seTPM: Towards Flexible Trusted Computing on Mobile Devices based on GlobalPlatform Secure Elements

Sergej Proskurin[1], Michael Weiß[2], and Georg Sigl[1]

[1] Technische Universität München, Munich, Germany
{sergej.proskurin, sigl}@tum.de
[2] Fraunhofer Institut AISEC, Garching, Germany
michael.weiss@aisec.fraunhofer.de

Abstract. Insufficiently protected mobile devices present a ubiquitous threat. Due to severe hardware constraints, such as limited printed circuit board area, hardware-based security as proposed by the Trusted Computing Group is usually not part of mobile devices, yet. We present the design and implementation of seTPM, a secure element based TPM, utilizing Java Card technology. seTPM establishes trust in mobile devices by enabling Trusted Computing based integrity measurement services, such as IMA for Linux. Our prototype emulates TPM functionality on a GlobalPlatform secure element, which allows seamless integration into the Trusted Software Stack of Linux-based mobile operating systems like Android. With our work, we provide a solution to run Trusted Computing based security protocols while supplying a similar security level as provided by hardware TPM chips. In addition, due to the flexible design of the seTPM, we further increase the security level as we are able to selectively replace the outdated SHA-1 hash algorithm of TPM 1.2 specification by the present Keccak algorithm. Further, our architecture comprises hybrid support for the TPM 1.2 and TPM 2.0 specifications to simplify the transition towards the TPM 2.0 standard.

1 Introduction

The shift from the era of primitive mobile devices to full-blown general purpose smart phones significantly affected the importance of mobile security. Becoming omnipresent, the mobile market turned out to be very attractive for cybercriminals: The continuously rising computing power, management of personal information, as well as the potential interoperability with corporate networks and services present an ideal base for various cyber-attacks. Software security hardening solutions, such as antivirus products, try to mitigate the stated issues. However, there is no guarantee that software-based security hardening measures themselves cannot become victims of cyber-attacks and thus be intentionally misused by malware. Because of this, software-based security hardening solutions alone cannot always provide trust.

For this purpose, hardware supported security hardening measures are needed. Therefore, modern embedded systems rely on the concepts of *Trusted Computing* [1] integrated into isolated, hardware supported, environments such as *Trusted Platform Module*s (TPMs) [2, 3] and *Trusted Execution Environment*s (TEEs) [4, 5]. A TPM presents a dedicated microprocessor that is utilized to establish trust between communication partners. It comprises secure storage capabilities for cryptographic keys and cryptographic co-processors to provide reliable integrity measurement and remote attestation services. On the other hand, a TEE is usually part of the main processor and provides an isolated execution environment for *Trusted Application*s (TAs), which can be executed concurrently to a rich operating system. The *Trusted Computing Group* (TCG) and the *GlobalPlatform* jointly elaborated a TPM 2.0 Mobile architecture [6] that can be implemented as a TA [7]. However, TEE-based solutions cannot provide the same isolation as TPMs, e.g., concerning side-channel vulnerabilities. Although a dedicated *Mobile Trusted Module* (MTM) [8] has been specified, it has not yet been accepted within the world of mobile devices. One reason for this is that mobile devices suffer from severe hardware constraints, such as printed circuit board area. Another reason is that usually a dedicated security chip is already deployed in most modern smartphones for other purposes like secure payment.

Our work introduces a TPM implementation for GlobalPlatform specified *secure element*s (SEs) providing flexibility of software- and tamper resistance of hardware supported approaches without the need for additional circuit area. We based our work on the software TPM emulator for Linux [9]. In contrast to this software-only solution, our implementation has been designed to run on Java Card technology [10] based SEs. In short, the seTPM presents a highly portable and tamper resistant TPM emulator that can be easily integrated in modern mobile architectures. Our contributions comprise the following features:

- The seTPM extends the idea of a portable TPM emulator in [9] in a way that it becomes applicable to virtually every platform being able to interface with GlobalPlatform specified secure elements.
- The seTPM software architecture enables transparent integration into Trusted Computing applications like *Integrity Measurement Architecture* (IMA) [11].
- We show that the seTPM can be functionally extended according to individual requirements. Our implementation extends the TPM 1.2 specification [12] on demand, by exchanging the SHA-1 with the Keccak [13] hash algorithm.
- We provide a Keccak implementation for Java Card to further increase the security level compared to modern hardware TPM chips.
- We present an architecture of a hybrid system combining the TPM 1.2 and TPM 2.0 standards.
- Further security enhancement is discussed by facing hardware reset attacks.

The rest of this paper is organized as follows: After providing related work in Section 2, we present background information in Section 3. The architecture and design of seTPM is described in Section 4. We discuss and evaluate our implementation in Section 5. Finally, we conclude the paper in Section 6.

2 Related Work

Strasser and Stamer [9] introduce a software-based TPM emulator. Their implementation presents a Unix-platform independent realization of the TPM 1.2 specification [12] in software, providing a perfectly suitable environment for testing, research, and educational purposes. seTPM adopts the concepts of the TPM emulator within the context of Java Card technology based SEs. Thus, seTPM extends capabilities of the TPM emulator by providing a hardware supported, isolated, execution environment.

Costan et al. [14] introduce a concept called *Trusted Execution Module* (TEM), providing a secure and tamper resistant execution environment in form of a Java Card applet. The execution environment of a TEM comprises a virtual machine executing user provided execution primitives, called closures. In contrast to seTPM, TEM does not focus on system integrity measurement and attestation capabilities but rather provides a general-purpose execution environment that is similar to TEEs of modern processors.

Portable TPM (PTM) has been presented by Zhang et al. [15]. Similar to seTPM, PTM has been implemented for Java Cards providing TPM-like functionality. Unlike seTPM, Zhang et al. follow a user centric approach by binding the PTM to users. This concept establishes trust between users and remote challengers and provides the flexibility of being able to be utilized across different platforms. However, the PTM requires migratable Storage- and Attestation Root Keys, managed by additional TPMs on the respective platforms.

