INVITED PAPER

Mobile Trusted Computing

This paper surveys the trusted computing features in mobile computing platforms.

By N. Asokan, Jan-Erik Ekberg, Kari Kostiainen, Anand Rajan, Carlos Rozas, Ahmad-Reza Sadeghi, Steffen Schulz, and Christian Wachsmann

ABSTRACT | Trusted computing technologies for mobile devices have been researched, developed, and deployed over the past decade. Although their use has been limited so far, ongoing standardization may change this by opening up these technologies for easy access by developers and users. In this survey, we describe the current state of trusted computing solutions for mobile devices from research, standardization, and deployment perspectives.

KEYWORDS | Attestation; low-cost trusted execution; mobile security; physically unclonable functions (PUFs); trusted computing

I. INTRODUCTION

The term trusted computing is used to collectively describe technologies enabling the establishment of trust in local and remote computing systems by using trustworthy components, trust anchors, to ensure the integrity of other parts of the system. In this paper, we survey recent research and industry efforts in designing and deploying trusted computing solutions, especially in the context of mobile systems.

Security in general and trusted computing technologies in particular have had a very different trajectory in the history of mobile devices compared to that of personal computers [54]. Various stakeholders had strict security requirements, some of which date back two decades ago, right at the beginning of the explosion of personal mobile communications. For example, standards specifications required ensuring that the device identifier resists manipulation and change [1]; regulatory guidance called for secure storage for radio-frequency parameters calibrated during manufacture; business requirements necessitated ways of ensuring that subsidy locks[1] cannot be circumvented.

These requirements incentivized mobile device manufacturers, chip vendors, and platform providers to deploy hardware and platform security mechanisms for mobile platforms from early on. Hardware-based trusted execution environments (TEEs) were seen as essential building blocks in meeting these requirements. A TEE is a secure, integrity-protected processing environment, consisting of processing, memory, and storage capabilities. It is isolated from the "normal" processing environment, sometimes called the rich execution environment (REE) [39], where the device operating system and applications run. The term "rich" refers to the extensive functionality and, hence, the increased attack surface, in mass market operating systems today. TEEs enable improved security and usability for REE applications by ensuring that sensitive operations are restricted to the TEE and sensitive data, such as cryptographic keys, never leave the TEE.

The academic research community has been engaged in research in hardware-based trusted computing, although not using that specific term, for a long time dating back to the 1970s [5], [106]. Recent research efforts have focused on investigating alternative architectures for trusted computing, developing novel trust anchors using physically unclonable functions (PUFs), and enabling TEEs for resource-constrained devices [21], [97], [98].

Some of the results of these research activities have led to implementation proposals and large-scale deployment via standardization bodies and industry efforts. The Trusted Computing Group (TCG) [101] has been leading the standardization efforts in trusted computing. Global Platform [39] is specifying TEE functionality in mobile devices. Various application-specific standardization bodies, such as the Car Connectivity Consortium [67],

Manuscript received September 2, 2013; revised June 3, 2014; accepted June 13, 2014. Date of publication July 15, 2014; date of current version July 18, 2014.
N. Asokan is with the Intel Collaborative Research Institute for Secure Computing, University of Helsinki & Aalto University, Aalto FI-00076, Finland (e-mail: asokan@acm.org).
J.-E. Ekberg is with Trustonic, Aalto FI-00076, Finland (e-mail: jan-erik.ekberg@trustonic.com).
K. Kostiainen is with ETH Zurich, Zurich 8092, Switzerland (e-mail: kari.kostiainen@inf.ethz.ch).
A. Rajan and **C. Rozas** are with Intel Labs, Hillsboro, OR 97124-5961 USA (e-mail: anand.rajan@intel.com; carlos.v.rozas@intel.com).
A.-R. Sadeghi is with the Technische Universität Darmstadt, Darmstadt 64289, Germany, and also with the Center for Advanced Security Research Darmstadt (CASED), Darmstadt D-64293, Germany (e-mail: ahmad.sadeghi@trust.cased.de).
S. Schulz and **C. Wachsmann** are with the Intel Collaborative Research Institute for Secure Computing, Technische Universität Darmstadt, Darmstadt D-64293, Germany (e-mail: steffen.schulz@trust.cased.de; christian.wachsmann@trust.cased.de).
Digital Object Identifier: 10.1109/JPROC.2014.2332007

[1]A subsidy lock prevents mobile phones subsidized by a mobile operator from being used by subscribers of a different mobile operator.

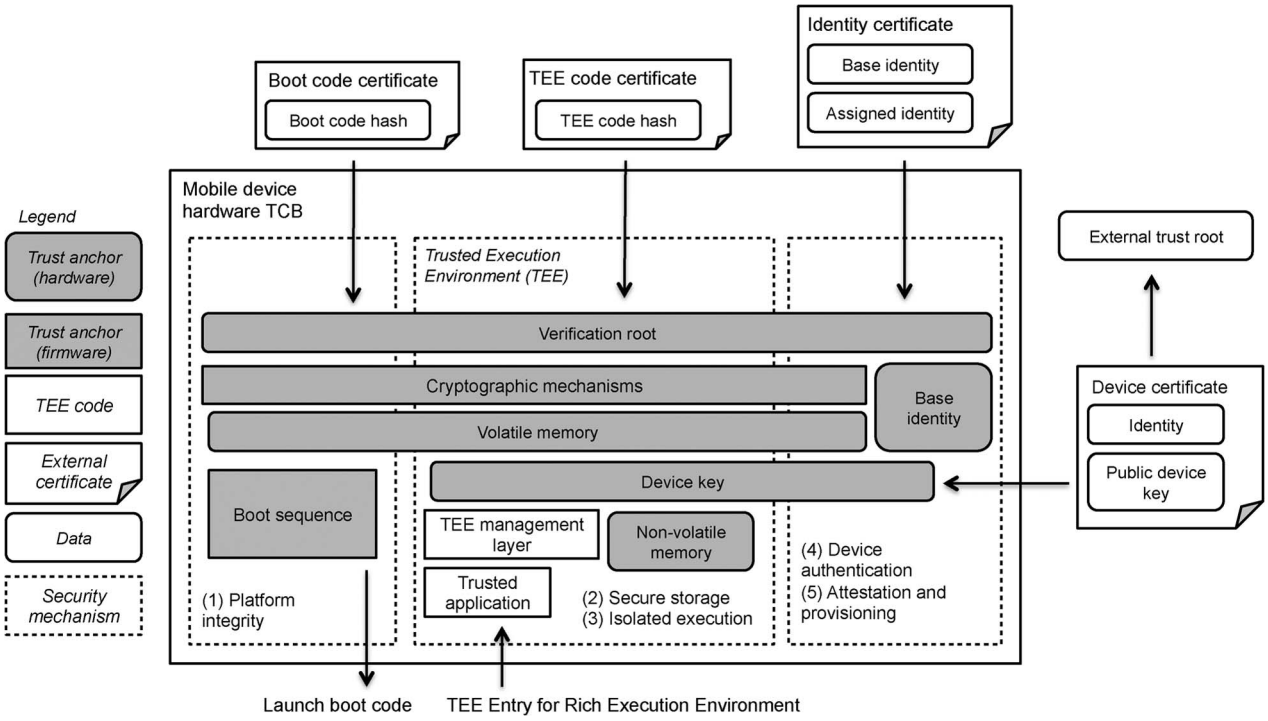

Fig. 1. ***Common hardware security concepts in mobile devices (adapted from [27]).***

are formulating specifications for using trusted computing technologies to address security and privacy problems in specific application areas.

The first mobile phones with hardware-based TEEs appeared almost a decade ago [96]. A common way to realize a TEE in mobile devices is implementing a secure processor mode. An example of such an implementation is ARM TrustZone [9], which is present in smartphones and tablets today. Despite TEE deployment in many of these application areas, there has been no widely available means for application developers to benefit from existing TEE functionality apart from research or proprietary efforts [53].

With emerging standardization, this situation is about to change [27]. In the near future, we expect to see implementations of standardized interfaces for accessing and using TEEs emerging across different platforms. By making trusted computing technologies widely accessible, such a change will spur further research into novel trusted computing technologies and their applications in mobile and embedded devices.

Our goal in writing this survey is to explain the state of trusted computing solutions for mobile devices from research, standardization, and deployment perspectives. While we focus on mobile computing platforms, we also discuss technologies and approaches, such as lightweight trust establishment mechanisms and PUFs, which are relevant to mobile systems. We begin by outlining basic trusted computing concepts and introducing terminology (Section II). We then discuss recent research (Section III), outline current and forthcoming new standard specifications (Section IV), and discuss various solutions proposed and deployed by the industry (Section V). Finally, we conclude by taking a step back and providing a perspective for the outlook of trusted computing technologies in mobile devices, outlining open issues (Section VI).

II. BASIC CONCEPTS

The trusted computing base (TCB) of a mobile device consists of hardware and firmware components that need to be trusted unconditionally. In this survey, we denote such hardware and firmware components as trust anchors of the computing system.

Fig. 1, adapted from [27], illustrates trust anchors present in a typical mobile device. Individual trust anchors are shown in gray. The numbered dotted boxes (1–5) represent common security mechanisms and illustrate the trust anchors needed to implement each mechanism. In Sections II-A1–II-A5, we describe the security mechanisms. We use **bold font** whenever we introduce a concept shown in the figure for the first time.

A. Basic Security Mechanisms

1) Platform Integrity: The integrity of platform code (e.g., the device OS) can be verified either during system boot or at device runtime. This allows device manufacturers and

platform providers to prevent or detect usage of platform versions that have been modified without authorization. Two variations of boot time integrity verification are possible.

In secure boot, the device startup process is stopped if any modification of the launched platform components is detected. A common approach to implement secure boot is to use code signing combined with making the beginning of the boot sequence immutable by storing it within the TCB (e.g., in ROM of the mobile device processor chip) during manufacturing [8]. The processor must unconditionally start executing from this memory location. Boot code certificates that contain hashes of booted code, signed with respect to a verification root, such as the device manufacturer public key stored on the device, can be used to verify the integrity of the booted components. The mobile device must be enhanced with cryptographic mechanisms to validate the signature of the system component launched first (e.g., the boot loader) that can in turn verify the next component launched (e.g., the OS kernel) and so on. If any of these validation steps fail, the boot process is aborted. Integrity of the cryptographic mechanisms can be ensured by storing the needed algorithms in ROM. The immutable boot sequence and a verification root together with an integrity-protected cryptographic mechanism provide the needed trust anchors for secure booting.

In authenticated boot, the started platform components are measured but not verified with respect to any reference values. Instead these measurements are logged in integrity-protected volatile memory. The boot loader measures the first component launched which in turn measures the next one and so on. The recorded measurements represent the state of the platform components after boot, and can be used for local access control enforcement or remote attestation (cf., Section II-A5). Two trust anchors are used to implement authenticated boot: integrity-protected volatile memory and a cryptographic mechanism.

Boot time integrity alone is not sufficient if an attacker can modify the system after it has been booted. In runtime platform integrity verification, a trusted software (or firmware) component monitors the integrity of the platform code continuously [76] and repairs modified components automatically if possible [50]. The integrity of the monitor itself can be verified using the above described boot integrity verification techniques.

