System-on-Chip Security Assurance for IoT Devices: Cooperations and Conflicts

Sandip Ray
NXP Semiconductors, Austin, TX 78735. USA
sandip.ray@nxp.com

Abstract—**Security is a critical component for computing devices targeted towards Internet-of-Things applications. Unfortunately, IoT security assurance is a challenging enterprise, involving cooperation and conflicts among a variety of stake-holders working in concert with a variety of architecture and design collateral generated across various points in a complex design life-cycle. Furthermore, the long life and aggressive energy/performance needs of IoT applications bring in new challenges to security designs. In this paper we discuss some of the challenges, the current industrial practice to address them, some gaping holes in the state of the practice, and potential research directions to address them.**

I. Introduction

The emergence of the Internet-of-Things (IoT) has led to a new era of computing when the number of smart, connected computing devices exceeds the human population [1]. It is an exciting time for computing, witnessing an explosive growth in the smart, connected computing devices — with estimates between 50 billion and 75 billion by 2020 and running into the trillions within a decade more. The number and kind of computing applications has seen a corresponding diversity, ranging from smart dust to smart cities.

Security trustworthiness of computing systems are crucial to the IoT regime. With computing devices being employed for a large number of highly personalized activities (*e.g.*, shopping, banking, fitness tracking, providing driving directions, etc.), these devices have access to a large amount of sensitive, personal information which must be protected from unauthorized or malicious access. Furthermore, communication of this information to other peer devices, gateways, and datacenters is in fact crucial to providing the kind of adaptive, "smart" behavior that the user expects from the device. For example, a smart fitness tracker must detect from its sensory data (*e.g.*, pulse rate, location, speed, etc.) the kind of activity being performed, the terrain on which the activity is performed, and even the motivation for the activity in order to provide anticipated feedback and response to the user; this requires a high degree of data processing and analysis much of which is performed by datacenters or even gateways with higher computing power than the tracker device itself. The communication and processing of intimate personal information by the network and the cloud exposes the risk that it may be compromised by some malicious agent along the way. In addition to end user information, most computing systems contain confidential collateral from architecture, design, and manufacturing, *e.g.*, cryptographic and digital rights management (DRM) keys, programmable fuses, on-chip debug instrumentation, defeature bits, etc. Malicious or unauthorized access to secure assets in a computing device can result in identity thefts, leakage of company trade secrets, even loss of human life. A crucial component of a modern System-on-Chip (SoC) architecture includes mechanisms to protect these assets.

In this paper, we look at the spectrum of challenges for security assurance in SoC designs targeted for Internet-of-Things, and the state of the practice in developing security architecture for these devices. Unfortunately, and in spite of significant work on SoC security architecture, the current state of the practice [2], [3] does not satisfy the requisite constraints to enable smooth usage for IoT applications. In particular, IoT security involves conflicts and trade-offs between a large number of stake-holders, including energy, performance, intelligence, and validation. We discuss the current challenges in IoT security, and touch upon some emergent research to address them.

The remainder of the paper is organized as follows. Sections II and III provide a high-level overview of the current state of the practice in security assurance for modern SoC designs. Section V discusses some of the unique challenges in IoT security. In Section VI we briefly introduce a new approach to address the IoT security needs. This approach is work in progress, but provides a promising direction towards addressing the in-field configurability needs as well as security/performance/energy trade-offs necessary for IoT devices. We conclude in Section VII.

II. SoC Security Policies

SoC security is driven by the requirement protect system assets against unauthorized access. The assets in an SoC design include cryptographic and DRM keys, premium content, defeaturing bits, configuration fuses as well as personal end user information, etc., and are sprinkled across different IP blocks. Access control can be defined by confidentiality, integrity, and availability requirements [4]. The goal of a security policy is to map the requirements to "actionable" design constraints that can be used by IP implementers or SoC integrators to develop protection mechanisms. Following are two representative examples for a typical SoC.

- *Example 1:* During boot, data transmitted by the crypto engine cannot be observed by any IP in the SoC fabric other than its intended target.

