Secure and Trusted Execution: Past, Present and Future – A Critical Review in the Context of the Internet of Things and Cyber-Physical Systems

Carlton Shepherd[†], Ghada Arfaoui[‡], Iakovos Gurulian[†], Robert P. Lee[†], Konstantinos Markantonakis[†], Raja Naeem Akram[†], Damien Sauveron[§¶], and Emmanuel Conchon[§]
[†]*ISG-SCC, Royal Holloway, University of London, Egham, United Kingdom*
[§]*XLIM (UMR CNRS 7252 / Université de Limoges), Département Mathématiques Informatique. Limoges, France*
[¶]*LaBRI (UMR CNRS 5800 / Université de Bordeaux), Talence, France*
[‡]*Orange Labs, Châtillon, France.*
Email: {carlton.shepherd.2014, iakovos.gurulian.2014, robert.lee.2013}@live.rhul.ac.uk, ghada.arfaoui@orange.com, {k.markantonakis, r.n.akram}@rhul.ac.uk, {damien.sauveron, emmanuel.conchon}@unilim.fr

***Abstract*—Trust has various instantiations: some rely on real-world relationships between entities, while others depend on robust hardware and software technologies to establish it post-deployment. In this paper, we focus on the latter, analyse their evolution in previous years, and their scope in the near future. The evolution of such technologies has involved diverse approaches; consequently, trust is understood and ascertained differently across heterogeneous systems and domains. We look at trusted hardware and software technologies from a security perspective – revisiting and analysing the Trusted Platform Module (TPM); Secure Elements (SE); hypervisors and virtualisation, including Java Card and Intel's Trusted eXecution Technology (TXT); Trusted Execution Environments (TEEs), such as GlobalPlatform TEE and Intel SGX; Host Card Emulation (HCE); and the Encrypted Execution Environment (E3). In our analysis, we focus on these technologies and their application to the emerging domains of the Internet of Things (IoT) and Cyber-Physical Systems (CPS).**

***Index Terms*—Trust, Trustworthiness, Trusted Computing, Trusted Platform Module, Trusted Execution Environment, GlobalPlatform, Java Card, Intel SGX, Host Card Emulation, Encrypted Execution Environment, Internet of Things, Cyber-Physical Systems**

1. Introduction

The Trusted Computing Group (TCG) defines trust as the *"expectation that a device will behave in a particular manner for a specific purpose"* [50]. In highly constrained environments, where all devices are vetted off-line beforehand and can be continually vetted afterwards by a human actor, establishing trust in devices can be straightforward. However, off-line vetting can quickly become infeasible when the number or geographical dispersion of devices is great enough. As such, a wide range of robust technologies have been developed to provide assurances that individual devices are secure and running in a trusted state. Such technologies can be pivotal in designing, implementing and deploying secure computing that has a secure, trusted and standardised environment. The applications running in such an environment have assurances that no malicious entity may interfere with its operation within the protection scope of the chosen trust technology.

Trust technologies have predominantly evolved across diverse computing environments, like smart cards, e.g. Java Card; virtualised platforms, e.g. Intel TXT; and mobile devices, e.g. ARM TrustZone. Each one typically offers certain services and capabilities to realise secure computing in that particular setting. Commonly, however, trust technologies help to establish trust in the host device's operating state. With the increased reliance on remote on-demand services, establishing trust has greater importance when connecting a range of devices outside the remit of a single organisation or individual. The problem of creating trust across heterogeneous devices and stakeholders can only increase with the underlying drivers of cloud computing, the Internet of Things (IoT) and Cyber-Physical Systems (CPS).

Secure and trusted computing environments are critical for emerging paradigms like IoT and CPS, where devices can interact directly with the physical world to potentially assist individuals in their daily activities – requiring increased levels of assurance that devices are performing within expected parameters to prevent harmful behaviour. Indeed, the trustworthiness of devices may be the differentiating factor in realising the projected potential of IoT and CPS. In Section 2, we briefly discuss an adversarial model and criteria for evaluation of secure and trusted execution technologies in the context of IoT and CPS. We discuss the evolution of secure and trusted execution proposals over the years in Sections 3 to 8, and evaluate them in Section 9 with respect to the criteria defined in Section 2.2.

2. Threat Model and Evaluation Criteria

In this section, we define the potential threat model and evaluation criteria for such technologies, in the context of their application in the IoT and CPS.

2.1. Threat Model

We discuss two generic types of adversary [41] based on the level of access to a particular device. Note that, in IoT and CPS environments, an adversary can reasonably gain physical access to a device and its hardware, in addition to potentially exploiting software- and network-based vectors. The trust technologies are evaluated for their coverage against these adversaries.

2.1.1. On-Device Adversary. On-device adversaries are malicious users that can potentially compromise the device. By doing so, depending upon the depth of compromise, they can control the execution of any sensitive applications and kernel space services. We divide the capabilities of this adversary further:

1) *Malicious Application Adversary:* An adversary that develops a malicious application and has it installed on the target device (Ring 3).
2) *Malicious Root Adversary:* An adversary that comprises the kernel space (Ring 0) of a device, thus providing root system access.
3) *Malicious Hardware Adversary:* An adversary with the ability to perform arbitrary hardware attacks, including hardware trojans and power analysis.

2.1.2. Off-Device Adversary. Off-device adversaries are malicious users that are located remotely and aim to manipulate the target's communication interfaces. The objective is to exfiltrate stored data or introduce malicious code to provide control the device to the remote adversary.