Within the context of multi-application smartcard environments, Akram et al. [16] present the *Trusted Environment and Execution Manager* (TEM) that acts as a TPM for smartcards. Hence, TEM introduces the concepts of Trusted Computing directly into secure elements ensuring trustworthiness of the individual smart card applications.

With vTPM, Berger et al. [17] integrate their own TPM implementation into a Xen-hypervisor providing a software-based trusted environment for virtual machines. This way, Berger et al. establish a flexible multi-context TPM, providing TPM functionality to multiple virtualized environments. Similar to vTPM, given sufficient memory resources, the isolation capabilities of Java Card applets in SEs may manage multiple instances of seTPM, providing similar functionality.

3 Background

To maintain the property of trust, the TCG defines a *Trusted Computing Base* (TCB) to be a part of the system. The TCB comprises so called *roots of trust* that are considered as inherently trustworthy: These are the *Root of Trust for Measurement* (RTM), *Root of Trust for Storage* (RTS) and *Root of Trust for Reporting* (RTR). Usually RTS and RTR are represented by a TPM. A TPM symbolizes a *trust anchor* simplifying detection of potentially malicious code and protection of cryptographic keys from physical theft and distribution.

3.1 TCG Software Architecture

The TCG software architecture provides a modular design that distributes tasks across layers in user- and kernel space (Figure 1). Every layer of the TCG architecture provides an abstract interface towards its upper layer. The user space components, also referred to as the *TCG Software Stack* (TSS), comprise the *TCG Device Driver Library* (TDDL), *TCG Software Stack Core Services* (TCS), *TCG Service Provider* (TSP), and an application making use of these layers. In short, the TPM device driver establishes a communication with the TPM, providing an interface towards the user space layers. The TDDL layer maintains a uniform and operating system independent interface towards the TCS layer. The TCS layer yields core services, such as key-, credential- and context-management. The TSP layer comprises a C-library, utilized by Trusted Computing applications. On Linux-based devices, the open source TrouSerS library [18] provides the TSS of the platform, which is accessed through the *tcsd* user space daemon. As shown in Figure 1, the user space layers of the TSS-2.0 differ entirely from its TSS-1.2 counterpart. The *System API* (SAPI) replaces the TDDL and thus directly interfaces and communicates with the TPM device driver via raw byte commands. The upper layers – *Feature API* and *Extended System API* (ESAPI) – serve as abstraction layers. The *Resource Manger*, *TPM Access Broker* (TAB), and *TPM Command Transmission Interface* (TCTI) layers are optional, transparent, layers providing further features such as the scheduling of multiple TPM sessions and TPMs in user space. For further reading on TSS-2.0, we refer to [19].

3.2 Java Card Technology

The Java Card technology provides a uniform platform, implemented on various SE architectures. Its specification comprises a *Java Card Virtual Machine* (JCVM), a *Java Card Runtime Environment* (JCRE), and a *Java Card API* [10]. These components represent the actual Java Card platform. The JCVM and JCRE enable hardware abstraction and provide a universal API for Java Card applet developers. The communication between the host and the Java Card is established via a message passing based protocol, exchanging *Application Protocol Data Units* (APDU) [20]. An APDU represents a self-contained message

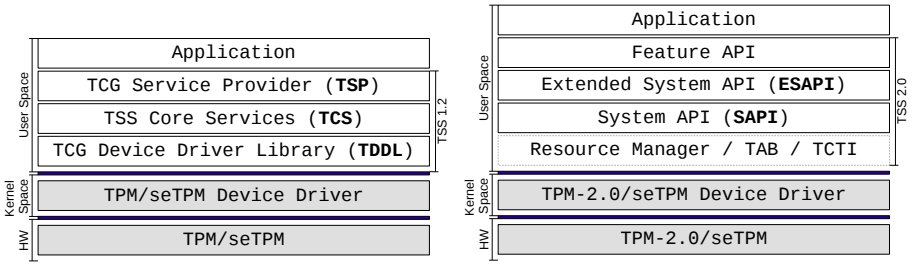


Figure 1: TCG software interfaces and services to communicate with a TPM-1.2 [1] and TPM-2.0 [19]. Highlighted layers present modifications of the TCG architecture.

enclosing either command- or response-information. A command APDU comprises a header and a body. The header contains information, specifying the instruction class and type (`CLA` and `INS`) and instruction parameters (`P1` and `P2`). The body of a command APDU indicates the size of the data to follow (`Lc`) and the maximum number of bytes of the response APDU to receive (`Le`). The Response APDU comprises a body, including the response information, and a trailer, indicating the status of the processed instruction.

Shareable Interface A Java Card application comprises multiple Java Card applets associated with a single package. In other words, applets belonging to the same package share a joint object space, also referred to as a context. The applet firewall enforced by the JCRE strictly constrains the access to data belonging to another context. To permit communication and data sharing throughout the applet firewall and thus between different contexts, applets have to implement the *Shareable* interface of the Java Card API. This way, so called server applets may share selected functionality with chosen client applets from different contexts. To acquire access to this functionality an applet has to request a reference of the particular *Shareable Interface Object* (SIO) through defined system calls.

4 Architecture and Design of seTPM

The idea behind seTPM is a transparent integration into Trusted Computing based applications and the ability to functionally extend the emulated TPM according to individual requirements. These characteristics present an inherent component of the seTPM architecture. Figure 2 gives an overview of our system architecture, which integrates Trusted Computing applications into mobile devices using a GlobalPlatform defined secure element, e.g., in form of a microSD card. Due to its popularity and open source character, our architecture particularly targets mobile devices running the Android operating system. In general, applications communicate with the seTPM by setting off TPM commands. TPM commands traverse the entire TSS before they are forwarded by the Data Transfer Management System to the seTPM applets running on the GlobalPlatform SE. Our architecture introduces a switchable hashing engine allowing to choose between the SHA-1 and the Keccak algorithm. The Keccak algorithm can produce hash values of arbitrary sizes, and thus can be seamlessly integrated into the TPM 1.2 specified protocol. Further, to ease the transition towards the TPM 2.0 standard, we propose a hybrid system supporting both TPM specifications by two interacting seTPM applets residing on the same SE and sharing key resources by means of a server applet.