978-1-5090-5191-5/17/$31.00@2017 IEEE

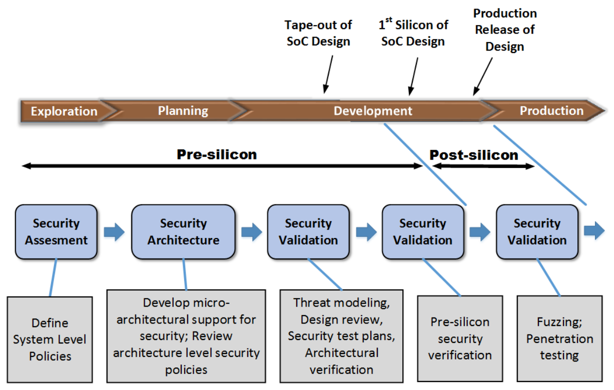
Tape-out of SoC Design
1st Silicon of SoC Design
Production Release of Design
Exploration
Planning
Development
Production
Pre-silicon
Post-silicon
Security Assesment
Security Architecture
Security Validation
Security Validation
Security Validation
Define System Level Policies
Develop micro-architectural support for security; Review architecture level security policies
Threat modeling, Design review, Security test plans, Architectural verification
Pre-silicon security verification
Fuzzing; Penetration testing

Fig. 1. Life cycle of a typical SoC from exploration to production.

- *Example 2:* A secure key container can be updated for silicon validation but not after production.

Example 1 is a confidentiality requirement while Example 2 is an integrity constraint; The policies provide definitions of (computable) conditions to be satisfied by the design for accessing a security asset. Furthermore, these may vary depending on the state of execution (e.g., boot time, normal execution, etc.), or position in the development life-cycle (e.g., manufacturing, production, etc.). In addition to access control, security policies can capture requirements from information flow, liveness, time-of-check vs. time-of-use (TOCTOU), etc.

III. Security Across SoC Design Life-Cycle

Fig. 1 provides a high-level overview of the SoC design life-cycle. Each component of the life-cycle involves a large number of design, development, and validation activities. Here we summarize the key activities involved along the life-cycle, that pertain to security.

Risk Assessment. Security requirements definition is a key part of product planning, and happens concurrently with (and in close collaboration with) the definition of architectural features. This process involves identifying the security assets in the system, their ownership, and protection requirements, collectively defined as *security policies*. The result of this process is the generation of a set of documents, often referred to as *product security specification* (PSS), which provides the requirements for downstream architecture, design, and validation activities.

Security Architecture. The goal of a security architecture is to design mechanisms for protection of system assets. It includes (1) identifying and classifying potential adversary for each asset; (1) determining attacker entry points, also referred to as threat modeling; and (3) developing protection and mitigation strategies. The security definition typically proceeds in collaboration with architecture and design of other system features, including speed, power management, thermal characteristics, etc.

Security Validation. Security validation spans the architecture, design, and post-silicon components of the system life-cycle. The actual validation target and properties validated at any phase depends on the collateral available, *e.g.*, we target, respectively, architecture, design, implementation, and silicon artifacts as the matures. One key activity is to subvert the advertised security requirements in PSS, and identify mitigation measures.

IV. Security Architectures: State of the Practice

How would we go about designing authentication mechanisms to ensure policy enforcement? Unfortunately, the state of the practice today depends heavily on human creativity. The typical approach is to develop a baseline architecture definition which is then repeatedly refined through the following two steps:

- identify potential threats to the current architecture; and
- refine the architecture with mitigation strategies.

The baseline architecture is typically derived from legacy architectures for previous products, adapted to account for the policies defined for the SoC design under exploration. In particular, for each asset, the architect must identify (1) who can access the asset, (2) what kind of access is permitted by the policies, and (3) at what points in the system execution or product development life-cycle such access requests can be granted or denied. Note that not all assets are statically defined; many assets are created at different IPs during the system execution. For example, a fuse or e-wallet may have a statically defined asset such as key configuration modes. During system execution, these modes are passed to the cryptographic engine, which generates the cryptographic keys for different IPs and transmits them through the system NoC to the respective IPs. Each participant IP has sensitive assets (either static or created) during different phases of the system execution, and the security architecture must account for any possible access to these assets at any point.

