Trusted Execution Environment: What It is, and What It is Not

Mohamed Sabt, Mohammed Achemlal, Abdelmadjid Bouabdallah

Trusted Execution Environment: What It Is, and What It Is Not

Mohamed Sabt*‡, Mohammed Achemlal*† and Abdelmadjid Bouabdallah‡
*Orange Labs, 42 rue des coutures, 14066 Caen, France
{mohamed.sabt, mohammed.achemlal}@orange.com
†Greyc ENSICAEN, 6 Bd Maréchal Juin, 14050 Caen, France
‡Sorbonne universités, Université de technologie de Compiègne,
Heudiasyc, Centre de recherche Royallieu, 60203 Compiègne, France
{mohamed.sabt, madjid.bouabdallah}@hds.utc.fr

***Abstract*—Nowadays, there is a trend to design complex, yet secure systems. In this context, the Trusted Execution Environment (TEE) was designed to enrich the previously defined trusted platforms. TEE is commonly known as an isolated processing environment in which applications can be securely executed irrespective of the rest of the system. However, TEE still lacks a precise definition as well as representative building blocks that systematize its design. Existing definitions of TEE are largely inconsistent and unspecific, which leads to confusion in the use of the term and its differentiation from related concepts, such as secure execution environment (SEE). In this paper, we propose a precise definition of TEE and analyze its core properties. Furthermore, we discuss important concepts related to TEE, such as trust and formal verification. We give a short survey on the existing academic and industrial ARM TrustZone-based TEE, and compare them using our proposed definition. Finally, we discuss some known attacks on deployed TEE as well as its wide use to guarantee security in diverse applications.**

I. Introduction

User requirements for security are increasingly becoming demanding. New challenges arise, since modern systems are becoming more and more complex, open and connected. Traditional security technologies can no longer meet the security requirements of such architectures. This explains the recent trend to integrate trusted computing concepts into different systems, such as embedded systems [1].

Trusted Computing was defined to help systems to achieve secure computation, privacy and data protection. Originally, trusted computing relies on a separate hardware module that offers a functional interface for platform security. The trusted platform module (TPM) [2] allows a system to provide evidence of its integrity and to protect cryptographic keys inside a tamper-evident hardware module. The main shortcoming of the TPM is that it does not provide an isolated execution environment for third-party, thereby reducing its functionality to a predefined set of APIs. A new approach to address trusted computing is to allow the execution of arbitrary code within a confined environment that provides tamper-resistant execution to its applications. In the literature, many names exist to this environment. Examples are closed-box VM [3], operator virtual machine (OVM) [4], TrustZone software (TZSW) [5], and trusted language runtime [6]. In this paper, we are going to refer to this environment using the term coined by GlobalPlatform in [7], that is *trusted execution environment* (TEE).

A TEE is a secure, integrity-protected processing environment, consisting of memory and storage capabilities [8]. For marketing purposes, the term TEE is heavily used in advertisements of chip vendors and platform providers [9], [10]. The first deployed system with TEE appeared almost a decade ago demonstrated by a joint venture of Orange, Trusted Logic and STMicroelectronics [11]. Nevertheless, no common and precise understanding for this term has been established so far, and no framework has been proposed to evaluate and compare TEE solutions. To underline this observation, we cite, in a chronological order, four definitions of TEE demonstrating the inconsistent use and understanding of the term:

1) Ben Pfaff, *Terra, 2003* [3] The TEE is a "dedicated closed virtual machine that is isolated from the rest of the platform. Through hardware memory protection and cryptographic protection of storage, its contents are protected from observation and tampering by unauthorized parties."
2) OMTP, *Advanced Trusted Environment, 2009* [12] "The TEE resists against a set of defined threats and satisfies a number of requirements related to isolation properties, lifecycle management, secure storage, cryptographic keys and protection of applications code."
3) GlobalPlatform, *TEE System Architecture, 2011* [7] "The TEE is an execution environment that runs alongside but isolated from the device main operating system. It protects its assets against general software attacks. It can be implemented using multiple technologies, and its level of security varies accordingly."
4) Jonathan M. McCune, *Trustworthy Execution on Mobile Devices, 2013* [13] "The set of features intended to enable trusted execution are the following: isolated execution, secure storage, remote attestation, secure provisioning and trusted path."

All definitions somehow mention isolated execution and secure storage. In these two points there appears to be some consent. However, definitions (1) and (3) are less explicit about secure storage. Definition (1) specifically states crypto-

graphic protection as the only means to achieve secure storage, whereas definition (3) tries to capture secure storage in a generic way as a 'protection of assets'. In addition, definition (1) describes isolation as the protection of the integrity and confidentiality of the TEE runtime states. Regarding the required security level, definitions largely differ from each others. Definitions (1) and (4) do not specify a threat model for the TEE. Definition (3) vaguely includes all software attacks in the threat model, while definition (2) clearly specifies the threats against which the TEE must resist. Definition (1) describes the TEE as a 'dedicated closed virtual machine', while the other definitions do not provide any detail about the nature of the execution environment. Some definitions are concerned by particular properties. For instance, definition (2) and McCune's definition (4) involve content management by indicating that TEE should remotely manage and update its data in a secure way (secure provisioning). Furthermore, the McCune's definition is specific to the context of human-interface devices, and therefore it includes the requirement of interaction between the TEE and end-users (trusted path).

We argue that existing definitions of TEE fail to capture the core aspects of this term in a clear and unambiguous manner and are even contradictory in some parts. To address this issue, in this paper, we propose a new refined definition of TEE considering its core aspects and the 'separation kernel' trusted model (Section II). Thereby, we differentiate TEE from some related concepts. In Section III, we present the building blocks that capture the TEE design, followed by Section IV, in which we review the principal formal methods used in the context of TEE. Section V gives a brief survey on the existing industrial and academic TEE, and compares them using our proposed definition. In Section VI, we discuss some known attacks on deployed TEE, and classify the research propositions defined in the literature that rely on TEE to guarantee security. We end with a brief summary.