4.1 Hardware Design

The conceptual hardware design of our project comprises three main components: a *host*, a *card acceptance device* (CAD), and a *secure element*. Within the context of seTPM, the host represents an Android mobile device, such as a smart

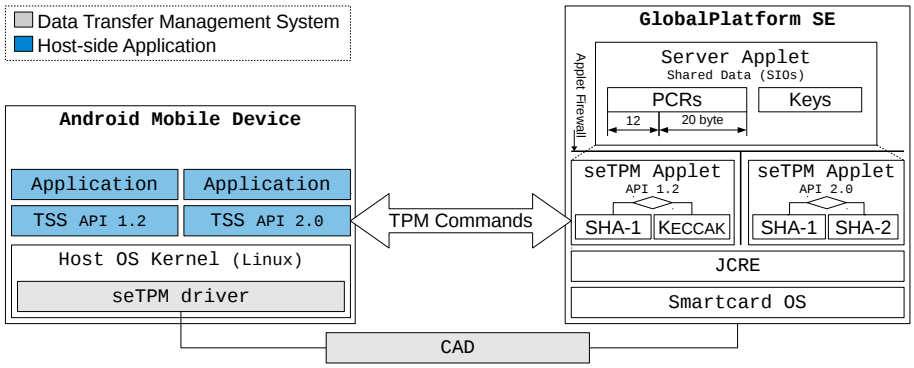


Figure 2: seTPM system architecture

phone or tablet. The employed SE supports Java Card technology with cryptographically relevant functions. The CAD bridges the interface between the host and SE by transparently forwarding requests and replies in form of APDUs. A CAD should be seen as a means for the purpose of provisioning a communication medium and can be realized, e.g., by conventional card readers.

4.2 Software Design

The software design of seTPM has been adopted from the TCG software architecture [1] being presented in Section 3.1. Along with security aspects, seTPM has been designed with flexibility concerning the scope of application and ease of extensibility in mind. To meet these goals, the software architecture of seTPM is divided into three isolated components, which can be incorporated into TCG-compliant systems: These comprise a *host-side application*, a *data transfer management system*, and a *secure element based Trusted Platform Module*. They are described in detail in the following.

Host-side Application The host-side application basically combines all user space TCG components, namely the TSS and the actual application making use of the TSS layers (Figure 1). It is conceivable to extend the TDDL and the SAPI (TSS 2.0), respectively, in order to preprocess and forward APDUs to and from the seTPM device driver. This way, the TDDL/SAPI layer would be able to directly communicate with the seTPM and simultaneously maintain its initial functionality comprising abstraction of the underlying hardware implementation towards its upper layers. However, the kernel should be able to start measuring integrity of the underlying system before userspace is initialized and the associated libraries are loaded. Therefore, we shifted the TCG architecture adjustments to the lower levels residing in kernel space (highlighted layers in Figure 1). Thus, the initial behavior of the user space components of the TCG architecture remains unchanged. This allows to transparently incorporate the seTPM into existing Trusted Computing applications without the need for additional user space libraries.

Secure Element based Trusted Platform Module Our Java Card technology based SE needs to implement the TPM 1.2 [12] and the TPM 2.0 command specification [21] as two isolated applets, each with its own *Application Identifier* (AID). The idea is to share resources, such as *Platform Configuration Register*s (PCRs) and root keys, between both instances using the *Shareable* interface [22] of the underlying Java Card API, as being discussed in Section 3.2. This interface allows to share objects throughout the applet firewall in form of so called SIOs. Our server applet manages the creation of and the access to these SIOs. In terms of compatibility, the size of PCRs must be increased from 20 to 32 bytes. The lower 20 bytes can be used by the TPM 1.2 protocol whereas the upper 12 bytes are additionally utilized by the TPM 2.0 implementation. Thus, TPM 2.0 compliant integrity measurements utilizing the SHA-256 hash algorithm can be assured. In short, this approach meets SE related hardware constraints and additionally allows to support a set of Trusted Computing applications implementing different TPM protocols. For instance, in the event of a legacy trusted boot process, the seTPM may measure integrity and extend its PCR values by applying the original TPM 1.2 functionality. Further integrity measurements may, however, appropriately pad and extend the stored 160 bit values by utilizing the specified SHA-256 hash algorithm. For this, the associated remote attestation party needs to consider this chain of events or simply maintain valid hash values to preserve anonymity. In addition, our design implements an explicit hash algorithm switch inside the TPM 1.2 applet. For legacy/backward compatibility or if not explicitly specified by the host we make use of the SHA-1 algorithm. The host may issue a switch command to change the hash engine to KECCAK at any time by providing owner credentials. However, in that case, a reset is necessary to get consistent PCR states.

Data Transfer Management System The data transfer management system presents a kernel module in form of a device driver. The general idea is to enable a communication between the host-side application and seTPM so that the integrity measurement services provided by seTPM are transparently made available to both, kernel- and user space applications. As a result, not only user space applications but also the operating system kernel can perform integrity measurements, e.g., of kernel modules, security-critical system configuration, and certain applications, even before user space components have been initialized. The seTPM device driver hides details concerning low level communication between the host and seTPM. This way, it is possible to communicate with a particular seTPM via different communication standards, such as PC/SC[3].

For the actual communication, we define a protocol that wraps incoming TPM requests into case-4 command APDU messages, which are then forwarded to the seTPM (left part of Figure 3). To simplify the interpretation of incoming APDUs on the seTPM side, individual TPM request meta information fields are mapped to command APDUs as shown in Table 1. The command APDU entry

[3] Personal Computer/Smart Card (PC/SC) specifies the communication between a host computer and a smartcard.

