simTPM: User-centric TPM for Mobile Devices

Dhiman Chakraborty, *CISPA Helmholtz Center for Information Security, Saarland University;*
Lucjan Hanzlik, *CISPA Helmholtz Center for Information Security, Stanford University;*
Sven Bugiel, *CISPA Helmholtz Center for Information Security*

https://www.usenix.org/conference/usenixsecurity19/presentation/chakraborty

This paper is included in the Proceedings of the 28th USENIX Security Symposium.

August 14–16, 2019 • Santa Clara, CA, USA

978-1-939133-06-9

Open access to the Proceedings of the 28th USENIX Security Symposium is sponsored by USENIX.

simTPM: User-centric TPM for Mobile Devices

Dhiman Chakraborty
CISPA Helmholtz Center for Information Security, Saarland University

Lucjan Hanzlik
CISPA Helmholtz Center for Information Security, Stanford University

Sven Bugiel
CISPA Helmholtz Center for Information Security

Abstract

Trusted Platform Modules are valuable building blocks for security solutions and have also been recognized as beneficial for security on mobile platforms, like smartphones and tablets. However, strict space, cost, and power constraints of mobile devices prohibit an implementation as dedicated on-board chip and the incumbent implementations are software TPMs protected by Trusted Execution Environments.

In this paper, we present simTPM, an alternative implementation of a mobile TPM based on the SIM card available in mobile platforms. We solve the technical challenge of implementing a TPM2.0 in the resource-constrained SIM card environment and integrate our simTPM into the secure boot chain of the ARM Trusted Firmware on a HiKey960 reference board. Most notably, we address the challenge of how a removable TPM can be bound to the host device's root of trust for measurement. As such, our solution not only provides a mobile TPM that avoids additional hardware while using a dedicated, strongly protected environment, but also offers promising synergies with co-existing TEE-based TPMs. In particular, simTPM offers a user-centric trusted module. Using performance benchmarks, we show that our simTPM has competitive speed with a reported TEE-based TPM and a hardware-based TPM.

1 Introduction

Trusted computing technology has become a valuable building block for security solutions. The most widely deployed form of trusted computing on end-consumer devices is the Trusted Platform Module (TPM), a dedicated hardware chip that offers facilities for crypto co-processing, protected credentials, secure storage, or even the attestation of its host platform's state. By today, software and system vendors have built various security solutions on top of TPM. For instance, Microsoft's BitLocker uses it to release disk-encryption credentials only to a trustworthy bootloader [49]; or Google's Chromium uses the TPM for a range of objectives [60], such as preventing software version rollback, protecting RSA keys, or attesting protected keys.

TPM is also of interest for the different stakeholders on mobile devices. However, the particular benefits that the TPM offers have historically hung on the TPM's implementation as a dedicated security chip that can act as a "local trusted third party" on devices. Mobile devices are, however, constrained in space, cost, and power consumption, which prohibits a classical deployment of TPM. To address the particular problems of the mobile domain, the Trusted Computing Group (TCG) introduced the Mobile Trusted Module (MTM) specifications [61]. Although the MTM concept has never left the prototype status, its ideas influenced the latest TPM2.0 specification [64]. The TPM2.0 mobile reference architecture [63] proposed different alternatives for implementing a TPM on a mobile device, including virtualization, dedicated cores, or hardware-based isolation. The de-facto implementation of mobile TPMs today are protected environments through hardware-based trusted execution environment (TEE) [23, 24, 32, 45, 46, 54], like ARM TrustZone that is available on virtually all mobile platforms today, where the TPM is implemented as protected software application inside the TEE.

Given the different proposals for realizing TPMs on mobile platforms, we conduct a systematic comparison of the different solutions in terms of security of the TPM itself, their applicability in current systems, and deploy-ability in the specific setting of mobile devices. While the solutions naturally differ in their security guarantees for the TPM (i.e., TPM state or execution) due to differences in the underlying technology (e.g., dedicated hardware chip vs. virtual machine), we see particularly shortcomings of the current solutions in terms of applicability and deploy-ability. In particular, the currently incumbent fTPM (firmware TPM) is strictly bound to the platform vendors and serves their purposes (e.g., securing vendor credentials), but is not or only very limited available to other stakeholders in the system, such as the user. Moreover, an fTPM [54] that is based on a TEE falls short on providing a fully measured boot by itself. The availability of an fTPM

depends on the availability of the TEE during boot, which is one of the last steps in the long boot-chain. In light of recent attacks against mobile bootloaders [55] and trusted software in TEE [3, 8, 18, 39, 41, 51, 56–58], this lacking support to attest the entire, early boot-chain, including the software in the TEE, is unsatisfactory.

To put a new perspective on solving those issues of fTPM, we add in this paper an alternative implementation of a hardware TPM called *simTPM* to the landscape of mobile TPM implementations by using the subscriber identity module (SIM) card. We have implemented a prototype of our solution on a Hikey960 reference board [1] and using a Gemalto Multos card as SIM card. Our simTPM solves the technical challenge of implementing TPM2.0 compliant functionality on the SIM card, which does not require any additional hardware for the TPM. This approach keeps the costs down and leverages dormant hardware capabilities of mobile devices. Through performance tests, we show that simTPM is competitively fast to reported fTPM implementations. A particular challenge of this design is the lack of the usual physical binding between the TPM and its host platform's root of trust for measurement (RTM), that is, a SIM card can be moved to another platform. We discuss two strategies in the particular setting of mobile devices on how to bind the simTPM to a device's RTM, either through an extended secure boot and TEE proxy or through a distance bounding protocol. Once bound to the device's RTM, we also integrated simTPM with the ARM Trusted Firmware (ATF) boot chain to augment the ATF secure boot with an authenticated boot. Our solution not only fills the gap of TEE-based TPMs for measured boots, but the co-existence of a fTPM and simTPM on a mobile device creates also promising synergies between the two TPMs (e.g., to support multiple stakeholders). Our contribution can be summarized as follows:

1. A systematic comparison of existing solutions for mobile TPMs and their enabling technologies. We discover that incumbent solutions fall short on applicability and deploy-ability aspects.

2. We implemented the first SIM card based TPM2.0 for mobile devices by developing a simTPM, which can be executed in this constrained environment. Our solution enables a user-centric trusted module offering a portable sealed storage.

3. We propose an integration with the on-board TEE to solve the problem of binding the simTPM to the RTM and discuss an alternative solution based on distance bounding. As a result of this binding, a fully measured boot on the ARM Trusted Firmware (ATF) secure boot chain is possible.

4. The performance of our simTPM is competitively fast to a reported fTPM implementation and is comparable with existing hardware TPMs.

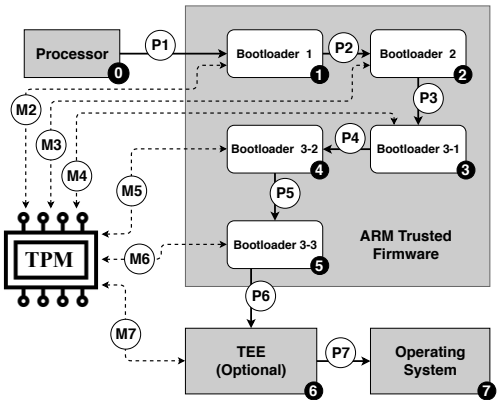

Figure 1: Trusted Boot Process with TPM; P(#) = boot chain path; M(#) = measurement of component #

2 Background

We briefly introduce necessary background information about ARM Trusted Firmware, TPM, and SIM cards.

2.1 ARM Trusted Firmware (ATF)

ATF implements a subset of the trusted board boot requirements for ARM reference platform [5]. Figure 1 illustrates the bootloader settings and boot chain. ATF is triggered when the platform is powered on. After the primary CPU and all other CPU cores are initialized successfully, the primary core triggers the ATF (P1). ATF is divided in five steps depending on modularity: ❶ BootLoader stage 1 (BL1) for AP trusted boot ROM, ❷ BootLoader stage 2 (BL2) for Trusted Boot Firmware, ❸ BootLoader stage 3-1 (BL3-1) for EL3 Runtime Firmware, ❹ BootLoader stage 3-2 (BL3-2) for Secure-EL1 Payload (optional), ❺ BootLoader stage 3-3 (BL3-3) for Non-trusted Firmware.

Secure boot: ATF implements a secure boot in which every component along the boot chain (P#) verifies the authenticity and integrity of the next component. Since BL1 does not have a preceding component, it has to be axiomatically trusted. Thus, BL1 verifies BL2, BL2 verifies BL3.x, and so forth. Verification is usually based on certificates, where a hash of a trusted (vendor) public key is fused into the hardware and is available to BL1 to ensure a trustworthy signature of BL2. At the end of a successful secure boot, every component in the boot chain has been checked for integrity and authenticity before handing control to it. If any verification fails, the boot aborts.

2.2 Trusted Platform Module (TPM)

TPM by the Trusted Computing Group is the most widespread trusted computing technology on end-user devices. By today, the TPM specification is in its version 2.0, addressing

many of the security issues and practical concerns of previous versions 1.0–1.2. According to this specification, a TPM provides a number of desirable hardware and security features. It is equipped with secure non-volatile memory, a set of platform configuration register (PCR) banks, a processor to run TPM code in isolation, co-processors for common cryptographic primitives (e.g., RSA, ECC, SHA-1, SHA-256), a clock, and a random number generator. By default, a TPM is deployed as a hardware chip soldered onto a platform's motherboard. Besides acting as a cryptographic co-processor, a TPM provides the facilities to securely store measurement about the host platform's configuration (e.g., software state) in its PCRs and to reliably report those measurements to a remote verifier (remote attestation based on a pre-installed endorsement key), as well as creating secure storage through TPM protected credentials and data sealing with extended authorization policies. Further, the TPM non-volatile memory, including secure monotonic counters, can be attractive for building security solutions, e.g., version rollback prevention for software updates.

By now, a number of real world applications make use of TPM. For instance, IBM's password manager uses it for storing keys, Microsoft windows management instrumentation uses TPM for cryptographic co-processing, Intel's Trusted eXecution Technology or AMD's Secure Technology rely on a hardware TPM, several VPN apps can make use of it, TPM is used in full disk encryption (e.g., Microsoft Bitlocker, dm-crypt), and even browsers like Chrome make use of TPM for different purposes.

Measured boot: Of particular relevance for this paper is measured (or authenticated) boot based on TPM (see Figure 1). During a measured boot, every component in the boot chain (P#) measures the next component—a cryptographic hash of the component—and then stores this measurement in the PCR of the TPM ((M#)) before passing on control. Since BL1 does not have a preceding component, it is not measured and acts as the *Root of Trust for Measurement*, which starts the measurement chain. In contrast to a secure boot, the components are not verified and the boot is not aborted, however, after a measured boot the software configuration of the boot components can be attested by the TPM or used to seal storage to this configuration (i.e., values in the PCR).

