SCHOOL OF COMPUTATION,
INFORMATION AND TECHNOLOGY —
INFORMATICS

TECHNISCHE UNIVERSITÄT MÜNCHEN

Master's Thesis in Informatics

Establishing trust in an updatable fTPM using remote attestation

Andreas Korb

SCHOOL OF COMPUTATION, INFORMATION AND TECHNOLOGY — INFORMATICS

TECHNISCHE UNIVERSITÄT MÜNCHEN

Master's Thesis in Informatics

Establishing trust in an updatable fTPM using remote attestation

Herstellung von Vertrauen in ein aktualisierbares fTPM durch Remote Attestierung

Author:	Andreas Korb
Supervisor:	Prof. Claudia Eckert
Advisor:	Albert Stark
Submission Date:	15.11.2023

I confirm that this master's thesis is my own work and I have documented all sources and material used.

Munich, 15.11.2023 Andreas Korb

Acknowledgments

Abstract

Contents

1 Introduction

1.1 Motivation

1.2 Goal

1.3 Threat Model

The main threat is the modification of the binary of the fTPM before or during boot. For example, by exchanging the SD card storing the binary. However, we assume that the fTPM cannot be modified by malicious parties after the boot process (regardless of whether the fTPM is benign or compromised) because we trust the OP-TEE environment. Out-of-scope are hardware attacks, side-channel attacks, control-flow attacks, and Denial of Service attacks.

1.4 Environment

This work was created at the ’Fraunhofer-Institut für Angewandte und Integrierte Sicherheit AISEC’ in Garching. It is part of the ’Fraunhofer Society for the Promotion of Applied Research e. V.’, which is an organization distributed over Europe with main focus on applied research. In the roughly 35 years of its existence, it rose to become the largest research institute in Europe with around 30,000 employees.

1.5 Outline

2 Background

This chapter discusses the relevant background knowledge required to understand the remainder of this work.

2.1 Trusted execution environment

One of the core security concepts of operating systems are the privilege levels of processes. Thereby, processes are protected against other processes with the same or lower privilege level. However, they are not protected against more privileged processes. This bears problems for example for cloud computing and edge computing. In cloud computing, other services, the hypervisor, or the cloud provider in general could potentially access sensitive data of the cloud tenant. In edge computing, the edge applications deal with plain text data, while they are potentially running on insecure edge devices. Hence, protection against more privileged processes is desired.

A Trusted execution environment (TEE) is an integrated hardware extension to processors. Effectively, the execution environment is separated into the Rich execution environment (REE) and the TEE by hardware. The REE runs the common software, e.g., a Linux-based operating system and the applications. The TEE allows code to be executed and memory separately to be used on a device in a hardware-protected manner that ensures a high level of confidentiality and integrity. Previous technologies ensure protection of data-in-transit, and data-at-rest, while TEE additionally protects data-in-use. Since it is integrated into the processor, there is no separate chip required. However, it is common to give the TEE a dedicated volatile memory chip, namely the secure RAM (sRAM), which is ensured to be exclusively accessible to the TEE by hardware.

Moreover, the TEE commonly follows the same user and kernel space separation as REE operating systems. The kernel space is running a trusted OS kernel, and the user space is running the trusted applications.

One such TEE is ARM's TrustZone [1]. It partitions all software and hardware resources of the containing system into the Normal world (NW) and the Secure world (SW). While the SW can access the resources of the SW and the NW, the NW is restricted to its own resources. Since ARM is the dominant processor architectures for IoT devices with a market share of 86 % [2], many of the approaches in this field

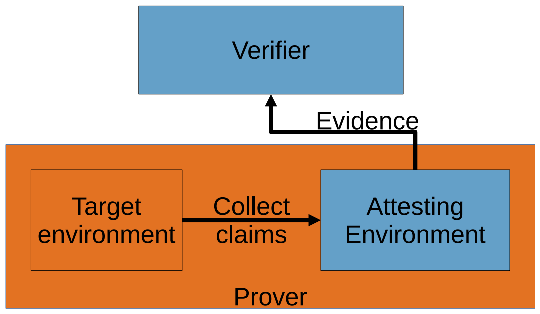


Figure 2.1: Data flow of remote attestation [5]. Initially, only the blue areas are trusted by the verifier. After the attestation, the verifier also trusts the target environment.

of research rely on ARM technology such as TrustZone. Our approach also leverages TrustZone to enable the execution and the remote attestation of an fTPM.