Table 1: TPM request header to APDU header mapping

APDU header	TPM 1.2	TPM 2.0
CLA	0xB0 | CLA xor 0x01	
INS	TAG[0]	
P1	ORDINAL[3]	TPM_CC[1]
P2	ORDINAL[0]	TPM_CC[0]
Lc	PARAMSIZE[0] | MAX_APDU_PAYLOAD	

Lc, representing APDU size, is limited to $min(\texttt{MAX\_APDU\_PAYLOAD}, 255)$ byte. This limitation is explained by vendor specific maximum APDU sizes of the Java Card. In case a TPM request exceeds this size, the APDU entry CLA signalizes data to follow with a set *chaining bit* – the chaining bit is the least significant bit of the CLA header entry. Chained TPM requests reuse the header of the first transmitted command APDU. The end of a TPM request chain is identified by a clear chaining bit. Besides, only two bytes (MSB and LSB) of ordinals within TPM requests carry significant information. Thus, remaining bytes are left out of the mapping, as shown in Table 1. The same applies to TPM 2.0 specified command codes. Only the first two bytes are relevant as shown in [21]. The format of TPM responses embedded in response APDU messages is visualized in the right part of Figure 3. Received response APDU messages contain TPM responses, which are extracted and subsequently passed to the TDDL/SAPI layer of the TCG architecture (Figure 1). TPM responses exceeding the limit of $min(\texttt{MAX\_APDU\_PAYLOAD}, 256)$ byte are divided into multiple chained response APDUs. Chained response APDU messages are detected by analyzing the actual size of the TPM response embedded within the body of the respective APDU. In case of a chained response, the data transfer management system needs to assemble the complete TPM response before forwarding it to the TDDL or the SAPI layer, respectively.

4.3 Extension of TPM 1.2

The TPM 1.2 specification utilizes the SHA-1 algorithm for hash values and as part of the *Message Authentication Code* (MAC) computation. The SHA-1 cryptographic hash algorithm generates 160-bit (i.e. 20 bytes) hash values from messages of arbitrary sizes. To find a collision using a brute force method, one

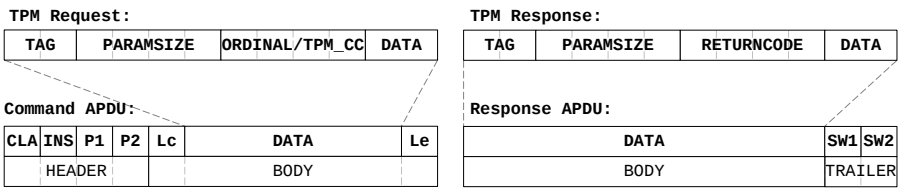


Figure 3: Integration of TPM requests and extraction of TPM responses from APDUs.

would need to calculate hash values of about 2^{80} random messages. However, Wang et al. [23] showed that for the SHA-1 algorithm collisions can be found with complexity of only 2^{69}. To increase the collision-resistance of the SHA-1 hash algorithm and simultaneously to maintain the field lengths of the TPM protocol according to the TPM 1.2 specification, our design introduces the KECCAK hash algorithm [13]. The KECCAK hash algorithm belongs to the *sponge function* family being able to produce hash values of any desired length and is the algorithm behind SHA-3, the latest successor of SHA-1. Because NIST specified SHA-3 by limiting arbitrary hash lengths to four values, namely to 224, 256, 384, and 512 bit [24], we decided to utilize the original KECCAK algorithm to generate hashes of 160 bit and simultaneously provide collision-resistance of the strength of 2^{160}. As specified in [13], the KECCAK sponge function makes use of the parameters KECCAK[*r,c,d*] with *bitrate* r, *capacity* c, and *diversifier* d [25]. The bitrate r presents the block size, meaning the maximum number of bits that are processed at every iteration step. The capacity c can be seen as a security parameter. KECCAK claims to be resistant against collision-attacks with complexity of $2^{c/2}$ and against preimage-attacks with complexity of 2^c although the output length N may be higher. The diversifier d is meant to make two KECCAK instances utilizing equal parameters r and c to produce different results. KECCAK applies so called KECCAK-f permutations on its internal state with the width of $b = r + c$ bit, whereas the state width b is limited to the values $b \in \{25, 50, 100, 200, 400, 800, 1600\}$. The state represents a three-dimensional $5 * 5 * w$ bit array. To achieve best performance, w may be chosen similarly to the word size on the CPU with $w \in \{1, 2, 4, 8, 16, 32, 64\}$ (i.e. 16 bit CPUs should use the parameter $w = 16$). Consequently, seTPM supports KECCAK [*80,320,0*], providing maximum security for the resulting $N = 160$ bit hash values.

4.4 Countering Reset Attacks

Earlier TPM 1.1b [26] specified chips were prone to so called *reset attacks* [27]. In this scenario, the weakest link was given by the *Low Pin Count* (LPC) bus connecting TPMs with the southbridge of their host system. Kauer [27] showed that a TPM can be reset through an induction of the *LRESET#* signal without restarting the system. A subsequent invocation of `TPM_Startup(TPM_CLEAR)` puts the TPM and its PCRs in a default state. Ergo, an attacker can extend the PCRs to reproduce a consistent state despite system integrity manipulations.

The TPM 1.2 specification approached this issue by introducing the concept of *Locality*. Apart from Legacy Locality, this concept classifies execution entities, such as the RTM, trusted OS, and trusted hardware, on the basis of five Locality levels. These levels represent a certain level of trust that is encoded into the address of the LPC start cycle. Thus, upon command reception, a TPM 1.2 specified chip identifies the active level of trust and regulates access, especially reset, to certain PCRs. The Locality level of 4 is applied only within the context of special CPU instructions such as those introduced by Intel *Trusted eXecution Technology* (TXT) and AMD *Secure Virtual Machine* (SVM) (`SENTER`/`SKINIT`).