2) Secure Storage: A mechanism to store data on the device to disallow unauthorized access by REE components is called secure storage. Sensitive data kept in secure storage should not leak to an attacker even if the REE is compromised. A common way to implement secure storage is to augment the device hardware configuration with a confidential and integrity-protected device-specific key that can be accessed only by authorized code. Such a device key may be initialized during manufacturing and stored in a protected memory area on the processor chip. To protect against key extraction by physical attacks, manufacturing techniques like protective coatings may be used. In addition to the device key, implementation of secure storage requires trusted implementations of necessary cryptographic mechanisms, such as an authenticated encryption algorithm. Data rollback protection requires the inclusion of writable nonvolatile memory (e.g., a monotonic counter) that persists its state across device boots.

To summarize, two trust anchors are needed for secure storage: a device key and cryptographic mechanisms. Note that securely storing cryptographic keys is useful only if cryptographic algorithms using these keys are protected as well.

3) Isolated Execution: The term "isolated execution" refers to the ability to run security-critical code outside the control of the untrusted REE. Isolated execution combined with secure storage constitutes a TEE, which allows implementation of various security applications that resist REE compromise. We explain possible TEE architectures in Section II-B. Here, we introduce the trust anchors needed to implement a TEE, which are a subset of the mobile device hardware TCB. Conceptually, the TEE can be seen as a component of the TCB.

A TEE can expose the functionality of predefined cryptographic mechanisms to the REE with the guarantee that the cryptographic keys never leave the TEE. While predefined common cryptographic operations are sufficient for many services, certain applications require isolated execution of application-specific algorithms. Proprietary one-time password algorithms for online banking constitute one such example. To support isolated execution of arbitrary code, the device hardware configuration must provide an interface (TEE entry) through which the executable code (trusted applications) can be loaded for execution using the protected volatile memory.

A TEE code certificate can authorize code execution within the TEE and authorize trusted applications to access the device key and other device resources such as confidential data (e.g., digital rights management keys) and hardware interfaces (e.g., the cellular modem or near-field communication interface). Furthermore, the access that any trusted application has to the device key and other device resources may be controlled based on the platform state that was measured and saved during an authenticated boot process.

A software or firmware component called TEE management layer provides a runtime environment for trusted applications and enforces access control to protected resources like the device key (more details in Section II-B). The integrity of the management layer must be verified either as part of the boot time platform integrity verification (and runtime monitoring) or on demand when trusted applications are loaded for execution [64]. Realization of isolated execution can make use of the following trust

anchors: isolated memory (volatile or nonvolatile), cryptographic mechanisms, and verification root.

4) Device Authentication: An external service provider can use device authentication to verify the identity of the mobile device (and its TEE). The identity may include device manufacturer information that can imply compliance to external service provider requirements.

The mobile device hardware configuration typically has a unique immutable base identity, which may be a serial number from a managed namespace or a statistically unique identifier initialized randomly at manufacture. A combination of a verification root and the base identity allows flexible device identification. An identity certificate that is signed with respect to the aforementioned verification root binds an assigned identity to the base identity. International mobile equipment identifier (IMEI) and link-layer identities such as Bluetooth and WiFi addresses are examples of device identities.

A device certificate signed by the device manufacturer can bind any assigned identity to the public part of the device key. Signatures over device identities using the device key provide device authentication toward external verifiers.

5) Attestation and Provisioning: An externally verifiable statement about the software configuration running on a device is called remote attestation. Remote attestation allows an external service provider to verify that a device is running a compliant platform version. A common way to implement remote attestation is to provide statements signed with the certified device key over authenticated measurements (e.g., cryptographic hash digests) of the firmware and software components loaded at boot time.

The process of securely sending secrets and code to the TEE of the target device is called provisioning. Many security services require a security association between an external service provider and the TEE of the correct user device. For example, a bank might want to provision a key to the TEE of a customer device for online banking authentication. In some cases, service providers also need to provision TEE code that operates on the provisioned secrets, such as proprietary one-time password algorithms. Device authentication provides the basis for TEE provisioning. Data can be provisioned encrypted under a certified device key. Device certificates do not include user identities and thus provisioning user authentication must be implemented by other means.

Note that all cryptographic keys needed for secure storage, isolated execution, device authentication, attestation, and provisioning can be derived from the same device key.

B. TEE Architecture

The isolation needed for a TEE can be realized in various ways, ranging from separate security elements to secure processor modes and virtualization. Depending on the used isolation technique, different TEE architectures are possible. Fig. 2, adapted from [39], depicts a generic, high-level TEE architecture model that applies to different TEE architecture realizations.

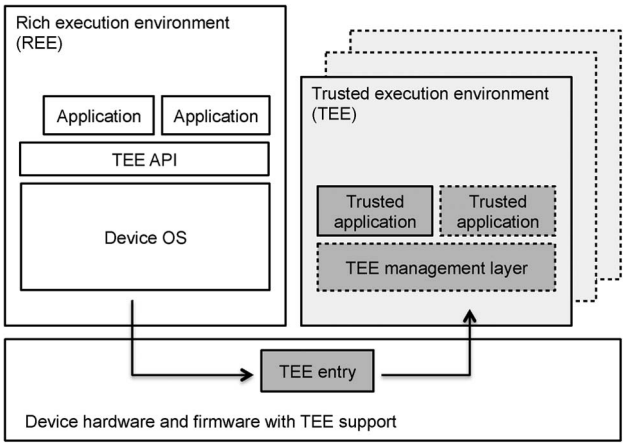

Fig. 2. ***Generic TEE architecture model (adapted from [39]). Trusted applications are executed within a TEE instance that is isolated from the REE device OS. One or more TEE instances may be supported. Dashed boxes illustrate entities that are not present in all TEE architectures; gray boxes are not controlled by the REE device OS.***

We call a processing environment that is isolated from the REE device OS as TEE instance. A TEE architecture realization may support one or more TEE instances. In TEE architectures that are based on dedicated security chips [102] and processor modes [9], [96], typically a single TEE instance is available. Virtualization [63] and emerging processor architectures [65], [72] are TEE examples in which each REE application may create its own TEE instance. TEE instances are created (or activated) and accessed using the TEE entry interface. Applications running in the REE device OS access TEE services through a TEE application programming interface (API) that allows REE applications to execute trusted applications and to read and write data to and from them.

If only a single TEE instance is available, the same TEE instance typically allows execution of multiple trusted applications. The TEE management layer can be implemented in software as a full-fledged embedded OS, a set of libraries, a small interpreter that runs within the TEE or in device hardware and firmware. It provides the interface through which trusted applications communicate with REE applications and invoke cryptographic operations within the TEE. In terms of size and complexity, the management layer is likely to be significantly smaller than the REE device OS, and thus, its attack surface is smaller. In TEE architectures, where each REE application creates its own TEE instance, a management layer may not be used.

III. RESEARCH SOLUTIONS

In the following, we discuss various concepts and research efforts that continue to extend and improve trusted computing research. Observe that we mainly focus on the recent research directions in mobile and embedded systems, while trusted computing following the approach used by the TCG is covered in Section IV. A detailed survey on research in traditional trusted computing is available in [75].

A. Alternative Trusted Computing Designs

One of the earliest works that describe the use of secure coprocessors to assure isolated execution (cf., Section II-A3) is the report on the 4758 Secure Coprocessor [24]. It describes the design of a physically isolated, tamper-resilient execution environment which implements a TEE that communicates with the central processing unit (CPU) to execute certain tasks securely on a separate processor and memory [108]. Following this work, it was investigated how remote parties can gain assurance that a particular application has been executed in the TEE of some particular device (cf., Section II-A5). A trust chain was devised by which the TEE itself can vouch for the execution of a particular code, which in turn may load and execute other code [95]. The device key of the TEE is embedded by its manufacturer, who vouches for the correct operation of that TEE.

Drawbacks of secure coprocessors are the high additional costs and the generally low computation performance. Copilot [76] alleviates this problem by using the coprocessor only to monitor and assure the integrity of the actual computation performed by the main CPU. Overshadow [20] uses hardware-assisted virtualization to enforce different views on memory for user applications and OS kernels, thus ensuring the integrity and confidentiality of applications despite OS compromise.

Some works have also investigated the extension of the CPU itself to enable the measurement of executing code and to establish a TEE. For instance, the AEGIS system architecture [99] extends the CPU interface with facilities for loading, measuring, and authenticating software modules, and uses these facilities to provide authenticated execution of tasks in real (nonvirtual) memory. Similarly, it was proposed that a CPU vendor could provide trusted software modules (TSMs) [23]. The code segments of TSMs are extended with authentication codes which are automatically verified when they are loaded into the cache banks of a CPU during execution.

Leveraging such a trusted loader or regular secure/authenticated boot (cf., Section II-A1), a minimal security kernel can be launched which then in turn ensured a measured and isolated execution of software tasks. In particular, the PERSEUS system architecture [77] proposes to leverage a secure microkernel for strong isolation between a multitude of software security services. The next-generation secure computing base (NGSCB) [29] proposes an ultimately trusted security kernel to support secure applications in a secure world mode, while Terra [34] argues that a chain of trust must be established from platform initialization to the hypervisor and the individual executed applications. Trusted hypervisors such as sHype [85] and TrustVisor [63] follow this design and use a minimal security kernel that provides strong isolation between higher layer applications.

B. Remote Attestation

Remote attestation (cf., Section II-A5) begins with the initial measuring of the bootloader and OS [48], [91]. Integrity measurement architecture (IMA) [44], [86] extends the Linux kernel with facilities to measure loaded code and data according to predefined policies. During attestation, the software measurements maintained by the kernel can then be signed by the device key (cf., Section II-A2) and the kernel in turn can be verified based on the measurements performed by the bootloader and platform firmware. As an alternative, secure OS kernels such as PERSEUS or TrustVisor only measure certain isolated security services, which are then used by regular applications to perform secure transactions on their behalf [63], [77]. The security services are designed to provide maximum flexibility while maintaining low internal complexity and external dependencies, thus simplifying the process of measuring, validating, and establishing trust in a particular software [3], [88].

When extending a secure channel protocol with remote attestation, care must be taken that the reported measurements actually originate from the particular platform that is to be attested [40]. Multiple works have proposed protocol extensions for secure channels such as SSL and IPsec [10], [88] and extend the resulting networks into security domains of assured distributed access control enforcement (e.g., [17] and [62]).

A general problem in remote attestation is the disclosure of the often privacy-sensitive software state to the verifying entity (verifier). To address the problems of privacy but also scalability when dealing with large amounts of software integrity measurements, property-based attestation [19], [83] proposes to attest concrete properties of software. For this purpose, the loaded software is equipped with property certificates which ensure that the software has certain properties. During attestation, the platform then proves the existence of the required properties of the loaded software to the verifier. However, the identification and extraction of the desired software security properties from software remains an open problem [70].

C. Low-Cost Trusted Execution Environments

With the rise of resource-constrained embedded systems as part of complex monitoring and control infrastructures, a recent line of research investigates the possibility to perform attestation (cf., Section II-A5) and isolated execution (cf., Section II-A3) even on such

low-end devices. These works typically assume that common approaches like secure coprocessors or complex CPU modes are too expensive in terms of production cost or energy consumption. Instead, they aim to provide a limited trusted computing functionality for the purpose of automated verification and trust establishment in larger IT infrastructures.

1) Software-Based Attestation: If the mobile device does not support a hardware-protected key needed for remote attestation (as described in Section II-A5), attestation can be implemented in software. A typical software-based attestation scheme exploits the computational constraints of a device to make statements about its internal software state [92], [93]. The prover must compute a response to a given attestation challenge within a certain time. When receiving the correct response in the expected time, the verifier has assurance that only a specific attestation algorithm could have been executed within that time frame. The attestation algorithm is typically implemented as a specific checksum function that iteratively merges information gathered from the device. A formal analysis of software-based attestation [12] has shown the challenges of formalizing the underlying assumptions.