Other TEE technologies are Intel Software Guard Extensions (SGX), and AMD Secure Encrypted Virtualization (SEV), in the future also Intel Trusted Domain Extensions (TDX), and ARM Confidential Computing Architecture (CCA). Since we focus on the implementation of our concept with ARM TrustZone, we do not go into detail about these other technologies here. However, since our concept is not tied to ARM processors and can also be applied to others, they are mentioned for the sake of completeness.

2.2 Attestation

According to the Cambridge Dictionary an attestation is “a formal statement that you make and officially say is true”. Specifically in our context, attestation is a mechanism for software to prove its identity. In the following, the two types are discussed.

2.2.1 Local attestation

Local attestation enables assertions between two environments on the same system [3]. The claim of an environment can be verified by another environment, usually with the help of message authentication codes (MAC) [4]. For example, Intel SGX uses this mechanism to establish assertions between two enclaves [3].

2.2.2 Remote attestation

Remote attestation is a challenge-response protocol initiated by a remote attestor. Figure 2.1 depicts a simplified overview of the data flow of a remote attestation. The process is initiated by a remote trusted party (called "verifier") to verify that a target environment on the end-device (called "prover") has not been tampered with [4, 6]. This challenge contains a nonce, enforcing a fresh response. The response must be an evidence of the challenged system that it is trustworthy. To build that, an attesting environment on the prover device generally inspects the following properties of a program: (i) its code and data has been correctly loaded into memory for execution, and (ii) its data has not been maliciously modified at runtime.

The attesting environment acts as a trust anchor for the verifier. A trusted anchor is required on the device to be attested because at least one trusted component is necessary to extract the data from the remote device to be verified. In many cases, TEE's act as a trust anchor because they are hardware-protected, making it an excellent candidate for a trust anchor.

2.3 Trusted Platform Module

The Trusted Computing Group (TCG) proposed the TPM specification. It describes a Hardware Security Module (HSM) that increases trust in the underlying platform. They support three main use-cases: secure key generation, remote system attestation, and secure storage. Allows remote attestation with Platform Configuration Register (PCR) values in the form of a digitally signed quote to a remote attestor.

TPMs send PCR values in the form of a digitally signed quote to a remote attestor.

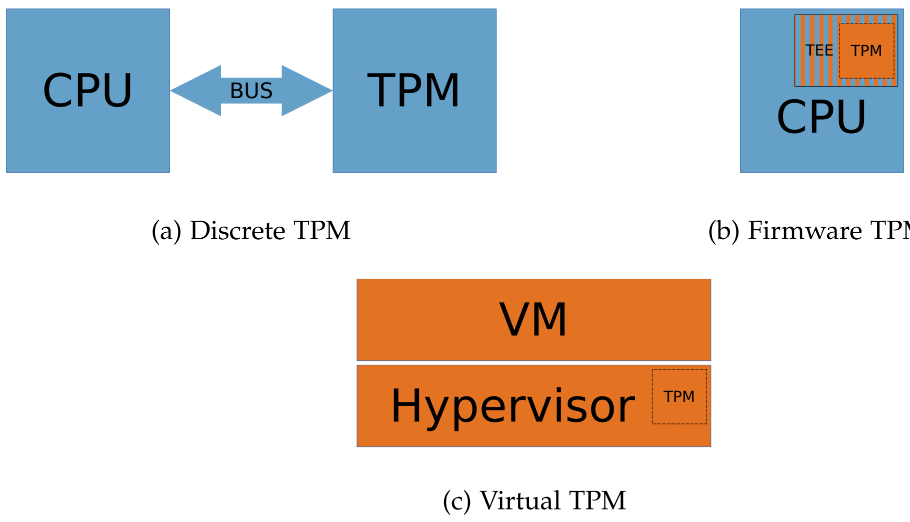
While TPM 1.2 is limited to SHA-1 hashes which are considered broken [7], TPM 2.0 offers crypto agility and therefore, also allows newer algorithms such as SHA-256. In general, TPM 2.0 is more flexible, and is always turned on, while TPM 1.2 components needed to be turned on manually.