2.2. Criteria

In this section, we introduce the evaluation criteria of the trust technologies:

1) *User Control:* The user may freely (un-)install arbitrary applications that use the chosen technology.
2) *Centralised Control:* The trust technology is under the control of its issuer or maintainer – a centralised authority other than the end-user.
3) *Managed Access:* To use the features and services of the trust technology, prior authorisation is required from the device manufacturer. Managed access can be implemented as part of the both user and centralised control.
4) *User Managed Access:* To use the features and services of the trust technology, prior authorisation from the end-user of the device is required. Similar to managed access, this may also be implemented as part of the user and centralised control.
5) *Open Access:* No prior authorisation is required to use the features and services of the trust technology.
6) *Static Verification:* The technology has the ability, or can be extended, to provide static integrity verification of the device platform and applications running on it.
7) *Continuous Verification:* The technology has the ability, or can be extended, to provide continuous integrity verification of the device platform and applications running on it to monitor run-time instructions.
8) *Tamper-Resistant Hardware:* The trust technology is implemented in tamper-resistant hardware.
9) *Secure Storage (Internal):* The device has substantial internal storage for protecting sensitive data/applications.
10) *Secure Storage (External):* The device has substantial external storage for protecting sensitive data/applications. This also includes encrypted data/applications stored on external storage, where the device storage itself may be insecure.
11) *Isolated Execution:* The trust technology provides services to enable an application to execute in complete isolation, without interference from untrusted device applications.
12) *Software/Hardware Binding:* Individual applications and services installed on the device are securely bound to the underlying hardware. No new software can be introduced to such devices without knowing the correcting binding method and key.
13) *Remote Attestation:* The trust technology can provide evidence to assure remote devices that it is functioning to expectations, including its platform and applications.
14) *Protection Against Adversary:* The trust technology can protect the device against any modification by the adversaries described in the previous section.

3. Trusted Platform Modules (TPMs)

In this section, we examine how the TPM provides a secure boot to the host device along with trustworthy reporting and attestation operations. For a detailed discussion of individual components and their functionality, the reader is referred to [12] and [28].

Secure Boot (Measurement Operation). When a user boots up their computer, the first component to power up is the system BIOS (Basic Input/Output System). On a trusted platform, the boot sequence is initiated by the Core BIOS (i.e. CRTM: Core Root of Trust Measurement), which first measures its own integrity. This measurement is stored in PCR_0[1] and later it is extended to include the integrity measurement of the rest of the BIOS. The Core BIOS then measures the motherboard configuration setting, and this

1. A Platform Configuration Register (PCR) is a 160-bit (20 bytes) data element that stores the result of the integrity measurement, which is a generated hash of a given component (e.g. BIOS, operating system, or an application). Therefore, a group of PCRs form the integrity matrix. The process of extending PCR values is: $PCR_i = Hash(PCR'_i||X)$, where i is the PCR index, PCR'_i represents the old value stored at index i, and X is the sequence to be included in the PCR value. "||" indicates the concatenation of two data elements in the given order. The starting value of all PCRs is set to zero.

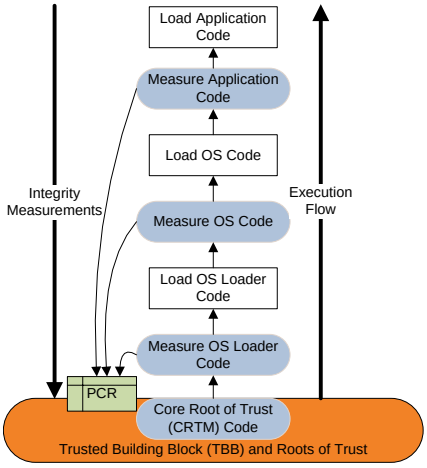


Figure 1: Trusted Platform Boot Sequence

value is stored in PCR_1. After these measurements, the Core BIOS will load the rest of the code of the BIOS.

The BIOS will subsequently measure the integrity of the ROM firmware and the ROM firmware configuration, storing them in PCR_2 and PCR_3 respectively. At this stage, the Trusted Building Block (TBB) is established and CRTM will proceed with integrity measurement and loading of the Operating System (OS).

The CRTM measures the integrity of the "OS Loader Code," also termed the Initial Program Loader (IPL), and stores the measurement in the PCR. The designated PCR index is left to the discretion of the OS. Subsequently, it will execute the "OS Loader Code" and on its successful execution, the TPM will measure the integrity of the "OS Code". After measurement is made and stored, the "OS Code" executes. Finally, the relevant software that initiates its execution will be subjected to an integrity measurement, and values will be stored in a PCR. Then the software will be sanctioned to execute. This process is shown in Figure 1, which illustrates the execution flow and integrity measurement storage.

By creating a chain of integrity measurements, a TPM provides a trusted and reliable view of the current state of the system. Any software, whether part of an OS or an application, has an integrity measurement stored in a PCR at a particular index. If the value satisfies the requirements of the software or requesting entity, then it can ascertain the trustworthiness of the system or otherwise take action. As discussed before, a TPM does not make any decisions: it only measures, stores, and reports integrity measurements in a secure and reliable manner. When a TPM reports an integrity measurement, it is recommended that it should generate a signature on the value, thus avoiding replay and man-in-the-middle attacks [49].

Reporting and Attestation Operations. The attestation process, whether initiated by the relevant user/administrator/third-party locally or remotely, involves the generation of a signature using the respective Attestation Identification Key (AIK) on the (associated/requested) PCR values [28]. The signature assures requesters of the validity of the integrity measurement stored in the PCRs. The choice of the AIK and PCR index is dependent on the respective user, platform (OS) or application.

4. Secure Elements (SEs)

A secure element (SE) is a tamper-resistant platform capable of securely hosting code and confidential data, such as cryptographic keys, according to the security processes established by its owner. Historically, in the mid-1970s, the first secure element was the smart card, which was based on a one-chip, secure microcontroller running a minimal secure operating system, which was, for a long time, restricted to only one application. In contrast to TPMs, SEs are able to execute secure code and not limited to performing only cryptographic operations. For a considerable period, smart cards were the only type of SE used under different form factors and with different types of connectivity, such as USB dongles and contactless cards. However, following the introduction of Near Field Communication (NFC) in mobile phones, there are now three different form factors of SEs: the Universal Integrated Circuit Card (UICC), which, essentially, is a smart card often under the SIM card format; the embedded SE (eSE), which is usually a smart card microcontroller integrated in the NFC chip or directly in the hardware of the mobile phone; and the secure microSD card, which is based on a smart card microcontroller. Both the UICC and microSD card are removable.

