integrity. The second layer includes the SSL handshake, SSL change cipher, and SSL alert protocols. Let us now examine how the SSL record protocol is used to process application data. The first step involves breaking the application data into smaller fragments. Each fragment is then optionally compressed. The next step involves computing a message authentication code (MAC), which facilitates message integrity. The compressed message plus MAC is then encrypted using a symmetric cipher. If the symmetric cipher is a block cipher, then a few padding bytes may be added. Finally, an SSL header is attached to complete the assembly of the SSL record. The header contains various fields including the higher-layer protocol used to process the attached fragment.

Of the three higher-layer protocols, SSL handshake is the most complex and consists of a sequence of steps that allows a server and client to authenticate each other and negotiate the various cipher parameters needed to secure a session. For example, the SSL handshake protocol is responsible for negotiating a common suite of cryptographic algorithms (cipher-suite), which can then be used for session key exchange, authentication, bulk encryption, and hashing. The cipher-suite RSA-3DES-SHA1, for example, indicates that RSA can be used for key agreement (and authentication), while 3DES and SHA1 can be used for bulk encryption and integrity computations, respectively. More than 30 such cipher suite choices exist in the OpenSSL implementation [OpenSSL] of the SSL protocol, resulting from various combinations of cipher alternatives for implementing the individual security services.

Finally, the SSL change cipher protocol allows for dynamic updates of cipher suites used in a connection, while the SSL alert protocol can be used to send

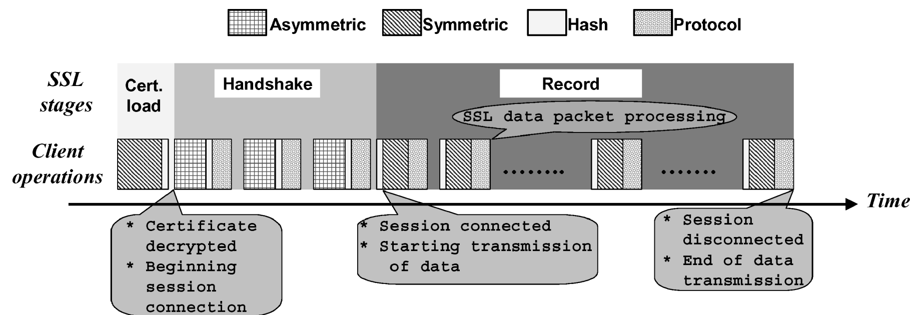


Fig. 5. Typical sequence of client-side operations performed during an SSL session.

alert messages to a peer. Further details of the SSL protocol can be found in SSL and Stallings [1998].

5.2 Security Processing Workloads

We will now examine the workload imposed by security processing by considering the SSL protocol as an example.

Figure 5 shows the typical (client-side) sequence of operations for a secure session that uses the SSL protocol. The first stage involves loading the client certificate from local storage, decrypting it using a symmetric cipher, and performing an integrity check. Once the SSL handshake is initiated, the client and server exchange a sequence of messages that result in the client-side operations as shown in the figure. The objectives of these operations include (i) *server authentication*, where the client verifies the digital signature of the trusted certificate authority (CA) on the server certificate using the public key of the CA, followed by an integrity check, (ii) *client authentication*, where the client optionally generates a digital signature by hashing some data using the MD5 or SHA-1 algorithms, concatenating the digest, and signing the result with its private key, and (iii) *key exchange*, where the client generates a 48-byte premaster secret (used to generate the secret key for the SSL record protocol) and encrypts it with the public key of the server. The specific manner in which these objectives are met depends on the cipher suite being used, but the ultimate result of a successful handshake is that the client and server have completed any necessary authentication steps and share a secret key. Once the connection is established, secure transmission of data proceeds through the SSL record protocol.

Figure 6 shows a function call graph obtained for an SSL session using the OpenSSL implementation [OpenSSL]. The profile has several hundred functions which can be classified from a functional perspective into four categories: asymmetric computations (e.g., function *RSA_public_decrypt*, which implements the RSA algorithm), symmetric computations (e.g., function *des_encrypt*2, which implements the basic DES cipher), hashing operations (functions such as *SHA*1*_Update* and *MD*5*_Update*, which implement the SHA-1 and MD5 algorithms), and protocol processing computations (e.g., functions such as *initialize_ctx* that initializes the session context, *block_host_order* and *block_data_order* that reorder bytes for endianness, record assembly, and so on).

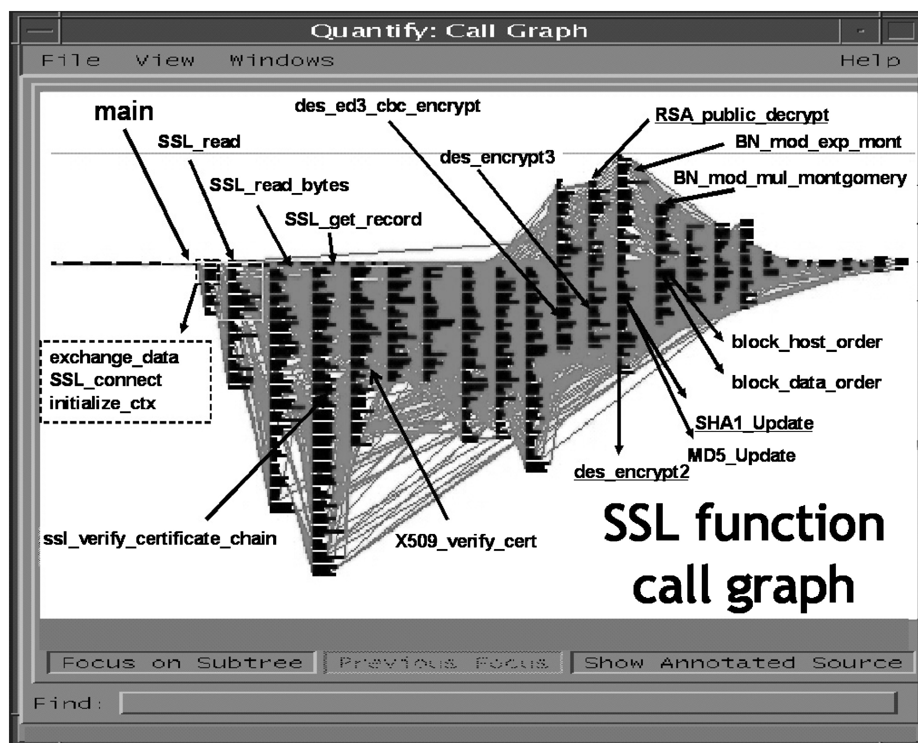


Fig. 6. Function call graph for the OpenSSL implementation of the SSL protocol (client-side).

A quantitative breakup of the processing workload imposed by the above components is given in Figure 7, for three different transaction sizes (1 KB, 100 KB, 1 MB). The measurements were performed on an iPAQ H3670 PDA, which contains an Intel SA-1110 StrongARM processor clocked at 206 MHz. From the figure, we can see that the protocol processing workload, which refers to the workload due to noncryptographic computations, increases with transaction size. The cryptographic processing workload is dominated by asymmetric ciphers for small transactions, and by symmetric ciphers for large transactions.