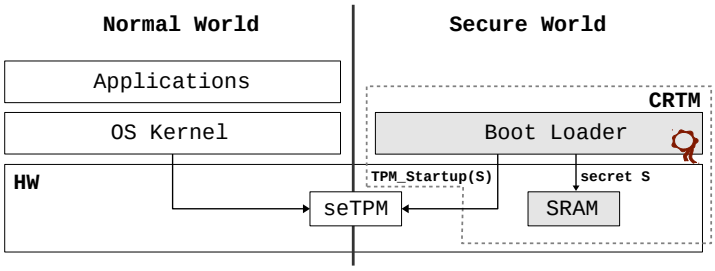


Figure 4: seTPM grants state resets originated only from the CRTM.

Consequently, illegal attempts to reset the associated dynamic PCRs (PCR17-22) by software are detected resulting in a prevention of reset attacks. Yet, Winter et al. [28] manage to hijack the LPC bus to effectively emulate Locality level 4 interactions so that it becomes possible to reset the TPM state.

In mobile devices, SEs in form of SD cards can be physically accessed by potential adversaries and even reset by software at any time. This way, all volatile state of SE applications gets lost. Since the design of seTPM maintains PCR values in persistent memory (EEPROM), a volatile state loss is not a concern. On the other hand, the Java Card framework cannot distinguish between hardware and software card resets. Within the context of seTPM, hardware resets are tolerated. Software induced card resets through special APDUs must be intercepted to prevent reset attacks. This leads to the question how to determine when the `TPM_Startup` command is allowed to reset the PCR values of seTPM.

To face reset attacks on smartcards that cannot distinguish between hard and soft reset, we propose a hardware-assisted solution that can be realized with modern mobile device architectures, such as ARM *TrustZone* (TZ) [29]. A simplified architecture is shown in Figure 4. In general, ARM TZ introduces the concept of two strongly isolated worlds: The *secure* and the *normal world*. The normal world executes a rich *operating system* (OS), whereas the secure world hosts trusted applications. ARM TZ provides among others secure SRAM for key storage. Access to the secure SRAM is regulated by the so called *TrustZone Memory Adapter* (TZMA), whereas the access is granted only when the CPU is in a privileged mode called secure world. We propose to use an immutable, signed, boot loader or locked flash memory as the *Core Root of Trust for Measurement* (CRTM). As shown in Figure 4, the boot loader is placed inside of the high-privileged secure world and starts the OS kernel in the less-privileged normal world. Our design assumes the boot loader to generate a random secret that is stored in the secure, persistent memory. Due to the flexibility of seTPM, we are able to parametrize the `TPM_Startup` command so that the generated secret can be persistently stored inside of seTPM. Section 4.5 discusses the individual steps required for a secure secret transfer. During the seTPM initialization, the `TPM_Startup` handler of seTPM determines when the TPM state must be reset.

1. The initial setup of seTPM is determined by a missing secret. In this case, the `TPM_Startup` command initializes the secret on the SE and sets its state to default values, respectively.
2. Further setup attempts compare the maintained secret with the provided secret value of the `TPM_Startup` command. In case the compared values match, the seTPM continues setting its state to default values. Otherwise, the command is aborted keeping the old PCR values.

This method assures that only the boot loader is able to reset the seTPM state. Reset attack attempts could be reported through custom seTPM commands.

4.5 Secure System Initialization

As being discussed in Section 4.4, it is the task of the immutable boot loader to generate a secret that is shared between the mobile host and seTPM. To prevent potential abuse, the secret must be maintained on both sides in a secure way so that it cannot be accessed by adversaries. Thus, on the mobile host side, the secret is stored within the secure SRAM that can be accessed only from the secure world. The seTPM side provides tamper resistant capabilities preventing illegal access. Yet, the communication channel between the host and seTPM can be attacked. Thus, the secret must be delivered in a way that provides integrity, confidentiality, and prevents eavesdropped messages from being replayed.

As shown in Figure 5, to provide integrity and confidentiality, the bootloader must encrypt the generated secret and its hash value with a public key of seTPM. For this, the bootloader can make use of a crypto co-processor that is often part of the ARM architecture. Since the bootloader is signed, the cryptographic functionality must already be part of the code base and can be reused for our purposes. Replay attacks can be prevented by interpreting the secret value as a counter. After initial transfer of the secret towards seTPM, the associated `TPM_Startup` handler will reset the TPM state only on reception of the value secret+1. In case the adversary manages to inject a crafted `TPM_Startup` request into the communication channel before the host, she could take over control of seTPM. Because of this, we suggest to bind seTPM to the associated mobile device as part of a device pairing process in a physically secured environment.

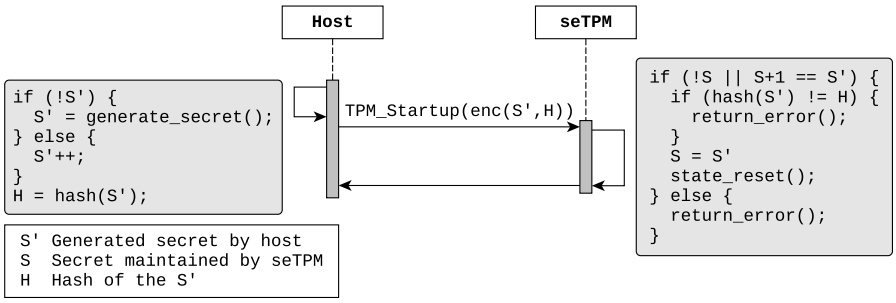


Figure 5: Tasks performed by the mobile host and seTPM on `TPM_Startup` invocation.

Table 2: TPM commands supported by seTPM.