2.3 Subscriber Identification Module (SIM)

SIM card is the module that authenticates the mobile device in the network. The primary job of the SIM card is to prove the identity of the owner of subscription to the cellular carrier to enable services like calling, Internet, and various others.

Through physically separated pins, a SIM module can achieve the same degree of independence from power supply, reset capability, clock signal, and separated I/O communication with the host platform like a TPM.

Since SIM cards are smart cards, they use command-response communication and the application protocol data unit (APDU) to communicate with their reader. The Android radio interface layer can be extended to send specialized APDU commands to the SIM card, which we use in simTPM. It is worth noting, that this APDU command sent by the Android radio interface has to go through the baseband processor. The structure of the APDU commands are defined in the ISO/IEC 7816-4 standard and are recalled later on in Section 4.

3 Requirement Analysis & Systematization of Existing Solutions

There exists many approaches to realize TPM in a way different than using a dedicated hardware TPM. In this section, we systematically compare different solutions of trusted computing procedures using both hardware and software that are representative for the different implementation options. For comparison, we first re-enumerate the objectives a secure and practical TPM implementation needs to fulfill (Section 3.1) and then discuss the existing solutions (Sections 3.2 through 3.4). In particular, this systematization should help to understand the trade-off of the proposed solutions in comparison to the default hardware TPM and where our simTPM solution fits into. Table 1 summarizes the discussion in the remainder of this section.

3.1 Objectives

We start by briefly formulating the objectives a trusted module, in particular for mobile devices, should fulfill. We group them into security of the TPM itself, the applicability of the implementation, and desirable deploy-ability objectives.

3.1.1 Security of TPM

These are objectives that should be fulfilled to ensure the security of the TPM state, its execution and trustworthiness, and secure operations.

S1 Confidentiality and integrity of TPM state: The TPM state should be confidential and protected against untrusted code (e.g., host platform, non-TEE apps) and only be available to authorized entities. We assign ✔ if the confidentiality and integrity of the state is protected through strong security means (e.g., physical isolation), ✪ if they depend on software integrity (e.g., of the OS), and ✘ in other cases.

S2 Rollback Protection: Reverting the TPM state back to a former version must be prevented or at least be detectable. We assign ✔ if rollback protection is guaranteed through hardware means (e.g., hardware counters), ✪ if there is a dependency on untrusted OS but rollbacks can be detected, ✘ if no rollback protection or detection is provided.

S3 Trustworthy Endorsement: A TPM should be carrying an asymmetric encryption key called Endorsement key (EK) that can live as long as the TPM and for which credentials exist that verify the authenticity of the TPM and allow a verifier to recognize a genuine TPM. We assign ✔ if endorsement credentials are available to the TPM (e.g., pre-installed at manufacturing time or derived from other verifiable credentials), ✪ if the TPM has to create an EK and prove it is genuine through a remote verification, ✘ otherwise.

S4 Secure Counter: TPM has to provide secure, persistent monotonic counters, e.g., for its clients or extended authorization policies. We assign ✔ if the TPM provides such counters backed by hardware support or NV-storage of the TPM software state that is protected (i.e., **S1, S2** both ✔). We assign ✪ if the security of the counter depends on software integrity (e.g., of the OS or hypervisor). Otherwise ✘.

S5 Secure Clock: A clock is needed for attestation, for generation of timed attestation keys, and for authorization policies with lock-out time. If a secure clock is available to the TPM (e.g., its own hardware clock), we assign ✔; if the clock depends on shared resources but manipulation can be detected we assign ✪, otherwise ✘.

S6 Security of TPM Execution: The execution of the TPM code or firmware has to be protected against compromise. We assign ✔ if a strong security boundary exists between untrusted code and the TPM execution environment (e.g., dedicated physical chip). If the execution environment shares hardware resources (e.g., CPU or RAM) with untrusted code and the shared resources provide isolation (e.g., modes of operation of CPU and separate memory regions), we assign ✪, since the shared resources open an attack surface. If the security of the TPM execution environment is based purely on software means (e.g., hypervisor or OS), we assign ✘ for this weakest form of isolation.

3.1.2 Applicability

These are objectives related to the application of TPM, such as authenticated boot or providing secure storage to clients.

A1 Secure Persistent Storage: TPM provides a persistent storage to securely store limited amounts of data (e.g., certificates). We assign ✔ if the TPM provides such storage (e.g., NV-RAM in a dedicated chip) and ✪ if the persistent storage is part of an outsourced TPM state that is protected (i.e., **S1, S2** both ✔). We assign ✘ in other cases.

A2 Early Availability: A main use-case for TPM is storing the measurement of loaded software components, i.e., measured boot. To be able to attest the entire software stack, the TPM has to be early available during the boot sequence. If the trusted module is available as soon as the platform has power, we assign ✔. Otherwise, if the TPM becomes available at late stage during boot (e.g., after initializing a separate execution environment), we assign ✘.

A3 Multiple Stakeholders: Computer systems, in particular mobile platforms and enterprise devices, usually have multiple stakeholders co-existing with an interest in protecting credentials and software on the platform (e.g., end-user, administrator, network operator, software vendor). If the TPM was designed to support both platform software and users (e.g., distinct hierarchies), we assign ✔. If the TPM primarily supports the platform but offers limited functionality to the end-user, we give ✪. If the TPM was designed solely as support for the platform vendor, we give ✘.

3.1.3 Deploy-ability

Objectives related to the deployment of TPM, in particular if deployment complies with the requirements of mobile devices or if it is bound to a specific platform.

D1 Mobile Availability: We want to have the TPM available for mobile devices. This imposes strict constraints, such as not changing the current architecture by adding a new on-board chip. If the TPM implementation adheres to this constraints, we assign ✔, otherwise ✘.

D2 Movability: The TCG specification has introduced the TPM as being bound to its host platform (e.g., fixed part of the motherboard). However, depending on the context, the movability of the TPM to another platform is desirable, e.g., if an associated virtual machine migrates to another platform. If the TPM is generally easily moved to another platform, we assign ✔, if it is bound to a specific platform, we assign ✘.

D3 Bound RTM: The measurements during a measured boot are given to the TPM by the host platform, starting with the Root of Trust for Measurement (RTM). To ensure that the provided measurements indeed describe the TPM's host platform's configuration, TPM and RTM must be bound together on the same platform. If this binding is achieved via physical means (e.g., TPM and RTM are fixed parts of the same motherboard), we assign ✔. If the TPM receives those measurements from another trusted entity (e.g., another, bound TPM, or a secure boot anchored at the RTM), we assign ✪. If the TPM cannot establish trust into the RTM, we assign ✘.

In Section 2.2, while introducing the hardware TPM, we explained all its properties, which allow the TPM to achieve the objectives we defined in Section 3.1 and summarized in Table 1. Objectives **S1** to **S6** and **A1** to **A3** are our interpretation of properties derived from TCG's mobile TPM [61, 63] and standard TPM specification [62, 64]. We define *Deploy-ability* as added objectives that simTPM should achieve. The current TCG specifications do not stipulate a removable TPM. We will use the standard hardware TPM as the baseline that simTPM should achieve.

Table 1: Comparison of existing TPM implementations

Category		Objective	fTPM [54]	vTPM† [9]	Intel SGX [19]	simTPM	Hardware TPM
Security of TPM	Security of TPM state	**S1.** Confidentiality and integrity	✔	✔/ ✪	✲	✔	✔
		S2. Rollback protection	✔	✔/ ✪	✔	✔	✔
		S3. Trustworthy Endorsement	✔	✪/ ✪	✔	✔	✔
		S4. Secure counter	✔	✔/ ✪	✔	✔	✔
		S5. Secure clock	✪	✔/ ✘	✘	✔	✔
		S6. Security of TPM execution	✪	✔/ ✘	✲	✔	✔
Applicability		**A1.** Secure persistent storage	✪	✔/ ✘	✪	✔	✔
		A2. Early availability	✘	✔/ ✔	✘	✔	✔
		A3. Multiple stake holder	✘	✔/ ✔	✔	✔	✔
Deploy-ability		**D1.** Mobile availability	✔	✘/ ✘	✘	✔	✘
		D2. Movability	✘	✔/ ✔	✘	✔	✘
		D3. Bound RTM	✔	✘/ ✪	✔	✪	✔

✔ = fulfilled by the implementation; ✪ = partially fulfilled by the implementation; ✘ = not fulfilled by the implementation; ✲ = not applicable for the implementation

† First column is for *Secure co-processor based vTPM* (SCoP) implementation and second column is for *Software only vTPM* (SW-only) implementation

3.2 fTPM

Specifically for the mobile domain, a number of past implementations [25, 54, 66] leveraged trusted execution environments (TEE) to realize a software-based TPM. We use Microsoft's fTPM [54] as a representative for those implementations, since it is one of the most recent solutions. The fTPM implementation is widely deployed in Microsoft mobile devices using a TEE on top of ARM TrustZone (**D1:** ✔). TrustZone creates a memory and process isolation between the protected environment ("secure world") running inside the TEE and the "normal world" (i.e., Android or similar), and allows the execution to switch contexts between those two worlds via a secure monitor.

fTPM provides confidentiality, integrity (**S1:** ✔), and rollback protection (**S2:** ✔) for fTPM states by creating a trusted storage through a combination of encryption with fused keys, device UUID, and Replay Protected Memory Block (RPMB) with authenticated writes and write counter. Any form of secure persistent storage the fTPM offers to clients is based on this securely outsourced state (**A1:** ✪), which is also used to provide secure counters to clients (**S4:** ✔).

Due to ARM TrustZone, the execution of the fTPM environment is isolated from the normal world, however, both worlds still share the CPU and RAM (**S6:** ✪), which has opened TrustZone TEEs to attacks (e.g., [41]).

fTPM does not have a separate secure clock. It uses the clock of the system in cooperation with the untrusted OS (**S5:** ✪). To handle the shared clock situation, fTPM implements *fate sharing*, where fTPM refuses to provide any functionality if the OS does not cooperate.

fTPM is primarily designed to provide TPM support to the platform vendor (**A3:** ✘). The fTPM is a software implementation and bound to one device (**D2:** ✘), since it derives many of its credentials from device-specific keys or UUIDs, including its endorsement credentials (**S3:** ✔).