5. Hypervisors and Virtualisation

In this section, we discuss Intel TXT and Java Card, as examples of hypervisors and virtualisation technologies for enabling secure execution.

5.1. Java Card

Java Card [43] provides an interoperable secure platform for secure elements, which enables them to run multiple applications, called applets, written in a subset of the Java language. Fundamentally, the components of a Java Card platform are: a Virtual Machine (VM) composed of a byte-code interpreter providing hardware-independence and, from version 3.x (for the classic edition), a mandatory embedded bytecode verifier; a set of APIs to hide the complexity of the underlying smart card communications protocols, to offer access to cryptographic functionalities and to enable features for secure execution, e.g. transaction and inter-applet sharing mechanisms; a Java Card Runtime Environment (JCRE), which provides security mechanisms such as the atomic update of persistent data; and a firewall which ensures a strong isolation between applets and the system, and between applets coming from different contexts (a context being related not to an application provider but to the package from which the applets are instantiated). As illustrated in Figure 2, applets from the same package are in the same context and

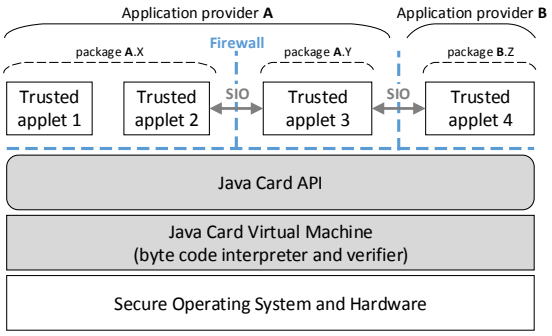


Figure 2: Java Card Architecture

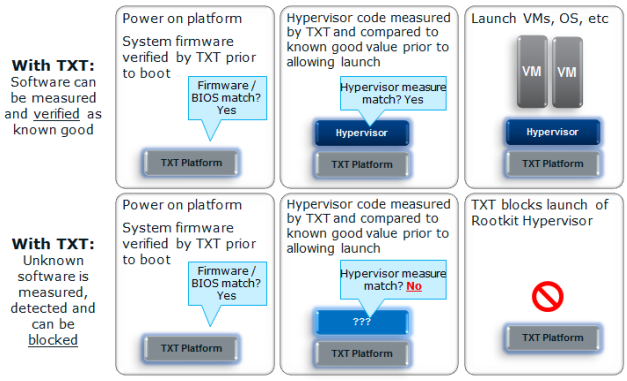


Figure 3: Using Intel TXT for Trusted Hypervisor Launch and VM Instantiation. (Reproduced from [23]).

classical access level modifiers are valid in this space. For applets from the same applications provider but which are located in different packages or for applets from different application providers, the firewall enforces the security rules by restricting the rights of an applet to access objects owned by another applet in a different context to only items that are explicitly tagged as shared (illustrated Figure 2 by the Shareable Interface Object (SIO), since the object must implement an interface extended from `Shareable`, which manages items authorized to be called).

Though Java Card can host multiple applets, this is only a single threaded environment. The applet provisioning mechanism is not defined by Java Card but by GlobalPlatform [44]. The regular ownership model for Java Card is issuer-centric (ICOM) [46], which means that only the issuer of a Java Card can load and install applications. However, GlobalPlatform provides different ways to authorize some third parties – application providers, for instance – to control some spaces on the card, called Security Domains. According the level of entitlement given to the third party and to its representative Security Domain on the card, applets can be loaded and installed based on an authorization token, or if the trust that the issuer grants to the third party is lower, authorization can be given only if the applets are signed by the issuer or by a trusted certification authority. In most of the devices running Java Card nowadays, similar mechanisms apply, so that only trusted applets can be loaded and executed on the platform. However, some more open but secure provisioning mechanisms have been considered, such as the GlobalPlatform Consumer-Centric Model (GP-CCM) [47] and the User-Centric Ownership Model (UCOM) [45], [46].

5.2. Intel Trusted Execution Technology (TXT)

Intel TXT [21], formerly known as LaGrande technology, aims to defend computing platforms from generalised software attacks, including firmware, BIOS and rootkit attack vectors. Broadly, TXT constructs a chain of trust from an on-board TPM that extends to the Virtual Machine Monitor (VMM) to enable trusted hypervisor and OS launch. At each stage of the launch sequence – beginning with the BIOS, option ROM and Master Boot Record (MBR) – a series of SHA-1 hashes are measured and compared with those extended from the TPM's Platform Configuration Registers (PCRs) to verify launch integrity. Any underlying system modifications, such as by rootkits and other surreptitious software, can be detected due to deviations in the launch configuration from known stored values. TXT interoperates with Intel's Virtualisation Technology (Intel VT) [22], which provides native extensions for hardware virtualisation to provide trusted launched of VMs with a high degree of isolation. Facilities are also offered for platform attestation in which a local or remote party is able to determine the trust status of the launch configuration of the platform or VM, i.e. whether it was instantiated correctly or not. A disadvantage of TXT is its significant hardware Trusted Computing Based (TCB), comprising the TPM, CPU chipset, motherboard and system buses; its advantage, however, is its wide availability on consumer devices on commercially-available Intel CPUs. Figure 3 illustrates a high-level architecture for instantiating trusted launch of VMs using TXT.

