International Journal of Science and Research (IJSR)
ISSN (Online): 2319-7064
Index Copernicus Value (2015): 78.96 | Impact Factor (2015): 6.391

Relative Study in Architecture and Design Methodologies for Secured Real Time Embedded Systems

Anupama S

Assistant Professor, Department of Electronics & Communication Engineering, Vidya Vikas Institute of Engineering and Technology, Mysuru

Abstract: ***The embedded systems field is growing rapidly,with devices such as cellular phones, PDAs, smart cards, and digital music players permeating society. On the horizon are futuristic technologies such as embedded network sensors and wearable computers, which promise even greater interaction between humans and machines. As embedded devices are increasingly integrated into personal and commercial infrastructures, security becomes a paramount issue. For example, if a patient is wearing a heart-monitoring device that sends data wirelessly to a doctor, the embedded system must keep this information confidential and deliver it uncorrupted to the doctor. An embedded network sensor monitoring water quality to prevent bioterrorism must have multiple methods to detect tampering in both hardware and software, lest an attacker bypass security measures and corrupt the water supply. The design of security for embedded systems differs from traditional security design because these systems are resource-constrained in their capacities (and consequently in their defenses) and easily accessible to adversaries at the physical layer. Embedded securv can't be solved at a single security abstraction layer, but rather is a system problem spanning multiple abstraction levels. 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.***

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, and storage and web servers. Technological developments 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 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,

- 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.
- 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.

2. Embedded System Challenges

Embedded systems are essentially processor-based devices operating under resource-constrained conditions. Embedded devices include systems as diverse as automobile microcontrollers, cellular phones, smart cards, embedded network sensors, and digital cable boxes. These devices are often portable, communicate via wireless channels, and are battery-powered or otherwise energy-limited. Because these systems are often considered small computers, it's tempting to port workstation based security techniques directly onto the devices to make them secure. However, embedded systems have characteristics that differentiate their security Architecture from that of workstations and servers. We group these characteristics into two categories: resource limitation and physical accessibility.

Embedded devices pose severe resource constraints on the security architecture in terms of memory, computational capacity, and energy. For example, the Smart- Dust node is a battery-powered device possessing an 8-bit, 4-MHz CPU with 8,000 bytes of instruction flash memory, 512 bytes of RAM, and 512 bytes of EEPROM. Clearly, such a platform severely limits potential security scenarios. In terms of memory, sophisticated public key cryptography techniques such as RSA or elliptic-curve cryptography might simply be infeasible. Considering the device's 4-MHz computational horsepower,certain protocols could cause too much latency tobe useful. Furthermore, an energy-intensive security scheme can cause the node to perish from battery exhaustion before it can perform useful work. Power dissipation and other resources are infrequently major concerns when dealing with workstation-based security.concerns. On the one hand, storing sensitive information on a device rather than on multiple servers minimizes the number of locations where an attack can occur. On the flip side, small devices can be easily lost or stolen, and hence must have extra security measures built in to ensure that private data can't be compromised.

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 checkingthe 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 plain text) and transmit the encrypted data (usually called the cipher text) to the receiver. On receiving the cipher text, the receiver then uses the same secret key to decrypt it and regenerate the plain text. The cipher text should have the property that it is very hard for a third party to deduce the plain text, 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 "thumb prints" 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 afore mentioned 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] andSSL [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 withan 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 associated with a given application or a process [13], HW/SW techniques to preserve the privacy and integrity ofdata throughout the memory hierarchy [14], execution of encrypted code in processors to prevent bus probing [5, 6]*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.

Three factors, which we call the *Trinity of Trouble* complexity, extensibility and connectivity conspire to make managing risks in software a major challenge.

a) Tamper-resistant hardware

There are a wide range of 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 [14]. 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.

b) 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 [5]. 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 depackaged 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.

c) 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.

5. Embedded Processing Architectures for Security

At the architecture level, the protocol and algorithms must be mapped onto an embedded architecture platform. At this stage, we partition the device into secure and insecure modules. Secure modules are hardware modules that run secure functions, house secure memory, and use various hardware and software techniques to protect themselves. Insecure modules run insecure functions, house insecure memory, and aren't protected from attack.Because providing security has a cost overhead in terms of area, power, and computation, we must clearly distinguish between the protocol's secure and insecure parts. Then, we can provide security to the secure parts only, reducing the overhead. Basically, we try to confine the secure parts to the smallest possible portion of the system.At the protocol level, we move secure parts from server to device; at the architecture level, we move them to a limited area of the device (the secure module). We call this security-driven hardware–software partitioning, or security partitioning.