Since the fTPM is implemented as software in the TEE on top of ARM TrustZone, the fTPM becomes only available once the TEE has been initialized during the boot sequence (see also Section 2). That means the fTPM (or any TEE-based TPM) is not early enough available to store measurements of the early boot stages (**A2:** ✘). But this can be alleviated by introducing shared memory between the bootloaders and TEE for measurement storage. We will discuss this solution in more details in Section 4.3.

Although the fTPM is only available after the bootchain has created the TEE, the secure boot transitively extends the trust put into the RTM (BL1) to the remainder of the secure bootchain on the same platform as the TEE. Thus, fTPM can assume that the measurements are done as if by the RTM on the same platform (**D3:** ✔) if the measurements comes from a component of the secure bootchain.

3.3 vTPM

Another way of implementing a software TPM is by creating virtual instances over a physical TPM [9]. This, in particular, targets cloud environments in which virtual machines need a TPM, but sharing a single physical TPM (or providing an array

of physical TPM) is not an option. The representative work for virtual TPM, or vTPM, is based on the Xen hypervisor and proposes two different implementation options: 1) a software only implementation with vTPM instances running inside a privileged VM, and 2) a secure co-processor (SCoP) to run all vTPM instances with better isolation at the cost of additional hardware. Both options are not feasible for mobile TPMs (**D1:** ✘), since virtualization is not sufficiently supported or effective, and adding a secure co-processor is too costly in terms of space and power. However, by design vTPMs must be movable to different platforms to support migration of associated VMs between platforms (**D2:** ✔).

In both deployment options, a vTPM has to create its endorsement key at creation time. To establish trust into the EK for a remote verifier, a genuine, primary TPM on the platform (hardware TPM) must attest the trustworthiness of the vTPM's EK (**S3:** ✪).

In case of SCoP-vTPM, the TPM logic and vTPM instances are executed inside the secure co-processor (**S6|SCoP-vTPM:** ✔). Further, the secure co-processor used in [9] (an IBM PCIXCC) provides CMOS RAM backed persistent storage. We assume it provides the confidentiality, integrity, and rollback protection of the vTPM states as well as sufficient secure persistent storage to the vTPM clients (**S1, S2, A1|SCoP-vTPM**: ✔). The same co-processor also offers facilities for secure counters (**S4|SCoP-vTPM:** ✔) and a secure clock (**S5|SCoP-vTPM:** ✔).

For SW-only-vTPM the vTPM instances reside in Xen's privileged *dom0*. Thus, their execution is protected from untrusted VMs by only the Xen hypervisor (**S6|SW-only-vTPM:** ✘), and their state, when stored in persistent storage in *dom0*, is also protected by only the access control and isolation of the hypervisor and *dom0* (**S1, S2|SW-only-vTPM:** ✪). Similar, the protection of any persistent storage offered to vTPM clients depends on the integrity and trustworthiness of *dom0* (**A1|SW-only-vTPM:** ✘) as does any counter stored in the vTPM state (**S4|SCoP-vTPM:** ✪). A vTPM relies on the platform's clock shared between all vTPMs including untrusted code and not specifically protected (**S5|SCoP-vTPM:** ✘). Although vTPM instances are created after the host platform has booted up, a vTPM receives the initial measurement from the underlying hardware TPM of its platform, which also attests the vTPM trustworthiness, and *dom0* protects the vTPM state from migrating to an untrusted platform (**D3|SW-only-vTPM:** ✪). Further, vTPM instances are created together with their associated VM, hence, allowing the VM to measure its entire bootchain and store the measurements in its vTPM (**A2:** ✔).

In case of SCoP-vTPM, the TPM resides entirely in the IBM PCIXCC, a removable peripheral. Thus, no physical binding to the RTM exists and no authenticity/trustworthiness of the RTM is being ensured (**D3|SCoP-vTPM:** ✘), hence, an attacker could move the TPM to an untrusted platform that feeds the TPM with arbitrary measurements. This situation is very similar to our simTPM, which is also removable, and we discuss solutions to this challenge in Section 4.3, which might also be applicable to SCoP-vTPM.

The vTPM does not make any assumptions about which stakeholder—user or platform—within the associated VM uses the vTPM and supports, like a regular hardware TPM, multiple hierarchies (**A3:** ✔).

3.4 Intel SGX

Although Intel SGX is not an implementation of a TPM but a solution to allow applications to establish a TEE, *enclave* in SGX jargon (**S1, S7:** ✲), we include it here for comparison because it offers in many dimensions similar protections as a hardware TPM and shares a lot of a TPM's objectives (we mark non-applicable objectives with ✲ in Table 1). For this work we have only considered stock SGX implementations in Intel processor to keep the comparison on par with other candidates. SGX is currently only supported by desktop and server class Intel processors (**D1:** ✘) and binds any credentials, like generated and derived keys, and transitively sealed data strictly to the CPU (**D2:** ✘).

In SGX, *attestation* means verifying that a certain enclave code was initialized correctly and not tampered with by the untrusted host OS. For *remote attestation* in SGX an Intel-provided *Quoting Enclave* provides the facilities to enclaves to do direct anonymous attestation (DAA) using attestation keys endorsed by Intel (**S3**: ✔). The SGX extensions to the CPU measure the enclaves, hence, the enclaves are physically bound to their RTM (**D3**: ✔).

SGX supports enclaves in sealing data for storing it on untrusted persistent storage, since enclaves themselves do not have any persistent storage like NV-RAM (**A1:** ✪). In addition, Intel has added support for monotonic counters [30, 43] that allow rollback protection of sealed data (**S4, S2:** ✔).

It is a processor based technology, so it can fully utilize the clock of the system. But the current SGX implementation does not accommodate a trusted and fine grained clock for the user-level enclaves. There is an API provided by Intel, e.g., `get_trusted_time`, but this call can be arbitrarily modified by the untrusted OS, since it requires to make an *OCALL* [4, 6, 17, 31, 37]. Moreover, any timing mechanism must account for the fact that the OS can interrupt the enclave at any point in its execution, wait for an arbitrary period of time, and then resume the enclave using *ERESUME* (**S5:** ✘).

Both regular applications and system software can use enclaves and SGX is not restricted to particular stakeholders (**A3:** ✔). However, an early firmware initialized enclave is not possible, since the OS is needed for memory management of enclaves (**A2:** ✘).

3.5 Java-card based MTM

Dietrich and Winter proposed a way of implementing a mobile trusted module (MTM) in a Java-based smart-card for mobile devices [21, 26]. The implementation is for applications running on mobiles and the TPM communicates through NFC.

The TPM is installed as a set of applets in the Java-card, where a *master-applet* provides services to other applets, like TPM command handling and controls the access to the endorsement key. The actual processing of TPM commands is handled by specific applets implementing those commands.

Although this implementation seems like closest related work to our simTPM, their work described a proof-of-concept prototype and is unfortunately silent about many aspects, such as secure persistent storage, and some functionality is not available, such as attestation of the system or authenticated boot. The Java-card communicates with the system over NFC, so binding the card with the system is not possible and early availability of the trusted module is also not possible before the NFC driver is loaded.

Their implementation provides important insights on the implementation of MTM on mobile devices through a programmable TPM and presented pioneering work, but given the lack of documentation and also differences in engineering (see Section 4), we cannot provide a full and fair comparison with simTPM and exclude it from our systematization.

4 System Design and Security Analysis

The main component of simTPM is a smart card based implementation of a SIM TPM. However, to properly work it also requires changes in the bootloader and the operating system (i.e., Android). In this section, we describe the design and implementation of simTPM in more details. We also discuss how our solution solves the shortcomings described in Section 3 and argue about our design's security. Along with the design descriptions, we indicate how the objectives shown in Table 1 are met by simTPM.

4.1 SIM TPM

Modern SIM cards are usually general purpose smart cards running an applet created by the mobile network provider. The two most prominent smart card technologies are Java Cards and Multos cards. Both introduce a custom OS (i.e., Java Card OS and Multos OS) and APIs that can be used by programmers for cryptographic (e.g., encryption, signing) and non-cryptographic (e.g., memory allocation and copy) operations that are implemented and executed directly on the microprocessor. Depending on the technology, applets can be programmed in C/C++ (e.g., Multos cards) or in Java (e.g., Java Card). Additional cryptographic algorithms, not provided by the API, can be implemented in software.

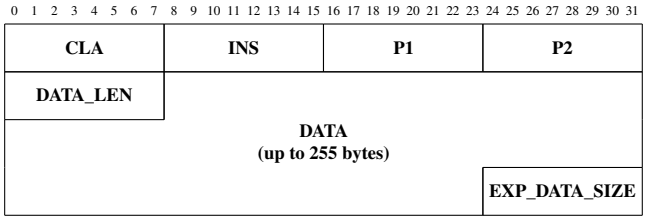

Figure 2: Generic APDU command structure

Both card technologies have support for multiple applets. To properly manage them, cards provide a specialized security manager that is responsible for installing and deleting of user defined applets. Once an applet is uploaded, the security manager creates its instance and allows the applet to create necessary objects and allocate memory.

4.1.1 API Limitations of Smart Cards

As mentioned above, each Smart Card OS provides a card specific API that allows applets to perform extended operations. This forces the programmer to use only a predefined set of functions. For example, in case of Java cards the API supports only a subset of the standard Java language and is limited to high level cryptographic operations (e.g., encryption, hashing, signing). There is no support for mathematical functions like modular multiplication or elliptic curve point addition, which are one of the main building blocks of public key cryptography. In other words, the developer cannot use hardware support for those low-level operations and is limited to software implementations that are inefficient due to the overhead of the virtualization layer.

Obviously, those limitation do not directly concern TPM commands that only use basic cryptographic operations. Unfortunately, the TPM standard defines a remote attestation scheme that is not supported by the cards API, because it uses, e.g., zero-knowledge proofs. This constitutes an interesting engineering problem that we solve. In particular, we were able to implement simTPM on a Gemalto MultiApp Multos smart card with an Infineon SLE78CLX family microprocessor. This card also helped us achieving process isolation from the general-purpose processor (**S6:** ✔). It is worth noting, that in this paper we focused mainly on the Multos API [42], because it supports a broader range of functions than the Java card API. In particular, we were able to efficiently implement a remote attestation scheme on-card.

4.1.2 Smart Cards and TPM Command Parsing

SIM cards are connected to the main processing unit over a separate bus and available for mobile telephony services (**D1:** ✔). Smart cards work in a command/response manner, i.e., given an input the card executes the code and returns a response. The input data is defined by an APDU command

(see Figure 2), which consists of a class byte (CLA), an instruction byte (INS), two bytes for parameters (P1, P2), one byte for the expected response length, one byte for the data length (DATA_LEN), and DATA_LEN bytes of data. The cards' response contains the response data and two bytes that constitute the status word (not shown in the figure). The data field is limited to 255 bytes. There exist an extended length APDU specification that allows for a larger data field but it is not widely implemented.

