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*—Notions like security, trust, and privacy are crucial in the digital environment and in the future, with the advent of technologies like the Internet of Things (IoT) and Cyber-Physical Systems (CPS), their importance is only going to increase. Trust has different definitions; some situations rely on real-world relationships between entities while others depend on robust technologies to gain trust after deployment. In this paper we focus on these robust technologies, their evolution in past decades and their scope in the near future. The evolution of robust trust technologies has involved diverse approaches; as a consequence trust is defined, understood and ascertained differently across heterogeneous domains and technologies. In this paper we look at digital trust technologies from the point of view of security and examine how they are making secure computing an attainable reality. The paper also revisits and analyses the Trusted Platform Module (TPM), Secure Elements (SE), Hypervisors and Virtualisation, Intel TXT, Trusted Execution Environments (TEE) like GlobalPlatform TEE, Intel SGX, along with Host Card Emulation, and Encrypted Execution Environment (E3). In our analysis we focus on these technologies and their application to the emerging domains of the IoT and 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 System**

1. Introduction

Computer devices, large or small, now form heterogeneous interconnected networks that consist of complex sub-systems and sub-networks, managed and maintained by divergent organisations. In a centralised environment, where all devices are vetted (offline) beforehand, establishing trust in individual devices or groups of devices can potentially be straightforward. In contrast, with increased numbers of devices, offline vetting can be difficult in some situations, and the situation becomes more challenging. For both of these scenarios, a wide range of robust technologies are deployed to provide assurance that individual devices are secure and running in a trusted state. Such technologies, in the context of this paper, are referred to as trust technologies. These technologies are pivotal in designing, developing and implementing secure computing. A secure computing scheme is one that has a secure, trusted and verifiable environment. The applications running in such an environment have the assurance that no malicious entity can interfere, as any such intervention would be detected by the robust trust technologies used.

Diversification of trust technologies has occurred in several different settings. Each inception provides additional services to further develop secure computing environments. Each new trust technology helps to define and articulate the notion of trust in a device's state and security in a distributed environment. For example, Trusted Computing Group (TCG) [48], [49], ARM TrustZone [13] and M-Shield [50] are based on the effectiveness of trust evaluation and validation mechanisms [51]. With increased reliance on on-demand and ubiquitous services, connecting with a wide range of devices that might not be under the control of a particular organisation/individual is becoming commonplace. In such a chaotic and ever-changing environment, dynamic establishment and verification/validation of digital trust is crucial. The need for digital trust will only grow with increasing adoption of cloud computing, mobile platforms, the Internet of Things (IoT) and Cyber-Physical Systems (CPS).

A secure computing environment is crucial for emerging technologies like the IoT and CPS, as these technologies will be interacting directly with the physical world and potentially assisting individuals and societies in their daily activities. Modern life will be heavily dependent upon them, requiring increased levels of assurance that devices forming part of the IoT and CPS will perform as stated. The security, trustworthiness and reliability of such devices may become the differentiating factor and in section 2 we briefly discuss an adversarial model and criteria for evaluation of secure and

trusted execution technologies, in the context of the IoT and CPS. We discuss the evolution of secure and trusted execution proposals over the years in sections 3 to 8. We evaluate these technologies in section 9 for their effectiveness based on 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

In this section we discuss two generic types of adversary [41]. This categorisation is based on the level of access an adversary has to a particular device and not on the basis of an individual adversary's ability to compromise a given device. This categorisation is valid because in the IoT and CPS, an adversary can have physical access to individual devices.

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 further divide the capabilities of this adversary:

1) Malicious Application Adversary: An adversary that develops a malicious application and gets it installed on to the target device(s).
2) Malicious Root Adversary: An adversary that has compromised the kernel space of a device, giving himself root access. He cannot compromise the hardware of the device.
3) Malicious Hardware Adversary: An adversary that has the ability to introduce hardware Trojans.