6. Trusted Execution Environments

Trusted Execution Environments (TEEs) are execution environments that operate alongside standard mobile environments, such as Android. This family of execution environments provides two security properties: (1), strong isolated execution of applications (usually hardware-enforced) running on top of the TEE, also called Trusted Applications (TAs); and (2), secure storage of data and cryptographic keys. Standards-wise, GlobalPlatform specifies a TEE system architecture [1], a range of TEE APIs [1]–[8], a TEE protection profile [9] for Common Criteria compliance, and documentation about functional testing of two TEE APIs [10] the TEE Client and Internal Core APIs. Despite standardisation efforts, however, a range of TEE implementations have arisen with often diverse characteristics, some of which are wholly GlobalPlatform compliant, while others are partially or non-compliant (Intel SGX). The availability of the solution also varies: some solutions are

commercial with paid licences, while others are open-source and available freely. We now detail the GlobalPlatform TEE and briefly examine its main implementations.

6.1. GlobalPlatform TEE

GlobalPlatform published the first TEE specification in July 2010. Surprisingly, the first specification did not provide a TEE overview description, but detailed a software interface (the TEE Client API [2]) enabling applications running in a standard mobile OS, also called the Rich Execution Environment (REE), to communicate with TAs. The specification of this API was strongly inspired by/ based on the ARM TrustZone [13] proposal, which provides the hardware means for instantiating a TEE. Thereafter, GlobalPlatfrom specified a hardware and a software architecture [1], both of which highlight full isolation between REE and TEE. The environments allocate their own resources – e.g. RAM, ROM, CPU, and OS – while communications between the two environments are only performed via the TEE Client API.

Figure 4 presents the main software components of a TEE. The trusted kernel maintains TAs and provides facilities to TA developers for accessing cryptographic resources, secure storage, networking, and others. The trusted kernel must also be instantiated using secure boot at every start-up (with authentication and isolation from the REE). The TEE Internal API [3], [4] interface enables TAs to communicate with other TAs and to access trusted hardware and kernel functionality. Finally, two interfaces are provided for the REE: the TEE Client API, described previously, and TEE Functional API, which offers applications running in REE a set of rich OS-friendly APIs.

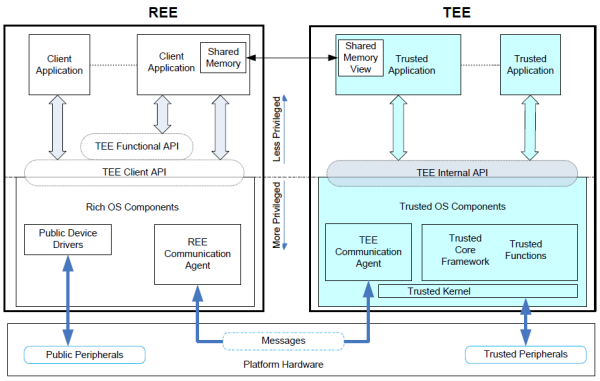


Figure 4: Software TEE Architecture [1]

Besides these interfaces. GlobalPlatform specifies three further specifications: the Trusted User Interface API [7] for enabling secure display and user input, which can be used by TAs to protect PIN or password entry; the TEE Secure Element API [5] that allows TAs to communicate with applications running on an attached secure element; and the TEE Sockets API [6] that lets TAs to establish and use network communications using POSIX-style sockets.

6.2. Intel Software Guard eXtensions (SGX)

Intel SGX (Software Guard eXtensions) [37] is an extension of the x86-64 instruction set that allows portions of an application to be executed in a secure container referred to as an 'enclave'. An enclave provides a protection against other applications or privileged system software running at any protection ring after initialisation, i.e. user mode (ring 3) or even kernel mode (ring 0). With SGX, only the data and code deemed security-critical is relocated to the enclave by defining 'trusted' functions in SGX specification language controlled by the developer, while the enclave application itself is written in a subset of C/C++. Enclaves may read/write to memory spaces in its host application, but not from the host application to enclaves outside of these pre-defined functions; enclaves are also unable to access other enclaves' contents arbitrarily. Currently, only enclaves are launched by a special launch enclave, which enforces that only signed by Intel may be executed in production, 'in the wild' deployments.

While enclave code is linked and compiled alongside the host application, sensitive data can be provisioned by bootstrapping a shared session key from SGX's remote attestation mechanism. This mechanism is based on the Enhanced Privacy ID (EPID) protocol – a Direct Anonymous Attestation (DAA) protocol that uses a group signature scheme in conjunction with a CPU-bound EPID key. A separate, special enclave (the quoting enclave) is used to measure, sign and certify the state of the enclave application, along with collecting other platform-specific data, which is transmitted back to the attestation challenger. Local attestation is also available for enabling two enclaves to verify the operating state of each other running on the same physical CPU. In this process, the enclaves produce separate operating reports containing their identifies. Next, MACs are computed over these reports using 128-bit AES-CMACs under a common report key bound to the underlying CPU. The MACs and common report key are used used to verify whether the enclaves are executing on the same platform. For secure storage, a separate key derived from the hardware-bound storage key (the root seal key) is used for binding sensitive data such that only the calling enclave may access that data. While Intel SGX achieves most goals of the GlobalPlatform TEE, such as secure storage and isolated execution, it does not provide native trusted UI or network communication directly from enclaves.

6.3. Technologies

Next, we discuss a range of proposals that – aside from SGX, which is a complete hardware-software solution for Intel CPUs (from Skylake chipsets and onwards) – have helped instantiate TEEs in mobile systems.

6.4. TEE Hardware Technologies

Aegis is a single-chip secure processor. It ensures the authentication of platforms and software using cryptographic keys computed by a physical unclonable function

(PUF) [14]. In addition, it guarantees the integrity and privacy of applications from both hardware and software attacks by using four secure execution modes with different privileges and memory protection approaches.