In a multi applet system, an APDU command will be forwarded to the currently selected applet. To select an applet, the `SELECT` APDU command with an unique applet identifier has to be sent to the card. This command is then recognized and executed by the OS. Once selected, the applet can parse incoming commands according to its work flow. In particular, this means that the developer can use the instruction and parameters bytes to program the behavior of the card.

The APDU data structure provides a convenient way to communicate with the card. We designed a custom APDU command that implements TPM commands. The data length size of up to 255 bytes is sufficient for the payload sizes of most TPM commands, and for TPM commands with larger payload sizes (e.g., sealed data blobs), we send the payload split across multiple APDU messages and use the parameter bytes to communicate the card if more data is to be expected.

Changes to Android's radio interface: The Android Radio interface layer (RIL) is responsible for communicating with the device's SIM card. To allow RIL to communicate with simTPM, we introduced a set of TPM commands. We implemented a custom RIL as a shared library, which sends APDU commands as bulk transfer to the simTPM and receives its responses.

4.1.3 TPM Commands

We now briefly discuss how we designed the card to handle basic TPM commands related to PCR banks and sealing. The former case is easy, the applet reserves enough non-volatile memory to store the PCRs. The number of banks is defined by the installation parameter of the TPM applet, which also defines the algorithm we use to extend the PCR (e.g., `SHA1` or `SHA256`). In a standard setup we use 24 PCRs. For the `TPM_EXTEND` and `TPM_READ` commands we used two separate instruction bytes (respectively, `0x10` and `0x20`) to form the APDU. In both cases the number of the PCRs is given using parameter P1.

To design (un-)sealing on a smart card was a bit harder. Due to the limited input data size, the card has to encrypt/decrypt the input in chunks, which are split across multiple APDU messages. The storage key for sealing is generated by the card after receiving the `TPM_INIT` command. The key is stored in the non-volatile memory that is allocated during installation of the applet (see next Section 4.1.4).

It is worth noting that smart cards can be programmed to execute all TPM commands that require basic cryptographic algorithms, on-card key generation, key agreement, or storing data in volatile/non-volatile memory. Unfortunately, the privacy-preserving variant of remote attestation (i.e., direct anonymous attestation, DAA) requires zero-knowledge proofs and other unsupported crypto operations. What is more, in versions below TPM 2.0 the specification defined only one algorithm for anonymous attestation [14], which is based on groups with hidden order (i.e., using a RSA modulus) and Camenisch-Lysyanskaya signatures. The TPM 2.0 specification, however, allows for algorithm agility. We leveraged this fact and used a custom scheme. We present the full scheme and security proofs in the technical report of our paper [16]. Here, we only draft the idea behind the scheme, which follows the generic approach used by other DAA schemes: The TPM receives a signature/certificate under its secret DAA key from an authority. It then uses this secret key to certify its attestation key using a proof. In this zero-knowledge proof the TPM shows that it knows a certificate under a DAA key and a signature created using this key under an attestation key. The scheme uses Boneh and Boyen [11] signatures and an efficient zero-knowledge proof for the above statement that is made non-interactive using the Fiat-Shamir transformation [28]. The main advantage of the scheme is that it can be executed solely by the TPM (i.e., on-card) and does not require any involvement of the host platform. To further improve efficiency of our scheme, we decided to optimize the workload between commands, i.e., if the `TPM_CREATE` command recognizes that the TPM is creating an attestation key, it already does some pre-computation for the DAA certification.

4.1.4 PCR and NV storage

All smart cards implement a small amount of non-volatile storage that can be used for various purposes. This memory of the smart card is by design tamper-resistant and therefore offers memory isolation from the rest of the system (**S6:** ✔). Modification of this memory is only possible by the applet that reserved it and we reserve some of the NV storage for the simTPM (**A1:** ✔). Smart cards are equipped with features preventing updates of its internal state by the outside world. To update stored content (e.g., applets), one has to issue an authorized command to the card manager to update storage or perform applet specific commands, e.g., PCR extension (**S1:** ✔). Our simTPM is equipped with PCR banks that are initialized when power cycling the device and, hence, the SIM card, and can only be changed between power cycles using `PCR_EXTEND`.

System software or user level software can keep a counter containing the current version of the software inside the NV-storage and updates to the counter are only allowed via authorized commands. This provides an easy setup for secure counter and rollback protection (**S4:** ✔).

4.1.5 Trustworthy endorsement & Clock

Trustworthy endorsement of a TPM is very important. The standard solution is to use an asymmetric encryption key called endorsement key. This key is unique per TPM and should stay alive as long as the TPM is alive. This key differentiates a genuine from a rogue TPM. simTPM can achieve secure endorsement by putting a (vendor) certified endorsement key inside its NV-storage and implementing TPM logic that ensures that the private portion of the key is never released to the outside world (**S3:** ✔).

SIM cards are equipped with a clock pin connected to the baseband processor. Thus, they cannot be clocked higher or lower by an untrusted application or OS. This separate clock helps simTPM to work on a different clock frequency not under direct influence of the main processor. What is more, the baseband processor can be used as a secure external clock. In particular, since the baseband processor is by default isolated with a strong security boundary from untrusted code on the platform, it can prepend any APDU command with an APDU command containing the current time (this can also be limited to time-sensitive TPM commands only). This way simTPM can be provided with a secure clock (**S5:** ✔).

4.1.6 Movability & Stakeholders

The other unique feature of the simTPM architecture is its movability (**D2:** ✔). simTPM implements the TPM inside the SIM card. So by design, simTPM can be transferred to a different device. This creates some interesting use-cases, which we discuss in more details in Section 6.2, but also challenges, which we discuss separately in Section 4.3. simTPM is not specifically bound to one particular stakeholder and supports the multiple stakeholder model proposed by TCG (**A3:** ✔), although we think the end-users and their apps are the primary beneficiaries of simTPM.

4.2 ATF boot-loader changes

In Section 2.1, we have briefly introduced ATF and its boot-loader chains. In this section we describe the changes we have implemented to enable communication between the bootloader components and the simTPM. Figure 1 can be helpful as a visual aid for understanding.

After turning on the secondary cores on the cold boot path, the processor kicks in the first stage BL1 of the bootloader (❶). Current bootloaders are not implemented such as to be able to communicate with a device like a SIM card and to run a command response protocol. Thus, we have extended all the boot-loaders with the capability to communicate with the SIM card via bus communication. This modification in ATF makes the simTPM already available to the early BL1 stage (**A2:** ✔). The bootloader software is capable of translating TPM commands to APDU commands, sending them to simTPM, receiving responses, and translating them to a meaningful response that can be used to make decisions (e.g., failed/successful PCR extension commands). One thing that needed to be addressed here is that except for BL3-3, all bootloaders are secure mode software (i.e., secure world in TrustZone). So during execution, simTPM has to be initialized as secure mode hardware to be available to the bootloader. We initialize the simTPM as a secure mode hardware, but after a successful boot chain verification, we switch simTPM to normal mode (of TrustZone). This allows us to maintain normal efficiency in the normal world, since the SIM card functionality (e.g., calls or text messages) is accessed by Android and switching context from normal world to secure world every time before accessing the SIM card in Android can interrupt the normal world execution and would be highly inefficient.

4.3 Bootstrapping trust for movable simTPM

Parno [53] was first to identify the problem of how to bootstrap trust into a hardware TPM and the possibility of *cuckoo attacks*. A fundamental problem of TPM is that the verifier (e.g., local user) does not know if they are talking to the *intended* (e.g., local) TPM, just that they are talking to *a genuine* TPM. In a cuckoo attack, an attacker that compromised the local platform can exploit this problem and fool the verifier into trusting the compromised platform: the attacker simply relays the verifier's communication to another (remote) TPM on an attacker-controlled platform, which then can attest an arbitrary, trustworthy state to the verifier. The preferred solutions to prevent cuckoo attacks are hardwired channels via a special purpose hardware interface to the on-board TPM or, alternatively, a cryptographically secured verifier-TPM communication where the verifier has knowledge of the public key of the TPM on the intended platform.

However, those solutions make an implicit assumption: Historically TPMs are soldered onto the motherboard, eliminating the issue of ensuring proper binding to the device's root of trust of measurement (RTM), usually in form of an immutable piece of trusted code in the BIOS. Due to this static design a TPM is ensured that the very first received measurement in a chain-of-trust is coming from a trusted, local RTM. Only a sophisticated hardware attack can break this binding. A TPM that is by-design movable, such as our simTPM or the PCI-attached secure co-processor for vTPM [9], raises an interesting question about how to re-establish this bond between TPM and RTM.

Lack of chain-of-trust: Without binding the TPM to a trusted, local RTM, the measurements of any authenticated boot cannot be trusted. An adversary could simply plug the simTPM into an attacker-controlled platform and replay[1] any desired measurements sequence, i.e., create arbitrary PCR values akin to a TPM reset attack [29, 36]. This allows the

[1] The TPM is a passive device to which the measurements have to be provided by its caller.

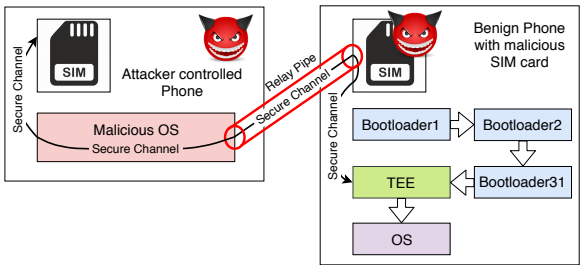

Figure 3: Using TEE as TPM proxy to bind simTPM with RTM and to mitigate the effects of relay attacks.

attacker to fool a remote verifier during remote attestation but also to gain access to sealed secrets, whose release is bound to the platform state (i.e., PCR values).

Binding simTPM and RTM: To create a binding between the simTPM and a trusted, local RTM, we need the simTPM to 1) authenticate the RTM to ensure its a trusted code (e.g., BL1 of ATF); and to 2) ensure policies (e.g., for data release) and commands (e.g., attestation) are only executed for exactly the platform for which the simTPM stores the measurements. To address those challenges, we identified two possible solutions, using the device's TEE as a proxy to the simTPM (see below) or using distance bounding protocols (discussion deferred to the technical report of our implementation [16]).