Security partitioning isolates the device's sensitive data and functions so both software (architecture-level) and physical (circuit-level) mechanisms can protect them. Security partitioning is the application of a variant of Kerckhoffs' principle, which is to minimize the number of secrets in a system. If the device as a whole is physically compromised, it remains secure as long as the secure module is intact. The technique addresses the software bypass attack as well as resource limitation and physical accessibility issues. Later, we'll discuss a technique to secure a system at the circuit level, which requires approximately twice the normal area and power. Securing only a required subset of the system thus addresses area and energy limitations.

Partitioning the device into secure and insecure modules involves four primary steps at the architecture level: determine partitioning topology, determine coupling and secure-to-insecure bus structure, partition functions and their data, and construct a secure instruction set.

A) Microarchitecture level: Hardware design

We designed and simulated the architecture's hardware implementation at the microarchitecture level. This involved designing the hardware for the insecure and secure modules and their interfaces. We used the Leon processor with a configurable Amba bus structure for the insecure module and designed a memory-mapped interface on the Amba peripheral bus to communicate with the secure module.The secure module is a custom-designed secure coprocessor. The coprocessor consists of a top-level controller, a cryptographic engine, and a matching engine (with template storage). The coprocessor includes two categories of buses: public and private. The public buses are the coprocessor's interface to the outside world (that is, the secure-to-insecure bus structure) and are thus insecure. The private buses are secure buses internal to the coprocessor, such as those between the matching and cryptographic submodules. These private buses ensure that sensitive data remain local to the coprocessor and aren't directly accessible to the insecure module. The top-level controller monitors for illegal and out-of-sequence instructions. Hence, the insecure processor can't access the coprocessor's internal data; it can access the coprocessor only using instructions on the public buses, which the top-level controller monitors for foul play.

One challenge at the microarchitecture level is to accurately cosimulate the secure and insecure modules. In our design, this implies a hardware/software co-simulation, including a thorough sequence of valid and invalid instructions to protect against false-instruction attacks. For such a co-simulation, we need a design and simulation environment that can simulate the secure and nonsecure modules together. This is because the boundary between hardware and software is often a weak part of secure systems, simply because they're designed and developed separately (and often by different teams).

The micro architecture level implements the design and security features described at the architecture level, but suppose an attacker chooses an out-of-band attack at the physical level. In this case, the architecture and microarchitecture defenses are meaningless. We therefore need new defenses at the circuit level.

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, webriefly 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, and the Common Criteria for information technology security evaluation 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.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 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 survey.

7. Conclusion

Today, secure embedded system design remains a field in its infancy terms of pervasive deployment and research. The good news is that unlike the problem of providing security in cyberspace, 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 andusability.

References

[1] D. Lie, C. A. Thekkath, M. Mitchell, P. Lincoln, D. Boneh, J. C. Mitchell, andM. Horowitz, "Architectural support for copy and tamper resistant software," inProc. ACM Architectural Support for Programming Languages and OperatingSystems (ASPLOS), pp. 168–177, 2016.
[2] 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 2013.
[3] R. M. Best, Crypto Microprocessor for Executing Enciphered Programs. U.S.patent 4,278,837, July 1981.

[4] M. Kuhn, The TrustNo 1 Cryptoprocessor Concept. CS555 Report, PurdueUniversity (http://www.cl.cam.ac.uk/˜mgk25/), Apr. 1997.
[5] G. Hoglund and G. McGraw, Exploiting Software: How to Break Code(http://www.exploitingsoftware.com). Addison-Wesley, 2004.
[6] J. Viega and G. McGraw, Building Secure Software(http://www.buildingsecuresoftware.com). Addison-Wesley,2001.
[7] G. McGraw, "Software Security," IEEE Security & Privacy, vol. 2, pp. 80–83,March–April 2004.
[8] R. Anderson and M. Kuhn, "Tamper resistance - a cautionary note," 1996.
[9] R. Anderson and M. Kuhn, "Low cost attacks on tamper resistant devices," inIWSP: Intl. Wkshp. on Security Protocols, Lecture Notes on Computer Science,pp. 125–136, 1997.
[10] O. Kommerling and M. G. Kuhn, "Design principles for tamper-resistantsmartcard processors," in Proc. USENIX Wkshp. on Smartcard Technology(Smartcard '99), pp. 9–20, May 1999
[11] Ravi, A. Raghunathan, and S. Chakradhar, "Tamper Resistance Mechanismsfor Secure Embedded Systems," in Proc. Int. Conf. VLSI Design, Jan. 2004.
[12] 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.
[13] T. S. Messerges, E. A. Dabbish, and R. H. Sloan, "Examining Smart-CardSecurity under the Threat of Power Analysis Attacks," IEEE Trans. Comput.,vol. 51, pp. 541–552, May 2002.
[14] P. Kocher, J. Jaffe, and B. Jun, "Differential Power Analysis," Advances inCryptology – CRYPTO'99, Springer-Verlag Lecture Notes in Computer Science,vol. 1666, pp. 388–397, 1999.