There has been significant work towards standardizing architecture to implement access control for different assets. Most of this work has taken the form of developing a Trusted Execution Environment (TEE), *viz.*, a mechanism for guaranteeing isolation between code and sensitive data at different points of the system execution. TEEs, of course, have been a part of computer security for a long time, with a large number of mechanisms and architectures. One of the most common TEE architectures is the Trusted Program Module (TPM), which is an international standard for a secure cryptoprocessor designed to secure the hardware by integrating cryptographic keys into devices [5]. It covers methods for secure generation of cryptographic keys and limitation of their use, the requirements from random number generator, as well as capabilities such as remote attestation and sealed storage. In addition to TPM, there has been significant work on architecting other TEEs, both in industrial platform and in academic research [6], [7]. Some important TEE frameworks specifically developed for SoC designs include Samsung KNOX [8], Intel® Software Guard Extension (SGX) [9], and ARM Trustzone® [3]. Note that in spite of differences motivated by the isolation and separation targets, the underlying architectural plans for these

TEEs are similar, *viz.*, a combination of hardware support (*e.g.*, secure operating modes, virtualization), and software mechanisms (*e.g.*, context switch agents, integrity check).

The TEEs provide a foundation (*i.e.*, a mechanism of isolation) for implementing security policies. However, they are a far cry from a standardized approach for implementing policies themselves. To provide such approaches, it is necessary to (1) develop a language for succinctly expressing security policies; (2) architecting a parameterized "skeleton" design that can be easily instantiated to diverse policy implementations; and (3) developing techniques for synthesizing policy implementation from high-level descriptions. Recent academic and industrial research has attempted to mitigate some of these issues. Li *et al.* [10] provide a language and synthesis framework for certain security policies. Basak *et al.* [11] provide a microcontrol-based framework for implementing certain class of security policies. There have been optimized architectural support for specific classes of policies,*e.g.*, control-flow integrity [12], Trojan resistance [13]. However, in spite of such work on pieces of the problem, we are still far away from a robust, configurable security architecture as necessary for robust system design. Some key deficiencies include interplay of secure access control with on-chip instrumentation, definition of security architectures that are configurable for different phases of system life-cycle, and lack of a centralized IP for policy implementation in the SoC design, which makes it difficult to evaluate policy compliance.

V. IoT Security: Configurability, Performance, and Energy Trade-offs

IoT applications introduce additional constraints on the SoC security architecture. Most of the constraints arise from the following three requirements.

Long Field Life. Unlike traditional desktop, laptop, or mobile computing devices, many devices targeted for IoT (such as a smart automotive or smart meter) have a long in-field life often spanning over a decade or more. The consequence to security is that security requirements may evolve significantly over this period, *e.g.*, consider the security requirements for today's systems vis-a-vis those of a decade back. Requirements can change because of (1) a new security exploitation discovered when the device is in-field; or (2) new requirements based on an unanticipated use case or change in performance or energy requirements; or (3) obsolescence of a protection mechanism due to changing computing paradigm, *e.g.*, emergence of quantum computers subverting cryptographic algorithms.

Bulk Production. As IoT devices are getting deployed in quantities ranging from billions to trillions, it is impossible to design each system individually by accounting for its unique use-case constraints. Indeed, in our current swift-changing environment of computing, all the use-case constraints are not even clearly defined by the time a system design is initiated. On the other hand, security requirements are typically unique to each class of device, *e.g.*, a home appliance, an automotive, and an implant has completely different security requirements. Addressing the challenge of unique security needs while permitting development of systems *en masse* is a critical requirement for IoT applications.

Tight Energy/Performance Requirements. Many IoT devices operate within a very strict thermal, power, performance, and real-time constraints. For example, an implant connected to a human body must operate within the thermal constraints defined by physiology; a wearable such as a smart watch must operate within energy budget that precludes the device getting uncomfortably hot; a processor in a car engine must withstand potentially high temperatures and enable real-time communication with the highway system. Security solutions must account for these constraints. For example, one cannot use a TEE that consumes high energy or significantly affects real-time performance on a smart implant or automotive system respectively/ Furthermore, the requirements may change dynamically during execution, *e.g.*, one may withstand more energy consumption when wearing a smart watch at a coffee shop in return for a higher level of security protection, while one may trust the home network to enable a lower protection level to conserve battery.