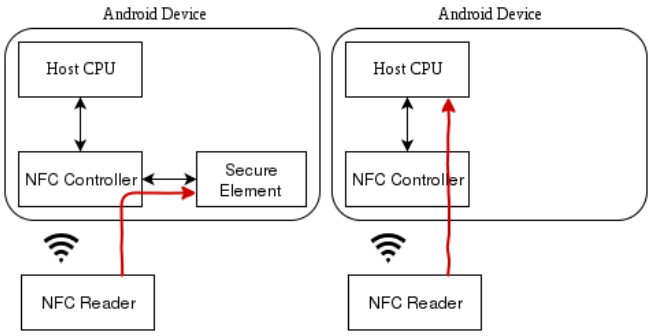
ARM TrustZone is a security architecture for instantiating an independent trusted world from the REE. In design, it shares similarities to the GlobalPlatform proposals; indeed, the first standardized TEE API – the TEE Client API – was originally a TrustZone Client API. TrustZone allocates two virtual cores to each physical core, and differentiates between execution modes using a non-secure (NS) bit to distinguish between trusted and untrusted world execution. Platform authentication is assured via a secure ROM, which measures and performs authenticated boot of the trusted world, similar to TPMs (Section 3). Additionally, TrustZone enables the use of peripherals, e.g. input/output (I/O) devices, such that only designated I/O device interrupts are routed to the trusted world. This is achieved using a TrustZone Protection Controller (TZPC), which secures on-chip peripherals, and the TrustZone Address Space Controller (TZASC), which protects DRAM and memory-mapped devices. These technologies allow secure accesses to sensitive I/O devices; for example, keyboard input to be routed securely to a sensitive, TEE-based authentication application.

6.5. Programmable TEEs

Programmable TEEs are sets of software and frimware components that enable the development, development and debugging of TAs. In this section, we focus explore a range of such solutions within the commercial world.

Nokia's On-board Credentials (ObC) [15] is one of the first programmable TEEs for an open security framework. Its software architecture is significantly different to the GlobalPlatform specifications, and provides six components: a provisioning module, interpreter, management module,a key engine and cryptographic library inside the TEE, and a credential manager on the REE. The only similarity with GlobalPlatform lies within the credential manager, which operates similarly to the GlobalPlatform TEE Client API. Because of this structure, the ObC platform provides dynamic credential provisioning and update. A major issue with ObC is that was narrowly-deployed, and available only in Symbian 3 phones.

The Trustonic TEE [16] is the first commercial solution with paid licenses. The Trustonic TEE is a merger and enhancement of technologies provided by ARM [17], Giesecke & Devrient (G&D) [18] and Gemalto [19]. Trustonic is underpinned significantly by ARM TrustZone, which provides the hardware means for instantiating the Trustonic TEE. G&D provided the secure mobile operating system Mobicore, while Gemalto provided the Trusted Foundations system; the Trustonic TEE is mostly GlobalPlatform compliant. The Open Portable TEE (OP-TEE) [20] is the first open source TEE enabling developers to develop, upload and test TAs (along with the TEE) in a widely-used open-source setting. The OP-TEE project was initiated by STMicroelectronics [24] and the Linaro Security



(a) SE-based (b) HCE-based

Figure 5: Card emulation using SE and HCE

Working Group [25], and its software software provides a range of GlobalPlatform-compliant APIs, such as the TEE Client, Internal and Sockets APIs. OP-TEE is available for a range of ARM-based, single-board computers, such as the Raspberry Pi 3 and the HiKey LeMaker boards. Another open-source, GlobalPlatform-compliant TEE is the OpenTEE [26] project led by the Intel Collaborative Research Institute for Secure Computing [27].

Qualcomm SecureMSM [29] is a Qualcomm proprietary TEE solution. It proposes a secure mobile operating system (called Qualcomm Secure Execution Environment, QSEE), which leverages TrustZone with enhanced cryptographic capabilities, such as device attestation and secure key provisioning methods. QSEE is used widely on Android OSs for implementing the KeyMaster service and performing fingerprint authentication.

More recently, Google published the Trusty TEE [30], which forms part of the Android open source project. It comprises a secure mobile operating system (Trusty OS) and a set of libraries and APIs to set communications between applications running within Trusty OS and Android. Like the other major current solutions, Trusty TEE is also based on ARM TrustZone technology.

7. Host Card Emulation (HCE)

Host-based Card Emulation (HCE) allows NFC-enabled devices to act as contactless smart cards without relying on a dedicated secure element (SE). It was first introduced in Android 4.4 (API 19) [31] and is used by mobile applications to perform transactions through NFC, such as banking and transport-related applications. Its ability to emulate contactless smart cards facilitates interoperability with existing card-reader infrastructures based on NFC [32].

When an SE is used, typically all of the data received by the NFC controller is routed directly to the SE (Figure 5a). In the case of HCE, however, data is directed to the device's CPU, which is responsible for handing it to the corresponding application via the operating system (Figure 5b). In both SE- and HCE-based deployments, data is passed to an application based on an Application ID (AID), as stipulated in the ISO/IEC 7816-4 specification [33].

Naturally, because HCE is software-based, it cannot provide the level of tamper-resistance that a hardware SE can. Device storage may only be protected by the security mechanisms of the operating system, such as application sandboxing [34]. Therefore, it is not recommended that critical transactions, like mobile payments and high-security access control, to be based entirely on HCE [32], [35]. In such scenarios an SE- or TEE-based solution would be more appropriate for safeguarding critical information.

Android devices do not always contain an SE or TEE. In order to make HCE more widely-accessible, cloud-based storage that acts as a remote SE is one proposed solution. However, Internet connectivity is typically required and achieving authentication via a remote cloud platform can be challenging. Various proposals have been made to enchangce the security of HCE, some of which are being used by stakeholders. Such methods include verification of the user and/or the hardware (e.g. using PINs or biometrics), limitations to transaction amounts, tokenised payments, operating system checks, and white-box cryptography [36].

8. Encrypted Execution Environments (E3)

An Encrypted Execution Environment (E3) is an execution environment in which the executed application software is encrypted; its goal is to enable application execution without revealing the instructions that comprise the application.