Several variations and extensions to software-based attestation have been proposed, ranging from implementations for different platforms to more fundamental changes to the software-based attestation concept, such as repeated challenge–response procedures [45], [58] or using memory constraints [33], [104], and self-modifying or obfuscated algorithms to prevent manipulation of the attestation algorithm [37], [92], [94]. Multiple works consider the combination of software-based attestation with hardware trust anchors such as TPMs [55], [87] and SIM cards [45] to authenticate the prover device.

2) Minimal Attestation Hardware: The secure minimal architecture for root of trust (SMART) [21] is designed to enable remote attestation and isolated execution at the lowest possible hardware cost (see also [31]). SMART realizes this using a custom access control enforcement on the memory bus, allowing access to a particular secret key in memory only if the current CPU instruction pointer (IP) points to a known trusted code region in ROM (secure storage). This way, the secret key is only accessible when the CPU is executing that trusted code and can thus be used to authenticate the execution of that ROM code to other parties. In particular, by letting the trusted ROM code measure and execute arbitrary code, the design can be extended to a freely programmable trusted execution mechanism or simply be used to attest the local platform.

While SMART is more efficient and easier to validate than software-based attestation, it suffers from certain practical drawbacks. In particular, SMART offers no exception or interrupt handling, requiring a platform reset and memory clearing in case of unexpected errors. To prevent interruption of the trusted code, the hardware access control in SMART assures that the corresponding code region can only be entered at the first address and exited at its last address. However, memory protection based on the CPU instruction pointer may still be exploited with code reuse attacks, where the semantics of code is changed based on stack or other data manipulation [30].

3) CPU-Based Task Protection: Another approach to isolated execution and possibly low-cost trusted execution are self-protected modules (SPMs) [98]. They extend the CPU instructions to provide trusted execution based on execution-dependent memory protection, allowing tasks to request protected memory regions and query the protection status of other tasks in the system directly from the CPU. This way, protected tasks can inspect and attest each other in local memory. For communication and multitasking, protected tasks can declare code entry points which may be called by other tasks with the desired arguments, while other portions of code are protected by the platform. However, when communicating with other tasks on the local platform, one needs to assure that the other task's entry points and protection status have not been changed since the last local attestation.

Sancus [72] extends an openMSP430 CPU to implement SPMs in hardware. However, the problem of handling interrupts and unexpected software faults remains unsolved, and additional modifications are required to sanitize the platform memory upon device reset. To assure to local tasks that a particular other task has not been modified (e.g., by malware), the CPU provides a number of cryptographic tokens and secure hashes of individual loaded tasks. As a result, Sancus imposes relatively high hardware costs for the targeted low-end systems, imposing a 100% area increase for providing eight secure modules [72]. By managing tasks through CPU instructions, Sancus imposes certain restrictions on the memory layout of a task, e.g., limiting capabilities for shared memory or peripherals input/output (I/O). Another implementation of SPMs is provided in the Fides hypervisor [97]. Fides can provide secure interruption and communication between processes, which, however, seems to be achievable also with typical task isolation by trusted hypervisors or security kernels.

4) Execution-Aware Memory Protection: TrustLite [51] extends the concepts of SMART [21] and SPM [98] to provide a programmable, execution-aware memory protection subsystem for low-cost embedded devices. TrustLite's execution-aware memory protection unit (EA–MPU) allows running a number of protected tasks (trustlets) in parallel without requiring additional CPU instructions. Moreover, the EA–MPU can be programmed to provide individual trustlets with shared memory and exclusive peripherals access, enabling the construction of secure device drivers and other platform services.

TrustLite also proposes a modified CPU exception engine to prevent information leakage to OS interrupt handlers. This allows the OS to perform preemptive multitasking of trustlets similar to regular tasks, thus facilitating integration of trustlets with existing software stacks.

To address the assumption of SMART and Sancus that all system memory is cleared on platform reset, TrustLite deploys a secure loader that initializes the EA–MPU at boot time, thus allowing an efficient reallocation and protection of sensitive memory regions prior to REE invocation. Additionally, instead of having the hardware managing identification tokens for secure interprocess communication (IPC) as in Sancus, TrustLite assumes that low-cost embedded systems do not require the reallocation or upgrade of TEE tasks at runtime but that TEEs can remain in memory until platform reset.

D. Physically Unclonable Functions

In many scenarios, emerging low-cost devices are exposed to physical attacks. Thieves or even rightful users may attempt to extract cryptographic keys, to clone the device or to manipulate its software. However, protecting secure storage (cf., Section II-A2) against hardware attacks requires the integration of expensive physical security mechanisms that are not economical for low-cost devices. In this context, PUFs represent a promising new security primitive, which enables unique device identification and authentication [78], [84], binding software to hardware [25], [56], secure storage of cryptographic secrets [59], and remote attestation protocols [90].

1) PUF Concept, Properties, and Assumptions: A PUF is a physical object, e.g., an integrated circuit [60] that, when queried with a challenge, generates a response which depends on both the challenge and the unique device-specific physical properties of the PUF. PUFs are typically assumed to be robust, unclonable, unpredictable, and tamper evident [11]. Informally, robustness means that, when queried with the same challenge multiple times, the PUF returns a similar response with high probability. Unclonability demands that it is infeasible to produce two PUFs that cannot be distinguished based on their challenge–response behavior. Unpredictability requires that it is infeasible to predict the PUF response to an unknown challenge, even if the PUF can be adaptively queried for a certain number of times. A PUF is tamper evident if any attempt to physically access the PUF irreversibly changes its challenge–response behavior.

Since PUFs are affected by operating conditions, such as ambient temperature variations, they return slightly different responses when queried with the same challenge multiple times. Furthermore, PUF responses are not uniformly random. Hence, PUFs are typically combined with fuzzy extractors [22], which map similar PUF responses to the same value (error correction) and extract full-entropy bit strings from the PUF response (privacy amplification).

2) PUF Types: There is a variety of PUF implementations (see [60] and [82] for an overview). The most appealing ones for the integration into electronic circuits are electronic PUFs, which come in different flavors. Delay-based PUFs are based on race conditions or frequency variations in integrated circuits and include arbiter PUFs [57] and ring oscillator PUFs [36]. Memory-based PUFs exploit the instability of volatile memory elements, such as SRAM cells [41], flip-flops [103], and latches [56].

3) PUF-Based Device Authentication: Device authentication (cf., Section II-A4) typically relies on a secret key securely stored in the device. While classical approaches to secure storage (cf., Section II-A2) may be too expensive or even technically infeasible for resource-constrained embedded devices, PUFs promise to provide a lightweight alternative to secure device authentication. The most common approach [78] of using PUFs for device authentication is that the device manufacturer stores a set of challenge–response pairs (CRPs) in a database which can later be used by a verifier to identify the device. However, a general problem of this approach is that CRPs cannot be reused since this would enable replay attacks. A more practical approach is based on standard authentication protocols and stores the authentication secret in a PUF-based key storage (cf., Section III-D4).

4) Secure Key Generation and Storage: Classical approaches to secure storage (cf., Section II-A2) are often not suitable for low-cost embedded systems. In this context, PUFs can be used to securely bind secrets (such as cryptographic keys) to a device. Instead of storing the key in secure nonvolatile memory, the key is extracted from the physical properties of the underlying hardware each time it is used [25], [56]. This protects the key against unauthorized readout by invasive attacks, such as probing attacks against nonvolatile memory. Moreover, when using a tamper-evident PUF, any attempt to physically extract the key changes the PUF and securely deletes the key.

5) PUF-Based Remote Attestation: Software-based attestation (cf., Sections II-A5 and III-C1) implies that, due to the lack of secure storage, cryptographic schemes that rely on secrets cannot be used. However, software-based attestation assumes that the prover device is authenticated to the verifier, which is hard to achieve without using cryptographic authentication. To overcome this problem, the attestation algorithm executed by the prover must be linked to the hardware it was computed on, which can be achieved by using PUFs [89], [90]. To assure that the attestation algorithm is not outsourced to another device, the constraints of the communication interfaces of the prover are exploited similar to the way the computational constraints of the prover are exploited by standard software-based attestation. Due to the uniqueness of the PUF responses and their tight integration into the

attestation algorithm, a correct and timely attestation response provides assurance of the identity of a remote device as well as on the integrity of its software state.

6) Security of PUF-Based Solutions: In contrast to most cryptographic primitives, whose security can be related to well-established intractability assumptions, the security of PUFs relies on physical properties. Many PUF security models exist (see [11] for an overview) that, however, do not capture the properties of real PUF implementations, which can lead to attacks on PUF-based systems [81]. A PUF security framework that aims to capture the properties of real PUF implementations and that allows for empirically assessing and quantifying these properties for PUF implementations has been presented in [11]. A large-scale security analysis of application-specific integrated circuit (ASIC) implementations of the most popular electronic PUF types [13], [47] shows that PUF implementations are sufficiently robust but not all of them achieve the desired security properties (e.g., unpredictability).

Most known implementations of PUFs can be emulated in software [57], [80], either by reading them out completely or by model building attacks. The complexity of these attacks can be increased by obfuscating the actual PUF response [35], [61]. However, this requires protecting the implementation of the algorithms obfuscating the PUF response against invasive and side-channel attacks. Research on the side-channel analysis of PUF-based systems has recently started [46], [66], [74].

IV. STANDARDIZATION

Industry standards consortia have recently intensified efforts to standardize TEE functionality and its interfaces. Standardization aims at agreeing on common APIs for provisioning and trustworthy execution across devices and software ecosystems and the ability to subject the TEE to compliance and security certification. In this section, we provide a brief overview of relevant standards dealing with TEE functionality in a bottom–up manner.

Standards that define services which make use of TEE functionality are important as well. Examples include Car Consortium [67] and MobeyForum [68]. Due to lack of space, we do not discuss these standards further.

A. National Institute of Standards and Technology

The U.S. National Institute of Standards and Technology (NIST) draft 800.164 [18] provides definitions and terminology for many aspects of mobile hardware security ranging from ownership roles to policy enforcement and usage scenarios like "bring your own device" (BYOD). But its most significant contribution is its unified categorization for roots of trust (RoTs), which is NIST's term for hardware trust anchors.

NIST guidelines clearly and concisely collect and describe RoTs for reporting, verification, storage, and measurement. In particular, NIST identifies a new RoT for integrity, which was not previously discussed in other work. This RoT represents the isolated environment where measurements and trusted state assertions can be securely stored when the device is active.

The RoTs are one way to agree on a hardware security foundation for TEEs. Each RoT can be evaluated and graded according to the level of security it can provide to the system. More importantly, as we mentioned in Section II, RoTs are the abstract tools on which the main capabilities of a TEE system are built: isolation, secure storage, and integrity.

B. Global Platform (GP)

The Global Platform Device Specifications, in particular the architecture document [39], have established the reference model and terminology used for TEE software architectures we introduced in Section II. The trusted applications (TAs) that run in the TEE are written using the TEE internal API [38], which is the reference library interface for TAs. The internal API includes interfaces for cryptographic functions, secure storage, and I/O. Especially the parameter passing paradigm between the REE device OS and TAs is a significant divergence from what typically is available for traditional secure elements such as smart cards. A GP TA gets its input and provides its output using memory references residing in the address space of the caller in the REE device OS. This allows TAs to access and process client-side memory directly, e.g., for in-place decryption or the processing of vast client-side data structures, say, for runtime integrity checking.