II. Trusted Execution Environment

In this section, we first describe the model of 'separation kernel', which is a fundamental concept related to the dual-execution-environment approach [14], and thus to the TEE. We then present a new refined and comprehensive definition, as well as analyze its core aspects. Finally, we distinguish the TEE from some related terms: secure execution environment (SEE) and dynamic root of trust measurement (DRTM).

A. Prerequisite: Separation Kernel

The separation kernel is a foundation component of the TEE. It is the element that assures the property of isolated execution. The separation kernel, firstly introduced in [15], is a security kernel [16] used to simulate a distributed system. Its main design purpose is to enable the coexistence of different systems requiring different levels of security on the same platform. Basically, it divides the system into several partitions, and guarantees a strong isolation between them, except for the carefully controlled interface for inter-partition communication.

The security requirements for separation kernels are described in the Separation Kernel Protection Profile (SKPP) [17]. The SKPP defines separation kernel as "hardware and/or firmware and/or software mechanisms whose primary function is to establish, isolate and control information flow between [...] those partitions.". Unlike traditional security kernels, such as operating systems, micro-kernels and hypervisors, the separation kernel is quite simple, providing both time and space partitioning.

The security requirements are composed of four main security policies:

- *Data (spatial) separation*. Data within one partition cannot be read or modified by other partitions;
- *Sanitization (temporal separation)*. Shared resources cannot be used to leak information into other partitions;
- *Control of information flow*. Communication between partitions cannot occur unless explicitly permitted;
- *Fault isolation*. Security breach in one partition cannot spread to other partitions.

B. Definition

Trusted Execution Environment (TEE) is a tamper-resistant processing environment that runs on a separation kernel. It guarantees the authenticity of the executed code, the integrity of the runtime states (e.g. CPU registers, memory and sensitive I/O), and the confidentiality of its code, data and runtime states stored on a persistent memory. In addition, it shall be able to provide remote attestation that proves its trustworthiness for third-parties. The content of TEE is not static; it can be securely updated. The TEE resists against all software attacks as well as the physical attacks performed on the main memory of the system. Attacks performed by exploiting backdoor security flaws are not possible.

C. Discussion

We define TEE as an execution environment which protects both its runtime states and stored assets, hence the need for isolation and secure storage. Unlike dedicated hardware coprocessors, TEE is able to easily manage its content by installing or updating its code and data. In addition, it must define mechanisms to securely attest its trustworthiness to third-parties. The threat model includes all software attacks and the physical attacks performed on the main memory and its non-volatile memory by a powerful adversary.

We argue that our definition is more general and includes all the previously presented definitions of TEE. *Secure execution*, *openness* and *trust* are its main parts. Conceptually, our definition means that no untrusted code should be able to cause, enable, or prevent any event in the TEE. Events include not only the execution of instructions, but also traps, exceptions and interruptions. 'Trust' is discussed further in the next subsection.

D. How Trust Can Be Measured

As mentioned in the above definition, and as its name indicates, the concept of **trust** is crucial to the TEE. Thus,

a direct comparison between two systems in terms of TEE is only possible if trust can be quantified. The main problem is that trust is a subjective property, hence non-measurable. In English, trust is the "belief in honesty and goodness of a person or thing.". A belief is hard to capture in a quantified way. The notion of trust is more subtle in the field of computer systems. In the real world, an entity is trusted if it has behaved and/will behave as expected. In the computing world, trust follows the same assumption.

In computing, trust is either static or dynamic. A **static trust** is a trust based on a comprehensive evaluation against a specific set of security requirements. The Common Criteria (CC) [18] are an international standard that provides assurance measures for the security evaluation. The CC specify seven evaluation assurance levels (EAL1–EAL7), where levels with higher numbers include all requirements of the preceding levels. In static trust, the trustworthiness of a system is measured only once and before its deployment. **Dynamic trust** is quite different. It is based on the *state* of the running system, and thus it varies accordingly. A system continuously changes its "trust status". In dynamic trust, the trustworthiness of a system is constantly measured throughout its lifecycle.

The concept of dynamic trust is based on the existence of a secure and reliable means that provides evidence of the trust status of a given system. Trust, in this context, can be defined as an expectation that the system state is as it is considered to be: secure. This definition requires a trusted entity called **Root of Trust (RoT)** to provide trustworthy evidence regarding the state of a system. The role of RoT is divided into two parts. First is the **trusted measurement** and second is the function that computes the **trust score**. The trustworthiness of the system, namely the generated score, depends on the reliability of the trust measurement. If a malicious entity can influence the trust measurement, then the generated score of trustworthiness is of no value. Therefore, RoT is necessarily a tamper-resistant hardware module. RoT, sometimes called trust anchor, can be implemented using various technologies. This depends on the hardware platform that is used to guarantee the isolation properties in the separation kernel. For instance, TrustZone-based systems [19] rely on secureROM or eFuse technology as trust anchor. PUF, Physically Unclonable Function, is a promising RoT technology for TEE [20].

Trust in TEE is a **hybrid trust**; it is both static and semi-dynamic. Before deployment, a TEE must be certified by thoroughly verifying its security level in accordance of a *protection profile*, a document that contains a predefined set of security requirements. For instance, GlobalPlatform defines a protection profile that conforms to EAL2 [21]. In addition, during each boot, the RoT assures that the loaded TEE is the one certified by the platform provider. Strictly speaking, RoT protects the integrity of the TEE code. Once running, the integrity is protected by the underlying separation kernel. The trust in TEE is considered semi-dynamic because the TEE is not supposed to change its trust level while running because it is protected by the separation kernel. In this model of trust, the trust measurements are integrity measurements, and the trust score is a boolean that indicates the integrity state of the code. The TEE is trusted when its trust score is *true*, untrusted otherwise. The quality of the trust score depends on the defined measurements for integrity.