2.1.2. Off-Device Adversary. Off-device adversaries are malicious users that do not fall under any of the categories discussed in the previous section. This type of adversary manipulates the communication interface and tries to find any loopholes in the security of the device. The objective is to introduce a malicious code into the device that might then give the control of the device to the geographically remote adversary.

2.2. Criteria

In this section, we describe the evaluation criteria for the trust technologies discussed in this paper.

1) User Control: The trust technology deployed as part of the device is under the control of its user.
2) Centralised Control: The trust technology deployed as part of the device 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 the managed access, this can also be implemented as part of the user and centralised control.
5) Open Access: To use the features and services of the trust technology, no prior authorisation is required.
6) Static Verification: The trust technology has the ability or it can be extended to provide static integrity verification of the device platform and applications running on it.
7) Continuous Verification: The trust technology has the ability or it can be extended to provide continuous integrity verification of the device platform and applications running on it, monitoring individual runtime instructions.
8) Tamper-Resistant Hardware: The hardware on which the trust technology is implemented is tamper-resistant.
9) Secure Storage (Internal): The device has substantial internal storage used for protecting sensitive data/applications.
10) Secure Storage (External): The device has substantial external storage used for protecting sensitive data/applications. This also includes encrypted data/applications stored on external storage, which may itself not be secure (an adversary may be able to read its contents).
11) Isolated Execution: The trust technology provides services to enable an application to execute in complete isolation, without interference from any other application on the device.
12) Protection Against Bit Faults: The trust technology provides adequate protection against fault attacks that might change a byte or bit value (by laser beam) stored in non-persistent storage.
13) 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 technique.
14) Remote Attestation: The trust technology can provide evidence to assure a remote device that it and the associated device (platform and applications) are functioning properly.
15) Protection Against Adversary: The trust technology can protect the device against any modification by the adversaries described in this paper.

3. Trusted Platform Modules (TPMs)

For in-depth discussion of individual components and their functionality please refer to [12], [28]. In this section, we examine how the TPM provides a secure boot to the host device along with trustworthy reporting and attestation operations.

Secure Boot (Measurement Operation). When a user boots up her 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 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 daisy 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

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.

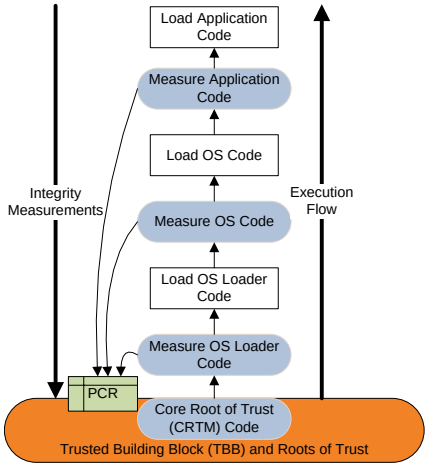


Figure 1: Trusted Platform Boot Sequence

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 (i.e. any assets like cryptographic keys) according to rules and security requirements established by its owner. Historically, invented 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 a TPM, SEs are able to run secure code and not limited to only performing 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) technology in mobile phones, there are now three different form factors of SE: the Universal Integrated Circuit Card (UICC) which is basically 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 virtualisations.

5.1. Java Card

Java Card [43] is a technology that provides an interoperable secure platform for secure elements, which enables them to run multiple applications, called applets, written in a subset of Java language. Basically, the components of