5.3 Security Processing Gap

We will now analyze how the workload imposed by security processing, in relation to the processing capabilities of embedded processors, leads to a "security processing gap." In order to quantify the security processing gap, we considered the client-side workload imposed by a single secure session between a client and server using the SSL protocol on various embedded processors. For our experiments, we considered the following platforms: (i) PDAs featuring StrongARM (206 MHz SA-1110) and XScale (400 MHz) processors, (ii) a workstation having a 768 MHz Pentium III Coppermine processor, and (iii) a server having a 2.8 GHz Xeon processor. The OpenSSL implementation [Open SSL] of the SSL protocol was used with varying data sizes (10K–1M) to obtain an estimate of the processing requirements in MIPS (millions of instructions per second). To

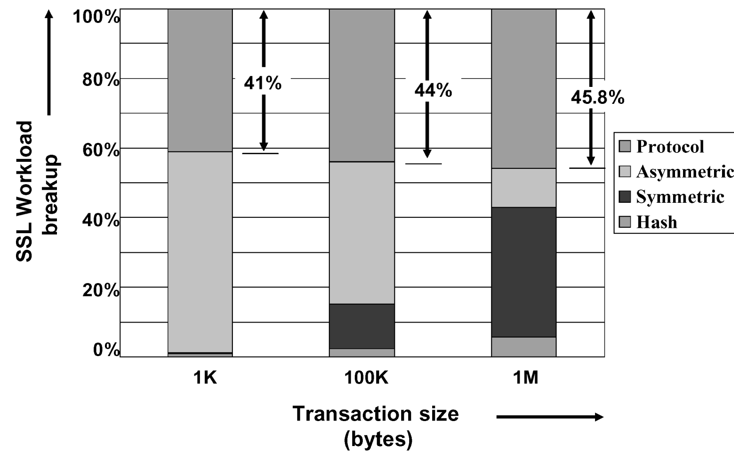


Fig. 7. Breakup of SSL workload into cryptographic and noncryptographic components.

avoid biases due to network effects, tests were performed with both the client and server running on the same processor.

Figure 8 plots the processing requirements in MIPS for performing SSL record protocol processing (using 3DES for bulk data encryption and SHA for integrity) for different data rates in both low- and high-end embedded systems. Data rates typical of cellular (128 kbps–2 Mbps), wireless LAN (2–60 Mbps) and the lower-end of access network (≈100 Mbps) technologies that will be supported by current and emerging low- and high-end embedded systems are indicated in the figure.

The MIPS capabilities of the considered processors are indicated by horizontal dashed lines in Figure 8. In low-end systems such as PDAs, the MIPS capabilities of embedded processors such as StrongARM SA-1110 and XScale are around 150 and 250 MIPS, respectively. If we assume that these processors are completely dedicated to SSL record protocol processing over a single session, they can sustain data rates of 1.8 and 3.1 Mbps, respectively. Any higher data rates are unattainable, leading to the so-called "security processing gap."

Note that the security processing gap is quite severe in practice. In a typical usage scenario, the processor executes multiple secure and nonsecure applications and is, therefore, not completely dedicated to security processing. For example, if the SA-1110 processor can devote only 10% of its resources to the secure SSL session, then the achievable data rates are less than 180 kbps.

The security processing gap is also seen in high-end systems. Processors such as Pentium III and Xeon are only capable of achieving SSL data rates of 7.3 and 29 Mbps, respectively. Thus, higher ranges of wireless LAN data rates as well as wired network data rates cannot be attained without architectural improvements.

5.4 Flexibility Concerns

A fundamental requirement of many embedded systems is the ability to interoperate in different environments. Hence, embedded systems are often required to

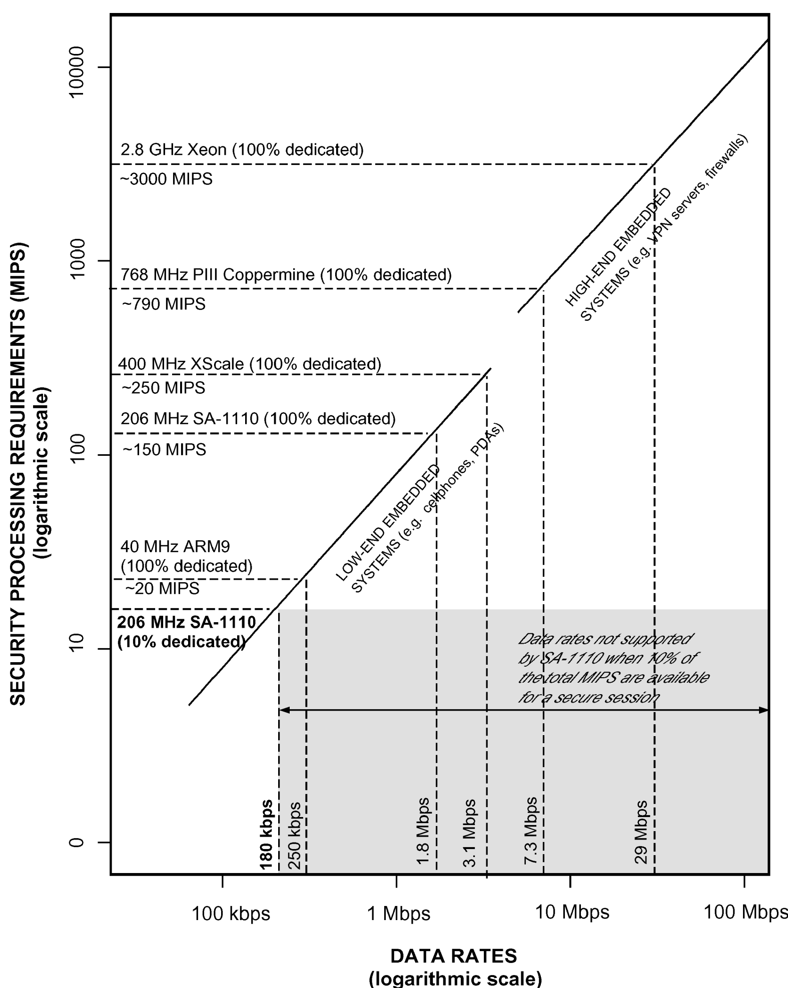


Fig. 8. Processing requirements for the SSL record protocol at different data rates.

support distinct security processing standards, for example, security protocols at different layers of the network protocol stack.

Complicating the above picture is the fact that even a single security protocol standard typically allows for a wide range of cryptographic algorithms. To illustrate this scenario, let us consider the SSL protocol [SSL], which supports the use of different ciphers for its operations (authenticating the server and client, transmitting certificates, establishing session keys, and so on). For key exchange, cryptographic algorithms such as RSA and Diffie–Hellman/DSA are possible choices. For symmetric encryption, an RSA key-exchange-based SSL cipher suite could need to support 3DES, RC4, RC2, or DES, along with the appropriate message authentication algorithm (SHA-1 or MD5). Since an embedded system may have to communicate with a variety of clients or servers, it is desirable to support most, if not all, of the combinations allowed in the standard.

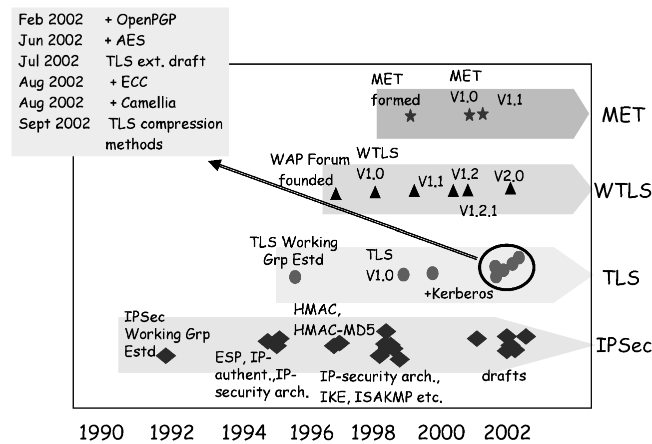


Fig. 9. Evolution of security protocols.

Finally, security protocols are not only diverse, but are also continuously evolving over time. This has been and is still witnessed in the wired network domain, wherein, protocol standards are revised to enable new security services, add new cryptographic algorithms, or drop weaker ciphers. Figure 9 tracks the evolution of popular security protocols in the wired domain (IPSec [IPSec] and TLS [TLS]). We can see that even a well-established protocol such as TLS is subject to constant modifications (e.g., in June 2002, TLS was revised to accommodate the new symmetric encryption standard, AES).