C. Mobile Hardware Security APIs

Some mobile platforms provide APIs for hardware-assisted cryptographic operations. Java MicroEdition, widely used in feature phones, defines JSR 177 [73] as a generic smartcard-like interface which can be used to expose a standard cryptographic API (the implementation may be provided by a mobile phone secure element such as a SIM card). Recent versions of the Android platform expose an API for hardware-assisted cryptography [28] in the form of a standard PKCS 11 interface [79], while in iOS similar functionality is provided through a proprietary API [7].

These hardware-security APIs have been modeled after usage paradigms of hardware security modules (HSMs), cryptographic tokens, and smart cards. A traditional hardware security API allows creation of hardware-protected keys and common cryptographic operations, such as encryption and signatures, using these keys. To take advantage of the programmability of mobile TEEs (isolated execution), a different kind of API abstraction is needed. The API should address provisioning of trusted applications and secrets into the device, authorization of trusted applications to access provisioned secrets and device keys, and control which REE application can execute trusted applications. None of the current standardized or

de facto proprietary hardware-security APIs provide such functionality.

D. Trusted Computing Group

The trusted platform module (TPM) [102], defined by the TCG, is a functional interface for platform security. TPMs are widely deployed today. A TPM contains functions for key generation and use, mainly with asymmetric encryption and signature primitives, but also for symmetric cryptography. All TPM processing happens in isolation from the REE OS and the caller, governed by the RoTs described in Section IV-A. Furthermore, TPMs provide primitives for data sealing and unsealing, monotonic counters, randomness, and some limited amount of nonvolatile storage.

1) Platform Configuration Registers (PCRs): One notable distinction of TCG specifications is that they provide platform binding as an inherent service for the authorization and attestation of TPM objects and functions. This feature sets it apart from other standards such as GP. All TPM implementations provide some number of PCRs which are integrity-protected writable volatile memory that reside within the TPM. PCRs are used to cryptographically aggregate successive measurements of software/hardware components or configurations originating from the REE. Aggregating a new measurement to a PCR is known as PCR extension and is realized using a cryptographic hash function. If the REE OS consistently provides measurements of all its code to the TPM, before executing the measured code, then the set of PCRs serve as a representation of the state of the currently running REE software. PCRs have two uses: binding TPM objects to REE state and reporting REE state to external verifiers as part of remote attestation (cf., Section II-A5). TPM objects such as keys and stored data can be associated with an assertion that restricts the use of the object only when the trusted platform is in a certain predefined state, expressed as set of predefined reference values for PCRs. Generating a signed report of PCR values with a certified TPM-specific key is the essential step in remote attestation of the REE state.

2) TPM Mobile: TPM mobile specifications [previously known as mobile trusted module (MTM)] allow a TPM to be realized in software within a TEE as a TA. This makes it possible to have more than one active TPM instance in a device. This multistakeholder model allows different stakeholders (such as device manufacturers, mobile operators, enterprises, and users) to run their "own" TPMs and populate their TPM with measurements relevant to their needs, independently of other TPMs active on the same system.

3) TPM2 Authorization: A new revision (v2.0) [102] of the TPM standards (TPM2) are currently on the road to publication. The new specifications make improvements in various aspects, such as algorithm agility and the use of nonvolatile memory. The most novel aspect of TPM2 is its enhanced authorization model. In this section, we illustrate the power of this model by showing how it can be used for secure boot, which is a common need for mobile and embedded devices.

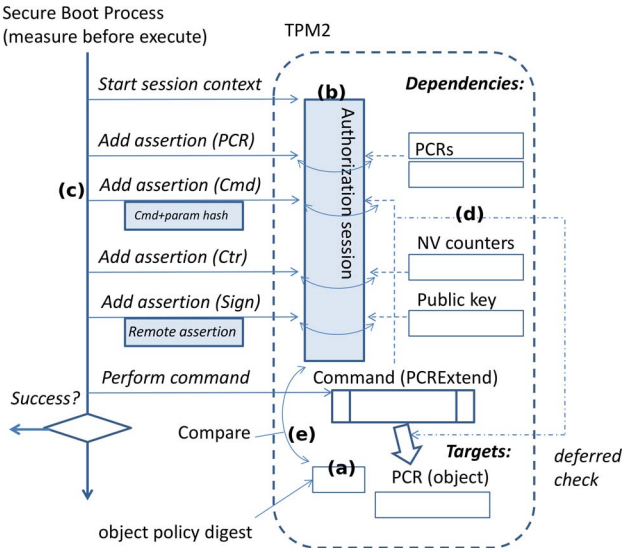

Fig. 3. ***Secure boot with TPM authorization.***

The guiding principle for the authorization model is separation of policy and mechanism [107]. This allows designing the secure boot of a platform from the OS upward in a hardware-independent manner, where any binary relation can be applied as a policy assertion. We will now explain an example secure boot policy depicted in Fig. 3.

In version 1.x of the TPM specifications, the means of expressing access control policies is mainly limited to passwords, and many object operations (e.g., extending PCRs) cannot be associated with a policy. For example, a device could conceivably implement secure boot by aggregating a sequence of measurements of launched software components in a PCR and aborting the boot process if the PCR does not reach an expected reference value. The reference value would, however, have to be encoded in data objects (certificates) outside the TPM. Naturally, this requires a rigid specification of the mechanism in order to make it apply across manufacturers and systems.

In the example in Fig. 3, we use TPM2 authorization to make secure boot conditioned on the successful completion of the TPM2 command PCRExtend to extend a specific PCR. We must, therefore, associate the policy for secure boot with the ability to perform this operation.

Most TPM2 objects can be associated with a single, unique object policy digest [cf., Fig. 3(a)], which is permanently stored with the object. In order to get authorization for object access, the caller must run an authorization session [cf., Fig. 3(b)] that contains a rollback-protected,

accumulated session policy digest. The session is updated using TPM policy commands [cf., Fig. 3(c)] that each implement a policy assertion, which consists of some system evidence [cf., Fig. 3(d)] that the TPM2 knows about. This evidence is aggregated into the session digest in a well-known and well-defined manner. Session policy digests are aggregated in the same way as PCRs are extended, i.e., by using a cryptographic hash function.

In our example session, the secure boot process makes three assertions at the beginning: the first one aggregates the current state of the system, represented by some source PCRs; the second assertion binds the exact target command (PCRExtend), along with its parameters such as the identity of the target PCR and the value it is going to be extended with; the third assertion binds the value of a counter (for example, specifying a bound for the current version number of the firmware being booted). After all these assertions have been aggregated, the session policy digest will end up with a specific value Q. The final assertion in our example (remote assertion) is an external signature on a structure asserting the expected value Q' of a session digest when all prior assertions have been completed. If this signature can be validated with a public key P loaded into the TPM and the session digest matches reference value $Q = Q'$, then the session digest is replaced with a unique representation of the public key[2] that validated the signature, say $W = H(P)$. If we had set the reference object policy for the target PCR [cf., Fig. 3(a)] to be the same value W, then issuing the PCRExtend command in the context of the authorization session will succeed as long as both the digests match [cf., Fig. 3(e)] and deferred checks on conditions implied by the provided assertions hold.[3] If the policy values do not match or the deferred assertions fail, then the PCRExtend invocation returns with a failure code. Based on this, the caller should take appropriate action, like resetting the device.

In other words, this process effectively allows having a remotely controlled authorization policy where an external signature controls which policies are allowed. The remote controller can issue signed remote assertions for multiple different session digests, each representing an alternate acceptable configuration for secure boot. TPM2 also supports ways of securely mapping accumulated session policy digests to perform logical access control decisions.

One example is the PolicyOR command. On the condition that the current policy digest value matches one of a number of values in a set provided as a PolicyOR parameter, then the current digest value is replaced with a representation of the entire set of values.

All the assertions listed above exist in the published TPM2 command set: a secure boot process along these lines is viable with a TPM2. As the caller cannot affect the contribution of any policy assertion, the only way the authorization can succeed is that the state described by the assertions actually is the one represented in the external signature, thus achieving separation of mechanism and policy.

Note that TPM2 is a passive entity. To implement secure boot using TPM2, there has to be an external immutable active entity, such as a boot loader, which enforces secure boot. It is customary for such secure boot implementations to allow the user to override secure boot [15].

V. INDUSTRY SOLUTIONS

Over the past years, several trusted computing research concepts have been realized in industry products, and in many cases such products have fostered new opportunities to build and research trusted systems. In the following sections, we review some of the main technologies as well as standardization efforts.

A. Virtualization and Dynamic Root of Trust

Many mobile and ultramobile laptop platforms feature hardware-assisted virtualization technology, such as Intel virtualization technology (Intel VT). A central design goal of Intel VT was to simplify the implementation of robust hypervisors. Intel VT adds two new operation modes: VMX root mode for hypervisors and VMX nonroot mode for virtual machines. VMX root mode is very similar to the normal Intel architecture without Intel VT while VMX nonroot mode provides an Intel architecture environment controlled by a hypervisor. A virtual-machine control structure (VMCS) was introduced to facilitate transitions between VMX root mode and VMX nonroot mode and can be programmed by the hypervisor to establish boundaries on a VM, including access to memory, devices, and control registers. While operating in VMX nonroot mode, the execution of certain instructions and events causes a transition to VMX root mode called a VMexit. The hypervisor can retrieve details as to the cause of the VMexit by reading the VMCS and process the event accordingly [71]. Intel VT introduced a generalized IO–MMU architecture which enables system software to define constraints on direct memory access (DMA) devices, restricting their access to specific subsets of physical memory allowing for a smaller TCB [2].

Another major capability of modern systems is the dynamic root of trust for measurement (DRTM). Available as Intel trusted execution technology (Intel TXT) or AMD secure virtual machine, this technique enables a CPU to perform a runtime reinitialization and establish a new software TCB (TEE payload), irrespective of the

[2]The public key used to verify the signature assertion is loaded into the TPM from the outside by the secure boot process. However, the TPM will extend the used public key into the session digest which ensures that the subsequent command will succeed only if the correct public key was used to verify the signature.

[3]Some assertions, like, e.g., the target command assertion, cause a deferred check to be logged in the session and validated during actual command execution in addition to the update of the session digest at the time the assertion is made.

trustworthiness of previously loaded software. For this purpose, the TCG TPM was extended with a set of DRTM PCRs which can be reset at runtime by the CPU by sending a TPM command from the appropriate operation mode (TPM locality). The Intel GETSECS [SENTER] instruction initiates the DRTM. The CPU resets the DRTM PCRs and loads an authenticated code module (ACM) into an isolated execution environment. The ACM performs a series of platform configuration checks, configures DMA protection for the TEE payload, and extends the TEE payload hashes into the TPM PCRs.

DRTM technology has been used to securely execute critical software payloads such as SSH logins, X.509 e-mail signatures, or to protect banking secrets [16], [32], [64]. Intel TXT has also been used in combination with Intel VT to initiate a trusted hypervisor, which in turn provides multiple TEEs to the individual running VMs [63]. The generalized IO–MMU allows hypervisors to be "disengaged," i.e., to only perform an initial configuration of VM boundaries, thus providing only a minimal external interface and complexity [49]. Alternatively, a "disaggregated" hypervisor may reduce its TCB by delegating drivers for peripherals control to other VMs [69], or to construct a trusted path, providing secure user I/O for TEEs [110].