The above requirements suggest the need for high extensibility built into the security architecture, to permit post-silicon, in-field, and potentially run-time configurability of security requirements. This would address both the challenge of evolving security requirements in course of the long life or run-time updates, and also permit bulk production while enabling post-manufacture reconfiguration of the policies based on deployment target. Unfortunately, this is hard to do in today's security architectures which implement a fixed set of security policies. These policies are conceived in advance during architecture exploration and risk assessment. Furthermore, their implementation is spread across hardware, firmware, and system software of different IPs. This makes any update (whether on-field or in advanced implementation stages) a highly non-trivial activity. Consequently, in-field updates today are based on creative and complex workarounds with software or firmware patches and point fixes. More often than not, the result is a modification in functionality not thought through in advance, a significant increase in system complexity in-field, and increased points of vulnerability and attack.

There is also a significant trade-off between configurability on the one hand and energy/performance on the other, *e.g.*, a highly configurable implementation typically involves a high software component which typically results in higher performance and energy overhead than a custom hardware.

VI. Emergent Approach: A Centralized Security Architecture

Recent work [11], [14] has attempted to address the problem of reconfigurability as well as potential trade-offs between security, energy, and performance. The idea is to provide an easy-to-integrate, scalable infrastructure IP that serves as a centralized resource for SoC designs to protect against diverse security threats at minimal design effort and hardware

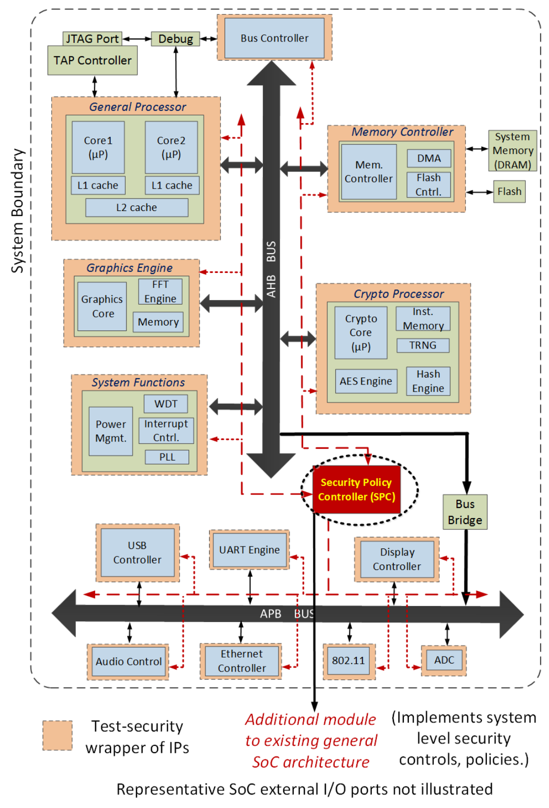

Fig. 2. SoC security architecture Based on E-IIPS for efficient implementation of diverse security policies.

overhead. Fig. 2 shows the overall architecture of E-IIPS. It includes a microcontroller-based firmware-upgradable module called SPC (for "security policy controller") that realizes system-level security policies of various forms and types using firmware code following existing security policy languages. The SPC module interfaces with the constituent IP blocks in a SoC using "security wrappers" integrated with the IPs. These security wrappers extends the existing test (*e.g.*, IEEE 1500 boundary scan based wrapper [15]) and debug wrapper (*e.g.*, ARM's CoreSight interface [16]) of an IP. These security wrappers detect local events relevant to the implemented policies and enable communication with the centralized SPC module. The result is a flexible architecture and approach for implementing highly complex system-level security policies, including those involving interoperability requirements and trade-offs with debug, validation, and power management. The architecture is realizable with modest area and power overhead [11]. Furthermore, more recent work has shown that the existing design instrumentations, *e.g.*, for DfD, could be exploited in implementing the architecture [14].