Index	TPM Command	Index	TPM Command
1	TPM_OIAP	11	TPM_TakeOwnership
2	TPM_OSAP	12	TPM_Init
3	TPM_GetCapability	13	TPM_Startup
4	TPM_ReadPubek	14	TPM_CreateWrapKey
5	TPM_OwnerReadInternalPub	15	TPM_LoadKey
6	TPM_GetRandom	16	TPM_Seal
7	TPM_FlushSpecific	17	TPM_Unseal
8	TPM_PcrRead	18	TPM_SelfTestFull
9	TPM_Extend	19	TPM_ContinueSelfTest
10	TPM_OwnerClear	20	TPM_GetTestResult

5 Prototype Implementation

The software implementation of seTPM comprises a host-side application, a data transfer management system, and a Java Card applet representing the seTPM. The host-side application may represent any Trusted Computing application utilizing the services provided by a conventional TPM. TrouSerS [18] provides a set of *tpm-tools* which enable the use of basic TPM services on Linux-based systems. These tpm-tools employ the TrouSerS library providing the TSS (Figure 1). To demonstrate the seamless integration of seTPM into Trusted Computing environments, we utilized tpm-tools to provide a TSS 1.2 compliant implementation of the host-side application. As being discussed in Section 4.2, the data transfer management system manages the communication between the host-side application and the seTPM. This part is implemented as a device driver maintaining a PC/SC connection with the SE. The last bit of our implementation concerns the seTPM Java Card applet providing TPM 1.2 functionality. Our current implementation supports 20 commands as specified by [12]. The supported commands are listed in Table 2. Every TPM command processes incoming data and subsequently generates a TPM response. If the TPM response exceeds the maximum number of bytes that can be embedded into a response APDU, the message is split up into chunks. Thus, the host's data transfer management system becomes responsible to request the remaining chunks of the TPM response.

5.1 Challenges

For our prototype implementation, we had to cope with various challenges which are discussed in the following. Severe resource constraints of SEs generally present one significant difficulty. As a result, the use of SEs with higher resource capacities may solve issues that might have affected earlier implementations.

Memory Management Java Card technology based secure elements utilize RAM and EEPROM for different purposes. Volatile RAM is used for temporary computations, whereas EEPROM is used to persistently store code and data of Java Card applets. Dynamic object construction allocates memory in EEPROM.

However, access to EEPROM is typically about 30 times slower than access to volatile data in RAM. Due to the limited amount of RAM and dramatically slow access of dynamically allocated memory in EEPROM, our implementation handles temporary values in RAM and stores only the data that must survive power loss in EEPROM. Yet, the Java Card technology does not support allocation of arbitrary data structures in RAM but rather manages data in form of transient byte arrays. Therefore, the state of the seTPM is managed by dedicated wrappers maintaining data structure elements in transient byte arrays. This way, temporary computations of arbitrary data structure elements can be performed in RAM, thus limiting the number of accesses to EEPROM.

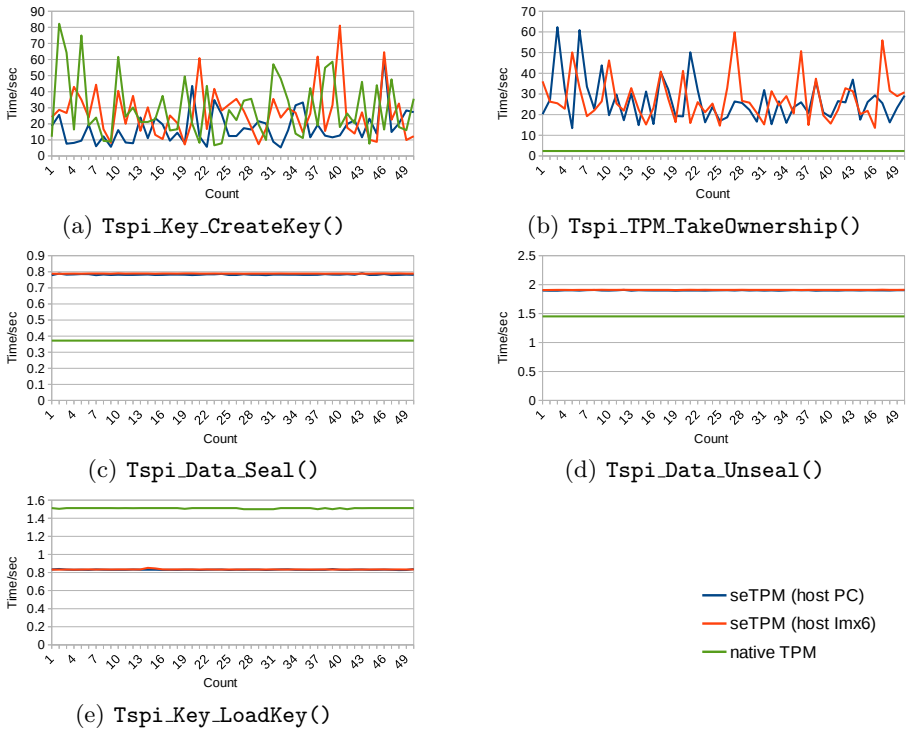
Key Encryption The TPM command `TPM_Create_WrapKey` initiates the generation of an asymmetric key pair that can be stored as an encrypted blob on the host side. To bind the key to the TPM, the TPM specification 1.2 [12] states that the newly created key pair should be encrypted by means of the non-migratable *Storage Root Key* (SRK). Therefore, essential parameters, which are used to restore the key at a later point in time, are encrypted with the SRK and sent to the host. However, the asymmetric key representation in CRT form used by the Java Card API requires more than 1024 bytes (the size of the utilized asymmetric key length). Since the asymmetric encryption is not capable of encrypting more bytes than the asymmetric key size, our implementation shifts towards a merged architecture combining the TPM 1.2 and TPM 2.0 specifications: The TPM 2.0 specification supports the encryption of asymmetric keys by means of a symmetric encryption algorithm (AES). As a consequence, keys of various lengths can be encrypted. Therefore, we introduce an additional non-migratable symmetric *Key Encryption Key* (KEK), which is generated during the installation process. As discussed in Section 4, our future implementation of a TPM 2.0 compliant seTPM applet can share the KEK with its TPM 1.2 counterpart.