An example E3 application is a company that wishes to control the provisioning of their developed software to devices. This is likely to be employed a company that allocates a large amount of monetary value and time to protect its investment from counterfeiting and other unauthorised duplication. In essence, if each device provisioned with software has its own E3 key, then E3 software for one device will not execute on another. The primary requirement for E3 is that the executed (encrypted) instructions must be revealed only within the E3; this does not necessarily imply that the instructions can be executed directly from their encrypted state. Indeed, it is acceptable for the E3 to decrypt and execute instructions so long as plain-text instructions are not revealed to the E3 environment. Certain users may apply extra requirements to the E3, such as encrypting any data stored by the application outside the E3 or in the registers. In general, however, it is necessary that the application be encrypted at all times outside the the encrypted environment.

8.1. Example E3 scheme

An example E3 scheme is the hardware-software binding scheme proposed by Lee et al. [38]. This proposal uses E3 to prevent platform counterfeiting and unauthorised firmware modification by 'binding' software to hardware. The work's assumption work is that multiple provisioned applications would be encrypted under different keys for each device [38]. As a corollary, the applications that have been personalised for each device can only execute under their E3. The scheme proposed by Lee et al. creates the E3 as a zone surrounding the processor of the device, which also includes

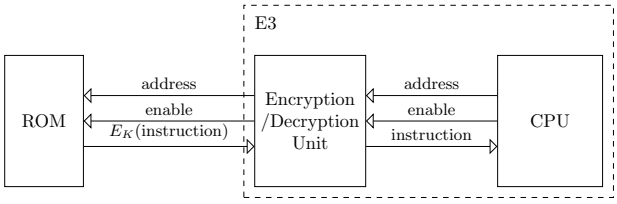


Figure 6: A diagram of an example E3 system as implemented by Lee et al.

a small encryption/decryption hardware element. This element is responsible for decrypting all instructions and data loaded from memory before the instructions are executed by the processor. This ensures that, while in memory, the application is protected by the E3 encryption scheme and that, after decryption, the program can execute normally [38]. This architecture is illustrated in Figure 6, which demonstrates the creation of an E3 using an extension that acts as an intermediary between the processor and memory. By positioning a unit that forwards the memory addresses, signals and instructions between the CPU and memory, an E3 is created for device applications to be executed without interfering with the underlying processor design.

9. Comparison

In this section, we revisit the trust technologies described in this paper and evaluate them using the criteria in Section 2.2. Table 1 illustrates the criteria comparison between the various technologies. Note that the no single technology satisfies all of the criteria. However, device manufacturers and system designers of IoT and CPS may utilise this to determine the suitability of each solution against their desired assurances.

The TPM, one of the earliest proposals for trusted computing, has significant limitations as it is directed primarily towards static validation of the integrity of a device's platform and applications. Notably, it offers tamper-resistant hardware and internal and external secure storage and limited ability for protecting applications against on-device adversaries 2 and 3 (discussed in Section 2.1.1).

In this analysis, we merged SE and Java Card despite being discussed separately in past sections; a substantial number of SEs support Java Card, and the criteria satisfied by SEs and Java Cards are uniformly common. A major issue with SEs, however, is the lack of end-user access to the devices. The next three proposals – Intel TXT, GP TEE, and Intel SGX – satisfy more or less the same criteria. The primary exception is that the GP-TEE defines no remote attestation mechanism, unlike Intel TXT and SGX. Secondly, most commercial implementations of the GP TEE, e.g. Trustonic Kinibi, are closed-access; they permit only authorised authorities to (un-)install trusted applications. While some open-source GP TEEs exist, like OP-TEE, these are found rarely on off-the-shelf consumer devices. Notably, all three TEEs provide poor protection against hardware-based adversaries, unlike SEs, and protect

TABLE 1: Comparison Trust Technologies Based on Defined Criteria

Criteria	TPM	SE / Java Card	TXT	GP-TEE	SGX	HCE	E3
User Control	✓	✗	✓	✗	✓	✗	✗
Centralised Control	✓	✓	✓	✓	✓	✓	✓
Managed Access	❋	✓	❋	❋	❋	✓	✓
User Managed Access	❋	✗	❋	❋	❋	✗	✗
Open Access	❋	✗	❋	❋	❋	✗	✗
Static Verification	✓	✓	✓	✓	✓	✗	❋
Continuous Verification	✗	✓	✗	✗	✗	✗	✗
Tamper-Resistant Hardware	✓	✓	✗	✗	✗	✗	✓
Secure Storage (Internal)	✓	✓	✗	✗	✗	✓	❋
Secure Storage (External)	❋	❋	❋	✓	✓	❋	❋
Isolated Execution	✗	✓	✓	✓	✓	✓	✓
Software/Hardware Binding	✗	❋	✗	✗	✗	✗	✓
Remote Attestation	✓	❋	✓	✗	✓	❋	✗
Protection Against Adversary - On-Device Adversary 1	❋	✓	✓	✓	✓	✓	✓
Protection Against Adversary - On-Device Adversary 2	▼	❋	✓	✓	✓	❋	▼
Protection Against Adversary - On-Device Adversary 3	▼	✓	▼	▼	▼	✗	▼
Protection Against Adversary - Off-Device	❋	❋	✗	✗	✗	❋	❋

✓: Supports, ✗: Does not support, ❋: In some situation supports, ▼: Limited to no support.

primarily against kernel- and user-space adversaries. HCE highlights a unique approach for attaining trusted computing by executing security-sensitive data in a remote location, e.g. cloud-based service, rather than a consumer device. This has its benefits, but, on an in-the-field device, it provides few security guarantees. Lastly, E3 demonstrates significant promise, especially for embedded devices with high-security requirements; namely, strong protection against adversaries that aim to compromise intellectual property protection by strongly binding software to host hardware.