To evaluate the actual trust, as a first step, we define a **trust function** $f(\text{TEE},\ \text{protection profile},\ \text{RoT},\ \text{measurements})$ as a function that returns the trust level of a given TEE depending on three parameters: the certificating protection profile, the reliability of RoT, and the integrity measurements. The precise definition of this function is beyond the scope of this paper.

E. Related Concepts

In this section, we highlight the conceptual differences between TEE and the related terms SEE and DRTM.

Secure Execution Environment (SEE) is a prerequisite for TEE, but it does not consider trust aspects. SEE is a processing environment that guarantees the following properties: (1) *authenticity*: the code under execution should not have been changed; (2) *integrity*: runtime states should not have been tampered with; and (3) *confidentiality*: code, data and runtime states should not have been observable by unauthorized applications, or even by the main OS of the system. In contrast to TEE, the design of SEE does not involve RoT to assert the integrity and authenticity of the loaded code. Moreover, it does not define secure mechanisms to update its applications and confidential data. In fact, in our definition, TEE is an *open* SEE that guarantees *trust*.

Dynamic Root of Trust Measurement (DRTM) is a group of techniques that enables some pieces of code to be executed in an isolated environment, without trusting the previously loaded software. This technology has been used to securely execute critical software applications. Unlike TEE, the trusted computing base (TCB) of DRTM is not limited to the code running in the isolated environment, but it also includes a small part of the main OS. Moreover, DRTM does not provide secure storage or include trusted mechanisms, such as remote attestation and integrity measurement, in its core design. In contrast to the TEE definition stated above, the isolated execution environment needs to use the main memory as its runtime memory, and hence is vulnerable to attacks on main memory. TEE is supposed to resist against physical attacks on main memory by executing its code in a protected memory. It is worth noting that DRTM does not allow concurrency. All software is frozen until the end of the isolated environment execution. Therefore, DRTM is not suitable for systems which execute non-secure applications with real-time constraints.

III. TEE Building Blocks

To capture the core design aspects of TEE, we propose the following definitions which are illustrated in the figure 1.

- **Secure Boot** assures that only code of a certain property can be loaded. If a modification is detected, the bootstrap process is interrupted. An example implementation of secure boot, as proposed by Arbaugh et al., is to verify the integrity of a succeeding component according to a given reference value [22]. Generally speaking, the design of secure boot

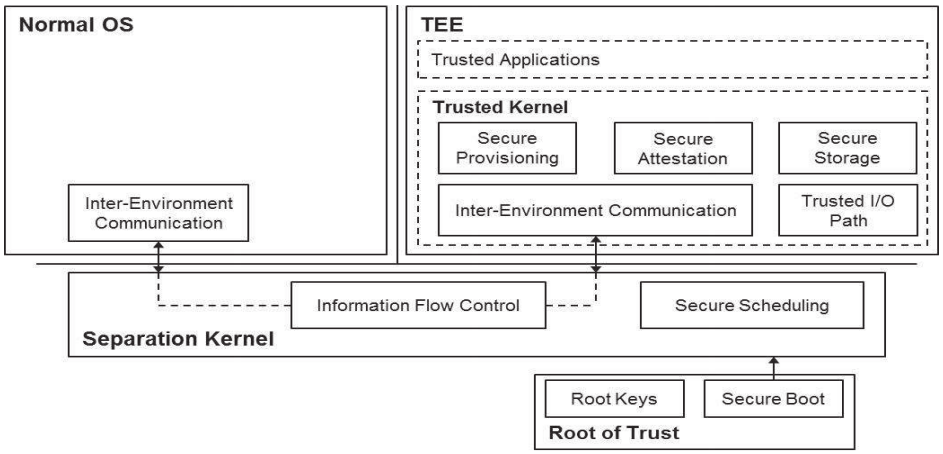

Fig. 1. An Overview of TEE Building Blocks

consists of various stages, and therefore, a chain of trust is established. This chain can be represented by the recurrence:

$$I_0 = \text{True};$$
$$I_{i+1} = I_i \wedge V_i(L_{i+1})$$

where I_i denotes the integrity of layer i and V_i is the corresponding verification function. The verification function performs cryptographic hash of the i^{th} layer, and compares the result to the reference value. We note that without the integrity of the initial boot code, represented by the I_0, any further integrity verification becomes pointless. Thus, the initial boot code is protected by a tamper-evident hardware module.

• **Secure Scheduling** assures a "balanced" and "efficient" coordination between the TEE and the rest of the system. Indeed, it should assure that the tasks running in the TEE do not affect the responsiveness of the main OS. Thus, the scheduler is often designed preemptive. Furthermore, the scheduler should take real-time constraints into consideration. Authors in [23] propose a secure scheduler that enhances the responsiveness of the main OS without compromising the real-time performance of the system.

• **Inter-Environment Communication** defines an interface allowing TEE to communicate with the rest of the system. Despite its numerous benefits, it introduces new threats: (1) message overload attacks [24]; (2) user and control data corruption attacks [25]; (3) memory faults caused by shared pages being removed; and (4) unbound waits caused by the non-cooperation of the untrusted part of system. There exist various methods and implementations of inter-environment communication. However, each mechanism should satisfy three key attributes: reliability (memory/time isolation), minimum overhead (unnecessary data copies and context switches), and protection of communication structures. In the literature, we identify three models of communication: (1) GlobalPlatform TEE Client API [26]; (2) secure RPC (Remote Procedure Call) of Trusted Language Runtime [6]; and (3) real-time RPC of SafeG [27]. Secure inter-environment communication is proposed in [28].