Using TEE as TPM proxy: One way to bind the simTPM with the device's RTM is by leveraging the platform security building blocks of mobile devices and using the TEE as a proxy to simTPM (see Figure 3). On a genuine device with secure boot in place, i.e., BL1 as a trusted RTM, the TEE has exclusive access to device-specific credentials that are certified by the device vendor. Using those credentials, the simTPM and TEE can establish a secure end-to-end channel. In this setup, simTPM will only respond to PCR extensions, attestation requests, or unsealing of encrypted data if the commands come via this secure channel. As a result, an attacker cannot forge arbitrary PCR values without compromising the device-specific key. Further, if the TPM enforces a particular device key, it can ensure that only the intended platform is using the simTPM; however, even without this strict set of device keys, this solution still ensures that any TPM commands, such as releasing data to the host platform, can only come from a genuine mobile platform with an intact secure boot from which it received the measurements. Considering previously mentioned software-based attacks against TEE (see Section 3.2), an attacker could compromise the TEE to steal the device-specific key and impersonate the TEE to the simTPM. This can be alleviated by using session keys instead of the long-term secret device-specific key for communication between TEE and simTPM, which could be setup during the bootstrapping and, hence, before untrusted code can attack the TEE. A drawback of this solution is that the simTPM requires the TEE to be bootstrapped to become itself operational, which prevents an early availability of the simTPM. Since the simTPM is not early available in this setup, ATF's secure boot has to be extended to store the measurements of verified software components and pass those measurements on to the TEE, which then can forward them to the simTPM via the secured channel (**D3:** ✪). It should be noted that while this extension to ATF would also provide a solution to the early availability of fTPM [54], simTPM gives a user-centric solution and additional interesting use-cases in comparison to fTPM (see Section 6). We discuss an alternative solution based on a distance bounding protocol in Appendix A.

4.4 Security analysis

Lastly, we analyze the security of simTPM in comparison to the closest solutions fTPM and hardware TPM, specifically considering the deployment of our TPM on a SIM card.

Off-chip protection: As mentioned in Section 3, fTPM depends on the integrity of the secure world, which has been under attack recently [3, 8, 18, 41, 51, 55–57]. Our simTPM implements an off-board TPM on the SIM card and, like a discrete TPM, is physically isolated from untrusted code. This provides a stronger protection of the simTPM's trusted computing base, however, we cannot fully exclude potential software attacks against the SIM card software. For instance, in the past smart cards have exhibited bugs [50] like hidden commands, buffer overflows, weaknesses of cryptographic protocols [52], or malicious applets [52]. Further, like a hardware TPM, simTPM is connected via a bus, which makes it prone to advanced bus attacks [12, 34, 36] that, however, are considered outside the attacker model for consumer grade hardware like the TPM.

SIM card cloning: Deployment on a SIM card also raises the concern of card cloning [65], which could easily enable impersonation attacks or theft of credentials. However, driven by the interests of telecommunication companies, modern SIM cards come with anti-cloning defenses that mitigate this attack vector [42].

SIM swapping attack: In SIM swap attack, an attacker obtains details about the victim and then tricks the telephony company to port the victim's phone number to a fraudulent SIM card owned by the attacker, usually with the goal to receive all SMS including highly sensitive information, like OTP for online banking. In our design, the TPM is not dependent on the SIM telephony functionalities. simTPM works as a local co-processor with desirable attributes. An attacker can port the telephony services to a fraudulent SIM card, but not the TPM state, as it is bound to the local SIM-card and would require explicit migration policies to other (SIM)TPM.

Side-channel attacks: To be compliant with the TPM 2.0 specification, the hardware has to implement cryptographic functions that are resilient to timing-based side-channel attacks. There exists a similar requirement for smart cards, which are designed to be resistant against various types of side-channel attacks. Thus, simTPM immediately benefits from the security features of the underlying smart card.

However, a motivated attacker can easily move simTPM to a controlled environment and mount different active side-channel attacks, such as clock frequency, heat measurement, probing [33], fault injection [35], or power analysis [40,47,48]. While similar attacks have been shown against ARM TrustZone (e.g., [39,58]) and discrete TPM chips [59], deploying the TPM on a removable card might ease mounting those attacks. Nevertheless, it should be noted that such sophisticated hardware attacks are not only strenuous, exorbitant, and inconsistent, but also beyond the protection that a consumer grade security chip can offer.

5 Performance Evaluation

We evaluate the performance of simTPM on a HiKey960 board in comparison with a hardware TPM. We focus on the most frequent commands executed by a TPM, i.e., key generation, sealing/unsealing of data, extending/reading a PCR, generating random bytes, and computing a hash value of an input. Beside simTPM we prepared two test setups equipped with an Infineon SLB 9670 TPM chip. One of these two test benches is a plug-able TPM on a Raspberry-Pi (*piTPM*) and the other one is an embedded TPM on a standard Lenovo laptop (*embTPM*). More details about the setups is given in [16]. We have used a TSS implementation by IBM [2] to communicate with the Infineon TPM. The results of our benchmarks are summarized in Figure 4. All results come from 50 measurements per command per device. We report the 95% confidence intervals.

5.1 Test cases and results

Key generation: We measured the time to generate a 256-bit ECC key and output the public part of the key. Our implementation of simTPM creates the key on average in $257 \pm 8.03ms$, comparable to the piTPM performance ($253 \pm 1.25ms$), but slower than the embTPM ($172 \pm 0.61ms$).

Create hash: We measured the time it takes for the TPM to hash 256 bits of input data with SHA-256 and output the digest. piTPM ($50 \pm 0.76ms$) and embTPM ($21 \pm 0.16ms$) outperform the simTPM ($72 \pm 10.13ms$) by a factor of 1.44 and 3.42, respectively.

Extending and reading a PCR: We evaluated the PCR extend and read commands. The former allows to extend the PCR with a new value, while the latter command is used to read the current value of a PCR. We use SHA-256 as hash algorithm and a 128 bit string as input value. For PCR extension, simTPM ($24 \pm 2.66ms$) is on par with embTPM ($21 \pm 0.11ms$), however, exhibits a higher instability of the performance. For reading PCRs, simTPM ($15 \pm 0.15ms$) is the fastest implementation, followed by embTPM ($21 \pm 0.13ms$). piTPM is the slowest implementation in both cases ($41 \pm 1.22ms$ and $57 \pm 2.58ms$) and exhibits an unstable performance, too.

Sealing and unsealing data: The TPM seal command takes a byte array, attaches a policy, encrypts it with a TPM storage key, and returns a blob to the caller. When unsealing, the TPM takes an encrypted blob, checks the policy, and decrypts the blob if the policy is satisfied by the TPM state. For our performance measurement we used 128 bits input data, a 256-bit ECC sealing key with ECIES, and an empty policy. The embTPM is the fatest solution for sealing and unsealing ($130 \pm 0.27ms$ and $89 \pm 0.46ms$) and outperforms our simTPM ($588 \pm 18.55ms$ and $376 \pm 22.30ms$) by a factor of 4.52 and 4.22, respectively.

Random number generation: We use the TPM to generate a 64 bit random number. Our simTPM is the fastest solution ($15 \pm 0.14ms$), followed by embTPM ($21 \pm 0.17ms$) and then piTPM ($63 \pm 1.63ms$).

5.2 Discussion of performance

Our test results show that there is no clear winner among our test systems. simTPM as well as embTPM excel for some commands and we would argue that our simTPM prototype shows a competitive performance. Unfortunately, the implementation of Infineon SLB 9670 TPM is not publicly available, thus commenting on the exact reasons for those differences would result in speculations. If we would venture to speculate, potential reasons for the differences could be the different communication buses. embTPM has a dedicated bus communication with the onboard processor and a faster processor, while piTPM is running on a Raspberry Pi and is connected over GPIO with lower bandwidth. On the other hand, simTPM is connected through the USB bus. Moreover, our simTPM implementation uses only the publicly available APIs of the smart card OS, which provide only an indirect access to hardware level commands. Hence, a vendor-supported implementation with direct access to the microprocessor would improve in efficiency.

The fTPM is unfortunately not available, precluding a direct comparison in our test suite; however, our observations for the embTPM speed are comparable to those reported by Raj et al. [54], although it is unclear which hardware TPM they evaluated. An fTPM, unsurprisingly, outperforms any other tested implementation here—e.g., slowest fTPM in [54]

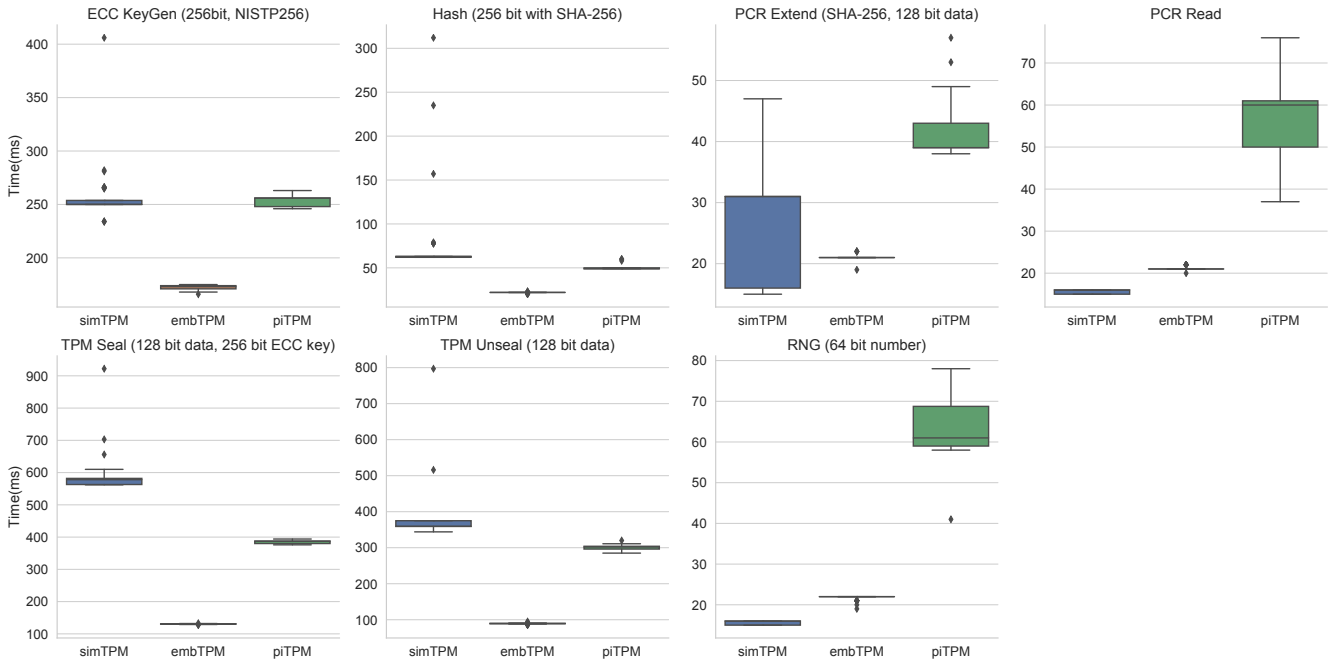

Figure 4: Performance comparison (in *ms*) of different TPM commands for simTPM and an Infineon SLB 9670 TPM2.0 on a Raspberry-Pi and Lenovo laptop.

was between 2.4–15.12 times faster than the fastest hardware TPM—since it is executed on the ARM Cortex main application processor, whereas discrete TPMs use slower microprocessors, as does our simTPM.