10. Conclusion

In this paper, we discussed the evolution of technologies aiming to provide secure and trusted computing by discussing the major proposals over the past 10-15 years. Such technologies have extended the basic notion of secure and trusted computing to divergent domains, such as payments to intellectual property protection. Notably, these technologies are required to accommodate the emergence of the IoT and CPS, in which adversaries may have greater physical access to deployed devices. We discussed evaluation criteria for trust technologies and their application in the IoT and CPS. Crucially, we showed that no single technology meets all of the criteria, but we illustrated how IoT and CPS designers may utilise these technologies to meet their chosen security objectives. Lastly, such an analysis is useful as a guide for future research, by suggesting the existing gaps in secure and trusted computing.

Acknowledgements

Carlton Shepherd and Robert P. Lee are supported by the EPSRC and the UK government as part of the Centre for Doctoral Training in Cyber Security at Royal Holloway, University of London (EP/K035584/1). Emmanuel Conchon and Damien Sauveron are supported by the IoTSec (IoT Security) project funded by Région Limousin. The authors would like to thank anonymous reviewers for their valuable comments that help us improve the paper.

References

[1] GlobalPlatform, "TEE System Architecture, version 1.0," GlobalPlatform Specifications, December 2011. [Online]. Available: https://www.globalplatform.org/specificationsdevice.asp

[2] ——, "TEE Client API Specification, version 1.0," GlobalPlatform Specifications, July 2010. [Online]. Available: https://www.globalplatform.org/specificationsdevice.asp

[3] ——, "TEE Internal Core API Specification, version 1.1 ," GlobalPlatform Specifications, July 2014. [Online]. Available: https://www.globalplatform.org/specificationsdevice.asp

[4] ——, "TEE Internal API Specification, version 1.0," GlobalPlatform Specifications, December 2011. [Online]. Available: https://www.globalplatform.org/specificationsdevice.asp

[5] ——, "TEE Secure Element API Specification, version 1.1," GlobalPlatform Specifications, September 2015. [Online]. Available: https://www.globalplatform.org/specificationsdevice.asp

[6] ——, "TEE Sockets API Specification, version 1.0," GlobalPlatform Specifications, July 2015. [Online]. Available: https://www.globalplatform.org/specificationsdevice.asp

[7] ——, "Trusted User Interface API Specification, version 1.0," GlobalPlatform Specifications, June 2013. [Online]. Available: https://www.globalplatform.org/specificationsdevice.asp

[8] ——, "TEE TA Debug Specification, version 1.0," GlobalPlatform Specifications, February 2014. [Online]. Available: https://www.globalplatform.org/specificationsdevice.asp

[9] ——, "TEE Protection Profile, version 1.2," GlobalPlatform Specifications, January 2015. [Online]. Available: https://www.globalplatform.org/specificationsdevice.asp

[10] ——, "TEE Initial Configuration Test Suite 1.1.0.1," GlobalPlatform Specifications, March 2016. [Online]. Available: https://www.globalplatform.org/specificationsdevice.asp

[11] ——, "TPM Main: Part 1 Design Principles," The Trusted Computing Group (TCG), Rev 1.4, March 2011. [Online]. Available: http://www.trustedcomputinggroup.org/resources/tpm%5C_main%5C_specification

[12] ——, "Trusted Computing Group, TCG Specification Architecture Overview," The Trusted Computing Group (TCG), Rev 1.4, OR, USA, August 2007.

[13] ARM Security Technology, "Building a Secure System using TrustZone Technology," April 2009. [Online]. Available: http://infocenter.arm.com/help/topic/com.arm.doc.prd29-genc-009492c/PRD29-GENC-009492C_trustzone_security_whitepaper.pdf

[14] B. Gassend, D. Clarke, M. van Dijk, and S. Devadas, "Silicon physical random functions," in *Proceedings of the 9th ACM Conference on Computer and Communications Security*, ser. CCS '02. New York, NY, USA: ACM, 2002, pp. 148–160.

[15] Kari Kostiainen, "On-board Credentials: An Open Credential Platform for Mobile Devices," Doctoral dissertation, Aalto University, Helsinki, 2012.

[16] TrustonIC, "Trusted Executed Environment (TEE)," Visited on April 2016. [Online]. Available: https://www.trustonic.com/technology/trusted-execution-environment

[17] ARM, "ARM the Architecture for the Digital World," Visited on April 2016. [Online]. Available: https://www.arm.com/

[18] Giesecke & Devrient, "Giesecke & Devrient Creating confidence," Visited on April 2016. [Online]. Available: https://www.gi-de.com/fr/index.jsp

[19] Gemalto, "Gemalto - Security to be free," Visited on April 2016. [Online]. Available: http://www.gemalto.com/france

[20] Linaro, "OP-TEE," Visited on April 2016. [Online]. Available: https://wiki.linaro.org/WorkingGroups/Security/OP-TEE

[21] Intel Corporation, "Intel Trusted Execution Technology: White Paper," Visited on April 2016 [Online]. Available: http://www.intel.com/content/www/us/en/architecture-and-technology/trusted-execution-technology/trusted-execution-technology-security-paper.html

[22] Intel Corporation, "Intel Virtualization Technology (VT)," Visited on April 2016 [Online]. Available: http://www.intel.com/content/www/us/en/virtualization/virtualization-technology/intel-virtualization-technology.html

[23] A. Sallam, "XenDesktop and The Evolution of Hardware-Assisted Server Technologies," Citrix: Trends and Innovations, February 2014. Visited on April 2016 [Online]. Available: https://www.citrix.com/articles-and-insights/trends-and-innovation/feb-2014/xendesktop-and-the-evolution-of-hardware-assisted-server-technologies.html#Intel%C2%AE%20Hardware-Assisted%20Security%20Technologies

[24] STMicroelectronics, "About STM," Visited on April 2016. [Online]. Available: http://www2.st.com/content/st_com/en.html

[25] Linaro, "Linaro Security Working Group," Visited on April 2016. [Online]. Available: https://wiki.linaro.org/WorkingGroups/Security

[26] Open-TEE project, "Secure Systems group - Intel Collaborative Research Institute for Secure Computing," Visited on April 2016. [Online]. Available: https://open-tee.github.io/

[27] "Intel Collaborative Research Institute for Secure Computing," Visited on April 2016. [Online]. Available: http://www.icri-sc.tu-darmstadt.de/icri-sc/institute/

[28] R. N. Akram, K. Markantonakis, and K. Mayes, "Trusted Platform Module for Smart Cards". In 6th IFIP International Conference on New Technologies, Mobility and Security (NTMS), O. Alfandi, Ed. Dubai, UAE. IEEE CS, March 2014.