$PCR(i)_{t+1} := hash(PCR(i)_t \parallel new\ measurement),\ PCR(i)_0 := 0$

There are three types of TPMs. They all offer the same functionality, but with different security guarantees and performance characteristics. Cheng et al. [9] conducted a detailed performance comparison between fTPMs and dTPMs. They found that fTPMs are faster overall, but especially in key loading, sealing, unsealing, signature verification, and encryption.

Table 2.1: The PCR register usages as defined by the TPM PC Client specification [8].

PCR Index	Usage
0	SRTM, BIOS, host platform extensions, embedded Option ROMs and PI Drivers
1	Host platform configuration
2	UEFI driver and application code
3	UEFI driver and application configuration and data
4	UEFI Boot Manager code (usually the MBR) and boot attempts
5	Boot Manager Code configuration/data and GPT/partition table
6	Host platform manufacturer specific
7	Secure boot policy
8-15	Defined for use by the static OS
16	Debug
23	Application support



(a) Discrete TPM

(b) Firmware TPM

(c) Virtual TPM

Figure 2.2: Schematic illustration of the different TPM types in their pure form. Blue: Hardware, Orange: Software.

2.3.1 Discrete TPM

This is the classical form of a TPM. It is a dedicated piece of hardware, connected to the CPU via a bus. It is designed and manufactured to be highly temper-resistant against hardware attacks. The TPM specifications [8, 10] do not demand a specific bus system, however, they define the interfaces between the TPM and the following bus systems: LPC, I^2C, and SPI.

The well-known 'TPM Reset Attack' was independently described in [11, 12]. It requires minimal hardware, precisely only a wire connecting the reset line of the LPC bus [13] to ground. This results in a reset signal for the TPM, yielding predictable values for the PCR registers, i.e., 0. This allows an attacker to replay the measurement log of a benign boot process to achieve valid PCR values, even though a modified chain has been booted. Since TPM 1.2, TCG provides a mitigation specification for this reset attack [14], requiring the BIOS to overwrite sensitive data after each unexpected reset, preventing an attacker to gain a valid measurement log. However, this mitigation is still vulnerable to cold boot attacks [15, 16].

Winter and Dietrich [16] demonstrate a bus modification attack at TPMs integrated with the LPC bus or the I^2C bus. Their approach, labeled 'Active LPC frame hijacking', allows them to "lift" commands to a higher locality than the one they were originally sent with. This allows them to evolve the 'TPM Reset attack' from being only usable for S-RTM, to also D-RTM systems. They also introduce a new approach of circumventing the TPM's measurement feature. Instead of resetting the TPM as previously described [11, 12], they reset the main device, i.e., the users' device like a desktop PC while preventing the TPM from receiving the reset signal. This keeps the state of the TPM, e.g., the valid PCR values of the previous boot procedure, and the attacker can hijack the boot procedure triggered by the platform's reset and boot a malicious operating system or firmware, while the TPM still stores the old and valid PCRs. While its conceptually easier since the attacker does not need to know the measurement log since the valid PCR values are already in-place, it requires active manipulation of bus transmissions to shield the TPM from the reset signal.

Seunghun Han et al. [17] report two attacks on discrete TPMs to reset the PCR registers. The first targets a gray area in the power management section of the TPM 2.0 specification. The TPM shall store its state into the (its?) non-volatile random access memory (NVRAM) before shutting down when the host platform goes to sleep, and restore it when it wakes up. However, the specification is missing a concrete description how to handle a lack of a stored state when waking up. Therefore, some implementations simply reset the state. Their second attack targets a DRTM, namely an implementation flaw in tboot [18], the most widely used measured boot environment used with Intel's Trusted Execution Technology. However, in their work, they found that some mutable function pointers are not measured, which allows attacks.

A time-based side-channel attack [19] during signature generation based on elliptic curves allows an attacker to recover 256-bit private keys for ECDSA and ECSchnorr signatures.

A passive sniffing attack is shown in [20]. It is applicable to TPM 1.1 connected to an LPC bus. They observed that the data of some operations like unsealing are transmitted via the bus in plain text. Since TPM 1.2, however, the modules no longer send sensitive data unencrypted [16].

That invasive hardware attacks against dTPMs are possible was already shown by Tarnovsky in 2010 [21]. However, this requires a lot of time, knowledge and resources, i.e., hardware and money.