Of course, the architecture itself is only one component of the policy definition. Several challenges remain, *e.g.*, (1) defining a language for security policy specification that can be efficiently compiled to SPC microcode, (2) study of bottlenecks related to routing and congestion across communication fabrics in implementing the architecture, (3) implementing security policies involving potentially malicious IPs (including malicious security wrappers or Trojans in the SPC itelf), etc. Nevertheless, the approach shows a promising direction towards systematizing policy implementations. Furthermore, by enclosing the policy definitions to a centralized IP, it enables security validation to focus on a narrow component of the design, thereby potentially reducing validation turn-around.

VII. Conclusion

We have discussed the current practice in development of security architecture for modern SoC designs. We described the new challenges induced by IoT applications and the gaps in the state of the practice in addressing these challenges. We discussed emergent research for addressing these challenges. Although the emergent research is promising, it is just a start. We believe that a comprehensive solution to the problem would require significant collaboration among security researchers as well as system and hardware architects, performance and power management experts, and system validators.

References

[1] D. Evans, "The internet of things - how the next evolution of the internet is changing everything," *White Paper. Cisco Internet Business Solutions Group (IBSG)*, 2011.

[2] S. Krstic, J. Yang, D. W. Palmer, R. B. Osborne, and E. Talmor, "Security of SoC Firmware Load Protocol," in *IEEE HOST*, 2014.

[3] ARM, "Building a secure system using trustzone technology," *ARM Limited*, 2009.

[4] S. J. Greenwald, "Discussion Topic: What is the Old Security Paradigm," in *Workshop on New Security Paradigms*, 1998, pp. 107–118.

[5] Trusted Computing Group, "Trusted Platform Module Specification," http://www.trustedcomputinggroup.org/tpm-main-specification/.

[6] A. Vasudevan, E. Owusu, Z. Zhou, J. Newsome, and J. M. McCune, "Trustworthy Execution on Mobile Devices: What Security Properties Can My Mobile Platform Give Me?" in *Trust and Trustworthy Computing*, ser. LNCS. Springer, vol. 7344, pp. 150—178.

[7] J. M. McCune, B. Parno, A. Perrig, M. K. Reiter, and H. Isozaki, "Flicker: An Execution Infrastructure for TCB Minimization," in *Proceedings of ACM EuroSys*, 2008.

[8] Samsung, "Samsung KNOX," www.samsungknox.com.

[9] Intel, "Intel® Software Guard Extensions Programming Reference," https://software.intel.com/sites/default/files/managed/48/88/329298-002.pdf.

[10] X. Li, V. K. anf J. Oberg, M. Tiwari, V. Rajarathinam, R. Kastner, T. Sherwood, B. Hardekopf, and F. T. Chong, "Sapper: A Language for Hardware-Level Security Policy Enforcement," in *International Conference on Architectural Support for Programming Languages and Operating Systems*, 2014.

[11] A. Basak, S. Bhunia, and S. Ray, "A Flexible Architecture for Systematic Implementation of SoC Security Policies," in *Proceedings of the* 34*th International Conference on Computer-Aided Design*, 2015.

[12] L. Davi, M. Hanreich, D. Paul, A.-R. Sadeghi, P. Koeberl, D. Sullivan, O. Arias, and Y. Jin, "Hafix: Hardware assisted flow integrity extension," in *Proceedings of the* 52*nd Annual Design Automation Conference*, 2015.

[13] L. Changlong, Z. Yiqiang, S. Yafeng, and G. Xingbo, "A System-On-Chip bus architecture for hardware Trojan protection in security chips," in *EDSSC*, 2011.

[14] A. Basak, S. Bhunia, and S. Ray, "Exploiting Design-for-Debug for Flexible SoC Security Architecture," in *IEEE DAC (accepted)*, June 2016.

[15] IEEE Joint Test Action Group, "IEEE Standard Test Access Port and Boundary Scan Architecture," *IEEE Std.*, vol. 1149, no. 1, 2001.

[16] E. Ashfield, I. Field, P. Harrod, S. Houlihane, W. Orme, and S. Woodhouse, "Serial Wire Debug and the CoreSight^{TM} Debug and Trace Architecture," 2006.