B. Userspace Trusted Execution

Intel software guard extensions (Intel SGX) are a set of new instructions and memory access changes to the Intel architecture to support TEEs. The extensions provide the ability to instantiate one or more TEEs (enclaves) that reside within an application inside an REE. Accesses to the enclave memory area against software (not resident in the enclave) are prevented by hardware. This restriction is enforced even from privileged software, such as operating systems, virtual machine monitors, and the basic input/output system (BIOS).

The enclave lifecycle begins when a protected portion of an application is loaded into an enclave by system software. The loading process measures the code and data of the enclave and establishes a protected linear address range for the enclave. Once the enclave has been loaded, it can be accessed by the application as a service or directly as part of the application. On first invocation, the enclave can prove its identity to a remote party and be securely provisioned with keys and credentials. To protect its data persistently, the enclave can request a platform-specific key unique to the enclave to encrypt data and then use untrusted services of the REE.

To implement Intel SGX memory protections, new hardware and structures are required. The enclave page cache (EPC) is a new region of protected memory where enclave pages and structures are stored. Inside the EPC, code and data from many different enclaves can reside. The processor maintains security and access control information for every page in the EPC in a hardware structure called the enclave page cache map (EPCM). This structure

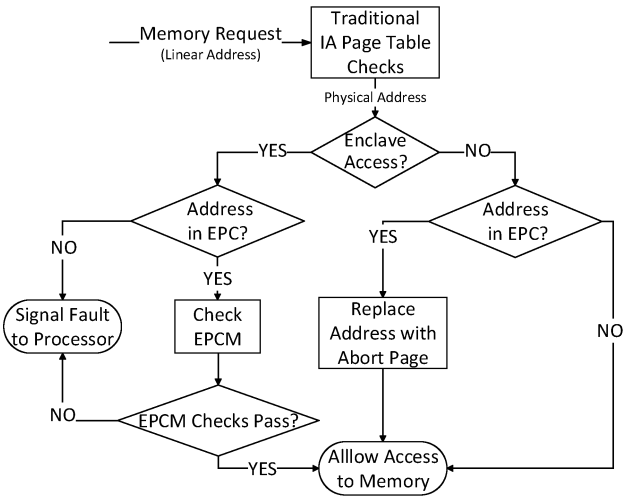

Fig. 4. ***High-level architecture of Intel SGX PMH. Processor memory requests are sent to the PMH for translation and access control checks. As part of Intel SGX, the PMH has been extended to check whether a memory access was initiated by an enclave. For nonenclave accesses, the PMH redirects any access to the EPC to nonexistent memory (abort page). For an enclave access (an access by enclave to its protected linear address range), the PMH checks that the translated address is an EPC page. Furthermore, the PMH consults the EPCM to verify that the EPC page belongs to enclave requesting access, the correct linear address was used to access the page, and access permissions are consistent with the request.***

is consulted by the processor's page miss handler (PMH) hardware module, as shown in Fig. 4. The PMH mediates access to memory by consulting page tables maintained by system software, range registers, and the EPCM. A memory encryption engine (MEE) protects the EPC when using main memory for storage [65].

Enclave binaries are loaded into the EPC using new instructions. ECREATE starts the loading process and initializes the Intel SGX enclave control structure (SECS) which contains global information about the enclave. EADD loads a page of content into a free EPC page and records the commitment into the SECS. Once the EPC page has been loaded, the contents of the page are measured using EEXTEND. After all the contents of the enclave have been loaded into the EPC, EINIT completes the creation process by finalizing the enclave measurement and establishes the enclave identity. Until an EINIT is executed, enclave entry is not permitted.

Once an enclave has been loaded, it can be invoked by application software. To enter and exit an enclave programmatically (e.g., as part of a call/return sequence), new instructions, EENTER and EEXIT, are provided. While operating in enclave mode, an interrupt, fault, or exception may occur. In this case, the processor invokes a special internal routine called asynchronous exit (AEX) which saves and clears the enclave register state and translation lookaside buffer (TLB) entries for the enclave. The

ERESUME instruction restores the processor state to allow the enclave to resume execution.

To enable attestation and sealing, the hardware provides two additional instructions EREPORT and EGETKEY. The EREPORT instruction provides an evidence structure that is cryptographically protected using symmetric keys. EGETKEY provides enclave software with access to keys used in the attestation and sealing process. A special quoting enclave is devoted to remote attestation. The quoting enclave verifies REPORTs from other enclaves on the platform and creates a signature using a device specific (private) asymmetric key [4].

Intel SGX minimizes the TCB of trusted applications to the critical portion of the application and hardware. This simplifies the creation and validation of remote attestation reports, as remote verifiers no longer have to understand multiple TEE management layers and their dependencies. While requiring CPU extensions, Intel SGX does not require any dependencies on the TPM, a hypervisor, or a separate trusted operating system. Further, it is protected against hardware and software attacks on RAM. Finally, Intel SGX enables application developers to directly deploy trusted applications inside REE applications [42].

C. Mobile Architectures Using Secure Processor Modes

ARM TrustZone [9] and TI M-Shield [96] are system-wide, mobile security architectures that leverage a secure execution mode provided by the device main CPU. TrustZone is deployed to the majority of current smartphones, whereas M-Shield is an example of a previous, similar architecture. In this section, we focus on TrustZone.

The main CPU of the mobile device supports two execution modes called normal world and secure world. The processor boots to the secure world which sets up the necessary runtime environment before switching to the normal world. Execution can switch back to the secure world when a special command is executed in the normal world. This command starts a monitor mode that performs the processor mode switch. The designer of a mobile device hardware configuration defines the hardware components that are accessible in these two modes.

Fig. 5 illustrates an example hardware configuration in a TrustZone-enabled mobile device [9]. The device main CPU, small amounts of RAM and ROM, and the cellular modem are integrated into a system on chip (SoC). These on-chip components are connected with an internal bus. The SoC also includes memory controllers for off-chip memory elements and peripherals.

The access control between these hardware elements is implemented using a status flag that the SoC internal bus carries. The status flag indicates the mode of the master device in bus communication. Bus communication slaves must enforce access control based on the flag. Hardware elements can be made aware of the status flag or dedicated access control hardware can be placed between the bus and the target hardware element. Access to memory elements and peripherals is typically controlled by adding dedicated access control hardware elements between the bus and the hardware element or its memory controller.

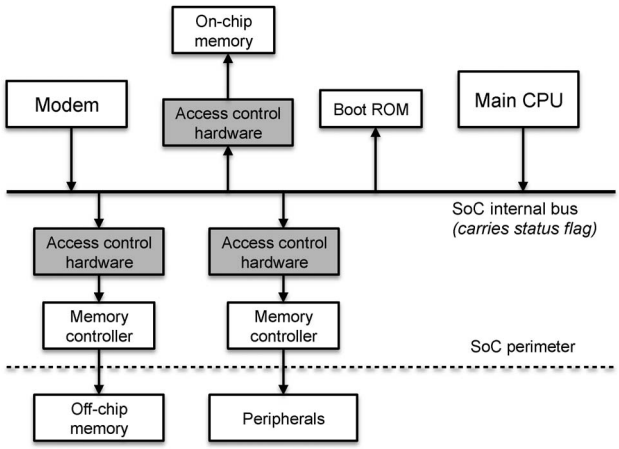

Fig. 5. ***Overview of the ARM TrustZone system architecture [9]. In a typical mobile device, many hardware elements are integrated into a single SoC. Access control between TrustZone normal world and secure world can be implemented with a system bus flag and dedicated access control hardware (gray boxes).***

Typically, on-chip memory is configured for secure world access only, while the off-chip memory elements and peripherals can be partitioned between the secure world and the normal world. Most peripherals are accessible by the normal world. Also interrupt handling can be configured; the processor can switch execution mode for dedicated interrupts, if needed.

Typical uses of TrustZone TEE include secure boot, secure storage, and isolated execution. The underlying trust anchors (device key, cryptographic mechanism, and verification root) can be configured during manufacturing into on-chip memory. The same trust anchors can be used for device authentication and attestation.

D. Secure Elements

Besides processor modes, smartphones support TEEs in the form of separate security elements, such as smart cards. Some mobile devices have extension slots for dedicated smart card TEEs and the majority of mobile devices are equipped with a SIM card. Smart card TEEs provide secure storage and isolated execution. Boot integrity verification, device authentication, and remote attestation are typically not supported by smart card TEE realizations.

E. Onboard Credentials

Although TEE architectures like TrustZone and M-Shield have been deployed to smartphones for almost a decade, and secure elements in the form of SIM cards are present in many mobile devices, the usage of

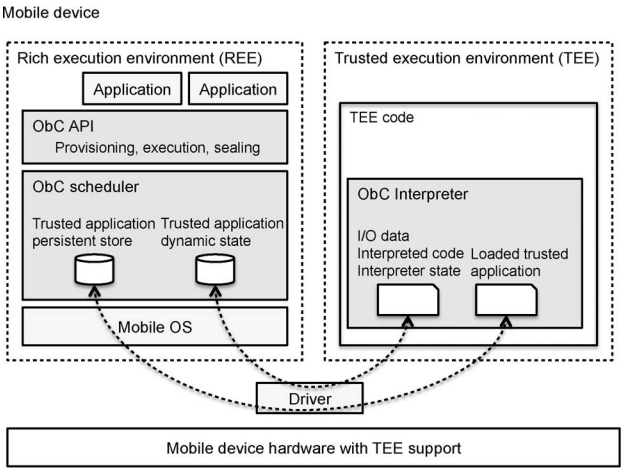

Fig. 6. ***ObC architecture [27]. ObC interpreter is implemented as TEE code. ObC scheduler controls execution of dynamically loaded ObC trusted applications and maintains persistent state. Mobile applications access ObC trusted applications through ObC API that provides provisioning, execution, and sealing functionality.***

hardware-security mechanisms by third-party developers has been limited [27]. Traditionally, mobile device manufacturers have leveraged mobile TEE capabilities only for their internal use cases, such as subsidy lock protection or secure boot and secure element application development has not been available for third parties.

Onboard credentials (ObC) [26], [52], [53] is a TEE architecture developed at Nokia Research Center and currently available in TrustZone-enabled Nokia Windows Phone 8 and Symbian phones. The ObC system serves as an example of TEE architecture that allows developers to utilize programmability of mobile TEEs.

Fig. 6 illustrates the ObC architecture [27]. Trusted applications are executed on top of a small virtual machine called the ObC interpreter (TEE management layer). The interpreter provides isolation between trusted applications originating from unknown developers. Trusted application development can be done in BASIC or using bytecode assembler. The ObC interpreter is implemented as a set of interacting TEE code components. Depending on the underlying TEE hardware the components may permanently reside inside the TEE or be loaded on demand.

To execute a trusted application, the ObC scheduler loads along with its inputs and stored data sealed by previous invocations. The ObC interpreter executes the trusted application bytecode. Some execution events will cause the interpreter to collect its runtime state, encrypt it, and return to the REE for scheduling. The ObC scheduler reinvokes the same or different trusted applications, attaching possible temporarily stored data or the interpreter state, and in this manner the bytecode execution continues to completion. Numerous context switches cause a significant execution overhead, but since the system runs on the mobile device main CPU, the achieved performance is comparable to solutions without scheduling on slower security chips, such as smart cards.

The ObC platform supports an open provisioning model in which any developer can, with the permission of the device user, deploy trusted applications. A device-specific and manufacturer-certified public key provides the basis for remote provisioning; service providers need to handle user authentication additionally. The certified device key can transport a provisioner-specific secret that defines a new security domain. Isolation between security domains inside the TEE is guaranteed by interleaving execution of different security domains in time and implementing local storage with distinct encryption keys.

