Masaryk University
Faculty of Informatics

Systematic collection of TPM 2.0 chips attributes on Linux

Bachelor's Thesis

Daniel Zaťovič

Brno, Spring 2020

Masaryk University
Faculty of Informatics

Systematic collection of TPM 2.0 chips attributes on Linux

Bachelor's Thesis

Daniel Zaťovič

Brno, Spring 2020

This is where a copy of the official signed thesis assignment and a copy of the Statement of an Author is located in the printed version of the document.

Declaration

Hereby I declare that this paper is my original authorial work, which I have worked out on my own. All sources, references, and literature used or excerpted during elaboration of this work are properly cited and listed in complete reference to the due source.

Daniel Zaťovič

Advisor: Ing. Milan Brož, Ph.D.

Acknowledgements

I wish to show my gratitude to my supervisor Ing. Milan Brož, Ph.D., for his help, advice and guidance throughout the writing of this thesis. I am also thankful to my loving parents and family.

Abstract

This work provides an overview of a Trusted Platform Module (TPM), related vulnerabilities, Linux software and kernel modules. A Linux-based bootable USB image for collecting TPM properties by volunteers is designed. The image is based on an existing command-line utility for TPM benchmarking. A hybrid user interface for the utility was implemented, which can run either in a terminal or in GUI. The anonymised result can be stored on a persistent partition on the flash drive or uploaded to a repository. The solution was verified on devices with TPM 2.0, ranging from embedded systems, laptop and desktop computers to dual-socket workstations and servers.

Keywords

TPM 2.0, Trusted Platform Module, TSS, vulnerabilities, bootable image

Contents

Introduction

With the rise of the Internet, a need to secure sensitive data has emerged. Since the software ecosystem became increasingly complex and, therefore, hard to secure a hardware solution was proposed. Trusted Platform Module (TPM) is a cryptography chip providing a stripped-down and secure environment to perform cryptographic operations without the keys ever leaving the TPM. The chip also acts as a passive element where the host uploads measurements of system components to ensure platform integrity.

To keep the chip as simple as possible, it performs only elementary operations, leaving most of the logic to the application software. TPM is designed with cost in mind to make it easy to include in as many consumer devices as possible. It has grown increasingly popular and has been included in many PCs and servers.

This work describes the general design, history and use-cases of a TPM in the first chapter. The next chapter discusses the various types of TPM, including a description of both, the software implementations and the physical forms and interfaces. Chapter 3 gives an overview of discovered flaws in the specification and in concrete TPM implementations. Then follows a chapter on the TPM Software Stack (TSS) and TPM software ecosystem with a focus on the GNU/Linux OS.

A tool called `tpm2-algtest` was developed [1] to give TPM software developers an insight into the capabilities of TPMs on the market. The tool collects supported algorithms, performs speed benchmarks and collects generated key pairs for analysis by security researchers. The goal of this thesis was to package `tpm2-algtest` utility as a bootable image, provide a convenient interface for the user and verify the solution by running it on volunteers' computers.

The process of creation of the bootable image and used technologies are described in Chapter 5. Results of the testing runs collected from the volunteers are presented in Chapter 6.

1 Trusted Platform Module

Trusted Platform Module (TPM) is a vendor-independent specification of a cryptographic coprocessor responsible for performing cryptographic operations in a secure separated environment and ensuring the integrity of the host system, which is continuously measured. It was invented by the Trusted Computing Group (TCG) consortium and the exact design is standardized as ISO/IEC 11889 [2]. The main functionality of a TPM includes cryptographically ensured device identification, secure key generation and storage, device health detection and remote attestation and finally, secure generation of random numbers.

1.1 Platform Configuration Registers

Platform Configuration Register (PCR) is a location in a TPM which can not be written directly but only extended. The extend operation is cryptographically guaranteed not to be reversible. The old value of a PCR is concatenated with the new extension and then hashed [2, p. 80]. These extensions (called measurements) are supplied to the TPM by an external application which measures critical components of the system (e.g. code of the BIOS, bootloader and kernel) and configuration files.