2.3.2 Firmware TPM

As seen in the previous section, the bus between the CPU and a TPM is the biggest attack vector. An fTPM circumvents this by being directly executed within the CPU, revealing no easily accessible bus. The trend is moving towards fTPMs, which can also been seen by the increasing efforts to bring an fTPM to the RISC-V processor family [22].

2.3.3 Virtual TPM

2.4 Secure Boot and Measured Boot

Secure boot is a concept of UEFI doing local attestation of components directly at boot-time. Based on signatures of next-to-boot components. It cancels the boot process as soon as deviations are detected. Binaries of components are first signed and then, deployed universally. Hence, binaries are not bound to the platform and can be considered portable in this context.

Measured Boot is a concept that is often implemented in interplay with a TPM. Measured Boot allows remote attestation to a later time. Uses sealing functionality of TPMs, therefore, bound to the exact platform.

Both technologies are often used in conjunction.

3 Related Work

In the following, we describe defense mechanisms for fTPMs that can be seen as complementary to our approach. They all have in common that they offer no way for a third party to ensure that the hardened fTPM is actually running on the device under test, which is exactly what our work aims to cover.

One approach is to verify the code of fTPMs [23]. Here, the TPM 1.2 code is written in a functional programming language that enables automatic verification.

There exist efforts to improve the security of TPM by introducing the concept of hybrid TPMs [24, 25]. Kim and Kim [24] extend a hardware TPM with software support, which they name hTPM. This increases the defense of the TPM, e.g., circumventing side-channel attacks, and also enables more secure TPM functions, e.g., enabling true random number generation. Their hTPM implementation also shows significantly better performance due to the use of modern CPU features. Vice versa, Gross et al. [25] propose the reverse approach of backing an fTPM with hardware. While their implementation has similar properties to hTPM, it inherits some downsides of fTPMs. For example, their fTPM is still started later in the boot chain than a dTPM, which is not the case for hTPM. However, it is easier to update than hTPM since the lack of a dTPM, and the overall design is simpler.

Abbreviations

TEE Trusted execution environment

REE Rich execution environment

HSM Hardware Security Module

PCR Platform Configuration Register

TCG Trusted Computing Group

List of Figures

List of Tables

Bibliography

[1] ARM Limited. *ARM Security Technology - Building a Secure System using TrustZone Technology*. Issue C. 2009.

[2] E. foundation. *IoT & Edge Developer Survey Report*. (Accessed July 2022). 2022. URL: `https://outreach.eclipse.foundation/iot-edge-developer-2021`.

[3] I. Anati, S. Gueron, S. Johnson, and V. Scarlata. “Innovative Technology for CPU Based Attestation and Sealing.” In: 2013.

[4] J. Ménétrey, C. Göttel, A. Khurshid, M. Pasin, P. Felber, V. Schiavoni, and S. Raza. “Attestation Mechanisms for Trusted Execution Environments Demystified.” In: *Distributed Applications and Interoperable Systems*. Springer International Publishing, 2022, pp. 95–113. DOI: `10.1007/978-3-031-16092-9_7`.

[5] H. Birkholz, D. Thaler, M. Richardson, N. Smith, and W. Pan. *Remote ATtestation procedureS (RATS) Architecture*. RFC 9334. Jan. 2023. DOI: `10.17487/RFC9334`. URL: `https://www.rfc-editor.org/info/rfc9334`.

[6] G. Coker, J. Guttman, P. Loscocco, A. Herzog, J. Millen, B. O’Hanlon, J. Ramsdell, A. Segall, J. Sheehy, and B. Sniffen. “Principles of remote attestation.” In: *International Journal of Information Security* 10.2 (Apr. 2011), pp. 63–81. DOI: `10.1007/s10207-011-0124-7`.

[7] M. Stevens, E. Bursztein, P. Karpman, A. Albertini, and Y. Markov. “The First Collision for Full SHA-1.” In: *Advances in Cryptology – CRYPTO 2017*. Springer International Publishing, 2017, pp. 570–596. DOI: `10.1007/978-3-319-63688-7_19`.

[8] Trusted Computing Group. *TCG PC Client Platform Firmware Profile Specification*. Level 00 Version 1.05 Revision 23. May 2021.