To develop a complete security service, a service provider needs to deploy a trusted application that handles the service-specific security logic within the TEE, and a REE application that triggers the trusted application execution (and provides a user interface).

Smartphone-based public transport ticketing is an example ObC application that has been deployed in practice [100]. In nongated transport systems, travel tickets are not verified at station gates, but instead travelers are requested to perform ticketing on their own accord but are subject to ticket inspections. Such a model allows a traveller to stop his phone from reporting evidence for trips during which he was not inspected. An ObC trusted application that implements an authenticated counter bound to identity verification signatures can address such situations. A traditional cryptographic API (e.g., PKCS11 [79] or TPM interface [102]) would not enable implementation of authenticated counters. With a programmable TEE, implementation of such a ticketing solution is simple, and deployment to devices already in the field is practical.

F. Physically Unclonable Functions

While PUFs and PUF-based security solutions are still investigated by the research community, security products based on PUFs are already announced for the market [43], [105]. These systems mainly target IP-protection and anti-counterfeiting applications as well as secure key storage and device authentication systems.

VI. OUTLOOK AND SUMMARY

The role of trusted computing features in mobile and embedded devices is at a crossroads. After years of limited use, the imminent arrival of new standards and increased interest on the part of the industry to make trusted computing widely accessible has the potential to increase and change the ways by which application developers and users benefit from these features. The fact that industry sees new opportunities in this domain is evident from the arrival of new products from established companies and the formation of new companies. With increased use of mobile devices, new research problems will become apparent.

On the other hand, several issues need to be addressed before such increased use becomes reality. We discuss some of them below.

Privacy and trust issues: With the improved scalability and security of modern TEEs, previously raised concerns [6] regarding privacy and vendor lock-in are becoming more important. For instance, the implications of combining TPM2 with secure boot are currently subject of intensive discussions in Europe [14], [109]. This calls for further research in industry and academia to analyze and improve the security and privacy implications of this emerging technology.

Future of PUFs as trust anchor: There are many yet unsolved challenges with regard to the scalable and secure integration of PUFs into IT systems. For instance, most existing PUF implementations can be emulated in software and require to obfuscate the actual PUF responses to prevent emulation attacks. In particular, side-channel attacks and invasive hardware attacks can be used to extract PUF responses. Hence, further research should investigate alternative PUF designs that are resistant to emulation attacks and/or the secure integration of the PUF and the logic processing the PUF responses.

Attacker models: Hardware security solutions need to consider a number of attack vectors such as side-channel attacks based on memory management and cache manipulation, power consumption analysis, pipeline analysis, or interface timing attacks. While protections against such threats are known, they are expensive and unsuitable for low-cost devices. The impact of possible attacks can be mitigated by suitable system designs. For example, TEE implementations that make use of chip-external memory effectively extend the TEE across multiple components within a device. Such TEEs are vulnerable to memory probing attacks using physical probes. If an application protected by such TEEs is designed to avoid the use of global keys (keys shared by all devices in the system) or "class keys" (keys shared by a large group of devices), then the impact of a successful attack on a single device can be minimized.

Mitigation of software attacks in hardware: One big challenge is the verification of the runtime integrity of an IT system. Existing approaches to detect and to prevent control flow attacks (such as return-oriented programming attacks) are typically implemented in software and involve a significant runtime overhead. An important line of research, therefore, is to investigate how mechanisms to protect the control flow integrity of IT systems can be effectively realized in hardware to achieve higher performance.

Provisioning: As discussed in Section V-E, the ability for developers to freely provision trusted applications and credentials subject only to user approval (and without necessarily requiring approval from any third parties like device manufacturers or mobile operators) has the potential to rapidly expand uses of TEE. Architectures like ObC (Section V-E) and Intel SGX (Section V-B) facilitate such open provisioning. Global Platform is working on a new "consumer-centric provisioning model" with the same intent.

Scaling down: Just as the popularity and deployment of smartphones exploded during the last decade, the widespread use of low-cost devices in security and privacy-critical infrastructures may follow. Remote attestation and isolated execution are important features impacting the feasibility and scalability of these emerging systems. We discussed multiple proposed solutions in Section III-C, each with their own limitations and cost constraints. However, currently no comprehensive security architecture exists to facilitate the secure operation of low-cost embedded systems.

Peripherals: In addition to secure execution and storage within the TEE, it is necessary to secure their interactions with peripherals. How this can be done in a flexible, yet economic manner is an open question.

Usability: How trusted applications and credentials in a device can be securely and easily backed up or migrated from one device to another on user demand are critical to the usability of trusted computing technologies. Equally important is the consideration of "usability" for developers. What programming paradigms should be used for the development of trusted applications? ■

REFERENCES

[1] 3GPP, "3GPP TS 42.009 security aspects," 2001. [Online]. Available: http://www.3gpp.org/ftp/Specs/html-info/42009.htm

[2] D. Abramson *et al.*, "Intel virtualization technology for directed I/O," *Intel Technol. J.*, vol. 10, no. 3, pp. 179–192, 2006.

[3] A. Alkassar, M. Scheibel, A.-R. Sadeghi, C. Stüble, and M. Winandy, "Security architecture for device encryption and VPN," in *Information Security Solution Europe (ISSE)*. Berlin, Germany: Vieweg-Verlag, 2006.

[4] I. Anati, S. Gueron, S. Johnson, and V. Scarlata, "Innovative technology for CPU based attestation and sealing," in *Hardware and Architectural Support for Security and Privacy (HASP)*. New York, NY, USA: ACM, 2013.

[5] J. P. Anderson, "Computer security technology planning study," U.S. Air Force, Tech. Rep. ESD-TR-73-51, 1972, vol. II. [Online]. Available: http://csrc.nist.gov/publications/history/ande72.pdf

[6] R. Anderson, "Trusted computing: Frequently asked questions," 2003. [Online]. Available: http://www.cl.cam.ac.uk/~rja14/tcpa-faq.html

[7] Apple Inc., "iOS security," 2012. [Online]. Available: http://images.apple.com/iphone/business/docs/iOS_Security_Oct12.pdf

[8] W. A. Arbaugh, D. J. Farber, and J. M. Smith, "A secure and reliable bootstrap architecture *Proc. IEEE Symp. Security Privacy*, 1997, pp. 65–71.

[9] ARM, "ARM security technology—Building a secure system using TrustZone technology," 2009. [Online]. Available: http://infocenter.arm.com/help/index.jsp?topic=/com.arm.doc.prd29-genc-009492c/index.html

[10] F. Armknecht *et al.*, "An efficient implementation of trusted channels based on OpenSSL," in *Proc. ACM Workshop Scalable Trusted Comput.*, 2008, pp. 41–50.

[11] F. Armknecht, R. Maes, A.-R. Sadeghi, F.-X. Standaert, and C. Wachsmann, "A formal foundation for the security features of physical functions," in *Proc. IEEE Symp. Security Privacy*, 2011, pp. 397–412.

[12] F. Armknecht, A.-R. Sadeghi, S. Schulz, and C. Wachsmann, "A security framework for the analysis and design of software attestation," in *Proc. ACM Conf. Comput. Commun. Security*, 2013, DOI: 10.1145/2508859.2516650.

[13] M. Bhargava, C. Cakir, and K. Mai, "Comparison of bi-stable and delay-based physical unclonable functions from

measurements in 65 nm bulk CMOS," in *Proc. IEEE Custom Integr. Circuits Conf.*, 2012, DOI: 10.1109/CICC.2012.6330625.

[14] BMI, "Key requirements on 'trusted computing' and 'secure boot'," German Government white paper, 2012. [Online]. Available: http://www.bmi.bund.de/SharedDocs/Downloads/DE/Themen/OED_Verwaltung/Informationsgesellschaft/trusted_computing_eng.html

[15] J. Bottomley and J. Corbet, "Making UEFI secure boot work with open platforms," Linux Foundation white paper, 2011. [Online]. Available: https://www.linuxfoundation.org/sites/main/files/lf_uefi_secure_boot_open_platforms.pdf

[16] F. F. Brasser, S. Bugiel, A. Filyanov, A.-R. Sadeghi, and S. Schulz, "Softer smartcards—Usable cryptographic tokens with secure execution," in *Financial Cryptography*. New York, NY, USA: Springer-Verlag, 2012.

[17] S. Cabuk *et al.*, "Towards automated security policy enforcement in multi-tenant virtual data centers," *Comput. Security*, vol. 18, pp. 89–121, 2010.

[18] L. Chen, J. Franklin, and A. Regenscheid, "Guidelines on hardware-rooted security in mobile devices," National Institute of Standards Technology (NIST), Tech. Rep. SP 800-164. [Online]. Available: http://csrc.nist.gov/publications/drafts/800-164/sp800_164_draft.pdf

[19] L. Chen, H. Löhr, M. Manulis, and A.-R. Sadeghi, "Property-based attestation without a trusted third party," in *Proc. 11th Int. Conf. Inf. Security*, 2008, pp. 31–46.

[20] X. Chen *et al.*, "Overshadow: A virtualization-based approach to retrofitting protection in commodity operating systems," *SIGOPS Oper. Syst. Rev.*, vol. 42, no. 2, pp. 2–13, 2008.

[21] K. E. Defrawy, A. Francillon, D. Perito, and G. Tsudik, "SMART: Secure and minimal architecture for (establishing a dynamic) root of trust," in *Proc. Netw. Distrib. Syst. Security Symp.*, 2012, Internet Society.

[22] Y. Dodis, J. Katz, L. Reyzin, and A. Smith, "Robust fuzzy extractors and authenticated key agreement from close secrets," in *Advances in Cryptology (CRYPTO)*. New York, NY, USA: Springer-Verlag, 2006.

[23] J. Dwoskin and R. Lee, "Hardware-rooted trust for secure key management and transient trust," in *Proc. 14th ACM Conf. Comput. Commun. Security*, 2007, pp. 389–400.

[24] J. Dyer *et al.*, "Building the IBM 4758 secure coprocessor," *IEEE Computer*, vol. 34, no. 10, pp. 57–66, Oct. 2001.

[25] I. Eichhorn, P. Koeberl, and V. van der Leest, "Logically reconfigurable PUFs: Memory-based secure key storage," in *Proc. 6th ACM Workshop Scalable Trusted Comput.*, 2011, pp. 59–64.

[26] J.-E. Ekberg, "Securing software architectures for trusted processor environments," Ph.D. dissertation, Dept. Comput. Sci. Eng., Aalto Univ., Aalto, Finland, 2013. [Online]. Available: http://urn.fi/URN:ISBN:978-952-60-3632-8

[27] J.-E. Ekberg, K. Kostiainen, and N. Asokan. "The untapped potential of trusted execution environments on mobile devices," 2014. *IEEE Security Privacy Mag.*, DOI: 10.1109/MSP.2014.38.

[28] N. Elenkov, "Jelly bean hardware-backed credential storage," 2012. [Online]. Available: http://nelenkov.blogspot.ch/2012/07/jelly-bean-hardware-backed-credential.html

[29] P. England, B. Lampson, J. Manferdelli, M. Peinado, and B. Willman, "A trusted open platform," *IEEE Computer*, vol. 36, no. 7, pp. 55–63, Jul. 2003.