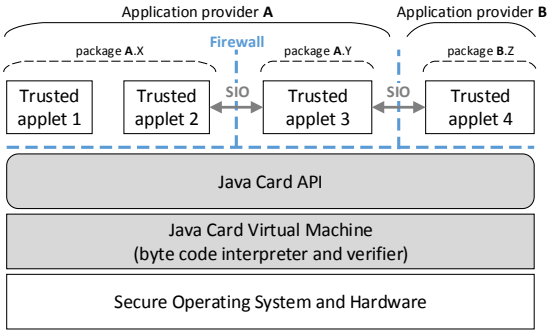


Figure 2: Java Card Architecture

a Java Card platform are: a Virtual Machine (VM) which is composed of a bytecode interpreter providing hardware-independence and, since 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 (transaction mechanisms, inter-applet sharing mechanisms, etc.); a Java Card Runtime Environment (JCRE), which is not illustrated in Figure 2, but 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 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 his applications on it. However, GlobalPlatform provides different ways to authorize some third parties (for instance applications providers) 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 provisioning but still secure mechanisms are considered, like 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's 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. Specifically, 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. Underlying system modifications, such as by rootkits or other surreptitious software, can be detected due to changes in the launch configuration that cause it to deviate from the stored known values. TXT interoperates with Intel's Virtualisation Technology (Intel VT) [22], which provides native extensions for hardware virtualisation, with the aim of providing trusted launched of VMs where applications can be executed with a high degree of isolation while retaining elements of trust. 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. One disadvantage of TXT is its significant hardware Trusted Computing Based (TCB), comprising the TPM, CPU chipset, motherboard and system buses; its advantage, however, is a widely-deployed means of spawning trusted VMs and performing measured launches. Figure 3 demonstrates a high-level architecture for instantiating trusted launch of VMs via TXT.

6. Trusted Execution Environments

Trusted Execution Environments (TEEs) are mobile execution environments running along side standard mobile environments (e.g., Android). This new family of execution environments provides two security properties: (1) secure execution isolation for applications running on top of a

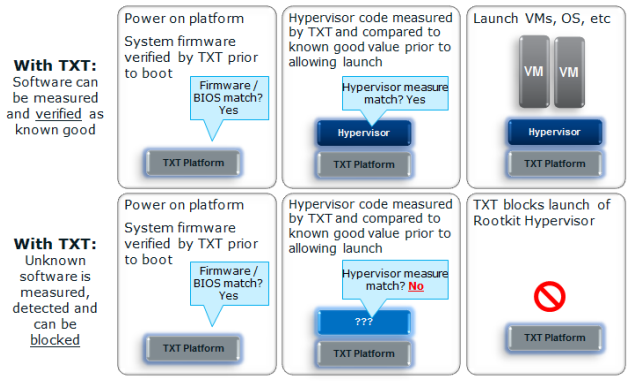


Figure 3: Using Intel TXT for Trusted Hypervisor Launch and VM Instantiation. (Reproduced from [23]).

TEE (also called Trusted Applications: TA) and (2) secure storage of data and cryptographic keys. In standardization, GlobalPlatform specifies TEE software and hardware architecture [1], a set of TEE APIs [1]–[8], a TEE protection profile [9] and documentation about the functional testing of two TEE APIs [10], i.e., TEE client API and TEE internal core API. Concurrently, various TEE implementations arise. These solutions have diverse characteristics. For instance, only some solutions are GlobalPlatform compliant. Some solutions are software-based (e.g., OP-TEE [20]). Others are hardware-software solutions (e.g., TrustonIC solution [16] requires TrustZone [13]). The availability of the solution also varies: some solutions are commercial, hence paid licenses, usually with specific rules (e.g., no testing is allowed) are needed.

We now detail the GlobalPlatform TEE proposal and briefly examine the main TEE implementations while highlighting its major characteristics.

6.1. GlobalPlatform TEE

GlobalPlatform published the first TEE specification in July 2010. Surprisingly, the first specification did not provide a TEE overview description. It detailed a software interface (i.e., TEE Client API [2]) enabling applications running in the standard mobile OS (also called Rich Execution Environment, REE) to communicate with TAs. The specification of this API was strongly inspired by/ based on the ARM TrustZone [13] proposal. Thereafter, GlobalPlatform specified a hardware and a software architecture [1]. Both architectures highlight full isolation between REE and TEE. Every environment has its own resources (e.g., RAM, ROM, CPU, TA, OS...) and 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 clearly runs TAs. In addition, it provides facilities to TA developers. The trusted kernel must also proceed to a secure boot (authentication and isolation from the REE) at every start-up. TEE Internal API [3], [4] is the interface that enables TAs to communicate with other TAs and to access trusted hardware resources (via trusted functions). Finally, two interfaces are provided in the REE: TEE Client API as previously described and TEE Functional API, which offers applications running in REE a set of rich OS-friendly APIs.