[9] J. L. Cheng, K. C. Chen, H. W. Zhang, W. Y. Chen, and D. Y. Wu. “Emulating Trusted Platform Module 2.0 on Raspberry Pi 2.” In: *International Journal of Security, Privacy and Trust Management* 9.3 (Aug. 2020), pp. 1–11. DOI: `10.5121/ijsptm.2020.9301`.

[10] Trusted Computing Group. *Trusted Platform Module Library Specification*. Family “2.0”, Level 00, Revision 01.59. May 2019.

[11] B. Kauer. "OSLO: Improving the Security of Trusted Computing." In: *16th USENIX Security Symposium (USENIX Security 07)*. Boston, MA: USENIX Association, Aug. 2007. URL: `https://www.usenix.org/conference/16th-usenix-security-symposium/oslo-improving-security-trusted-computing`.

[12] E. R. Sparks. "A Security Assessment of Trusted Platform Modules." In: *Dartmouth College Undergraduate Theses* (2007).

[13] Intel. *Intel® Low Pin Count (LPC)*. Aug. 2002. URL: `https://www.intel.com/content/dam/www/program/design/us/en/documents/low-pin-count-interface-specification.pdf`.

[14] Trusted Computing Group. *TCG PC Client Platform Reset Attack Mitigation Specification*. Family "2.0", Version 1.10 Revision 17. Jan. 2019.

[15] J. A. Halderman, S. D. Schoen, N. Heninger, W. Clarkson, W. Paul, J. A. Calandrino, A. J. Feldman, J. Appelbaum, and E. W. Felten. "Lest we remember." In: *Communications of the ACM* 52.5 (May 2009), pp. 91–98. DOI: `10.1145/1506409.1506429`.

[16] J. Winter and K. Dietrich. "A hijacker's guide to communication interfaces of the trusted platform module." In: *Computers & Mathematics with Applications* 65.5 (Mar. 2013), pp. 748–761. DOI: `10.1016/j.camwa.2012.06.018`.

[17] S. Han, W. Shin, J.-H. Park, and H. Kim. "A Bad Dream: Subverting Trusted Platform Module While You Are Sleeping." In: *27th USENIX Security Symposium (USENIX Security 18)*. Baltimore, MD: USENIX Association, Aug. 2018, pp. 1229–1246. ISBN: 978-1-939133-04-5. URL: `https://www.usenix.org/conference/usenixsecurity18/presentation/han`.

[18] Intel. *Trusted Boot*. URL: `https://www.sourceforge.net/projects/tboot`.

[19] D. Moghimi, B. Sunar, T. Eisenbarth, and N. Heninger. *TPM-FAIL: TPM meets Timing and Lattice Attacks*. 2019. DOI: `10.48550/ARXIV.1911.05673`.

[20] K. Kursawe, D. Schellekens, and B. Preneel. "Analyzing trusted platform communication." In: 2005.

[21] C. Tarnovsky. *Hacking the smartcard chip*. 2010. URL: `http://www.blackhat.com/html/bh-dc-10/bh-dc-10-archives.html`.

[22] M. Boubakri, F. Chiatante, and B. Zouari. "Towards a firmware TPM on RISC-V." In: *2021 Design, Automation & Test in Europe Conference & Exhibition (DATE)*. IEEE, Feb. 2021. DOI: `10.23919/date51398.2021.9474152`.

[23] A. Mukhamedov, A. D. Gordon, and M. Ryan. "Towards a Verified Reference Implementation of a Trusted Platform Module." In: *Security Protocols XVII*. Springer Berlin Heidelberg, 2013, pp. 69–81. DOI: `10.1007/978-3-642-36213-2_11`.

[24] Y. Kim and E. Kim. "hTPM." In: *Proceedings of the 1st ACM Workshop on Workshop on Cyber-Security Arms Race*. ACM, Nov. 2019. DOI: 10.1145/3338511.3357348.

[25] M. Gross, K. Hohentanner, S. Wiehler, and G. Sigl. "Enhancing the Security of FPGA-SoCs via the Usage of ARM TrustZone and a Hybrid-TPM." In: *ACM Transactions on Reconfigurable Technology and Systems* 15.1 (Nov. 2021), pp. 1–26. DOI: 10.1145/3472959.