[30] A. Francillon, Q. Nguyen, K. B. Rasmussen, and G. Tsudik, "Systematic treatment of remote attestation," 2012. [Online]. Available: http://eprint.iacr.org/2012/713.pdf

[31] A. Francillon, Q. Nguyen, K. B. Rasmussen, and G. Tsudik, "A minimalist approach to remote attestation *Proc. Design Autom. Test Eur. Conf. Exhibit.*, 2014, DOI: 10.7873/DATE2014.257.

[32] S. Gajek, H. Löhr, A.-R. Sadeghi, and M. Winandy, "TruWallet: Trustworthy and migratable wallet-based web authentication," in *Proc. ACM Workshop Scalable Trusted Comput.*, 2009, pp. 19–28.

[33] R. W. Gardner, S. Garera, and A. D. Rubin, "Detecting code alteration by creating a temporary memory bottleneck," *IEEE Trans. Inf. Forensics Security*, vol. 4, no. 4, pp. 638–650, Dec. 2009.

[34] T. Garfinkel, B. Pfaff, J. Chow, M. Rosenblum, and D. Boneh, "Terra: A virtual machine-based platform for trusted computing," in *Proc. 19th ACM Symp. Oper. Syst. Principles*, 2003, pp. 193–206.

[35] B. Gassend, D. Clarke, M. van Dijk, and S. Devadas, "Controlled physical random functions," in *Proc. IEEE Annu. Comput. Security Appl. Conf.*, 2002, pp. 149–160.

[36] B. Gassend, D. Clarke, M. van Dijk, and S. Devadas, "Silicon physical random functions," in *Proc. 9th ACM Conf. Comput. Commun. Security*, 2002, pp. 148–160.

[37] J. T. Giffin, M. Christodorescu, and L. Kruger, "Strengthening software self-checksumming via self-modifying code," in *Proc. IEEE 21st Annu. Comput. Security Appl. Conf.*, 2005, DOI: 10.1109/CSAC.2005.53.

[38] Global Platform, "TEE internal API specification," 2011. [Online]. Available: http://www.globalplatform.org/specificationsdevice.asp

[39] Global Platform, "TEE system architecture," 2011. [Online]. Available: http://www.globalplatform.org/specificationsdevice.asp

[40] K. Goldman, R. Perez, and R. Sailer, "Linking remote attestation to secure tunnel endpoints," in *Proc. 1st ACM Workshop Scalable Trusted Comput.*, 2006, pp. 21–24.

[41] J. Guajardo, S. Kumar, G.-J. Schrijen, and P. Tuyls, "FPGA intrinsic PUFs and their use for IP protection," in *Cryptographic Hardware and Embedded Systems (CHES)*. New York, NY, USA: Springer-Verlag, 2007.

[42] M. Hoekstra, R. Lal, P. Pappachan, V. Phegade, and J. Del Cuvillo, "Using innovative instructions to create trustworthy software solutions," *Proc. 2nd ACM Int. Workshop Hardware Architect. Support Security Privacy*, 2013, DOI: 10.1145/2487726.2488370.

[43] Intrinsic ID, "Products," 2013. [Online]. Available: http://www.intrinsic-id.com/products.htm

[44] T. Jaeger, R. Sailer, and U. Shankar, "PRIMA: Policy-reduced integrity measurement architecture," in *Proc. 11th ACM Symp. Access Control Models Technol.*, 2006, pp. 19–28.

[45] M. Jakobsson and K.-A. Johansson, "Retroactive detection of malware with applications to mobile platforms," in *Proc. USENIX Conf. Hot Topics Security*, 2010, article 1–13.

[46] D. Karakoyunlu and B. Sunar, "Differential template attacks on PUF enabled cryptographic devices," in *Proc. IEEE Int. Workshop Inf. Forensics Security*, 2010, DOI: 10.1109/WIFS.2010.5711445.

[47] S. Katzenbeisser *et al.*, "PUFs: Myth, fact or busted? A security evaluation of physically unclonable functions (PUFs) cast in silicon," in *Cryptographic Hardware and Embedded Systems (CHES)*. New York, NY, USA: Springer-Verlag, 2012.

[48] B. Kauer, "OSLO: Improving the Security of Trusted Computing," in *Proc. 16th USENIX Security Symp.*, 2007, article 16.

[49] E. Keller, J. Szefer, J. Rexford, and R. B. Lee, "NoHype: Virtualized cloud infrastructure without the virtualization," in *Proc. 37th Annu. Int. Symp. Comput. Architect.*, 2010, pp. 350–361.

[50] M. S. Kirkpatrick, G. Ghinita, and E. Bertino, "Resilient authenticated execution of critical applications in untrusted environments," *IEEE Trans. Dependable Secure Comput.*, vol. 9, no. 4, pp. 597–609, Jul./Aug. 2012.

[51] P. Koeberl, S. Schulz, A.-R. Sadeghi, and V. Varadharajan, "Trustlite: A security architecture for tiny embedded devices," in *Proc. Eur. Conf. Comput. Syst.*, 2014, DOI: 10.1145/2592798.2592824.

[52] K. Kostiainen, "On-board credentials: An open credential platform for mobile devices," Ph.D. dissertation, Dept. Comput. Sci. Eng., Aalto Univ., Aalto, Finland, 2012. [Online]. Available: http://urn.fi/URN:ISBN:978-952-60-4598-6

[53] K. Kostiainen, J.-E. Ekberg, N. Asokan, and A. Rantala, "On-board credentials with open provisioning," in *Proc. 4th ACM Symp. Inf. Comput. Commun. Security*, 2009, pp. 104–115.

[54] K. Kostiainen, E. Reshetova, J.-E. Ekberg, and N. Asokan, "Old, new, borrowed, blue—A perspective on the evolution of mobile platform security architectures," in *Proc. 1st ACM Conf. Data Appl. Security Privacy*, 2011, pp. 13–24.

[55] X. Kovah *et al.*, "New results for timing-based attestation," in *Proc. IEEE Symp. Security Privacy*, 2012, pp. 239–253.

[56] S. S. Kumar, J. Guajardo, R. Maes, G.-J. Schrijen, and P. Tuyls, "Extended abstract: The butterfly PUF protecting IP on every FPGA," in *Proc. IEEE Int. Workshop Hardware-Oriented Security*, 2008, pp. 67–70.

[57] J. W. Lee *et al.*, "A technique to build a secret key in integrated circuits for identification and authentication applications," in *Proc. Symp. VLSI Circuits*, 2004, pp. 176–179.

[58] Y. Li, J. M. McCune, and A. Perrig, "VIPER: Verifying the Integrity of PERipherals' firmware," in *Proc. 18th ACM Conf. Comput. Commun. Security*, 2011, pp. 3–16.

[59] D. Lim *et al.*, "Extracting secret keys from integrated circuits," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 13, no. 10, pp. 1200–1205, Oct. 2005.

[60] R. Maes and I. Verbauwhede, "Physically unclonable functions: A study on the state of the art and future research directions," in *Towards Hardware-Intrinsic Security*. New York, NY, USA: Springer-Verlag, 2010.

[61] M. Majzoobi, F. Koushanfar, and M. Potkonjak, "Lightweight secure PUFs," in *Proc. IEEE/ACM Int. Conf. Comput.-Aided Design*, 2008, pp. 670–673.

[62] J. M. McCune, S. Berger, R. Caceres, T. Jaeger, and R. Sailer, "Bridging mandatory access control across machines," IBM Res. Div., Res. Rep. RC23778, 2005.

[63] J. M. McCune *et al.*, "TrustVisor: Efficient TCB reduction and attestation," in *Proc. IEEE Symp. Security Privacy*, 2010, pp. 143–158.

[64] J. M. McCune, B. J. Parno, A. Perrig, M. K. Reiter, and H. Isozaki, "Flicker: An execution infrastructure for TCB minimization," in *Proc. 3rd ACM Eur. Conf. Comput. Syst.*, 2008, pp. 315–328.

[65] F. McKeen *et al.*, "Innovative instructions and software model for isolated execution," in *Proc. 2nd Int. Workshop Hardware Architect. Support Security Privacy*, 2013, DOI: 10.1145/2487726.2488368.

[66] D. Merli, D. Schuster, F. Stumpf, and G. Sigl, "Side-channel analysis of PUFs and fuzzy extractors," in *Trust and Trustworthy Computing (TRUST)*. New York, NY, USA: Springer-Verlag, 2011.

[67] Car Connectivity Consortium. [Online]. Available: http://www.mirrorlink.com/

[68] Mobey Forum, "Business models for NFC payments," 2011. [Online]. Available: http://www.mobeyforum.org/business-models-for-nfc-payments-white-paper

[69] D. G. Murray, G. Milos, and S. Hand, "Improving Xen security through disaggregation," in *Proc. 4th ACM SIGPLAN/SIGOPS Int. Conf. Virtual Execution Environ.*, 2008, pp. 151–160.

[70] A. Nagarajan, V. Varadharajan, M. Hitchens, and E. Gallery, "Property based attestation and trusted computing: Analysis and challenges," in *Proc. 3rd Int. Conf. Netw. Syst. Security*, 2009, pp. 278–285.

[71] G. Neiger, A. Santoni, F. Leung, D. Rodgers, and R. Uhlig, "Intel virtualization technology: Hardware support for efficient processor virtualization," *Intel Technol. J.*, vol. 10, no. 3, pp. 167–177, 2006.

[72] J. Noorman *et al.*, "Sancus: Low-cost trustworthy extensible networked devices with a zero-software trusted computing base," in *Proc. 22nd USENIX Conf. Security*, 2013, pp. 479–494.

[73] Java Community Process, "JSR 177: Security and Trust Services API for J2ME," 2007. [Online]. Available: http://jcp.org/en/jsr/detail?id=177

[74] Y. Oren, A.-R. Sadeghi, and C. Wachsmann, "On the effectiveness of the remanence decay side-channel to clone memory-based PUFs," in *Proc. 15th Int. Conf. Cryptogr. Hardware Embedded Syst.*, 2013, pp. 107–125.

[75] B. Parno, J. M. McCune, and A. Perrig, "Bootstrapping trust in commodity computers," in *Proc. IEEE Symp. Security Privacy*, 2010, pp. 414–429.

[76] N. L. Petroni, Jr., T. Fraser, J. Molina, and W. A. Arbaugh, "Copilot—A coprocessor-based kernel runtime integrity monitor," in *Proc. 13th Conf. USENIX Security Symp.*, 2004, vol. 13.

[77] B. Pfitzmann, J. Riordan, C. Stüble, M. Waidner, and A. Weber, "The PERSEUS system architecture," IBM Res., Tech. Rep. RZ 3335 (#93381), 2001.

[78] D. C. Ranasinghe, D. W. Engels, and P. H. Cole, "Security and privacy: Modest proposals for low-cost RFID systems," 2004.

[79] RSA Laboratories, "PKCS#11 v2.20: Cryptographic token interface standard," 2004. [Online]. Available: ftp://ftp.rsasecurity.com/pub/pkcs/pkcs-11/v2-20/pkcs-11v2-20.pdf

[80] U. Rührmair *et al.*, "Modeling attacks on physical unclonable functions," in *Proc. 17th ACM Conf. Comput. Commun. Security*, 2010, pp. 237–249.

[81] U. Ruhrmair and M. van Dijk, "PUFs in security protocols: Attack models and security evaluations," in *Proc. IEEE Symp. Security Privacy*, 2013, pp. 286–300.

[82] A.-R. Sadeghi and D. Naccache, Eds., *Towards Hardware-Intrinsic Security*. New York, NY, USA: Springer-Verlag, 2010.