The evolutionary trend is much more pronounced today in the wireless domain, where security protocols can be termed to be still in their infancy. Figure 9 also outlines the evolution of the wireless security protocols, WTLS [OMA] and MET [MeT 2001]. Many of the security protocols used in the wireless domain are adaptations of the wired security protocols. For example, WTLS bears a close resemblance to the SSL/TLS standards. However, it is possible that future security protocols would be specifically tailored from scratch for the wireless environment. This presents a significant challenge to the design of a security processing architecture, since flexibility and ease-of-adaptation to new standards become important design considerations in additional to traditional objectives such as power, performance, and so on.

5.5 Battery Life

The computational requirements of security protocols stemming from the inherent complexity of cryptographic algorithms suggest that the energy consumption of these algorithms can be very high. For battery-powered embedded systems, the energy drawn from the battery directly influences the system's battery life, and, consequently, the duration and extent of its mobility, and its overall utility. To illustrate the impact of security processing on battery life, consider the energy consumption of a wireless handheld (Symbol PPT2800 PocketPC) conducting a secure wireless session that employs 3DES for bulk data encryption and SHA for message authentication. When the handheld is

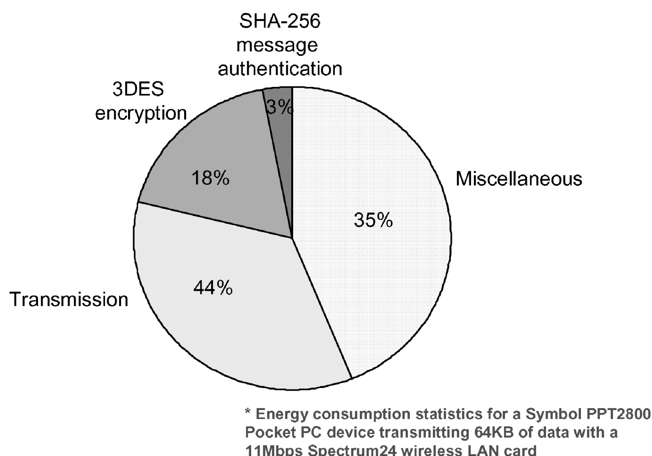
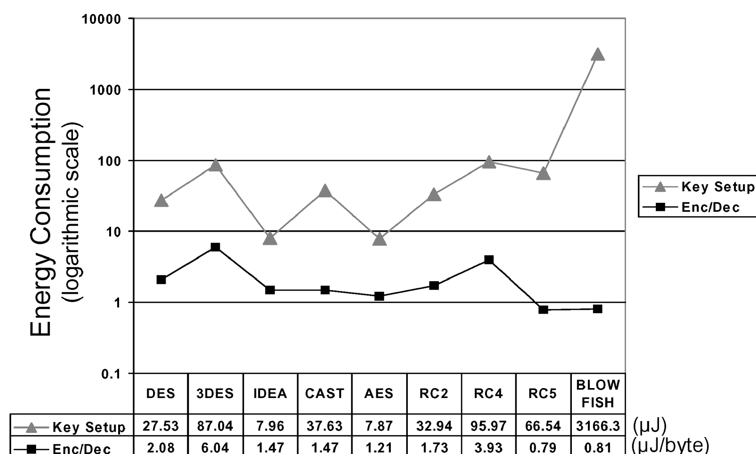


Fig. 10. The energy consumption profile of a sample secure wireless session (*source*: Karri and Mishra [2002]).



	DES	3DES	IDEA	CAST	AES	RC2	RC4	RC5	BLOW FISH	
Key Setup	27.53	87.04	7.96	37.63	7.87	32.94	95.97	66.54	3166.3	(μJ)
Enc/Dec	2.08	6.04	1.47	1.47	1.21	1.73	3.93	0.79	0.81	(μJ/byte)

Fig. 11. Energy consumption data for various symmetric ciphers (*source*: Potlapally et al. [2003]).

securely transmitting 64 kB of data, Figure 10 shows that a considerable part (nearly 21%) of the overall energy consumption is spent on security processing.

Figure 11, for example, shows significant variations in the energy consumption profile of various symmetric ciphers (measured on an iPAQ 3870 PDA running the OpenSSL cryptographic suite [OpenSSL]). Energy numbers for the key setup phase and energy-per-byte numbers for the encryption/decryption phases are shown for each cipher. The results are reported for one specific mode of each block cipher—ECB or electronic code book, where a given plaintext block always encrypts to the same ciphertext block for the same key. The only exception is RC4, which is a stream cipher. We can see from Figure 11 that the key setup costs are the smallest for AES and IDEA, and the largest for Blowfish. However, the per-byte energy cost of Blowfish encryption/decryption is the smallest. In the case of sufficiently large data transactions, one would

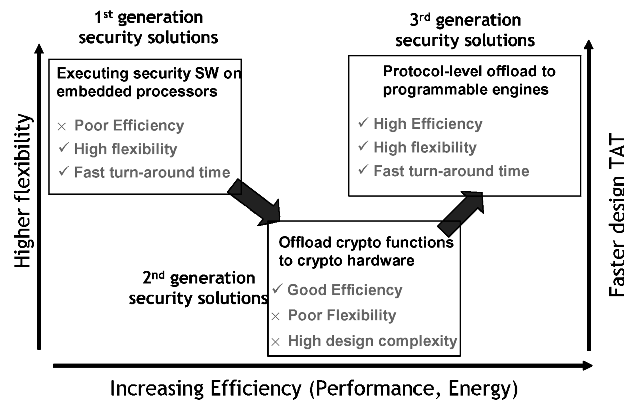


Fig. 12. Taxonomy of security processing architectures.

expect the cost of key setup to be amortized by the low per-byte encryption cost. A detailed analysis of the energy consumption of other cryptographic algorithms and the SSL protocol can be found in Potlapally et al. [2003]. The study also illustrates that significant "energy versus security" trade-offs can be explored by identifying and varying parameters provided in a security protocol—although embedded systems engineers would be well advised to avoid using new or modified algorithms (that have not been subject to widespread review) because of the extreme and often nonintuitive security risks involved.

While the above examples illustrate that the energy requirements for security must be reduced, improvements on the supply side (battery) can also benefit the so-called battery gap. It must be noted here that there has only been a slow growth (5–8% per year) in the battery capacities [Lahiri et al. 2002]. However, recent successes with alternative technologies such as fuel cells [SFC; PolyFuel] show considerable promise.

5.6 Architectures for Security Processing

A variety of approaches have been developed to address the challenges imposed by security processing. The security processing gap can be addressed either by reducing the security processing workloads (alleviating demand), or by enhancing the processing capabilities of the embedded system (improving supply). On the demand side, advances in cryptography have led to new and efficient cryptosystems such as ECC [Rosing 1998; Menezes 1993], NTRU [NTRU], and AES [AES], which provide efficient alternatives to conventional choices. However, their usage in security protocols is not yet widespread, in part due to the preference to use established (and hence higher assurance) algorithms instead of faster ones where the security risks are unknown. For example, flaws have been found in the original NTRU signature scheme, as well as the updated version intended to fix the problem [Gentry and Szydlo 2002]. Enhancements on the supply side of the security processing gap are, therefore, crucial. Figure 12 shows three generations of security processing architectures, and compares them in terms of performance and energy consumption efficiencies, flexibility, as well as design turn-around times.

First-generation solutions perform security processing by executing security software on the processors embedded in the system. While good flexibility and fast design times are possible with software-based solutions, these solutions are not efficient in terms of their performance and energy consumption characteristics.