This process creates a hash in the PCR, which represents a chain of the measurements leading to the current state. When before malicious code is loaded, it is measured, and therefore the expected PCR hash is corrupted. Particular secret or usage of specific cryptographic keys may be bound to a concrete PCR value. After loading malicious code under such conditions, the TPM would not release the secret to a host which is controlled by the attacker.

A common issue with TPM 1.2 was an update of a system component would break the unsealing because the secret would be sealed to a specific PCR value. TPM 2.0 supports sealing to any PCR value signed by a given key. The administrator can sign PCR hashes for multiple combinations of trusted configurations and system component versions. Apart from sealing objects to PCR values, the TPM can

also provide a signed report of the state of all PCRs (called *a quote* [3, p. 158]).

1.2 Cryptographic keys

TPM is suitable for storage of cryptographic keys because it is separated from the host and is significantly less complex and therefore has significantly reduced attack surface. TPM's main microcontroller is too slow for cryptographic operations. Therefore, a cryptographic accelerator is included in most TPMs to speed up the computations.

The keys are arranged into a hierarchy. On the TPM 1.2, only a single hierarchy existed and was represented by the Storage Root Key (SRK) [3, p. 105]. On the TPM 2.0, there are three persistent key hierarchies and one non-persistent, called *null*, for session keys [2, p. 73]. The three persistent hierarchies are *platform*, *owner* and *endorsement*. Its administrator can disable each hierarchy. This is an advantage over TPM 1.2 where disabling the single hierarchy means disabling the whole TPM.

Existing keys can be loaded or new can be generated directly on the TPM, as the TPM contains an RNG. Due to storage constraints, not all keys can be made persistent. However, the keys can be wrapped, exported, stored on the host and later reloaded.

Endorsement Key (EK), which is the root of the endorsement hierarchy, provides a guarantee the signature came from a TPM. The TPM manufacturer pre-generates the endorsement key and creates a certificate signed by the manufacturer's certification authority. On TPM 1.2, only a single endorsement key existed (because only RSA and SHA-1 were supported), and the certificate was stored persistently on the TPM.

A significant advantage of the TPM 2.0 is the support for multiple encryption, signature and hash algorithms [3, p. 49]. Therefore, multiple EKs may exist, one for every combination of algorithm and key size. Storing certificates for all the combinations would consume much of the limited persistent storage. Therefore, only one or a few of them are stored on the TPM. The rest can be looked up by the public key in the manufacturer's repository.

The same problem arises with the private keys. TCG addressed this by making the key generation reproducible on the same TPM. The same key is generated when the same key length, algorithm type and seed are supplied (collectively called *a template* [3, p. 121]). The template is combined with the TPM's primary seed (which is unique and known only by the specific TPM) using a key derivation function [2, p. 72]. The primary seed represents a root of all the keys stored and used by the TPM. This mechanism can be used for generating any key, including the persistent hierarchies' roots, eliminating the need to store all possible private keys.

1.3 Sessions and authorization

TPM provides a way to ensure the integrity and confidentiality of commands transmitted between the host and the TPM [2, p. 102]. TPM 1.2 offered only two ways of access management, either by verifying the physical presence of the owner or by an HMAC using a shared secret. TPM 2.0 extends this model by allowing additional authorization methods.

Authorization is usually performed over an established session. The exception is a password authorization in which the password is sent directly in the command data. Password is intended for local use only, because it is sent in plaintext and assumes the channel between the TPM user and the TPM is secure. The session includes nonces which are generated on both sides to prevent a replay attack. Every session can be optionally encrypted.

TPM 2.0 includes new kinds of authorizations called Enhanced Authorization (EA). These include plaintext password, signature, PCRs, counter values or value in the NV memory [2, p. 117]. Policy-based authorization allows any logical conjunction or disjunction of all possible authorization methods, making it the most expressive kind of authorization.

1.4 NV memory

The Non-Volatile (NV) memory can be used for permanent storage of user-defined data [2, p. 207]. An NV index identifies the data ob-

jects. TPM 1.2 supported only unstructured objects. Various read and write locks were provided based on HMAC, PCR values or physical presence.