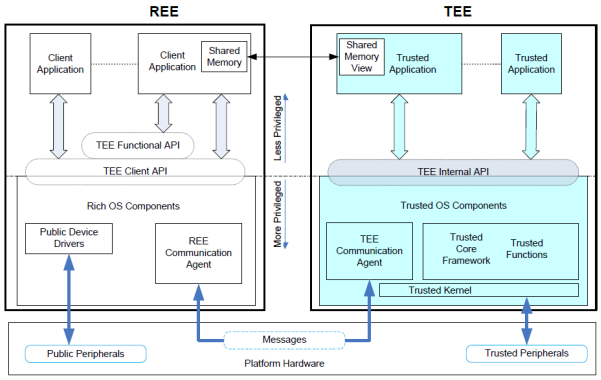


Figure 4: Software TEE Architecture [1]

Besides the interfaces previously mentioned, GlobalPlatform specifies three more interfaces. Trusted User Interface API [7] enables secure display and input. For instance, this API can be used by TAs to protect PIN or password entry. TEE Secure Element API [5] enables TAs to communicate with applications running on a secure element. TEE Sockets API [6] enables TAs to establish and use network communications using a socket-style approach.

6.2. Technologies

TEE technologies are key components in building a mobile platform with a TEE. We discuss three proposals.

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 Technology is a security architecture for building a TEE platform. It is very similar to the GlobalPlatform proposals. Indeed, the first standardized TEE API (i.e., TEE Client API) was originally a TrustZone Client API. Basically, ARM TrustZone ensures software authentication based on public key cryptography and platform authentication via a secure ROM. It also proposes memory security management approaches and three secure execution modes. Another TrustZone-like technology is Texas Instruments M-Shield Technology.

SGX (Software Guard eXtensions) [37] is a set of instructions developed by Intel for its processor, allowing applications (or parts 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 such as the OS, hypervisor or even BIOS. The main difference with TPM or TXT is that in SGX the whole application does not have to be stored securely. Only private data and the applications code that operates on it has to be stored in the Enclave. To upload these sensitive data, a software attestation mechanism is provided that relies on a cryptographic signature based on a SHA-256 digest. This digest certifies the identity of the software is verified, before the codes execution, against the identity enforced by a trusted third-party authority. SGX also provides a remote attestation feature that allows an Enclave to check the identity of another Enclave running on a remote host.

6.3. Programmable TEEs

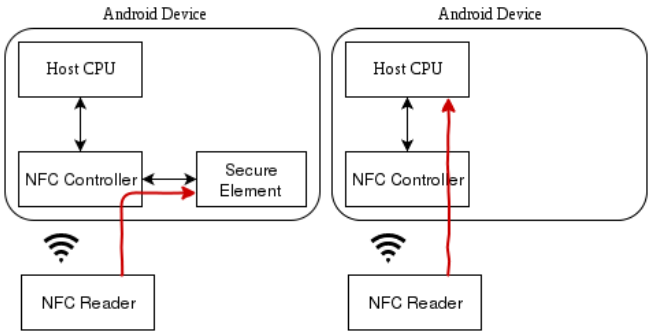
Programmable TEEs are sets of software components that enable the setup of a TEE, and the development and debugging of TAs. In this section, we focus on six solutions.

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

TrustonIC TEE [16] is the first commercial solution with paid licenses. TrustonIC TEE is a merger and enhancement of technologies provided by ARM [17], Giesecke & Devrient (G&D) [18] and Gemalto [19]. ARM provided the TrustZone technology. G&D provided the secure mobile operating system Mobicore. Gemalto provided the Trusted Foundations system. TrustonIC TEE is generally GlobalPlatform compliant. The main TEE API has been developed but specific APIs such as TEE arithmetical API, which enables the implementation of new cryptographic algorithms, are still under development.