• **Secure Storage** is storage where confidentiality, integrity and freshness (i.e., to protect against replay attacks and to enforce state continuity [29]) of stored data are guaranteed, and where only authorized entities can access the data [30]. A common way to implement secure storage is sealed storage. Sealed storage is based on three components: (1) integrity-protected secret key that can be accessed only by the TEE; (2) cryptographic mechanisms, such as authenticated encryption algorithms; and (3) data rollback protection mechanism, such as replay-protected memory blocks (RPMB) (see [31]).

• **Trusted I/O Path** protects authenticity, and optionally confidentiality, of communication between TEE and peripherals (e.g., keyboard or sensors) [32]. Thus, input and output data are protected from being sniffed or tampered with by malicious applications. To be more precise, trusted I/O path protects against four classes of attacks: screen-capture attack, key logging attack, overlaying attack, and phishing attack. Trusted path to user-interface devices enables broader functionality within TEE. It allows a human user to directly interact with applications running inside TEE. Examples of trusted user-interface can be found in [33]–[35].

IV. Formal Methods

The design of TEE, or any piece of software, consists of two aspects: requirements specification and implementation. A TEE is said to be *correct* if its implementation is verified to satisfy all the defined requirements. Formal methods, which are mathematically based languages and techniques, are used to prove correctness. Although formal methods do not *necessarily* guarantee correctness, they provide insights which prove useful in constructing better systems.

There are two goals for formal methods: specification and verification. **Formal specifications** aim at describing the requirements of a system in a syntax-based language. They are a necessary condition to perform proof-based verification on implementation. A set of formal specifications, combined with a formal language produce a *formal model*. In literature, there are several formal models for separation kernel. The most

TEE	Author laboratory/company	License	TCB Size	Supported Normal World	Supported Hardware Platform
ObC	Nokia	Close	10kB	Symbian OS	300 MHz OMAP 2420
<t-base	Trustonic	Close	Unknown	Android	Samsung Exynos platforms
Andix OS	TU Graz University of Technology	Open-source	Unknown	Linux	iMX53 QSB
TLK	NVidia	Open-source	128kB	Android	Tegra SoCs
TLR	Microsoft	Close	152.7 KLOC	.NET CLR	Tegra 250 Dev Kit
SafeG	Nagoya University	Open-source	1.96 kB	TOPPERS/ASP	PB 1176 JZF-S board

TABLE I
AN OVERVIEW OF THE COMPARED TEES

widely used formal specifications for separation kernel are those proposed by Greve, Wilding and Vanfleet (GWV) [36]. Concerning formal languages, the mainly used are Z notation, B method [37], HOL4 (a variation of High Order Logic) [38], and ACL2 (A Computational Logic for Applicative Common Lisp) [39]. **Formal verification** is used to analyze the formal model for the desired properties. Two general approaches to formal verification exist in practice today. The first, *model checking*, is a technique in which systems are modeled as finite state systems. The second, *theorem proving*, proves that a system satisfies the specifications by deductive reasoning. Although proofs can be constructed by hand, machine-assisted theorem provers are used in most cases. Theorem proving is used more often than model checking because it can efficiently deal with complex properties.

We illustrate with two formally verified separation kernels. *INTEGRITY-178B* [40] is a separation kernel of Green Hills Software that is certified EAL6+. It uses GWV as formal specifications, ACL2 as formal language, theorem proving as formal verification method, and ACL2 theorem prover. *SeL4* [41] is developed by NICTA and was formally verified for security critical domain. It uses information flow security as formal specifications, HOL as formal language, theorem proving as formal verification method, and Isabelle/HOL theorem prover.

Formal methods play an important role in computing the 'trust level' defined by the trust function (II-D), since the protection profile could be defined using formal specifications and proved using formal verification. This could highly improve the trust level. However, formal methods are not a silver bullet. The trust function has other parameters and they could negatively impact the global trust level, even though formal methods are employed. For the best of our knowledge, there is no TEE that is formally verified. We believe that formal characterization of TEE specifications will be regarded as a considerable contribution. The most difficult part will be to include all the components and building blocks in a single model, despite their heterogeneity. Any formal model must at least comprise the underlying separation kernel, the root of trust and the secure execution environment.

V. ARM TRUSTZONE-BASED TEE

ARM TrustZone technology can be seen as a special kind of virtualization with hardware support for memory, I/O and interrupt virtualization [42]. This virtualization enables ARM core to provide an abstraction of two virtual cores (VCPUs): secure VCPU and non-secure VCPU. The monitor is seen as a minimal hypervisor whose main role is the control of information flow between the two virtual cores.

A short survey on the existing TrustZone-based TEE solutions in both the academic and industrial worlds is presented.

A. Industrial TEEs

Established companies have invested to define their own TEE and integrate them in their devices. Some companies have published their architecture, while some have preferred secrecy over openness. Companies which open their TEE include Nokia and Samsung. Nokia, currently Microsoft integrate their TEE called ObC [43] into Nokia Lumia devices. Samsung define TZ-RKP [44] that is deployed on the latest Samsung Galaxy series. Closed-architecture TEEs include <t-base of Trustonic [9], SecuriTEE of Solacia [10], and QSEE of Qualcomm. Sierraware [45] propose two versions of TEE: open-source TEE that is called SierraTEE and a licensed TEE. Trusted Foundation and Mobiore, defined by Trust Logic and G&D respectively, are disappearing from the market because the two companies joined their efforts and formed Trustonic.

We also consider as industrial TEEs those which are defined by companies, but there is no public information about their deployment on commercial devices. STMicroelectronics, in collaboration with Linaro make their TEE called OP-TEE available on GitHub [46]. Nvidia proposes an open-source implementation of TEE called TLK.