With cryptographic algorithms being a significant part of the security processing workload, the embedded processor can offload cryptographic computations to custom cryptographic hardware, which can be designed to deliver high performance and consume lower energy. Such architectures constitute the second-generation solutions. These include, for example, embedded processors interfacing with cryptographic hardware on a high-speed bus or processors with special-purpose cryptographic hardware in the pipeline (instruction set extensions).

Since second-generation solutions sacrifice the flexibility and design times of the first-generation solutions, third-generation solutions have been proposed that can capture the benefits of both the first- and second-generation solutions. These solutions have a common characteristic that the embedded processor can offload large portions of the security protocol (not just cryptographic algorithms) to the security protocol processing engine. Since they are based on programmable engines, third-generation solutions are typically capable of performing the bookkeeping necessary to support multiple concurrent security processing streams.

We now provide a brief overview of various second- and third-generation security processing architectures. A more detailed description of architectural alternatives for security processing is provided in Kocher et al. [2004].

5.6.1 *Cryptographic Hardware Accelerators.* Highest levels of efficiency in processing are often obtained through custom hardware implementations. Since cryptographic (asymmetric, symmetric, hash) algorithms form a significant portion of security processing workloads, various companies offer custom hardware implementations of these cryptographic algorithms, for systems ranging for low-power mobile appliances and smartcards to high-performance network routers and application servers [Discretix; Safenet]. Several vendors also offer integrated microcontrollers that contain embedded processors, cryptographic accelerators, and other peripherals [Infineon Technologies; STMicroelectronics].

5.6.2 *Embedded Processor Enhancements for Accelerating Cryptographic Computations.* There have been several attempts to improve the security processing capabilities of general-purpose processors. Because most microprocessors today are word-oriented, researchers have targeted accelerating bit-level arithmetic operations such as the permutations performed in DES/3DES. Multimedia instruction set architecture (ISA) extensions such as those in PA-RISC's Max-2 [Lee 1996] or IA-64 [Intel Corp. 2000] already incorporate instructions for permutations of 8-bit or larger subwords. For arbitrary bit-level permutations, efficient instructions have been recently proposed [Lee et al. 2001]. Instruction set extensions have also been proposed

for other operations such as substitutions, rotates, and modular arithmetic [Burke et al. 2000].

Many such extensions have already been applied to low-end embedded processors used in the wireless handset domain. For example, the SmartMIPS [MIPS] cryptographic enhancements extend the basic 32-bit MIPS ISA to speed up security processing. Similar features are also found in the ARM SecureCore family [ARM-SEC]. The security processing capabilities of SecureCore processors can also be further extended by adding custom-designed cryptographic processing units through a coprocessor interface. This is useful for delivering high performance, without having to redesign the basic processor core. The MOSES security processor developed at NEC [Ravi et al. 2002; Potlapally et al. 2002a, 2002b] extends the basic instruction set of an embedded processor with additional instructions that accelerate various symmetric, asymmetric, and hashing algorithms used in security protocols.

5.6.3 *Security Protocol Engines.* While cryptographic accelerators alleviate the performance and energy bottlenecks of security processing to some extent, achieving very high data rates or extreme energy efficiency requires a holistic view of the entire security processing workload. In addition to cryptographic algorithms, security protocols often contain a significant protocol processing component, including packet header/trailer parsing, classification etc. Security protocol engines accelerate all or most of the functionality present in a security protocol, resulting in higher efficiency than cryptographic accelerators. For example, the 7811 security processor from HIFN [HIFN] can be used in VPNs to perform IPSec processing at very high data rates (nearly 250 Mbps). In addition, these protocol engines, if programmable, can be used to execute multiple protocols efficiently. Again, the 7811 security processor provides high-performance support for not only IPSec, but also layer 2 protocols such as PPP [PPP] and PPTP [PPTP].

Similar efforts have also been seen recently in security architectures for low-power embedded systems. For example, the MOSES security processor from NEC [Ravi et al. 2002] has been designed to function as a coprocessor in application chips for mobile terminals so that significant portions (more than just the cryptographic parts) of security protocols can be offloaded and accelerated.

In summary, programmable security protocol engines are being used increasingly by embedded system designers when both flexibility and efficiency are required.

6. TAMPER RESISTANCE AND THE ASSURANCE GAP

Modern cryptography can provide extremely robust security against specific (usually mathematical) attacks, such as brute force and factoring. Progress on the underlying mathematics has had an unintended consequence: rather than attempt easy-to-understand but futile attacks, intelligent adversaries focus their efforts on more subtle and complex attacks.

At the highest level, the process of creating security requires eliminating undesirable functionality. For example, a cellular telephone network should prevent unauthorized calls from being placed. A functional measure, such as a

cryptographic handshake, may be necessary to impose this restriction, but is irrelevant if an undocumented test mode or a buffer overflow enables adversaries to bypass the functional measure and make calls.

While today's engineering methods are well suited for developing complex functional systems, they are often poorly suited to the task of preventing undesired functionality. Conventional testing works well for detecting overt functionality-based flaws, but is notoriously ineffective at detecting latent security problems. Similarly, engineering abstractions frequently conceal security risks present in one layer from being visible to engineers working on others.

The strongest aspects of a design tend to be much more obvious than the weakest aspects, but the strength of a system is determined by the easiest attack. As a result, products are frequently built with lofty security objectives, but have low assurance of meeting these goals. For example, product advertisements frequently boast about the use of large keys, yet the difference between 128-bit and 256-bit AES is irrelevant in light of the risks posed by implementation weaknesses. Unfortunately, low-assurance designs are easy to produce and appear attractive to unsophisticated customers, even though they may only provide an illusion of security.

6.1 Understanding the Requirements

At the beginning of a security engineering effort, it is important to define what security capabilities are required, such as which attacks must be prevented. An orthogonal issue is the required level of assurance, that is, the probability that these goals have been met. Projects with overly ambitious security or assurance requirements will suffer from increased development cost, time, and effort. Similarly, systems whose requirements are ambiguous or specify inadequate capabilities or assurance have dramatically higher risk.

Hardware development projects always involve difficult trade-offs. For example, the choice of whether to support hardware testing capabilities (e.g., scan or JTAG) involves a trade-off between convenience and security risks. Similarly, choosing between the financial and schedule impact required to address physically invasive attacks, and the risk of leaving a system exposed is not easy. Therefore, it is critical to have a clear understanding of the different ways in which an embedded system can be "attacked." Figure 13 lists the various categories of techniques used to attack a device. At the top level, attacks have been classified into two broad categories: *physical and side-channel attacks*, and *logical attacks*.

Physical and side-channel attacks [Ravi et al. 2004; Anderson and Kuhn 1996; Anderson and Kuhn 1997; Kommerling and Kuhn 1999; Rankl and Effing; Hess et al. 2000; Quisquater and Samyde 2002; Kelsey et al. 1998] refer to attacks that exploit the system implementation and/or identifying properties of the implementation. Physical and side-channel attacks are generally classified into *invasive* and *noninvasive* attacks. Invasive attacks involve getting access to the appliance to observe, manipulate, and interfere with the system internals. Because invasive attacks typically require relatively expensive infrastructure, they are much harder to deploy. Examples of invasive attacks

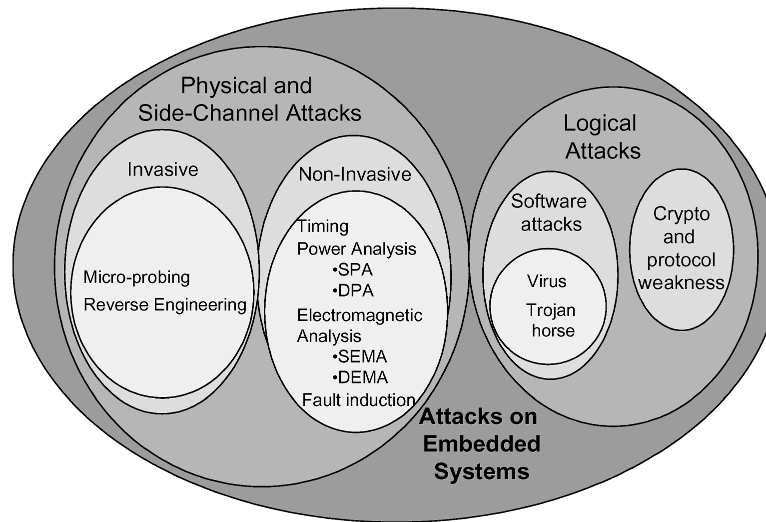


Fig. 13. Examples of attack threats faced by embedded systems.

include microprobing and design 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, and so on. In the sections that follow, we will be examining some of the noninvasive attacks in more detail.