6 Use Cases

We discuss briefly how simTPM fits into the trusted computing landscape and explain scenarios that are of particular interest when simTPM and fTPM co-exist.

6.1 Multiple stakeholder model

The TPM specifications [64] as well as the obsolete Mobile Trusted Module (MTM) specifications [61] acknowledged the fact that a trusted platform might have multiple stakeholders. In particular, mobile platforms are not considered under the full management of the user, but critical mobile network management is the domain of the mobile carrier/network operator and the device vendor has high interest in keeping highest privileged operations (e.g., TEE and OS) under their control. The old and new TCG specifications define recommended capabilities and various implementation alternatives to allow multiple stakeholders to safely coexist. For instance, the MTM specification clearly differentiates between remote stakeholders and local stakeholders, each with their own TPM under their control. This concept is reflected in the recommended capabilities for a mobile TPM2.0 [63], which advise the isolation between stakeholders and their resources and policy-based authorization of stakeholder sensitive data. To realize this multiple stakeholder model, the reference architecture outlines different implementation alternatives. For instance, multiple TPMs within a protected environment like TEE, or virtual TPMs supported by a hypervisor [9], where stakeholders are isolated from each other based on the compartmentalization provided by the TEE's trusted OS or the hypervisor, respectively.

Our particular setting also fits well into the defined multiple stakeholder model: two distinct TPMs co-exist, each with a distinct affinity to a different stakeholder. The fTPM is by design designated to the platform stakeholder (i.e., device manufacturer) and it is bound to the device through the device-specific credentials within the TEE (e.g., eFuses) from which fTPM derives its endorsement key and to which it anchors its key/storage hierarchies. For instance, the fTPM described in [54] is designated entirely to the platform and its services. In contrast, the simTPM is designated to the end-user. This intuition is based on the observation that users use the SIM to authenticate themselves to the mobile network and rather stick to one SIM (i.e., phone number) while changing more frequently the device. Moreover, users have to explicitly authenticate themselves to the SIM card, i.e., their mobile carrier issued PIN. In this setting we are going beyond the initial proposals by the TCG reference architecture by actually as-

Table 2: Migrating user data when switching SIM card or device

	Data bound to device	Data **not** bound to device
New SIM card	Key duplication	Key duplication
New device	`TPM_Authorize` key policy	—

signing two distinct stakeholders to two *physically* separated TPM instances, SIM card versus TEE.

6.2 Switching SIM card or device

Since the SIM card is removable and exchangeable, two scenarios have to be considered: the user switches devices but keeps the SIM card, or the user keeps the device and switches to a new SIM card. How this affects migration of the user data protected with the simTPM is summarized in Table 2 and explained in the following.

Switching device: When switching the mobile device and migrating the user data to a new device, the complexity of the operation is dependent on whether the user bound any data to the device. For instance, during secure boot, BL1 has access to device-specific information like the board_id (or potentially values derived from the device-specific vendor key) that uniquely identifies the current platform. This board_id (like derived values) can be included in the measurements collected during secure boot (see Section 4.3) and allow the simTPM to bind data or keys to this particular platform.[2] If the user did not bind any data/keys to the platform, no further action is required beyond moving the SIM card to the new phone. The entire simTPM state including the key hierarchy is inherently migrated to the new device and can be used to decrypt the user data—i.e., a form of *portable sealed storage*. If the data is bound to the board_id, a new feature of TPM2.0 called `TPM_Authorize` has to be used to avoid the problem of *"brittle policies."* Without `TPM_Authorize`, the user data would be bound to one particular board_id and could never be decrypted on another device. With `TPM_Authorize` different possible board_id values can be signed off as valid for a successful verification of the platform state and, hence, decryption of data migrated with the SIM card. The valid board_id values can be signed off by the user to endorse a new phone to which data should be migrated, or by another entity, like the mobile carrier or the employer in BYOD settings.

Switching SIM card: If the user switches the SIM card and hence moves to another simTPM, all user data has to be migrated to the new SIM card, i.e., the necessary simTPM keys have to be moved to the new simTPM. Independent of whether the user data is bound to the device or not, switching the SIM card requires the simTPM keys used for securing the data to be duplicated to the new simTPM. This is an example scenario for TPM2.0 key duplication to migrate keys and associated data to another TPM and is supported by simTPM.

The bottom line of those two scenarios is that a user that wants to keep the option to migrate data secured with the simTPM to both new SIM cards and new devices should use duplicable keys with `TPM_Authorize`.

[2] Assuming a bond between the RTM and simTPM was established.

7 Discussion

The fTPM [54] is the incumbent deployment for a TPM on mobile devices and was part of the Windows Phone platform. However, it was designed primarily for vendor services and did not specifically target the end-user. In this work, we add to the landscape of mobile trusted computing and advocate using the dormant hardware capabilities of SIM cards to provide (additional) TPM support on mobile devices. Our systematization of related works shows that a simTPM can take a niche among the existing works and, in particular, inherently avoids problems of TEE-based deployments (e.g., protected state or secure clock) that currently require compromises and modifications to the TPM specification (e.g., "dark period" or cooperative checkpointing of fTPM) or that make additional hardware requirements (e.g., replay-protected memory blocks). On the other hand, a movable TPM raises the challenge of how to bind the TPM and the platform RTM. In this work, we proposed using the unique features of mobile devices—secure boot and TEE with device-specific, certified keys—to address this challenge. However, we find that this problem also affects prior solutions, like a vTPM based on a PCI-attached secure co-processor, and our solution might give insights into how to establish the TPM-RTM binding in those prior works.

Our simTPM implementation is based on a physical SIM card, thus it is currently not suitable for phones using eSIM (e.g., Apple iPhone). However, eSIM solutions are supported by separate hardware modules (such as JEDEC SON-8) and it might be worthwhile to investigate how those modules can be extended to implement a full TCG compliant TPM2.0.

Recently, Google introduced their Titan chip [67] as part of their Nexus 3 phones, which shows the need for hardware-backed security features in addition to TEE-based implementations on mobile end-user devices. Similar to the simTPM, Titan chip also provides hardware-backed security for system operations like verified booting as well as a hardware-implemented keystore for apps and users. But Titan is exclusive for Google devices, whereas our simTPM is portable between mobile devices and provides TPM2.0 compliant features. Since implementation details are yet unknown, we excluded the Titan chip from our systematization in Section 3.

8 Conclusion

In this paper we proposed simTPM, a hardware-based TPM implementation for mobile devices using the SIM card. Performance evaluation of our prototype shows that our implementation is comparable with an existing discrete TPM chip. Thus, we think simTPM is a practical solution to add user-centric trusted computing technology to mobile devices without the need to add hardware. A particular challenge of a movable TPM is the binding between TPM and the device RTM, which we addressed through a TEE-proxy or a distance bounding protocol. Future work includes a more detailed and formal write-up of the custom DAA scheme we used in our prototype, since it is particularly fitting for implementation on a smart card. Also future implementations of simTPM in industrial IoT or automotive settings for hardware based attestation could be worthwhile to pursue.

9 Acknowledgment

We are grateful to N. Asokan for his insightful suggestions. We are also thankful to the anonymous reviewers for their valuable comments.

This work is supported by the German Federal Ministry of Education and Research(BMBF) through funding for the Center for IT-Security, Privacy and Accountability (CISPA)(AutSec/FKZ: 16KIS0753) and the CISPA-Stanford Center for Cybersecurity (FKZ: 16KIS0762).

References

[1] Hikey960 android development board. https://www.96boards.org/product/hikey960/. Accessed: 02.08.2018.

[2] IBM's TPM 2.0 TSS. https://sourceforge.net/projects/ibmtpm20tss/. Accessed: 06.08.2018.

[3] Trustzone downgrade attack opens android devices to old vulnerabilities. http://bits-please.blogspot.com/2015/03/getting-arbitrary-code-execution-in.html, March 2015. Accessed: 02.08.2018.

[4] Fritz Alder, N. Asokan, Arseny Kurnikov, Andrew Paverd, and Michael Steiner. S-FaaS: Trustworthy and Accountable Function-as-a-Service using Intel SGX. *CoRR*, abs/1810.06080, 2018.

[5] Arm Limited. Trusted board boot design guide. https://github.com/ARM-software/arm-trusted-firmware/blob/master/docs/trusted-board-boot.rst, March 2018. Accessed: 04.08.2018.

[6] N. Asokan. On secure resource accounting for outsourced computation, 2018. Invited keynote at 3rd Workshop on System Software for Trusted Execution (SysTEX 2018).

[7] Samy Bengio, Gilles Brassard, Yvo G Desmedt, Claude Goutier, and Jean-Jacques Quisquater. Secure implementation of identification systems. *Journal of Cryptology*, 4(3):175–183, 1991.

[8] Gal Beniamini. Getting arbitrary code execution in trustzone's kernel from any context. https://googleprojectzero.blogspot.com/2017/07/trust-issues-exploiting-trustzone-tees.html, July 2017. Accessed: 02.08.2018.

[9] Stefan Berger, Ramón Cáceres, Kenneth A. Goldman, Ronald Perez, Reiner Sailer, and Leendert van Doorn. vTPM: Virtualizing the Trusted Platform Module. In *Proc. 15th USENIX Security Symposium (SEC '06)*. USENIX Association, 2006.

[10] Thomas Beth and Yvo Desmedt. Identification tokens—or: Solving the chess grandmaster problem. In *Conference on the Theory and Application of Cryptography*, pages 169–176. Springer, 1990.

[11] Dan Boneh and Xavier Boyen. Short signatures without random oracles. In *Advances in Cryptology - EUROCRYPT 2004: International Conference on the Theory and Applications of Cryptographic Techniques*. Springer, 2004.

[12] Jeremy Boone. Tpm genie: Attacking the hardware root of trust for less than $50, 2018. Accessed: 02/13/2019.

[13] Stefan Brands and David Chaum. Distance-bounding protocols. In *Workshop on the Theory and Application of Cryptographic Techniques on Advances in Cryptology (EUROCRYPT '93)*. Springer, 1994.