B. Academic TEEs

In the academic world, many prototypes of TEE exist. We only mention seven of them: (1) Genode TEE defined by Genode Labs [35]; (2) Open TEE defined by Intel Collaborative Research Institute for Secure Computing [47]; (3) Andix OS defined at TU Graz University of Technology; (4) ARMithril defined at North Carolina State University; (5) SafeG defined at Nagoya University [48]; (6) ViMoExpress defined by Electronics and Telecommunication Research Institute [49]; and (7) TLR defined by Microsoft Research [6].

C. Comparative Study

We compare six TEE solutions using our proposed building blocks. An overview of these TEEs are presented in table I. We decided to compare only these TEEs because they represent well the wide spectrum of the different solutions. We do not

TEE	Provisioning	Secure storage	Secure UI	Inter-world communication
ObC	Open provisioning, which means that the content management does not need the approval of any trusted-party.	Sealing storage using AES-EAX authenticated encryption. The root key is derived from a one-time programmable (e-Fuse) persistent on-chip key.	Defined	Proprietary interface
<t-base	Owner-centric provisioning model in which an application to be installed on TEE needs to be encrypted using a key derived from the platform secret key.	Sealing storage which is not based on file systems. Instead, the unit of storage is an object. Objects are organized into a tree-like structure. Containers are protected by the secret key of their parent.	Defined	GlobalPlatform TEE Client API
Andix OS	Not defined	Sealing storage	Not defined	GlobalPlatform TEE Client API
TLK	Not defined	Sealing storage	Not defined	Proprietary interface
TLR	Not defined	Sealing storage with mechanisms to protect against rollback attack.	Not defined	.NET Remoting
SafeG	Not defined	Unknown	Defined	Secure RPC

TABLE II
A COMPARISON STUDY OF THE SIX REPRESENTATIVE TEE SOLUTIONS

include secure boot in our comparison criteria, since Non-disclosure agreements (NDA) prevent authors from providing details about their secure boot.

As illustrated in table II, only the two TEE widely deployed, namely ObC and <t-base, provide secure provisioning. Trustonic attempts to control the content management of its TEE, while ObC opens their TEE for any party to install secure applications. Similarly, due to its high complexity, secure UI is not defined for all TEE. Concerning secure storage, it is often accomplished using sealing storage with AES encryption. Our comparative table shows that there is no clear standard for inter-environment communication. The use of proprietary interface is common.

VI. ATTACKS AND APPLICATIONS

We first discuss the attacks conducted on TEE deployed on mobile devices. We then provide a classification of the proposed applications that rely on TEE to guarantee security.

A. Attacks

TEE has been heavily promoted as the *silver bullet* solution that provides secure processing in mobiles and embedded systems. However, far from speculative bubbles and marketing claims, security experts have not put TEE to the test, especially because of non-disclosure agreement (NDA). The attack surface of TrustZone-based TEE is: software exceptions (e.g. SMC call), hardware exceptions (e.g. interrupts), shared memory interface, peripherals, and TEE-specific calls. The threat model includes a powerful attacker who is able to execute an arbitrary code in the kernel privileges.

To the best of our knowledge, three attacks have been published against QSEE or a manufacturer-customized version of QSEE. QSEE is an enticing target for attackers, since Qualcomm controls the majority of the market of Android devices. In addition, it is easier to exploit security flaws, as the memory layout of QSEE is known. In fact, the QSEE resides unencrypted on eMMC flash and loaded at known physical address. Disassemblers are used to gain insight into QSEE implementation. In [50], authors present an exploit that is caused by code added by HTC. The exploit enables the execution of an arbitrary code within TrustZone in the secure region of the memory. D. Rosenberg unlocks the bootloader of Motorola Android phones using two different exploits. The first exploit is about overwriting part of the secure region of the memory with certain values [51]. This is used to bypass the check of the function that unlocks the bootloader. The exploit works only on Qualcomm-based Motorola Android phones. The second exploit affects all Android phones that utilize Qualcomm Snapdragon SoC [52]. By issuing specially crafted SMC requests, an attacker can execute an arbitrary code inside the QSEE. This vulnerability may be used to compromise any applications relying on TEE for security.

Besides exploiting SMC calls, the shared memory can be manipulated to find vulnerabilities. In [53], a module intercepting exchanged data between the normal world and the secure world is implemented. This module is integrated inside the kernel driver which interacts with the TEE. The targeted TEE by this attack is MobiCore and its goal is to better understand the internal workings of MobiCore trustlets.

B. Applications

The use of TEE paves the way for offering services requiring a high level of security in a complex and connected system. Most of TEE applications defined in the literature are designed for smartphones. It is used to provide a wide range of secure services: ticketing [54], [55], privacy-friendly public transport ticketing [56], online transaction confirmation [57], privacy-friendly online prepaid mobile payment [58], [59], media content protection [60], [61], authentication to access cloud storage services [62], [63], two factor authentication [64], [65], and trusted sensors [66].

TEE is also used to implement TPM (Trusted Platform Module) on a software-only basis, without the need for additional special purpose hardware [67]–[69]. There is a trend

to use TEE to secure various embedded system platforms, such as sensors and Internet of Things [70].

TEE was used recently to provide self-protection to autonomic systems. Azab et.al perform real-time protection for kernels of mobile devices [44], while authors of [71] propose introspection mechanisms for operating systems using TrustZone-based trusted execution environment.

VII. Related Work

This paper could be considered as a survey on the domain of trusted execution environment. Our approach is distinguished from existing surveys. The closest work to ours are [8], [72]. In [8], authors discuss trust computing in mobile devices. They focus on existing technologies and fail to provide a theoretical framework for TEE. Arfaoui et al. [72] present TEE uniquely according to GlobalPlatform standards.

The particularity of our work is that we present a refined definition of TEE. Its core properties are clearly defined. Other important related topics are discussed, such as formal verification, known attacks, and a classification of the proposed applications in the literature using TEE to guarantee security.