Logical attacks are easy to deploy against systems capable of executing downloaded software, and exploit weaknesses or bugs in the overall architecture (HW/SW) as well as flaws in the design of the cryptographic algorithm or security protocol. These attacks are discussed in the next section.

6.2 Logical Attacks

Logical attacks typically involve sending messages to a device and observing its responses. Often, the adversary's goal is to trick a device into revealing keys or running malicious software. Common examples include software attacks such as buffer overflow exploits, subverted device code updates, and so on.

Logical attacks often exploit design or implementation flaws. Although techniques such as code reviews and sandboxing can be used to reduce the density of critical defects, these gains are largely being overshadowed by increases in software complexity. As a result, logical attacks pose a massive challenge for virtually every security engineering effort—hence, the *assurance gap*.

The range of issues that can lead to logical attacks is extremely broad (see Howard and LeBlanc [2002] and Hoglund and McGraw [2004] for a good high-level introduction to these issues). While there is no way to make a complete list of security mistakes, some common design and implementation problems include: buffer overflows, failure to secure code update processes, use of insecure cryptographic algorithms, cryptographic protocol flaws, key

management failures, random number generator defects, use of debug modes that bypass security, improper error handling, incorrect algorithm implementations, security parameter negotiation weaknesses, improper reuse of keys, poor user interfaces, use of weak passwords, operator errors, pointer errors, operating system weaknesses, sequence counter overflows, solving the wrong problem, inability to reestablish security after compromises, and so on.

Countermeasures for logical attacks are typically designed with one or more of the following considerations:

- to ensure privacy and integrity of sensitive code and data during every stage of execution in an embedded system;
- to determine with certainty that it is safe from a security standpoint to execute a given program;
- to identify and remove software bugs and design flaws that make the system vulnerable to such attacks.

In many embedded systems, the first mentioned consideration is often addressed through various hardware and software changes so as to regulate the accesses of various software components (operating system, downloaded code, and so on) to different portions of the system (registers, memory regions, security coprocessors, and so on) during different stages of execution (boot process, normal execution, interrupt mode, and so on). Examples of such measures include the use of dedicated hardware to protect sensitive memory locations [Discretix], secure bootstrapping [Arbaugh et al. 1997], use of cryptographic file systems [Blaze 1993; Goh et al. 2003], and so on. Secure software execution is often achieved through software authentication and validation checks, sandboxing (restricted environments for code execution), run-time monitors that detect security policy violations [Kiriansky et al. 2002], use of safety proof carrying code [Necula and Lee 1996], and so on. Lastly, verification methods for finding security flaws in trusted software, security protocols and their implementations are also becoming important [Detlefs et al. 1998; Chess 2002; Clarke et al. 1998; Lowe 1998].

We refer the reader to Ravi et al. [2004] and Kocher et al. [2004] for a detailed survey of system-level tamper-resistance mechanisms against software attacks.

6.3 Fault Induction

Security depends on more than just 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 [Schneier 1996]. 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 divisor (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 [Boneh et al. 2001].

6.4 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 nonconstant 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 correct bit value (the one actually used by the target device) should show the strongest correlation to the observed times.

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 10 ms) 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 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 [Kocher 1996].

6.5 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 logic gates and losses due to the parasitic

capacitance of the internal wiring. Power consumption increases if more state transitions occur, or 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 Messerges et al. [2002] that the brute-force search space for a 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 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), 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 set where the predicted bit value is 0, and a second set where the predicted value is 1. Next, an average trace is computed for each subset, where the *n*th sample in each average trace is the average of the *n*th samples in all traces in the subset. Finally, the adversary computes the difference of the average 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, the 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. For a complete attack, the adversary would use a single data set to test hundreds of hypotheses until the entire key is known.

The approach 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 smartcards 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 Kocher et al. [1999].

Attacks such as DPA that involve many aspects of system design (hardware, software, cryptography, and so on) 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, nonintuitive, and require patent licensing [DPA PATENTS]. 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.

7. CASE STUDIES

Several innovative solutions are emerging to address the various design challenges outlined in this paper. In this section, we will consider two recent examples from commercial products to illustrate state-of-the-art solutions used to alleviate the challenges of security processing gap and software attacks.

7.1 Addressing the Security Processing Gap for Wireless Handsets: OMAP 1610

The OMAP 1610 processor is a single-chip application processor from Texas Instruments that is designed to deliver high performance for 2.5G and 3.5G mobile applications [Texas Instruments]. A system-level block diagram of the dual-core processor is shown in Figure 14. It consists of an enhanced ARM926 microprocessor plus the TMS320C55x DSP. The DSP can be used to not only enhance the performance of multimedia applications but also security computations. Public-key computations are typically offloaded to the DSP, while symmetric and hashing operations are offloaded to cryptographic hardware accelerators. Cryptographic hardware accelerators supporting DES, 3DES, AES, SHA-1, and MD5 are included.

The crypto engines (DSP and hardware accelerators) are accessible to security applications through Certicom's Security Builder cryptographic suite [Certicom] (Figure 15). Thus, they can be used to accelerate all applications such as Certicom's SSL, IPSec, and PKI toolkits as well as other third-party applications that use the Security Builder API. The small code size and efficient implementation of the Security Builder SW makes it suitable for the resource-constrained devices that use OMAP 1610.

Other features included in the OMAP 1610 processor for security processing are (a) a true hardware based random-number generator, (b) a secure bootloader for checking the integrity of device-code, and (c) a secure execution mode, enabling secure key storage and run-time authentication. To realize the latter two options, the OMAP 1610 architecture provides 48 kB of secure ROM and 16 kB of secure RAM on-chip.