[14] Ernie Brickell, Jan Camenisch, and Liqun Chen. Direct anonymous attestation. In *Proc. 11th ACM Conference on Computer and Communication Security (CCS '04)*. ACM, 2004.

[15] Broadchip. BCT4303 Dual Sim card controller. www.chinesechip.com/files/2015-03/912ed043-de27-4e8a-95f7-c009ad22dd92.pdf. Last accessed: 22/01/19.

[16] Dhiman Chakraborty, Lucjan Hanzlik, and Sven Bugiel. simTPM: User-centric tpm for mobile devices (technical report). *CoRR*, abs/1905.08164, 2019.

[17] Sanchuan Chen, Xiaokuan Zhang, Michael K. Reiter, and Yinqian Zhang. Detecting privileged side-channel attacks in shielded execution with déjà vu. In *Proc.*

12th ACM Symposium on Information, Computer and Communication Security (ASIACCS '17). ACM, 2017.

[18] Catalin Cimpanu. Trust Issues: Exploiting TrustZone TEEs. https://www.bleepingcomputer.com/news/security/trustzone-downgrade-attack-opens-android-devices-to-old-vulnerabilities/, September 2017. Accessed: 02.08.2018.

[19] Victor Costan and Srinivas Devadas. Intel sgx explained. *IACR Cryptology ePrint Archive*, 2016(086):1–118, 2016.

[20] Cas Cremers, Kasper B Rasmussen, Benedikt Schmidt, and Srdjan Capkun. Distance hijacking attacks on distance bounding protocols. In *Proc. 33rd IEEE Symposium on Security and Privacy (SP '12)*. IEEE Computer Society, 2012.

[21] Kurt Dietrich and Johannes Winter. Towards customizable, application specific mobile trusted modules. In *Proc. 5th ACM workshop on Scalable trusted computing (STC '10)*. ACM, 2010.

[22] Saar Drimer, Steven J Murdoch, et al. Keep your enemies close: Distance bounding against smartcard relay attacks. In *Proc. 16th USENIX Security Symposium (SEC '07)*. USENIX Association, 2007.

[23] J. Ekberg, K. Kostiainen, and N. Asokan. The untapped potential of trusted execution environments on mobile devices. *IEEE Security Privacy*, 12(4):29–37, July 2014.

[24] Jan-Erik Ekberg. *Securing Software Architectures for Trusted Processor Environments*. PhD thesis, Aalto University, Helsinki, Finland, 2013.

[25] Jan-Erik Ekberg and Sven Bugiel. Trust in a small package: Minimized MRTM software implementation for mobile secure environments. In *Proc. 4th ACM workshop on Scalable trusted computing (STC '09)*. ACM, 2009.

[26] Paul England and Talha Tariq. Towards a programmable TPM. In *Proc. 2nd International Conference on Trust and Trustworthy Computing (TRUST '09)*. Springer, 2009.

[27] ETSI. TS 151 011 V4.15.0 (2005-06) Technical Specification Digital cellular telecommunications system (Phase 2+); Specification of the Subscriber Identity Module - Mobile Equipment (SIM-ME) interface (3GPP TS 51.011 version 4.15.0 Release 4). https://www.etsi.org/deliver/etsi_ts/151000_151099/151011/04.15.00_60/ts_151011v041500p.pdf. Last accessed: 22/01/19.

[28] Amos Fiat and Adi Shamir. How to prove yourself: Practical solutions to identification and signature problems. In *Proc. on Advances in cryptology (CRYPTO '86)*. Springer, 1987.

[29] Seunghun Han, Wook Shin, Jun-Hyeok Park, and HyoungChun Kim. A bad dream: Subverting trusted platform module while you are sleeping. In *Proc. 27th USENIX Security Symposium (SEC' 18)*. USENIX Association, 2018.

[30] Intel. Software guard extensions sdk: sgx_create_monotonic_counter. https://software.intel.com/en-us/sgx-sdk-dev-reference-sgx-create-monotonic-counter, May 2018.

[31] Intel Developer Zone. Platform Service Enclave and ME for Intel Xeon Server. https://software.intel.com/en-us/forums/intel-software-guard-extensions-intel-sgx/topic/806502. Last accessed: 20/05/19.

[32] Jin Soo Jang, Sunjune Kong, Minsu Kim, Daegyeong Kim, and Brent Byunghoon Kang. SeCReT: Secure channel between rich execution environment and trusted execution environment. In *Proc. 22nd Annual Network and Distributed System Security Symposium (NDSS '15)*. The Internet Society, 2015.

[33] Timo Kasper, David Oswald, and Christof Paar. Information security applications. chapter EM Side-Channel Attacks on Commercial Contactless Smartcards Using Low-Cost Equipment, pages 79–93. Springer, 2009.

[34] Bernhard Kauer. Oslo: Improving the security of trusted computing. In *Proc. 16th USENIX Security Symposium (SEC '07)*. USENIX Association, 2007.

[35] Oliver Kömmerling and Markus G. Kuhn. Design principles for tamper-resistant smartcard processors. In *Proc. 1st Workshop on Smartcard Technology (Smartcard 1999)*, 1999.

[36] Nate Lawson. Tpm hardware attacks. https://rdist.root.org/2007/07/16/tpm-hardware-attacks/, July 2007. Accessed: 06.08.2018.

[37] Hongliang Liang and Mingyu Li. Bring the Missing Jigsaw Back: TrustedClock for SGX Enclaves. In *Proc. 11th European Workshop on Systems Security (EuroSec'18)*. ACM, 2018.

[38] Linear Technology. LTC4558 - Dual SIM/Smart Card Power Supply and Interface. https://www.analog.com/media/en/technical-documentation/data-sheets/4558fa.pdf. Last accessed: 22/01/19.

[39] Moritz Lipp, Daniel Gruss, Raphael Spreitzer, Clémentine Maurice, and Stefan Mangard. Armageddon: Cache attacks on mobile devices. In *Proc. 25th USENIX Security Symposium (SEC' 16)*. USENIX Association, 2016.

[40] Junrong Liu, Yu Yu, François-Xavier Standaert, Zheng Guo, Dawu Gu, Wei Sun, Yijie Ge, and Xinjun Xie. Small tweaks do not help: Differential power analysis of milenage implementations in 3g/4g usim cards. In *Proc. 20th European Symposium on Research in Computer Security (ESORICS 2015)*. Springer, 2015.

[41] Aravind Machiry, Eric Gustafson, Chad Spensky, Christopher Salls, Nick Stephens, Ruoyu Wang, Antonio Bianchi, Yung Ryn Choe, Christopher Kruegel, and Giovanni Vigna. Boomerang: Exploiting the semantic gap in trusted execution environments. In *Proc. 24th Annual Network and Distributed System Security Symposium (NDSS '17)*. The Internet Society, 2017.

[42] MAOSCO Limited. Multos standard c-api. https://www.multos.com/uploads/CAPI.pdf, 2016. Accessed: 02.08.2018.

[43] Sinisa Matetic, Mansoor Ahmed, Kari Kostiainen, Aritra Dhar, David Sommer, Arthur Gervais, Ari Juels, and Srdjan Capkun. ROTE: Rollback protection for trusted execution. In *Proc. 26th USENIX Security Symposium (SEC' 17)*. USENIX Association, 2017.

[44] Sjouke Mauw, Zach Smith, Jorge Toro-Pozo, and Rolando Trujillo-Rasua. Distance-bounding protocols: Verification without time and location. In *Proc. 39th IEEE Symposium on Security and Privacy (SP '18)*. IEEE Computer Society, 2018.

[45] Jonathan M. McCune, Bryan J. Parno, Adrian Perrig, Michael K. Reiter, and Hiroshi Isozaki. Flicker: An execution infrastructure for tcb minimization. In *Proc. 3rd ACM SIGOPS/EuroSys European Conference on Computer Systems (Eurosys '08)*. ACM, 2008.

[46] Brian McGillion, Tanel Dettenborn, Thomas Nyman, and N. Asokan. Open-tee – an open virtual trusted execution environment. In *Proc. IEEE Trustcom/BigDataSE/ISPA - Volume 01 (TRUSTCOM '15)*. IEEE Computer Society, 2015.

[47] Thomas S. Messerges and Ezzy A. Dabbish. Investigations of power analysis attacks on smartcards. In *Proc. 1st Workshop on Smartcard Technology (Smartcard 1999)*, 1999.

[48] Thomas S. Messerges, Ezzy A. Dabbish, and Robert H. Sloan. Power analysis attacks of modular exponentiation in smartcards. In *Proc. First International Workshop on Cryptographic Hardware and Embedded Systems (CHES'99)*, 1999.

[49] Microsoft. Secure the windows 10 boot process. https://docs.microsoft.com/en-us/windows/security/information-protection/secure-the-windows-10-boot-process, October 2017. Last accessed: 08/06/18.

[50] Wojciech Mostowski and Erik Poll. Malicious code on java card smartcards: Attacks and countermeasures. In *Proc. 8th IFIP WG 8.8/11.2 International Conference on Smart Card Research and Advanced Applications (CARDIS '08)*, 2008.

[51] Zhenyu Ning and Fengwei Zhang. Understanding the security of arm debugging features. In *Proc. 40th IEEE Symposium on Security and Privacy (SP '19)*. IEEE Computer Society, 2019.

[52] Karsten Nohl. Rooting sim cards. https://media.blackhat.com/us-13/us-13-Nohl-Rooting-SIM-cards-Slides.pdf, 2013. Blackhat USA 2013.

[53] Bryan Parno. Bootstrapping trust in a "trusted" platform. In *Proc. 3rd Conference on Hot Topics in Security (HOTSEC'08)*. USENIX Association, 2008.

[54] Himanshu Raj, Stefan Saroiu, Alec Wolman, Ronald Aigner, Jeremiah Cox, Paul England, Chris Fenner, Kinshuman Kinshumann, Jork Löser, Dennis Mattoon, Magnus Nyström, David Robinson, Rob Spiger, Stefan Thom, and David Wooten. fTPM: A Software-Only Implementation of a TPM Chip. In *Proc. 25th USENIX Security Symposium (SEC' 16)*. USENIX Association, 2016.

[55] Nilo Redini, Aravind Machiry, Dipanjan Das, Yanick Fratantonio, Antonio Bianchi, Eric Gustafson, Yan Shoshitaishvili, Christopher Kruegel, and Giovanni Vigna. Bootstomp: on the security of bootloaders in mobile devices. In *Proc. 26th USENIX Security Symposium (SEC' 17)*. USENIX Association, 2017.