The Open Portable TEE (OP-TEE) [20] solution is the first open source TEE enabling developers to develop, upload and test TAs and to test and contribute to OP-TEE software. The OP-TEE project was initiated by STMicroelectronics [24] and Linaro Security Working Group [25] and has also been supported by TrustonIC. OP-TEE software provides the two main GlobalPlatform TEE APIs, i.e., TEE Client API and TEE Internal API.

The Open-TEE [26] solution is also an open source TEE and is the result of the Open-TEE project led by the Intel Collaborative Research Institute for Secure Computing [27]. This solution has three main criteria. First, it is hardware-independent. Therefore, TAs that are implemented and validated by Open-TEE can run on top of any TEE hardware implementations. Second, the Open-TEE solution provides developer-friendly tools and frameworks for developing and debugging TAs. Finally, this solution is GlobalPlatform compliant.



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

Figure 5: Card emulation using SE and HCE

Qualcomm SecureMSM [29] is a Qualcomm proprietary TEE solution. It proposes a secure mobile operating system (called Qualcomm Secure Execution Environment, QSEE) with enhanced cryptographic capabilities and various secure key provisioning methods. The Qualcomm solution also ensures a secure boot functionality.

More recently, Google published a TEE called Trusty TEE [30]. Trusty TEE is the result of an Android open source project. It consists of 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 - Remote Execution

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

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

Because HCE is software-based, it cannot provide the amount of security that a hardware SE can. Device storage can 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, be based purely on HCE [32], [35]. In such scenarios an SE, TEE, or cloud-based secure storage can be used for safeguarding critical information.

Android devices do not always contain an SE or TEE. In order to make HCE available to more users, cloud-based storage that acts as a remote SE is a better solution. However, an Internet connection may be required in order to perform a transaction, and achieving authentication via the cloud platform can be a challenging task. Various proposals have been made with regard to enhancing 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 a type of execution environment in which the application software that is executed is encrypted. The goal of an E3 is to allow application execution without revealing the instructions that make up the application.

One example illustrating the use of E3s is a company that wants to control who is able to provision their software to devices. Devices could be mobile phones, multi-application smart cards or even more complicated systems such as PCs or servers. A software provisioned by a company is likely to be the result of a large amount of investment and/or development time. To protect its investment from counterfeiting and unauthorised copying the company provisions their software for execution in an E3. If each device provisioned with software has its own E3 key, then software for one device will not execute on another. In this setting, to control provisioning of the application it is critical to limit the knowledge that people can gain about the software. If the operations comprising the application are known then the software can easily be transferred from one device to another. Using an E3 allows software to execute on only devices that are in possession of the key with which the software is encrypted.

The main requirement for an E3 is that the instructions executed as part of the encrypted application must be revealed only within the E3. It is not a requirement that the instructions be executed directly from their encrypted state; it may be acceptable for the E3 to decrypt instructions and then execute them as long as the plain text instructions are not revealed to the outside world. Some users may apply extra requirements to the E3, such as encrypting any data stored by the application outside the E3 or in the registers. However, in general it is only necessary that the application be encrypted at all times when it is not within the encrypted environment.

8.1. Example E3 scheme

One example of an E3 scheme is the hardware-software binding scheme proposed by Lee et al. in 2016 [38]. Their proposal uses an E3 to bind hardware and software together on a device, to attempt to prevent platform counterfeiting or unauthorised firmware modification. The hardware-software binding case considered by the authors is not equal to the general E3 but is a narrower issue, with extra requirements for the solution.

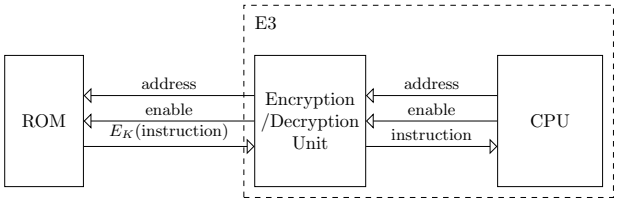


Figure 6: A diagram of an example E3 system as implemented by Lee et al.