VIII. Conclusion

TEE has practical interests and can be used to construct complex systems. Thus, we believe that deeper understanding of TEE is required in order to design better and more trustworthy systems.

In this paper, we proposed a refined definition of TEE to contribute in establishing a common understanding of this term in the context of trusted computing. Furthermore, we examined the building blocks of TEE explicitly differentiating it conceptually from the classical notions of SEE and DRTM. Moreover, we discussed how trust can be measured as well as formal verification for TEE. Finally, we discussed TrustZone-based TEE, by providing a short survey using our proposed definition, presenting the known attacks and classifying the proposed applications of TEE in the literature.

Many challenges exist. Our future work will focus on providing a formal definition for TEE, as well as explicitly defining the trust function in order to compute the trust level for the different TEE platforms.

References

[1] N. Anciaux, L. Bouganim, B. Nquyen, I. S. Popa, P. Pucheral, and P. Bonnet, *Trusted cells: a sea change for personal data services*, ser. ITU Technical Report Series. IT-Universitetet i København, 2012.

[2] "Trusted platform module (tpm) specifications." [Online]. Available: https://www.trustedcomputinggroup.org/specs/TPM

[3] T. Garfinkel, B. Pfaff, J. Chow, M. Rosenblum, and D. Boneh, "Terra: a virtual machine-based platform for trusted computing," *SIGOPS Oper. Syst. Rev.*, vol. 37, no. 5, pp. 193–206, Oct. 2003.

[4] "Secure payment via mobiles," *Card Technology Today*, vol. 17, no. 3, p. 5, 2005.

[5] P. Wilson, A. Frey, T. Mihm, D. Kershaw, and T. Alves, "Implementing embedded security on dual-virtual-cpu systems," *IEEE Des. Test*, vol. 24, no. 6, pp. 582–591, Nov. 2007.

[6] N. Santos, H. Raj, S. Saroiu, and A. Wolman, "Using arm trustzone to build a trusted language runtime for mobile applications," *SIGARCH Comput. Archit. News*, vol. 42, no. 1, pp. 67–80, Feb. 2014.

[7] GlobalPlatform, "TEE system architecture," 2011. [Online]. Available: http://www.globalplatform.org/specificationsdevice.asp

[8] N. Asokan, J. E. Ekberg, K. Kostiainen, A. Rajan, C. Rozas, A. R. Sadeghi, S. Schulz, and C. Wachsmann, "Mobile trusted computing," *Proceedings of the IEEE*, vol. 102, no. 8, pp. 1189–1206, Aug. 2014.

[9] "Trustonic," 2014. [Online]. Available: https://www.trustonic.com/

[10] Solacia, "SecuriTEE." [Online]. Available: http://www.sola-cia.com/en/securiTee/product.asp

[11] "STMicroelectronics, orange, and trusted logic to demonstrate secure mobile phone and payment application," 2005. [Online]. Available: http://phys.org/news3041.html

[12] OMTP limited, "Advanced trusted environment: omtp tr1 v1.1," 2009.

[13] A. Vasudevan, J. M. McCune, and J. Newsome, *Trustworthy execution on mobile devices*. Springer Publishing Company, Incorporated, 2013.

[14] M. Sabt, M. Achemlal, and A. Bouabdallah, "The dual-execution-environment approach: analysis and comparative evaluation," in *ICT Systems Security and Privacy Protection*, ser. IFIP Advances in Information and Communication Technology. Springer International Publishing, 2015, vol. 455, pp. 557–570.

[15] J. M. Rushby, "Design and verification of secure systems," *SIGOPS Oper. Syst. Rev.*, vol. 15, no. 5, pp. 12–21, Dec. 1981.

[16] J. Ames, Stanley R., M. Gasser, and R. R. Schell, "Security kernel design and implementation: an introduction," *Computer*, vol. 16, no. 7, pp. 14–22, Jul. 1983.

[17] "U.S. government protection profile for separation kernels in environments requiring high robustness," National Security Agency, Tech. Rep., Jun. 2007, version 1.03. [Online]. Available: https://www.niap-ccevs.org/pp/pp_skpp_hr_v1.03.pdf

[18] "The common criteria," 2014. [Online]. Available: https://www.commoncriteriaportal.org/

[19] P. Wilson, A. Frey, T. Mihm, D. Kershaw, and T. Alves, "Implementing embedded security on dual-virtual-cpu systems," *IEEE Des. Test*, vol. 24, no. 6, pp. 582–591, Nov. 2007.

[20] S. Zhao, Q. Zhang, G. Hu, Y. Qin, and D. Feng, "Providing toot of trust for arm trustzone using sram pufs," *IACR Cryptology ePrint Archive*, vol. 2014, p. 464, 2014.

[21] GlobalPlatform Device Committee, "Tee protection profile," 2014, version 1.2. [Online]. Available: http://www.globalplatform.org/specificationsdevice.asp

[22] W. A. Arbaugh, D. J. Farber, and J. M. Smith, "A secure and reliable bootstrap architecture," in *Proceedings of the 1997 IEEE Symposium on Security and Privacy*, ser. SP '97. Washington, DC, USA: IEEE Computer Society, 1997, pp. 65–71.

[23] D. Sangorrín, S. Honda, and H. Takada, "Integrated scheduling for a reliable dual-os monitor," *IPSJ Transactions on Advanced Computing Systems (ACS)*, vol. 5, no. 2, pp. 99–110, mar 2012.

[24] J. Regehr and U. Duongsaa, "Preventing interrupt overload," *SIGPLAN Not.*, vol. 40, no. 7, pp. 50–58, Jun. 2005.

[25] S. Chen, J. Xu, E. C. Sezer, P. Gauriar, and R. K. Iyer, "Non-control-data attacks are realistic threats," in *Proceedings of the 14th Conference on USENIX Security Symposium - Volume 14*, ser. SSYM'05. Berkeley, CA, USA: USENIX Association, 2005, pp. 12–12.