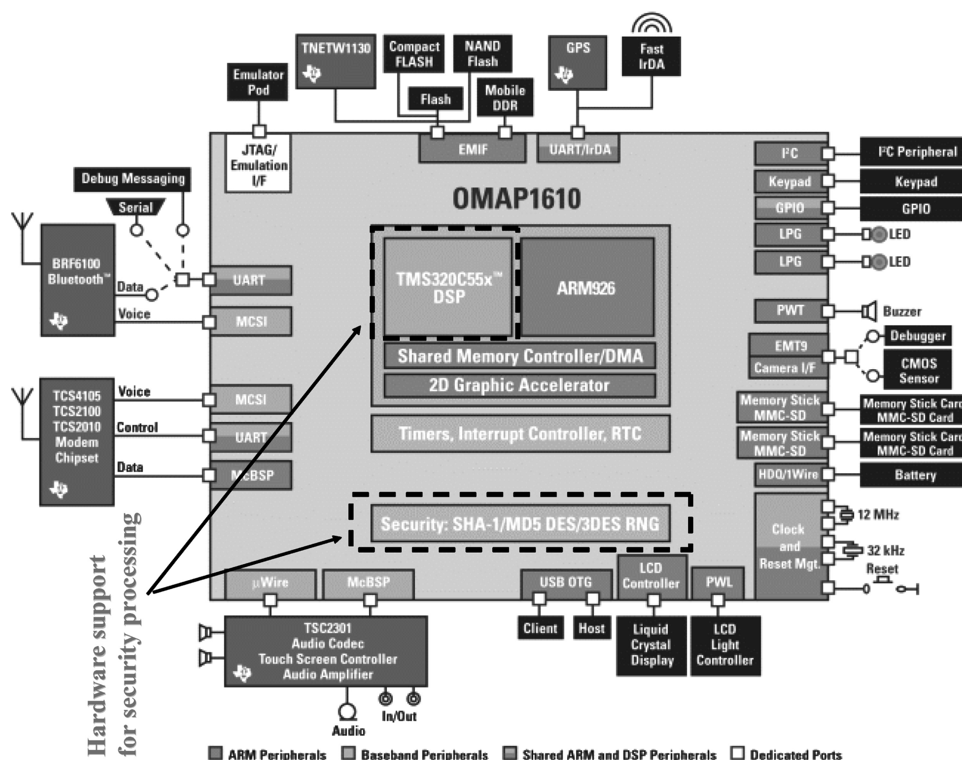


Fig. 14. System-level block diagram of the OMAP 1610 application processor [Texas Instruments].

7.2 Thwarting Software Attacks: ARM TrustZone

The TrustZone security technology [York 2003] from ARM provides an example of how hardware architectures can help provide tamper resistance against software attacks. The basic objective of TrustZone is to establish a clear separation of trusted code, including code that performs security critical operations, from untrusted code that can potentially compromise security. The trusted code is evolved from a "trusted code base" that resides in a secure area of the embedded system. The fundamental concept of evolving and enforcing a trust boundary at every stage of execution was first proposed in Arbaugh et al. [1997]. The trusted code base is responsible for regulating the security of the entire system, starting from the system boot sequence. In addition, the trusted code is responsible for all security tasks that involve manipulation of secret keys.

The trusted code base is protected by implementing a separate secure domain as shown in Figure 16. This is in addition to the user and privileged modes that are typically used to implement application-OS separation. Nonsecure applications are denied access to the secure domain, while trusted applications are identified before they are provided access. This access policy is enforced through the addition of a security tag called "S-bit" throughout the architecture. The S-bit defines the security operation state of the system and is used to denote

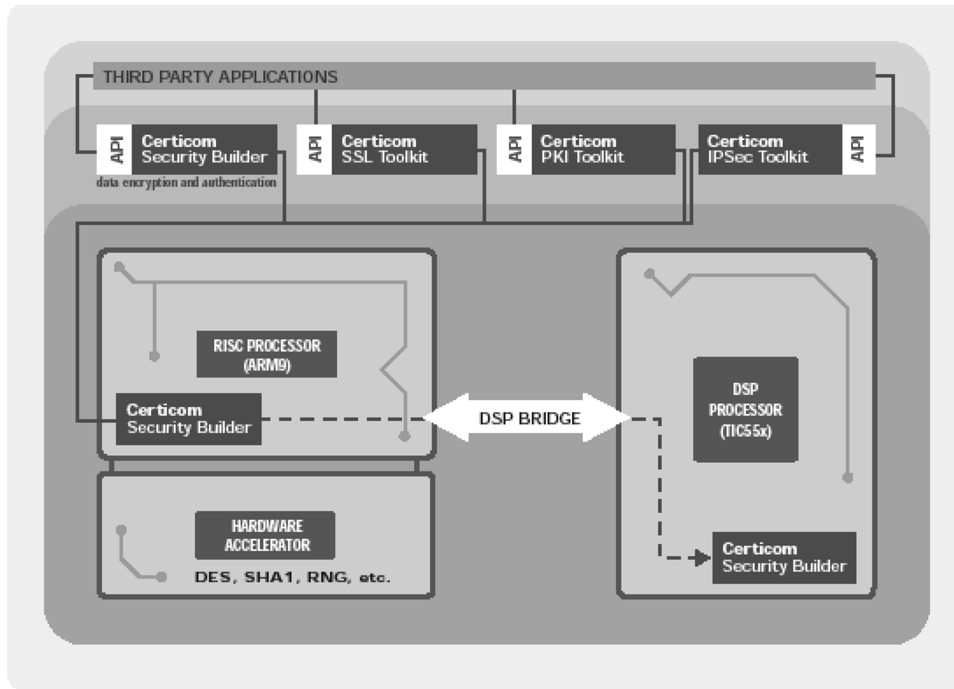


Fig. 15. Offload architecture for security in OMAP 1610 [Certicom-OMAP-WP 2003].

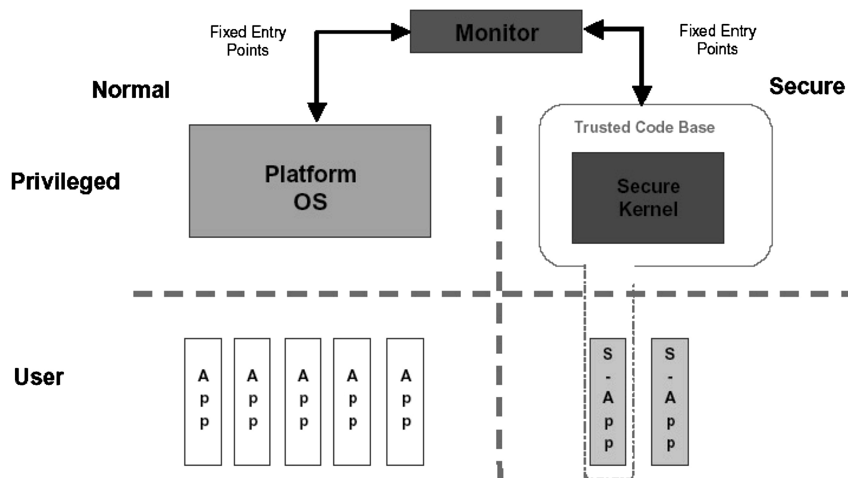


Fig. 16. Separation of secure and nonsecure domains in ARM TrustZone [York 2003].

parts of the system (ARM core, memory system, selected peripherals, and so on) that are secure. Access to the S-bit is through a separate processor operating mode called *monitor mode*, which itself can be accessed through a limited and predefined set of entry points. The monitor mode is responsible for controlling

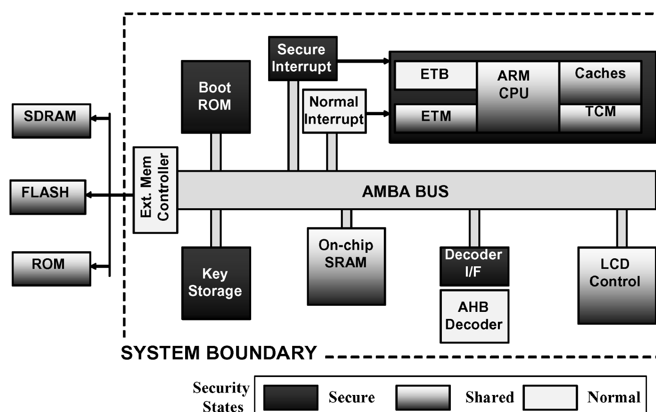


Fig. 17. Components of an embedded system demarcated into secure and nonsecure areas [York 2003].

the S-bit, verifying that data and instruction accesses made by an application are permitted, and ensuring a secure transition between secure and nonsecure states.

The use of TrustZone to secure a typical embedded system is shown in Figure 17, wherein the security perimeter of the system extends beyond the processor core to the memory hierarchy and peripherals. The overall system architecture is divided into secure and nonsecure regions. For example, the boot code is stored securely in the on-chip boot ROM, since modifications to the boot process would render any security scheme ineffective. The memory is segmented into secure and nonsecure areas. The S-bit and the monitor mode are used to ensure that secure data are not leaked to the nonsecure area. Exception handling is also partitioned into normal and secure areas. Because interrupts can be used to freeze the processor when it is processing sensitive information, the monitor mode is used to process critical interrupts.