[56] Dan Rosenberg. Reflections on trusting trustzone. https://www.blackhat.com/docs/us-14/materials/us-14-Rosenberg-Reflections-on-Trusting-TrustZone.pdf, 2014. Accessed: 02.08.2018.

[57] Di Shen. Exploiting trustzone on android. https://www.blackhat.com/docs/us-15/materials/us-15-Shen-Attacking-Your-Trusted-Core-Exploiting-Trustzone-On-Android-wp.pdf, 2015. Accessed: 02.08.2018.

[58] Adrian Tang, Simha Sethumadhavan, and Salvatore Stolfo. CLKSCREW: Exposing the perils of security-oblivious energy management. In *Proc. 26th USENIX*

Security Symposium (SEC' 17). USENIX Association, 2017.

[59] Christopher Tarnovsky. Deconstructing a 'secure' processor, 2010. BlackHat DC.

[60] The Chromium Projects. TPM Usage. https://www.chromium.org/developers/design-documents/tpm-usage. Last accessed: 08/06/18.

[61] Trusted Computing Group. Mobile phone work group mobile trusted module specification. https://trustedcomputinggroup.org/resource/mobile-phone-work-group-mobile-trusted-module-specification/, 2010.

[62] Trusted Computing Group. Tpm main part 1 design principles. https://trustedcomputinggroup.org/wp-content/uploads/TPM-Main-Part-1-Design-Principles_v1.2_rev116_01032011.pdf, 2011.

[63] Trusted Computing Group. Tpm 2.0 mobile reference architecture specification. https://trustedcomputinggroup.org/resource/tpm-2-0-mobile-reference-architecture-specification/, 2014.

[64] Trusted Computing Group. Tpm 2.0 library specification. https://trustedcomputinggroup.org/resource/tpm-library-specification/, 2016.

[65] David Wagner. Gsm cloning. http://www.isaac.cs.berkeley.edu/isaac/gsm.html. Last accessed: 02/13/19.

[66] Johannes Winter. Trusted computing building blocks for embedded linux-based arm trustzone platforms. In *Proc. 3rd ACM workshop on Scalable trusted computing (STC '08)*. ACM, 2008.

[67] Xiaowen Xin. Titan M makes Pixel 3 our most secure phone yet. https://blog.google/products/pixel/titan-m-makes-pixel-3-our-most-secure-phone-yet/, October 2018. Accessed: 13.11.2018.

A Binding RTM with distance bounding

In Section 4.3 we discussed using the TEE as proxy in order to assert the authenticity of the RTM and mitigate the risks of a relay attack. Another way to bind the simTPM with its RTM is by using a distance bounding (DB) protocol [7, 10, 13]. Distance bounding is widely used for card-based payment systems. When a credit card is punched to the card reader, the reader runs a distance bounding protocol to check the proximity of the card to prevent a possible relay attack. We are facing the opposite scenario, in which the card is trying to assert the proximity of the device where the communication partner, here the RTM, resides.

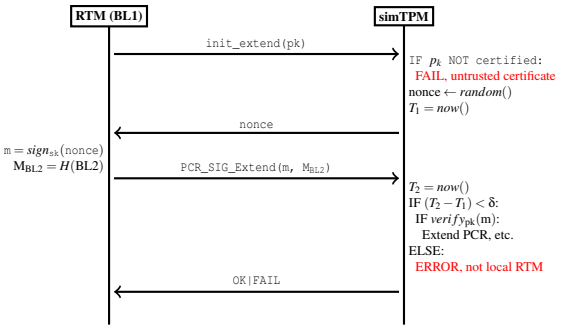

Figure 5: Prototypical distance bounding protocol for binding local RTM (BL1) and simTPM

Prototypical distance bounding: We assume the device vendors equipped the simTPM with certificates for their device-specific keys, which allows a verifier to distinguish trusted code with access to such secrets (e.g., early bootstages, like BL1 or the TEE) from untrusted code, like the host OS or apps. To assert the proximity of the RTM, only the very first measurement provided to the simTPM, i.e., the measurement by BL1 (RTM) of BL2, has to be checked for proximity. After that, the chain of trust of an authenticated boot will transitively extend this trust into the proximity of the RTM. Figure 5 illustrates a prototypical protocol for our scenario. We consider a two-step PCR extension by the RTM for verifying the proximity: (1) the RTM provides the public key *pk* of its device-specific key (or a key derived from it) to the TPM, which then can verify the authenticity of the RTM using the vendor-supplied certificate; (2) as in other distance bounding protocols, the simTPM (verifier) challenges the RTM (prover) with a nonce to which the RTM replies with the signed nonce value (using the authenticated private key) as well as the PCR extension arguments. If this reply of the signed nonce is received within a time threshold T and the signature verifies, simTPM assumes the RTM to be local and extends the PCR with the supplied measurement value M_{BL2}; if either condition fails, the simTPM aborts. For robustness of the protocol, the challenge-response can be repeated N times to decrease the chances of a legitimate, local RTM failing the threshold.

Prototypical setup: In general, calculating the threshold for distance bounding is difficult, because various factors can influence the response time. For instance, jitters of the network over which the verifier and prover communicate, interrupts of the prover's computation, cache and memory delays, etc. might introduce a high uncertainty of the expectable response time. At first glance, our particular scenario seems very favorable for a distance bounding protocol, since the prover (RTM) is the BL1 that has exclusively control of the CPU without interrupts or interference of an OS; and the RTM is connected

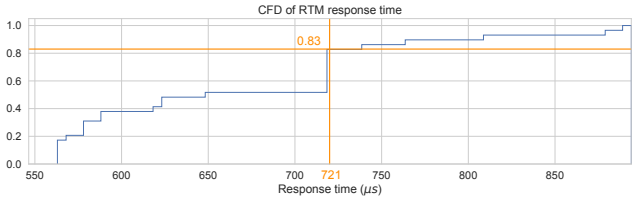

Figure 6: Cumulative frequency distribution of the RTM response time in our measurements ($N = 30$)

to the SIM card over a 480 mbps USB 2.0 bus, in modern devices even via a 5 gbps USB 3.0 bus, with no parallel transfers, providing favorable circumstances for a challenge-response protocol and small error-margin in which an attacker has to fall for a successful, undetected relay attack [20, 22, 44].
The SIM card is connected to the phone through a reader, which is directly connected to the baseband processor. The reader powers the smart card and provides it with the baseband's clock. The clock duty cycle shall be between 40% and 60% of the period during stable operation [27]. Modern smart cards support clock stop to allow preservation of power, which an attacker could use to tamper with the verifier's perception of time. However, this feature can be disabled by initializing the card as clock stop not allowed by setting the *VERIFY CHV* command to 0. Disabling this feature will increase the phone's battery consumption, but not in a significant amount, since the maximum current consumption of an idle SIM card should not exceed $200\mu A$.
The SIM card and the reader connection are in a contact connection and generally interfaces within $20ns$ [15, 38]. The reader connects to the baseband processor through Non-Level-Shifted bidirectional I/O. The connection in our test setup goes through an USB 2.0 bus with 480 mbps. Communication between SIM card and the CPU via this bus ranges between $35ns$ to $72ns$.

Measurements and threshold: We conducted measurements on our test device to evaluate the feasibility of distance bounding to bind the RTM and simTPM. We measured 30 times[3] the speed of the prover (RTM) for calculating the response to the challenge (64 bits nonce) using ECC with the NIST P-256 curve. In our test, the responses took 563–894μs, and the average response time was $669.759 \pm 49.804\mu s$ for a confidence level of 99%. Figure 6 shows the CFD of the RTM response time, where 83% of all responses were $\leq 721\mu s$ and 93% of all responses were $\leq 812\mu s$. The success chance of the distance bounding protocol P_{DB} for a single round is the cumulative probability sampled over the frequency distribution in Figure 6. If we were to set the threshold T for successful distance bounding to 721 μs:

[3] A single measurement requires ≈5min, since only a single measurement per power-cycle is possible on our test device.

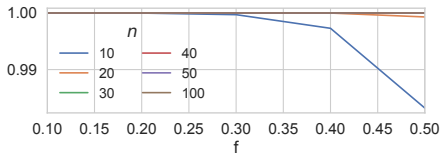

Figure 7: Success probability of local RTM for distance bounding depending on f and n for $T = 721\mu s$ ($p = 0.83$)

$$P_{DB} = \Pr[x \leq 721] = \sum_{i=563}^{721} \Pr[x = i] \approx 0.83$$

where 563 μs is the lowest latency in our dataset. Going below 721 μs reduces the probability of a successful bounding protocol for legitimate devices, i.e., to 0.52 for a threshold of $649\mu s$. To increase the chances for local RTM to pass the distance bounding check, a successful verification usually requires that the response is below T for a sufficient fraction f of the responses, means at-least $f \times n$ out of n responses should arrive within T. When modeling the challenge-response game as binomial distribution and requiring $f \times n$ responses within 721 μs out of n responses (i.e., success probability $P_{DB} = 0.83$), the cumulative probability distribution is:

$$\Pr[x \geq fn] = \sum_{i=fn}^{n} \binom{n}{i} (p)^i (1-p)^{n-i} \text{ where } p \in \{P_{DB}\}$$

Figure 7 shows the success probabilities for different choices of f and n. An optimal choice minimizes n (lower overall runtime overhead for the protocol) while maximizing $\Pr[x \geq fn]$ and minimizing the chance of the attacker to successfully relay. We have observed from our dataset that setting $f = 0.47$ for $n = 30$ (i.e., 14 out of 30 runs) offers a success rate $\Pr[x \geq fn]$=0.99999724049 for local RTM.

Attacker chances: The APDU package for the challenge is 112 bits and for the response 304 bits, which are transferred virtually instantly between verifier and prover ($\leq 1\mu s$). Thus, the response time measured in Figure 6 consists virtually only of the processing time of the RTM, which an attacker cannot speed up (see Figure 3). As a consequence, if an attacker requires more than $721 - 563 = 158\mu s$ to relay the challenge and the response, the relay attack has no chance of winning, since the RTM in our tests required at least $563\mu s$ to compute the response. Assuming a packet size of 55 bytes (minimal Ethernet frame size, IP header, and UDP package with 1 byte payload for the nonce/response), the attacker needs at least a relay bandwith of ≈ 5.87 mbps to have *any* chance of winning, which is a very reasonable assumption. Hence, attacks against this distance bounding are feasible. From our measurements it is hard to concretely model the attacker, however, the attack chance is already 0.1% when relaying via Ethernet and an IP network (55 bytes datasize) with a bandwith of ≈ 49 mbps, or when relaying only the APDU data of 14 bytes (e.g., via a custom build connection) with ≈ 10 mbps.