[26] GlobalPlatform Device Technology, "Tee client api specification," 2010, version 1.0. [Online]. Available: http://www.globalplatform.org/specificationsdevice.asp

[27] D. Sangorrín, S. Honda, and H. Takada, "Reliable and efficient dual-os communications for real-time embedded virtualization," *Information and Media Technologies*, vol. 8, no. 1, pp. 1–17, 2013.

[28] J. Jang, S. Kong, M. Kim, D. Kim, and B. B. Kang, "Secret: secure channel between rich execution environment and trusted execution environment," in *Proceedings of 2015 Annual Network and Distributed System Security Symposium (NDSS'15)*, February 2015.

[29] B. Parno, J. R. Lorch, J. R. Douceur, J. Mickens, and J. M. McCune, "memoir: practical state continuity for protected modules," in *Proceedings of the 2011 IEEE Symposium on Security and Privacy*, ser. SP '11. Washington, DC, USA: IEEE Computer Society, 2011, pp. 379–394.

[30] H. Löhr, A. R. Sadeghi, and M. Winandy, "Patterns for secure boot and secure storage in computer systems," in *Proceedings of the 2010 International Conference on Availability, Reliability, and Security*, ser. ARES '10. IEEE Computer Society, Feb. 2010, pp. 569–573.

[31] V. Tsai, "emmc v4.41 and v4.5. architecture for high speed functions and features," 2010. [Online]. Available: http://www.jedec.org/sites/default/files/Victor_Tsai.pdf

[32] "Trusted computer system evaluation criteria," U.S. Department of Defense, Tech. Rep. DoD 5200.28-STD, Dec. 1985.

[33] W. Li, M. Ma, J. Han, Y. Xia, B. Zang, C.-K. Chu, and T. Li, "Building trusted path on untrusted device drivers for mobile devices," in *Proceedings of 5th Asia-Pacific Workshop on Systems*, ser. APSys '14. New York, NY, USA: ACM, 2014, pp. 8:1–8:7.

[34] D. Liu and L. P. Cox, "Veriui: attested login for mobile devices," in *Proceedings of the 15th Workshop on Mobile Computing Systems and Applications*, ser. HotMobile '14. New York, NY, USA: ACM, 2014, pp. 7:1–7:6.

[35] GENODE Operating System Framework, "An exploration of arm trustZone technology," 2014. [Online]. Available: http://genode.org/documentation/articles/trustzone

[36] D. Greve, M. Wilding, and M. Vanfleet, "A separation kernel formal security policy," in *Fourth International Workshop on the ACL2 Prover and Its Applications (ACL2-2003)*, 2003.

[37] K. Kawamorita, R. Kasahara, Y. Mochizuki, and K. Noguchi, "Application of formal methods for designing a separation kernel for embedded systems," vol. 4, no. 8, pp. 1076–1084, 2010.

[38] "HOL4," 2014. [Online]. Available: http://hol.sourceforge.net/

[39] "ACL2," 2015. [Online]. Available: http://www.cs.utexas.edu/users/moore/acl2/

[40] R. J. Richards, "Modeling and security analysis of a commercial real-time operating system kernel," in *Design and Verification of Microprocessor Systems for High-Assurance Applications*, D. S. Hardin, Ed. Springer US, 2010, pp. 301–322.

[41] T. Murray, D. Matichuk, M. Brassil, P. Gammie, T. Bourke, S. Seefried, C. Lewis, X. Gao, and G. Klein, "Sel4: from general purpose to a proof of information flow enforcement," in *Proceedings of the 2013 IEEE Symposium on Security and Privacy*, ser. SP '13. Washington, DC, USA: IEEE Computer Society, 2013, pp. 415–429.

[42] ARMLtd, "Arm security technology - building a secure system using trustzone technology," 2009.

[43] K. Kostiainen, J.-E. Ekberg, N. Asokan, and A. Rantala, "On-board credentials with open provisioning," in *Proceedings of the 4th International Symposium on Information, Computer, and Communications Security*, ser. ASIACCS '09. New York, NY, USA: ACM, 2009, pp. 104–115.

[44] A. M. Azab, P. Ning, J. Shah, Q. Chen, R. Bhutkar, G. Ganesh, J. Ma, and W. Shen, "Hypervision across worlds: real-time kernel protection from the arm trustzone secure world," in *Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security*, ser. CCS '14. New York, NY, USA: ACM, 2014, pp. 90–102.

[45] SierraWare, "Sierratee for arm trustzone." [Online]. Available: http://www.sierraware.com/open-source-ARM-TrustZone.html

[46] P. Brand, "Op-tee." [Online]. Available: https://github.com/OP-TEE

[47] "Open-tee." [Online]. Available: https://github.com/Open-TEE

[48] D. Sangorín, S. Honda, and H. Takada, "Dual operating system architecture for real-time embedded systems," in *Proceedings of the 6th International Workshop on Operating Systems Platforms for Embedded Real-Time Applications (OSPERT)*, 2010, pp. 6–15.

[49] S.-C. Oh, K. Koh, C.-Y. Kim, K. Kim, and S. Kim, "Acceleration of dual os virtualization in embedded systems," in *2012 7th International Conference on Computing and Convergence Technology (ICCCT)*, Dec 2012, pp. 1098–1101.

[50] N. Keltner and C. Holmes, "Here be dragons: vulnerabilities in trustzone," 2014. [Online]. Available: http://atredispartners.blogspot.jp/2014/08/here-be-dragons-vulnerabilities-in.html

[51] D. Rosenberg, "Unlocking the motorola bootloader," 2013. [Online]. Available: http://blog.azimuthsecurity.com/2013/04/unlocking-motorola-bootloader.html

[52] ——, "Reflections on trusting trustzone," Presented at Black Hat 2014.