In summary, the TrustZone technology provides an architecture-level security solution to enforce a separation between trusted and untrusted code. More generally, the concepts of software isolation and sandboxing enable untrusted or less trusted code to coexist safely with trusted code, which allows system designers greater flexibility while providing higher levels of security against software attacks.

8. CONCLUSIONS AND A LOOK AHEAD

Security is critical to enabling a wide range of applications involving embedded systems. While some aspects of security have been addressed in the context of traditional general-purpose computing systems, embedded systems usher in many new challenges. This paper highlighted the security-related problems faced by designers of embedded systems, and outlined recent technological developments and innovations to address them. Several issues, however, remain open at the intersection of security and embedded system design.

The interplay of flexibility, performance, power consumption, and security level makes choosing the “right” security solution a highly complicated process. In addition to these metrics, cost and design turn-around times play a crucial role in deciding the security architecture. In many design scenarios today, it becomes hard to evaluate the effectiveness of a given security solution, or to trade-off between the above metrics, due to the absence of complete system-level analysis and evaluation tools.

Technological advances in allied areas will have an impact on secure embedded system design. For example, developments in the semiconductor fabrication industry can alter the choice of security hardware used. The increasing success of technologies such as field programmable logic devices (PLDs) in meeting good performance and lower design turn around times is prompting designers to examine (or re-examine) their usage as the underlying HW fabric. Consequently, their impact on performance and power consumption of any cryptographic architecture needs to be carefully studied.

Further efforts are also needed in designing the cryptographic algorithms and security protocols that are suited to the constraints and requirements of low-end embedded systems (e.g., in an ambient intelligence setup, where devices may need to support only specific security services). Scalability in algorithms and protocols makes it easier for a security scheme to be effective in a wide range of devices. However, because new lightweight cryptographic techniques require extensive review before they can be considered trustworthy, there is a gap of many years between when research is done in this area and when embedded systems developers can safely begin to take advantage of the results.

Efficient security processing alone is of limited use if an embedded system does not successfully address attacks that could potentially compromise its security. The attacks described in this paper are applicable to a wide range of embedded systems. A clear cost/risk analysis is necessary to determine the levels of attack resistance that a device must support. Since attacks continue to increase in sophistication, the development of countermeasures remains a challenging and on-going exercise. It is also important to remember that countermeasures applicable to one system (e.g., smartcards) may not be able to applicable to other embedded systems (e.g., PDAs or smart phones). Thus, system-specific attack-resistance measures are essential.

In summary, we envision that security will increasingly impact various aspects of the embedded system design process, including hardware circuits and microarchitecture, software, system architecture, and design methodologies.

REFERENCES

AES Algorithm (Rijndael) Information. Available at http://csrc.nist.gov/encryption/aes/rijndael.

ANDERSON, R. AND KUHN, M. 1996. *Tamper Resistance—A Cautionary Note*. Available at http://www.cl.cam.ac.uk/users/rja14/tamper.html.

ANDERSON, R. AND KUHN, M. 1997. Low cost attacks on tamper resistant devices. In *IWSP: International Workshop on Security Protocols*. Lecture Notes on Computer Science. 125–136.

ARBAUGH, A., FARBER, D. J., AND SMITH, J. M. 1997. A secure and reliable bootstrap architecture. In *Proceedings of IEEE Symposium on Security and Privacy*. 65–71.

ARM SecurCore. Available at http://www.arm.com.

BEST, R. M. 1981. *Crypto Microprocessor for Executing Enciphered Programs*. U.S. patent 4,278,837.

BLAZE, M. 1993. A cryptographic file system for UNIX. In *Proceedings of the ACM Conference on Computer and Communications Security*. 9–16.

BONEH, D., DEMILLO, R., AND LIPTON, R. 2001. On the importance of eliminating errors in cryptographic computations. *Cryptology 14,* 2, 101–119.

BURKE, J., MCDONALD, J., AND AUSTIN, T. 2000. Architectural support for fast symmetric-key cryptography. In *Proceedings of the International Conference on ASPLOS*. 178–189.

CARMAN, D. W., KRUS, P. S., AND MATT, B. J. 2000. Constraints and Approaches for Distributed Sensor Network Security. Tech. rep. #00-010, NAI Labs, Network Associates, Inc., Glenwood, MD.

CERTICOM CORP. *Security Builder*. Available at http://www.certicom.com/.

CERTICOM AND TEXAS INSTRUMENTS INC. 2003. *Wireless Security: from the inside out*. Available at http://focus.ti.com/pdfs/vf/wireless/certicom_ti_wp.pdf.

CHESS, B. 2002. Improving computer security using extended static checking. In *Proceedings of the IEEE Symposium on Security and Privacy*. 148–161.

CLARKE, E. M., JHA, S., AND MARRERO, W. 1998. Using state space exploration and a natural deduction style message derivation engine to verify security protocols. In *Proceedings of the IFIP Working Conference on Programming Concepts and Methods*.

COMPUTER SECURITY INSTITUTE. *2002 Computer Crime and Security Survey*. Available at http://www.gocsi.com/press/20020407.html.

Counterpane Internet Security, Inc. Available at http://www.counterpane.com.

DETLEFS, D. L., LEINO, K., NELSON, G., AND SAXE, J. 1998. Extended Static Checking. Tech. rep., Systems Research Center, Compaq Inc.

Cryptocell^{TM}. Discretix Technologies Ltd. Available at http://www.discretix.com.

Discretix Technologies Ltd. Available at http://www.discretix.com.

DPA PATENTS. 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. Available at http://www.cryptography.com/technology/dpa/licensing.html.

ePaynews—Mobile Commerce Statistics. Available at http://www.epaynews.com/statistics/mcommstats.html.

FIPS PUB 140-2. *Security Requirements for Cryptographic Modules*. Available at http://csrc.nist.gov/publications/fips/fips140-2/fips1402.pdf.

GENTRY, C. AND SZYDLO, M. 2002. Cryptanalysis of the revised NTRU signature scheme. In *Proceedings of EUROCRYPT*. 299–320.

GOH, E., SHACHAM, H., MODADUGU, N., AND BONEH, D. 2003. SiRiUS: Securing remote untrusted storage. In *Proceedings of the ISOC Network and Distributed Systems Security (NDSS) Symposium*. 131–145.

HESS, E., JANSSEN, N., MEYER, B., AND SCHUTZE, T. 2000. Information leakage attacks against smart card implementations of cryptographic algorithms and countermeasures. In *Proceedings of the EUROSMART Security Conference*. 55–64.

HIFN INC. Available at http://www.hifn.com.

HOGLUND, G. AND MCGRAW, G. 2004. *Exploiting Software: How to Break Code*. Pearson Higher Education.

HOWARD, M. AND LEBLANC, D. 2002. *Writing Secure Code*. Microsoft Press.

IEEE STANDARD 802.11. LAN/MAN Standards Committee of the IEEE. *Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specification*.

INFINEON TECHNOLOGIES. *SLE 88 family*. http://www.infineon.com.

INTEL CORP. 2000. *Enhancing Security Performance through IA-64 Architecture*. Available at http://developer.intel.com/design/security/rsa2000/itanium.pdf.

IPSec Working Group. Available at http://www.ietf.org/html.charters/ipsec-charter.html.

INTERNET STREAMING MEDIA ALLIANCE. Available at http://www.isma.tv/home.

KARRI, R. AND MISHRA, P. 2002. Minimizing energy consumption of secure wireless session with QoS constraints. In *Proceedings of the International Conference on Communications*. 2053–2057.