[29] Qualcomm, "Qualcomm Haven Security Platform ," Visited on April 2016. [Online]. Available: https://www.qualcomm.com/products/snapdragon/security

[30] Google, "Trusty TEE," Visited on April 2016. [Online]. Available: https://source.android.com/security/trusty/index.html

[31] Google, "Host-based Card Emulation," Visited on April 2016. [Online]. Available: http://developer.android.com/guide/topics/connectivity/nfc/hce.html

[32] ——, Smart Card Alliance, "Host Card Emulation (HCE) 101: White paper," August 2014

[33] "ISO 7816 Part 4: Interindustry Commands for Interchange," Visited on April 2016. [Online]. Available: http://www.cardwerk.com/smartcards/smartcard_standard_ISO7816-4.aspx

[34] M. Alattar and M. Achemlal, "Host-Based Card Emulation: Development, Security, and Ecosystem Impact Analysis", in Proceedings of *the High Performance Computing and Communications, 2014 IEEE 6th Intl Symp on Cyberspace Safety and Security*, August, 2014.

[35] Pannifer, Steve and Clark, Dick and Birch, Dave "HCE and SIM Secure Element: Its not black and white: White paper," June 2014

[36] UL, "HCE security implications - Analyzing the security aspects of HCE: White paper," January 2014

[37] McKeen, Frank and Alexandrovich, Ilya and Berenzon, Alex and Rozas, Carlos V. and Shafi, Hisham and Shanbhogue, Vedvyas and Savagaonkar, Uday R., "Innovative Instructions and Software Model for Isolated Execution," in *Proceedings of the 2Nd International Workshop on Hardware and Architectural Support for Security and Privacy*, HASP '13. Tel-Aviv, Israel: ACM, 2013, pp. 10-1–10-8.

[38] R. P. Lee, K. Markantonakis and R. N. Akram, "Binding Hardware and Software to Prevent Firmware Modification and Device Counterfeiting", in Proceedings of *the 2nd ACM Workshop on Cyber-Physical System Security (CPSS)*, J. Zhou and J. Lopez (eds.), Xian, China, May 30, 2016.

[39] M. G. Msgna, H. Ferradi, R. N. Akram, K. Markantonakis,"Secure Application Execution in Mobile Devices". The New Codebreakers, Ryan, A. Peter Y. and Naccache, David and Quisquater, Jean-Jacques (ends), Springer Berlin Heidelberg, 2016, pages 417-438

[40] K. Markantonakis and R. N. Akram, "A Secure and Trusted Boot Process for Avionics Wireless Networks", 16th Integrated Communication, Navigation and Surveillance Conference (ICNS), Aloke Roy (eds.), Herdon, VA, USA, April 2016, IEEE

[41] R. N. Akram and K. Markantonakis, "Challenges of Security and Trust of Mobile Devices as Digital Avionics Component", 16th Integrated Communication, Navigation and Surveillance Conference (ICNS), Aloke Roy (eds.), Herdon, VA, USA, April 2016, IEEE

[42] Raja Naeem Akram, Konstantinos Markantonakis, and Keith Mayes, "Trusted Platform Module for Smart Cards". In 6th IFIP International Conference on New Technologies, Mobility and Security (NTMS), O. Alfandi, Ed. Dubai, UAE. IEEE CS, March 2014.

[43] Oracle, "Java Card Technology", Visited on April 2016. [Online]. Available: http://www.oracle.com/technetwork/java/embedded/javacard

[44] GlobalPlatform, "GlobalPlatform Card Specification v2.3", Visited on April 2016. [Online]. Available: https://www.globalplatform.org/specificationscard.asp

[45] R. N. Akram, K. Markantonakis, and D. Sauveron, "A Novel Consumer-Centric Card Management Architecture and Potential Security Issues". Information Sciences, Volume 321, 10 November 2015, Pages 150-161, ISSN 0020-0255, http://dx.doi.org/10.1016/j.ins.2014.12.049.

[46] R. N. Akram, K. Markantonakis, and K. Mayes, "A Paradigm Shift in Smart Card Ownership Model". In Bernady O. Apduhan, Osvaldo Gervasi, Andres Iglesias, David Taniar, and Marina Gavrilova, editors, Proceedings of the 2010 International Conference on Computational Science and Its Applications (ICCSA 2010), pages 191200, Fukuoka, Japan, March 2010. IEEE Computer Society.

[48] *ISO/IEC 11889-1: Information Technology - Trusted Platform Module - Part 1: Overview*, Online, ISO Standard 11 889-1, May 2009.

[49] *TPM Main: Part 1 Design Principles*, Online, Trusted Computing Group (TCG) Specification 1.2, Rev. 116, March 2011.

[50] *TCG Glossary*, Online, Trusted Computing Group (TCG) Glossary 1.1, Rev. 1.00, March 2017.

[51] "M-Shield Mobile Security Technology: Making Wireless Secure," Texas Instruments, Whilte Paper, February 2008.

[52] A. Avizienis, J.-C. Laprie, B. Randell, and C. Landwehr, "Basic Concepts and Taxonomy of Dependable and Secure Computing," *Dependable and Secure Computing, IEEE Transactions on*, vol. 1, no. 1, pp. 11–33, 2004.

[47] GlobalPlatform, "A New Model: The Consumer-Centric Model and How it Applies to the Mobile Ecosystem", Visited on April 2016.