Garbage Collection Automatic memory reclamation is usually not part of the JCRE. To prophylactically prevent running out of memory at run-time, all objects need to be allocated during the applet installation and initialization stage. Thus, the allocated objects need to be made globally available to the rest of the implementation. Static object instances would meet this requirement. However, to additionally ensure uniqueness of these instances, we have decided to apply the Singleton pattern to globally used objects.

Stack Frames Method invocations and utilization of try-catch blocks create overhead concerning memory consumption through the dynamic generation of stack- and exception frames. This needs to be considered during implementation as the number of invoked methods may lead to memory shortage.

5.2 Evaluation

We evaluated our prototype by measuring the execution time of five of the most critical TPM commands from two different host systems (Freescale i.MX6 development board and a Lenovo Thinkpad X220) and compared the results with



(a) `Tspi_Key_CreateKey()` (b) `Tspi_TPM_TakeOwnership()`

(c) `Tspi_Data_Seal()` (d) `Tspi_Data_Unseal()`

(e) `Tspi_Key_LoadKey()`

Figure 6: Evaluation of TPM 1.2 commands. The high-level TSPI commands internally invoke the associated TPM commands.

performance measurements of a TPM natively built into a notebook (Lenovo Thinkpad X220). Our measurements consider the time required to traverse the entire TCG stack beginning from the TSP interface (TSPI) down to the seTPM/TPM (Figure 1). Therefore, we measured the time needed for invocations of the TSPI functions. The utilized SE implements the Java Card specification 2.2.2 and contains 4 kB of volatile RAM and about 80 kB of non-volatile EEPROM. The communication is established through a conventional card reader with PC/SC support. The following presents our evaluation results. However, the data transfer overhead caused, e.g., by a slow card reader needs to be considered: An internal card reader might result in a smaller transmission overhead.

- Figure 6(a) presents a non linear timing behavior in both seTPM setups and the native TPM. One explanation is that the `TPM_CreateWrapKey` command generates an asymmetric key pair on the seTPM/TPM and hence needs to perform primality tests. A random number that is not a prime must be rejected and regenerated.
- The invocation of the `TPM_TakeOwnership` command as well indicates a non-linear timing behavior of both seTPM setups in Figure 6(b). Interestingly,

Table 3: Mean time in ms of evaluated commands.

TSPI Command	**seTPM (PC)**	**seTPM (i.MX6)**	**TPM**
`Tspi_TPM_TakeOwnership`	26555,50	27845,42	2399,94
`Tspi_Key_CreateKey`	17667,39	26894,09	29689,12
`Tspi_Key_LoadKey`	831,10	833,84	1509,60
`Tspi_Data_Seal`	782,27	788,77	371,90
`Tspi_Data_Unseal`	1900,04	1909,89	1451,96
TPM Command	**seTPM (PC)**	**seTPM (i.MX6)**	**TPM**
`TPM_Extend (SHA-1)`	55,69	55,76	11,79
`TPM_Extend (Keccak)`	5775,32	5810,56	–

the native TPM takes constant time to perform the command. The seTPM creates an asymmetric SRK on every `TPM_TakeOwnership` invocation. Thus, seTPM needs to perform primality tests, as described above. We assume that the native TPM computes the SRK in advance, right after the TPM has been reset, thus reducing the execution time.

- The execution of the command `TPM_LoadKey` performs better in both seTPM setups than the native TPM, as shown in Figure 6(e). The reason for this is that symmetric cryptography is faster than asymmetric cryptography. As being discussed in Section 5.1, our prototype implements a modified version of `TPM_CreateWrapKey`: seTPM generates an asymmetric key pair, which is encrypted with the symmetric KEK instead of the asymmetric SRK. Ergo, `TPM_LoadKey` on the seTPM requires less time to decrypt the wrapped key.
- In both seTPM setups, the TPM commands `TPM_Seal` and `TPM_Unseal` perform slower than the native TPM, as shown in Figure 6(c) and Figure 6(d). Naturally, this is the accumulation of various factors, such as the bus transmission rate and different CPU rates of the native TPM and seTPM.

The mean time of the evaluated TSPI commands is shown in the first part of the Table 3. The table opposes the performance of the native TPM and the two seTPM implementations with different hosts. Except for `Tspi_Key_CreateKey()`, the table shows a minor variation between both seTPM setups, indicating a host-independent behavior. Consequently, our prototype can achieve similar results even on very resource-constrained devices.

We conclude our evaluation with a short demonstration of the functional extensibility of seTPM: The `TPM_Extend` command is responsible for updating specified PCR register values with integrity measurements provided by the host. This process concatenates a PCR register value with the provided measurement and subsequently generates a SHA-1 hash over the concatenated string. The resulted hash value is stored within the specified PCR register: $PCR_{new} = SHA1(PCR_{old}||measurement)$. The second part of the Table 3 presents the mean time required for the `TPM_Extend` command on both seTPM implementations as well as on the native TPM. One can observe that our implementation performs slightly slower that the native TPM. However, according to the data transfer overhead of contactless secure elements, our implementation requires about 40 *ms* to transfer 25-50 byte to and back from the seTPM. Consequently,

our implementation would be able to almost keep up with the performance of a native TPM, when connected, e.g., through an internal card reader. In addition, Table 3 shows performance results of the `TPM_Extend` command using the KECCAK hash algorithm. Obviously, a software implementation cannot keep up with a hardware accelerated SHA-1 implementation. However, this shows that our goal concerning functional extensibility has been met.