KELSEY, J., SCHNEIER, B., WAGNER, D., AND HALL, C. 1998. Side channel cryptanalysis of product ciphers. In *Proceedings of the ESORICS'98*. 97–110.

KIRIANSKY, V., BRUENING, D., AND AMARASINGHE, S. 2002. Secure execution via program sheperding. In *Proceedings of the 11th USENIX Security Symposium*.

KOMMERLING, O. AND KUHN, M. G. 1999. Design principles for tamper-resistant smartcard processors. In *Proceedings of the USENIX Workshop on Smartcard Technology (Smartcard '99)*. 9–20.

KOCHER, P., JAFFE, J., AND JUN, B. 1999. Differential power analysis. *Advances in Cryptology—CRYPTO'99. Lecture Notes in Computer Science, vol. 1666*. Springer-Verlag, Berlin, 388–397.

KOCHER, P., LEE, R., MCGRAW, G., RAGHUNATHAN, A., AND RAVI, S. 2004. Security as a new dimension in embedded system design. In *Proceedings of the Design Automation Conference*. 753–760.

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

KUHN, M. 1997. The TrustNo 1 Cryptoprocessor Concept. CS555 Report, Purdue University. Available at `http://www.cl.cam.ac.uk/mgk25/`.

LAHIRI, K., RAGHUNATHAN, A., AND DEY, S. 2002. Battery-driven system design: A new frontier in low power design. In *Proceedings of the Joint Asia and South Pacific Design Automation Conference / International Conference on VLSI Design*. 261–267.

LEE, R. B., SHI, Z., AND YANG, X. 2001. Efficient permutations for fast software cryptography. *IEEE Micro 21,* 6 (Dec.), 56–69.

LEE, R. B. 1996. Subword parallelism with Max-2. *IEEE Micro 16,* 4 (Aug.), 51–59.

LIE, D., THEKKATH, C. A., MITCHELL, M., LINCOLN, P., BONEH, D., MITCHELL, J. C., AND HOROWITZ, M. 2000. Architectural support for copy and tamper resistant software. In *Proceedings of the ACM Architectural Support for Programming Languages and Operating Systems (ASPLOS)*. 168–177.

LOWE, G. 1998. Towards a completeness result for model checking of security protocols. In *Proceedings of the 11th Computer Security Foundations Workshop.*

MENEZES, A. J. 1993. *Elliptic Curve Public Key Cryptosystems*. Kluwer Academic Publishers, Boston, MA.

MESSERGES, T. S., DABBISH, E. A., AND SLOAN, R. H. 2002. Examining smart-card security under the threat of power analysis attacks. *IEEE Trans. Comput. 51,* 5 (May), 541–552.

MOBILE ELECTRONIC TRANSACTIONS LTD. 2001. *MeT PTD Definition (version 1.1)*. Available at `http://www.mobiletransaction.org/`.

SmartMIPS. Available at `http://www.mips.com`.

MPEG Open Security for Embedded Systems (MOSES). Available at `http://www.crl.co.uk/projects/moses/`.

Moving Picture Experts Group (MPEG). Available at `http://mpeg.telecomitalialab.com`.

NECULA, G. C. AND LEE, P. 1996. Proof-Carrying Code. Tech. Rep. CMU-CS-96-165, Carnegie Mellon University.

NTRU Communications and Content Security. Available at `http://www.ntru.com`.

Open Mobile Alliance (OMA). Available at `http://www.wapforum.org/what/technical.htm`.

OPENIPMP. `http://www.openipmp.org`.

OpenSSL Project. Available at `http://www.openssl.org`.

PERRIG, A., SZEWCZYK, R., TYGAR, J. D., WEN, V., AND CULLER, D. E. 2002. SPINS: Security protocols for sensor networks. *Wireless Netw. 8,* 5, 521–534.

POLYFUEL, INC. Available at `http://www.polyfuel.com`.

POTLAPALLY, N., RAVI, S., RAGHUNATHAN, A., AND JHA, N. K. 2003. Analyzing the energy consumption of security protocols. In *Proceedings of the International Symposium on Low Power Electronics & Design*. 30–35.

POTLAPALLY, N., RAVI, S., RAGHUNATHAN, A., AND LAKSHMINARAYANA, G. 2002a. Optimizing public-key encryption for wireless clients. In *Proceedings of the IEEE International Conference on Communications*. 1050–1056.

POTLAPALLY, N., RAVI, S., RAGHUNATHAN, A., AND LAKSHMINARAYANA, G. 2002b. Algorithm exploration for efficient public-key security processing on wireless handsets. In *Proceedings of Design, Automation, and Test in Europe (DATE) Designers Forum*. 42–46.

Point-to-Point Protocol (PPP), RFC 1661. The Internet Engineering Task Force. Available at http://www.ietf.org/rfc/rfc1661.

Point-to-Point Tunneling Protocol (PPTP), RFC 2637. The Internet Engineering Task Force. Available at http://www.ietf.org/rfc/rfc2637.

QUISQUATER, J. J. AND SAMYDE, D. 2002. Side channel cryptanalysis. In *Proceedings of the SECI*. 179–184.

RANKL, W. AND EFFING, W. *Smart Card Handbook*. John Wiley and Sons, New York.

RAVI, S., RAGHUNATHAN, A., AND CHAKRADHAR, S. 2004. Tamper resistance mechanisms for secure embedded systems. In *Proceedings of the International Conference on VLSI Design*. 605–611.

RAVI, S., RAGHUNATHAN, A., POTLAPALLY, N., AND SANKARADASS, M. 2002. System design methodologies for a wireless security processing platform. In *Proceedings of the ACM/IEEE Design Automation Conference*, 777–782.

REID, P. 2003. *Biometrics and Network Security*. Prentice Hall PTR, Englewood Cliffs, NJ.

ROSING, M. 1998. *Implementing Elliptic Curve Cryptography*. Manning Publications Co.

SAFENET INC. *Safenet EmbeddedIP*^{TM}. Available at http://www.safenet-inc.com.

SCHNEIER, B. 1996. *Applied Cryptography: Protocols, Algorithms and Source Code in C*. John Wiley and Sons, New York.

SFC Smart Fuel Cell AG. Available at http://www.smartfuelcell.com.

SSL 3.0 Specification. Available at http://wp.netscape.com/eng/ssl3/.

STALLINGS, W. 1998. *Cryptography and Network Security: Principles and Practice*. Prentice Hall, Englewood Cliffs, NJ.

STMICROELECTRONICS INC. *ST19 Smart Card Platform Family*. Available at http://www.st.com.

SUH, G. E., CLARKE, D., GASSEND, B., VAN DIJK, M., AND DEVADAS, S. 2003. AEGIS: Architecture for tamper-evident and tamper-resistant processing. In *Proceedings of the International Conference on Supercomputing (ICS '03)*. 160–171.

TEXAS INSTRUMENTS INC. *OMAP Platform*. Available at http://focus.ti.com/omap/docs/omaphomepage.tsp.

TLS WORKING GROUP. Available at http://www.ietf.org/html.charters/tls-charter.html.

U.S. DEPARTMENT OF COMMERCE. 1999. *The Emerging Digital Economy II*. Available at http://www.esa.doc.gov/508/esa/TheEmergingDigitalEconomyII.htm.

WAP FORUM. 2002. *Wireless Application Protocol 2.0*. Technical White Paper. Available from http://www.wapforum.org.

WORLD WIDE WEB CONSORTIUM. 1998. *The World Wide Web Security FAQ*. Available at http://www.w3.org/Security/faq/www-security-faq.html.

YORK, R. 2003. *A New Foundation for CPU Systems Security*. ARM Limited. Available at http://www.arm.com/armtech/TrustZone?OpenDocument.

Received March 2003; revised August 2003 and October 2003; accepted November 2003