[83] A.-R. Sadeghi and C. Stüble, "Property-based attestation for computing platforms: Caring about properties, not mechanisms," in *Proc. New Security Paradigms Workshop*, 2004, pp. 67–77.

[84] A.-R. Sadeghi, I. Visconti, and C. Wachsmann, "Enhancing RFID security and privacy by physically unclonable functions," in *Towards Hardware-Intrinsic Security*. New York, NY, USA: Springer-Verlag, 2010.

[85] R. Sailer *et al.*, "Building a MAC-based security architecture for the Xen open-source hypervisor," in *Proc. 21st Annu. Comput. Security Appl. Conf.*, 2005, DOI: 10.1109/CSAC.2005.13.

[86] R. Sailer, X. Zhang, T. Jaeger, and L. van Doorn, "Design and implementation of a TCG-based integrity measurement architecture," in *Proc. 13th Conf. USENIX Security Symp.*, 2004, vol. 13.

[87] D. Schellekens, B. Wyseur, and B. Preneel, "Remote attestation on legacy operating systems with trusted platform modules," *Sci. Comput. Program.*, vol. 74, no. 1–2, pp. 13–22, 2008.

[88] S. Schulz and A.-R. Sadeghi, "Secure VPNs for trusted computing environments," in *Proc. 2nd Int. Conf. Trusted Comput.*, 2009, pp. 197–216.

[89] S. Schulz, A.-R. Sadeghi, and C. Wachsmann, "Lightweight remote attestation using physical functions," Cntr. Adv. Security Res. Darmstadt (CASED), Darmstadt, Germany, Tech. Rep., 2011.

[90] S. Schulz, A.-R. Sadeghi, and C. Wachsmann, "Short paper: Lightweight remote attestation using physical functions," in *Proc. 4th ACM Conf. Wireless Netw. Security*, 2011, pp. 109–114.

[91] M. Selhorst and C. Stüble, "TrustedGRUB Project," 2010. [Online]. Available: http://sf.net/projects/trustedgrub/

[92] A. Seshadri *et al.*, "Pioneer: Verifying integrity and guaranteeing execution of code on legacy platforms," in *Proc. 20th Symp. Oper. Syst. Principles*, 2005, DOI: 10.1145/1095810.1095812.

[93] A. Seshadri, A. Perrig, L. van Doorn, and P. K. Khosla, "SWATT: SoftWare-based ATTestation for embedded devices," in *Proc. IEEE Symp. Security Privacy*, 2004, pp. 272–282.

[94] M. Shaneck, K. Mahadevan, V. Kher, and Y. Kim, "Remote software-based attestation for wireless sensors," in *Security and Privacy in Ad-Hoc and Sensor Networks*. New York, NY, USA: Springer-Verlag, 2005.

[95] S. W. Smith, "Outbound authentication for programmable secure coprocessors," in *Proc. 7th Eur. Symp. Res. Comput. Security*, 2002, pp. 72–89.

[96] J. Srage and J. Azema, "M-shield mobile security technology, TI white paper," 2005. [Online]. Available: http://focus.ti.com/pdfs/wtbu/ti_mshield_whitepaper.pdf

[97] R. Strackx and F. Piessens, "Fides: Selectively hardening software application components against kernel-level or process-level malware," in *Proc. ACM Conf. Comput. Commun. Security*, 2012, pp. 2–13.

[98] R. Strackx, F. Piessens, and B. Preneel, "Efficient isolation of trusted subsystems in embedded systems," in *Security and Privacy in Communication Networks (SecureComm)*. New York, NY, USA: Springer-Verlag, 2010.

[99] E. Suh, C. O'Donnell, I. Sachdev, and S. Devadas, "Design and implementation of the AEGIS single-chip secure processor using physical random function," in *Proc. 32nd Int. Symp. Comput. Architect.*, 2005, pp. 25–36.

[100] S. Tamrakar, "Tapping and tripping with NFC," *Trust and Trustworthy Computing (TRUST)*, vol. 7904. Berlin, Germany: Springer-Verlag, 2013, pp. 115–132, ser. Lecture Notes in Computer Science.

[101] Trusted Computing Group. [Online]. Available: https://www.trustedcomputinggroup.org/

[102] Trusted Platform Module (TPM) Specifications. [Online]. Available: https://www.trustedcomputinggroup.org/specs/TPM/

[103] V. van der Leest, G.-J. Schrijen, H. Handschuh, and P. Tuyls, "Hardware intrinsic security from D flip-flops," in *Proc. 5th ACM Workshop Scalable Trusted Comput.*, 2010, pp. 53–62.

[104] A. Vasudevan, J. Mccune, J. Newsome, A. Perrig, and L. V. Doorn, "CARMA: A hardware tamper-resistant isolated execution environment on commodity x86 platforms," in *Proc. 7th ACM Symp. Inf. Comput. Commun. Security*, 2012, pp. 48–49.

[105] Verayo Inc., "Products," 2013. [Online]. Available: http://www.verayo.com/product/products.html

[106] M. Wilkes and R. Needham, *The Cambridge CAP Computer and Its Operating System*. Amsterdam, The Netherlands: Elsevier, 1979.

[107] W. Wulf *et al.*, "HYDRA: The kernel of a multiprocessor operating system," *Commun. ACM*, vol. 17, no. 6, pp. 337–345, 1974.

[108] B. S. Yee, "Using secure coprocessors," Ph.D. dissertation, Schl. Comput. Sci., Carnegie Mellon Univ., Pittsburgh, PA, USA, 1994, CMU-CS-94-149.

[109] "Bundesbehörden sehen Risiken beim Einsatz von Windows 8," *Zeit Online*, 2013. [Online]. Available: http://www.zeit.de/digital/datenschutz/2013-08/trusted-computing-microsoft-windows-8-nsa

[110] Z. Zhou, V. D. Gligor, J. Newsome, and J. M. McCune, "Building verifiable trusted path on commodity x86 computers," in *Proc. Symp. Security Privacy*, 2012, pp. 616–630.

ABOUT THE AUTHORS

N. Asokan received the Ph.D. degree in computer science from the University of Waterloo, Waterloo, ON, Canada.

Currently, he is a Professor of Computer Science and Engineering at Aalto University, Aalto, Finland and a Professor of Computer Science at the University of Helsinki, Helsinki, Finland. He is the lead academic principal investigator of the Intel Collaborative Research Institute for Secure Computing in Finland. Prior to joining academia, he spent several years in industrial research, first with IBM Research and then with Nokia Research Center. He is interested in understanding how to build systems that are simultaneously secure, easy to use and inexpensive to deploy.

Jan-Erik Ekberg received the Ph.D. degree in computer science from Aalto University, Aalto, Finland, and holds a Licentiate degree in electrical engineering from the same university.

Currently, he is the Director of Advanced Development at Trustonic, Aalto, Finland. His background is in research and the telecom industry, where he worked for 18 years in various research positions. His primary interests are with issues related to mobile platform security and trusted execution environments, but he has also background in securing network protocols and telecom systems, as well as with short-range communication technologies such as NFC, BT-LE, and WLAN. He has participated in several standardization activities (TCG/MPWG, BTSIG/ BTLE), and holds 30+ patents.

Kari Kostiainen received the Ph.D. degree in computer science from Aalto University, Aalto, Finland. In his dissertation he studied mobile platform security mechanisms and took part in developing onboard credentials platform. His dissertation results have been included in both deployed products and industry standards.

Currently, he is a Postdoctoral Researcher at System Security Group of ETH Zurich, Zurich, Switzerland. His research focus is on security and privacy issues of mobile devices. Before joining ETH, he spent several years at Nokia Research Center, Helsinki, Finland and also briefly in Palo Alto, CA, USA. His current research topics range from secure smartphone applications to hardware security mechanisms.

Anand Rajan is the Director of the Emerging Security Lab at Intel Labs, Hillsboro, OR, USA. He leads a team of senior technologists whose mission is to research novel security features that raise the assurance of platforms across the compute continuum (cloud to wearables). The topics covered by his team span trustworthy execution environments, mobile security, identity & authentication, cryptography, and security for emerging paradigms (IOT, wearables). He has been a Principal Investigator for Intel's research collaboration with academia, government, and commercial labs in the area of trustworthy platforms. He and his team developed the Common Data Security Architecture specification, adopted as worldwide standard by The Open Group. His team was also instrumental in several security standardization efforts (e.g., PKCS#11, BioAPI, UPnP-Security, & EPID). Prior to joining Intel in 1994, he was the Technical Lead for the Trusted-UNIX team at Sequent Computer Systems and worked on development and certification of a TCSEC B1-level Operating System.

Dr. Rajan was an active member of the IEEE WG that crafted the P1363 (public-key crypto) standard.

Carlos Rozas received the B.S. and M.S. degrees in computer engineering from the University of Michigan, Ann Arbor, MI, USA.

Currently, he is a Principal Engineer in Intel Labs, Hillsboro, OR, USA, where he leads research efforts in cloud security and is Intel Software Guard Extensions architect. In the past, he has led research in combining trustworthy computing and virtualization. This included the development of TPM virtualization technology and the development of runtime software verification technologies. He has worked on software tamper resistance and verification technologies used by the Common Data Security Architecture. He has published seven technical papers and holds 23 patents.

Mr. Rozas received an Intel Achievement Award for his work on Intel Software Guard Extensions.

Ahmad-Reza Sadeghi received the Ph.D. degree in computer science with the focus on privacy protecting cryptographic protocols and systems from the University of Saarland, Saarbrücken, Germany.

Currently, he is the Head of the System Security Lab at the Center for Advance Security Research Darmstadt (CASED), Technische Universität Darmstadt, Darmstadt, Germany and the Scientific Director of Fraunhofer Institute for Secure Information Systems (SIT). Since January 2012, he has been the Director of Intel-TU Darmstadt Security Institute for Mobile and Embedded Systems in Darmstadt, Germany. Prior to academia, he worked in Research and Development of Telecommunications enterprises, among others Ericson Telecommunications. He has led and been involved in a variety of national and international research and development projects on design and implementation of trustworthy computing platforms and trusted computing, security hardware, physically unclonable functions (PUFs), cryptographic privacy-protecting systems, and cryptographic compilers (in particular for secure computation).

Prof. Sadeghi has been continuously contributing to the IT security research community and serving as General Chair or Program Chair as well as Program Committee Member of many conferences and workshops in information security and privacy, trusted computing, and applied cryptography. He is on the Editorial Board of the *ACM Transactions on Information and System Security*.

Steffen Schulz received the Ph.D. degree, researching trusted channels and trust anchors in low-cost embedded systems at Macquarie University, Sydney, N.S.W., Australia and Ruhr-University Bochum, Bochum, Germany.

Currently, he is a Security Researcher at Intel Labs and is currently situated at the Intel Collaborative Research Institute for Secure Computing (ICRI-SC), Darmstadt, Germany. He was involved in multiple national and international research projects and coauthored several publications in the area of network security, lightweight/scalable attestation, and trusted execution.

Christian Wachsmann received the Ph.D. degree in computer science from the Technische Universitat Darmstadt, Darmstadt, Germany.

Currently, he is a Postdoctoral Researcher at the Intel Collaborative Research Institute for Secure Computing (IC-RISC), Technische Universitðt Darmstadt. His current research focuses on the design, development, formal modeling, and security analysis of security architectures and cryptographic protocols to verify the software integrity (attestation) of embedded systems. He is the main coauthor of more than 30 scientific articles in internationally renowned journals and conferences on IT security.