The reasoning behind the work of Lee et al. is that an application to be provisioned to many devices would be encrypted for each device under a different key [38]. Each application requires the E3 that possesses the correct key, in order to be correctly decrypted and executed. Similarly, the hardware devices only execute applications that have been personalised for their E3s. This personalisation is created by encrypting the application using a device-specific E3 key. Therefore the hardware and software are bound using this mutual dependency based on the E3.

The scheme proposed by Lee et al. creates the E3 as a zone surrounding the processor of the device, which also includes a small encryption/decryption element. This added element is responsible for decrypting all of the 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 encryption scheme used; however, the program is able to execute normally after decryption [38]. A diagram showing the architecture implemented by Lee et al. is found in Figure 6.

The diagram in Figure 6 demonstrates the creation of an E3 using a processor extension that acts as an intermediary between the processor and memory. By positioning a unit that forwards the memory addresses and enable signals and instructions between the CPU and memory, the authors created an E3 for device applications to execute from, without interfering with the processor design.

9. Comparison of Proposals

In this section, we revisit the trust technologies previously discussed and evaluate them based on the criteria defined in section 2.2. Table 1 shows whether each of the trust technologies satisfies individual criteria or not. The table indicates that no single technology satisfies all of the criteria. However, for device manufacturers and system designers of IoT and CPS deployments it can still be useful, as they can chose what services and assurances they would like to have. Based on these requirements, they can mix and match individual trust technologies from the table 1.

TPM, one of the earliest proposals for trusted computing, has limitations as it is more directed towards static validation of the integrity of device platforms and applications. It has tamper-resistant hardware and the capability for internal

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	✗	✓	✓	✓	✓	✓	✓
Protection Against Bit Faults	✗	❋	✗	✗	✗	✗	✗
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.

and external secure storage. It has limited ability to protect an application against on-device adversaries 2 and 3 (as discussed in section 2.1.1).

In this analysis, we have merged SE and Java Card even though they were discussed separately in the paper. The rationale behind this is that a substantial number of SEs would be supporting Java Card and most the criteria satisfied by SEs and Java Cards are common. One of the major issues with SEs is the lack of end-user access to the device. However, this assists the SE to protect against on-device adversaries, as only authorised entities have access to it, along with third party evaluations of hardware and software platforms.

The next three proposals, Intel TXT, GP-TEE, and Intel SGX, satisfy more or less the same criteria. The only exception is that GP-TEE does not support open access to application developers, who are required to gain prior authorisation from the device manufacturer to use GP-TEE. As all three of these provide strong isolated execution, they protect individual applications from other potentially malicious applications.

The discussion includes HCE to highlight that there is another way to achieve trusted computing, by not executing the security sensitive information on the device in the field. The actual application can execute in a secure location, while the mobile handset supporting HCE might be in the field, potentially in the hands of malicious users. Such an architecture has its benefits, but on the device in the field it does not provide strong security protection.

Finally, the E3 shows a lot of promise, especially for embedded devices, ranging from intellectual property protection to strong binding between software and host hardware, and protection against strong adversaries. In the table 1, we have mentioned that it has tamper-resistant hardware support to reflect the E3 security system shown in figure 6.

10. Conclusion

In this paper, we have discussed the evolution of technologies aiming to provide secure and trusted computing, by discussing the major milestone proposals. Such technologies have extended the basic notion of secure and trusted computing to divergent domains. Application of these technologies is now required to accommodate the imminent emergence of the IoT and CPS. We discussed evaluation criteria for trust technologies and their application in the IoT and CPS. Based on these criteria we evaluated each of the proposals. We showed that no single technology meets all of the criteria. However, IoT and CPS designers and developers can mix-and-match several of these technologies to achieve a desirable combination for their specific deployment. Furthermore, such an analysis is useful as a guide for future research, by indicating where the gaps are in term of 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.

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

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

[51] 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.