6 Conclusion

In this paper, we presented the architecture and design of seTPM, a secure element based Trusted Platform Module. To face current compatibility issues, we introduced an architecture of a hybrid system combining the TPM 1.2 and TPM 2.0 standards, the concepts of which may be implemented in the future. Our prototype implementation comprises 20 of the most important TCG specified TPM 1.2 commands as a framework that can be easily extended to finally introduce hardware supported Trusted Computing to mobile devices. The architecture dependent parts of seTPM have been implemented for Linux-based systems and can be easily ported to Android. We showed with our implementation that a seamless integration into the widely deployed TSS implementation TrouSerS is feasible. Further, we provided a proof-of-concept that our design is more flexible than hardware TPM chips by exchanging the SHA-1 hash algorithm with the KECCAK algorithm. With that approach we were also able to increase the security level compared to native TPM solutions. Our evaluation showed that seTPM performs similar to a native TPM. Although some TPM commands perform slower on the seTPM in comparison to a native TPM, it still presents an attractive solution for mobile devices. In summary, we believe that seTPM is capable of eliminating lots of today's issues concerning trust in mobile devices as our approach closes the gap between Trusted Computing and GlobalPlatform specified SEs of modern smart phones.

References

1. Trusted Computing Group, "TCG Specification Architecture Overview Specification, Revision 1.4," August 2007.
2. D. Challener, K. Yoder, R. Catherman, D. Safford, and L. Van Doorn, *A practical guide to trusted computing*. Pearson Education, 2007.
3. W. Arthur, D. Challener, and K. Goldman, *A Practical Guide to TPM 2.0*. Springer, 2015.
4. GlobalPlatform Inc., "TEE System Architecture – Public Release v1.0," GlobalPlatform Inc., December 2011.
5. *ARM Security Technology - Building a Secure System using TrustZone Technology*, Prd29-genc-009492c ed., ARM Limited, April 2009.
6. Trusted Computing Group, "TPM 2.0 Mobile Reference Architecture Family "2.0", Level 00 Revision 142," December 2014.
7. ——, "TPM MOBILE with Trusted Execution Environment for Comprehensive Mobile Device Security," 2012.

8. ——, "TCG Mobile Trusted Module Specification Version 1.0, Revision 6," June 2008.
9. M. Strasser and H. Stamer, "A software-based trusted platform module emulator," in *Trusted Computing-Challenges and Applications*. Springer, 2008, pp. 33–47.
10. Oracle, "Java Card Platform Specification 2.2.2."
11. R. Sailer, X. Zhang, T. Jaeger, and L. Van Doorn, "Design and Implementation of a TCG-based Integrity Measurement Architecture," in *USENIX Security Symposium*, vol. 13, 2004, pp. 223–238.
12. Trusted Computing Group, "TPM Main Specification Level 2 Version 1.2, Revision 116," March 2011.
13. G. Bertoni, J. Daemen, M. Peeters, and G. Van Assche, "Keccak specifications version 2," 2009.
14. V. Costan, L. F. Sarmenta, M. Dijk, and S. Devadas, "The trusted execution module: Commodity general-purpose trusted computing," in *Proceedings of the 8th IFIP WG 8.8/11.2 International Conference on Smart Card Research and Advanced Applications*, ser. CARDIS '08. Berlin, Heidelberg: Springer-Verlag, 2008, pp. 133–148.
15. D. Zhang, Z. Han, and G. Yan, "A Portable TPM Based on USB Key," in *Proceedings of the 17th ACM Conference on Computer and Communications Security*, ser. CCS '10. New York, NY, USA: ACM, 2010, pp. 750–752.
16. R. Akram, K. Markantonakis, and K. Mayes, "Trusted platform module for smart cards," in *New Technologies, Mobility and Security (NTMS), 2014 6th International Conference on*, March 2014, pp. 1–5.
17. S. Berger, R. Cáceres, K. A. Goldman, R. Perez, R. Sailer, and L. van Doorn, "vtpm: Virtualizing the trusted platform module," in *Proceedings of the 15th Conference on USENIX Security Symposium - Volume 15*, ser. USENIX-SS'06. Berkeley, CA, USA: USENIX Association, 2006.
18. "TrouSerS, The open-source TCG Software Stack."
19. Trusted Computing Group, "TSS System Level API and TPM Command Transmission Interface Specification Family "2.0", Level 00 Revision 01.00," January 2015.
20. ISO, "Identification cards – Integrated circuit cards – Part 4: Organization, security and commands for interchange," International Organization for Standardization, Geneva, Switzerland, ISO ISO/IEC 7816-4:2005, 2005.
21. Trusted Computing Group, "TPM Library Specification Family "2.0", Level 00, Revision 01.16," 2014.
22. M. Montgomery and K. Krishna, "Secure object sharing in java card," in *Proceedings of the USENIX Workshop on Smartcard Technology*, ser. WOST'99. Berkeley, CA, USA: USENIX Association, 1999, pp. 14–14.
23. X. Wang, Y. Yin, and H. Yu, "Finding Collisions in the Full SHA-1," in *Advances in Cryptology – CRYPTO 2005*, ser. Lecture Notes in Computer Science, V. Shoup, Ed. Springer Berlin Heidelberg, 2005, vol. 3621, pp. 17–36.
24. *SHA-3 standard: Permutation-based hash and extendable-output functions*, National Institute of Standards and Technology Std., Rev. DRAFT FIPS PUB 202, May 2014.
25. G. Bertoni, J. Daemen, M. Peeters, and G. Van Assche, "Keccak, note on parameters and usage," February 2010.
26. Trusted Computing Platform Alliance, "TCPA Main Specification Version 1.1b," February 2002.

27. B. Kauer, "Oslo: Improving the security of trusted computing," in *Proceedings of 16th USENIX Security Symposium on USENIX Security Symposium*, ser. SS'07. Berkeley, CA, USA: USENIX Association, 2007, pp. 16:1–16:9.
28. J. Winter and K. Dietrich, "A Hijacker's Guide to the LPC Bus," in *Proceedings of the 8th European Conference on Public Key Infrastructures, Services, and Applications*, ser. EuroPKI'11. Berlin, Heidelberg: Springer-Verlag, 2012, pp. 176–193.
29. ARM, *Designing with TrustZone® – Hardware Requirements.*