[53] S. Blog, "A software level analysis of trustzone os and trustlets in samsung galaxy phone," 2013. [Online]. Available: http://www.sensepost.com/blog/9114.html

[54] W. H. W. Hussin, P. Coulton, and R. Edwards, "Mobile ticketing system employing trustzone technology," in *Proceedings of the International Conference on Mobile Business*, ser. ICMB '05. Washington, DC, USA: IEEE Computer Society, 2005, pp. 651–654.

[55] W. H. Wan Hussin, R. Edwards, and P. Coulton, "E-pass using drm in symbian V8 os and trustzone: securing vital data on mobile devices," in *Proceedings of the International Conference on Mobile Business*, ser. ICMB '06. Washington, DC, USA: IEEE Computer Society, 2006.

[56] S. Tamrakar, J.-E. Ekberg, and N. Asokan, "Identity verification schemes for public transport ticketing with nfc phones," in *Proceedings of the Sixth ACM Workshop on Scalable Trusted Computing*, ser. STC '11. New York, NY, USA: ACM, 2011, pp. 37–48.

[57] L. Li, D. Huang, Z. Shen, and S. Bouzefrane, "A cloud based dual-root trust model for secure mobile online transactions," in *WCNC'13*, 2013, pp. 4404–4409.

[58] M. Pirker and D. Slamanig, "A framework for privacy-preserving mobile payment on security enhanced arm trustzone platforms," in *Proceedings of the 2012 IEEE 11th International Conference on Trust, Security and Privacy in Computing and Communications*, ser. TRUSTCOM '12. Washington, DC, USA: IEEE Computer Society, 2012, pp. 1155–1160.

[59] M. Pirker, D. Slamanig, and J. Winter, "practical privacy preserving cloud resource-payment for constrained clients," in *Proceedings of the 12th International Conference on Privacy Enhancing Technologies*, ser. PETS'12. Berlin, Heidelberg: Springer-Verlag, 2012, pp. 201–220.

[60] R. Tögl, J. Winter, and M. Pirker, "A path towards ubiquitous protection of media," in *Proceedings Workshop on Web Applications and Secure Hardware*, ser. CEUR Workshop Proceedings, vol. 1011. Technical University of Aachen, 2013, pp. 32–38.

[61] Z. Ahmad, L. Francis, T. Ahmed, C. Lobodzinski, D. Audsin, and P. Jiang, "enhancing the security of mobile applications by using tee and (u)sim," in *Proceedings of the 2013 IEEE 10th International Conference on Ubiquitous Intelligence & Computing and 2013 IEEE 10th International Conference on Autonomic & Trusted Computing*, ser. UIC-ATC '13. Washington, DC, USA: IEEE Computer Society, 2013, pp. 575–582.

[62] Y. Kim, J. Shin, C. Park, and W. Park, "Dfcloud: a tpm-based secure data access control method of cloud storage in mobile devices," in *Proceedings of the 2012 IEEE 4th International Conference on Cloud Computing Technology and Science (CloudCom)*, ser. CLOUDCOM '12. Washington, DC, USA: IEEE Computer Society, 2012, pp. 551–556.

[63] J.-E. Ekberg, A. Afanasyeva, and N. Asokan, "Authenticated encryption primitives for size-constrained trusted computing," in *Proceedings of the 5th International Conference on Trust and Trustworthy Computing*, ser. TRUST'12. Berlin, Heidelberg: Springer-Verlag, 2012, pp. 1–18.

[64] R. van Rijswijk-Deij and E. Poll, "Using trusted execution environments in two-factor authentication: comparing approaches," in *Open Identity Summit 2013, OID 2013*. Springer, 2013, pp. 20–31.

[65] C. Marforio, N. Karapanos, C. Soriente, K. Kostiainen, and S. Čapkun, "smartphones as practical and secure location verification tokens for payments," in *The Network and Distributed System Security Symposium (NDSS)*, 2014.

[66] H. Liu, S. Saroiu, A. Wolman, and H. Raj, "Software abstractions for trusted sensors," in *Proceedings of the 10th International Conference on Mobile Systems, Applications, and Services*, ser. MobiSys '12. New York, NY, USA: ACM, 2012, pp. 365–378.

[67] K. Dietrich and J. Winter, "Implementation aspects of mobile and embedded trusted computing," in *Proceedings of the 2nd International Conference on Trusted Computing*, ser. Trust '09. Berlin, Heidelberg: Springer-Verlag, 2009, pp. 29–44.

[68] J. Winter, "Trusted computing building blocks for embedded linux-based arm trustzone platforms," in *Proceedings of the 3rd ACM Workshop on Scalable Trusted Computing*, ser. STC '08. New York, NY, USA: ACM, 2008.

[69] S. Thom, J. Cox, D. Linsley, M. Nystrom, H. Raj, D. Robinson, S. Saroiu, R. Spiger, and A. Wolman, "Firmware-based trusted platform module for arm processor architectures and trustzone security extensions," January 2013, uS Patent App. 13/193,945. [Online]. Available: http://www.google.com/patents/US20130031374

[70] J. González and P. Bonnet, "Towards an open framework leveraging a trusted execution environment," in *Cyberspace Safety and Security*, ser. Lecture Notes in Computer Science. Springer International Publishing, 2013, vol. 8300, pp. 458–467.

[71] X. Ge, H. Vijayakumar, and T. Jaeger, "Sprobes: enforcing kernel code integrity on the trustzone architecture," in *Proceedings of the 3rd IEEE Mobile Security Technologies Workshop (MoST)*, May 2014.

[72] G. Arfaoui, S. Gharout, and J. Traoré, "Trusted execution environments: A look under the hood," in *Mobile Cloud Computing, Services, and Engineering (MobileCloud), 2014 2nd IEEE International Conference on*, April 2014, pp. 259–266.