TPM 2.0 assigns a data type to every defined NV index. The type is one of *ordinary*, *counter*, *bit-field* or *extend* [2, p. 208]. Ordinary is TPM 1.2-like unstructured data. A counter is a 64-bit number, which can only be increased. Bit field is a 64-bit value, initially set to 0, where every bit can be set, but never cleared. An *extend* type is similar to the PCRs. Its size is dependent on the used hash algorithm. It is zero-initialized and can be extended by hashing the new value with the previous value.

The permanent NV storage can withstand only a limited number of rewrites. A hybrid index was included in the TPM design to prevent the gradual wear-out of NV storage. It is an NV index, which is stored and modified in volatile memory and is only written on shutdown [3, p. 143].

1.5 Use cases

1.5.1 Remote health attestation

Remote health attestation is based on the TPM's capability to export a signed quote of the PCRs state. The same key may be used for signing arbitrary data and signing attestation quotes. The attacker can create a structure resembling a quote and then ask the TPM to sign it with the same key it uses to sign valid quotes. The quote begins with a constant value `TPM_GENERATED` to prevent this attack. The TPM refuses to sign provided data if they start with `TPM_GENERATED` constant [3, p. 128].

Next, the quote contains the qualified name of the key used to sign it (including all the ancestors of the key), the extra challenge provided by the caller (an anti-replay countermeasure), TPM firmware version and clock state and a list of included PCRs together with their digest [3, p. 158].

A server can use the described mechanism to determine the state of the remote system securely. To decide if the state should be trusted, he has to keep a database of known-good PCR states. When it receives the quote, it checks if the signing key used to produce it belongs to a

trusted computer and then checks the signature. Finally, it verifies if the PCR state is in the database and decides to grant access accordingly.

1.5.2 Full disk encryption

Combination of the approach described in the previous section and the NV storage capabilities of a TPM is used in disk encryption schemes. The disk encryption key or passphrase is stored in the TPM and is sealed to the expected PCRs values. Additionally, a second factor, like a smartcard signature, data signed by a fingerprint scanner [3, p. 166] or a PIN (available e.g. in Bitlocker [4]), may be added.

1.5.3 Cryptography coprocessor

TPM provides cryptographic primitives like sign, hash, encrypt or decrypt. They are much slower than their implementation on a regular PC. In some cases, however, it may be more convenient to use them instead. E.g., in embedded devices where the application CPU is too slow, or there is not enough storage for the cryptography implementation [3, p. 113].

2 TPM types

TPM chips exist in multiple forms. Discrete TPM is a separate physical chip on the motherboard. Firmware TPM is implemented inside a Trusted Execution Environment (TEE) of the application CPU or as a part of the chipset. Software TPM is used during application development or in cloud environments. The following chapters give a more detailed description of the different types.

2.1 Discrete TPM

In theory, discrete TPM provides the best security guarantees [5]. Its physical separation from the main CPU provides better isolation and removes some of the possible side channels. The separate IC package provides resistance against physical attacks like fault injection.

There are multiple ways a discrete TPM can be connected to the host CPU. A common approach for desktop computers is to use the Low Pin Count (LPC) bus, but also Inter-Integrated Circuit (I^2C) can be used. The TPM is usually sold on a daughterboard, which plugs into a vendor-specific connector on the motherboard. On laptop computers, the TPM is soldered directly to the motherboard. TPM chips with Serial Peripheral Interface (SPI) or I^2C interfaces also exist. Chips with these interfaces can be deployed in Internet of Things (IoT) or industrial environments. Figure 2.1 shows different types of TPMs. Left-to-right Infineon Iridium [6] with SLB9670 [7] (SPI interface), GIGABYTE TPM GC-TPM 2.0 [8] with SLB9665 [9] (LPC interface) and ASUS TPM-SPI [10] with Nuvoton NPCT750 [11] (SPI interface). TCG publishes a list of complaint TPM devices [12]. It shows three vendors - *Infineon Technologies*, *Nuvoton Technologies Corporation* and *STMicroelectronics*.

2.2 Firmware TPM

To further decrease price, increase the availability and bring TPM to already shipped hardware, vendors started to implement it in firmware. Such implementations might run inside a trusted execution environ-

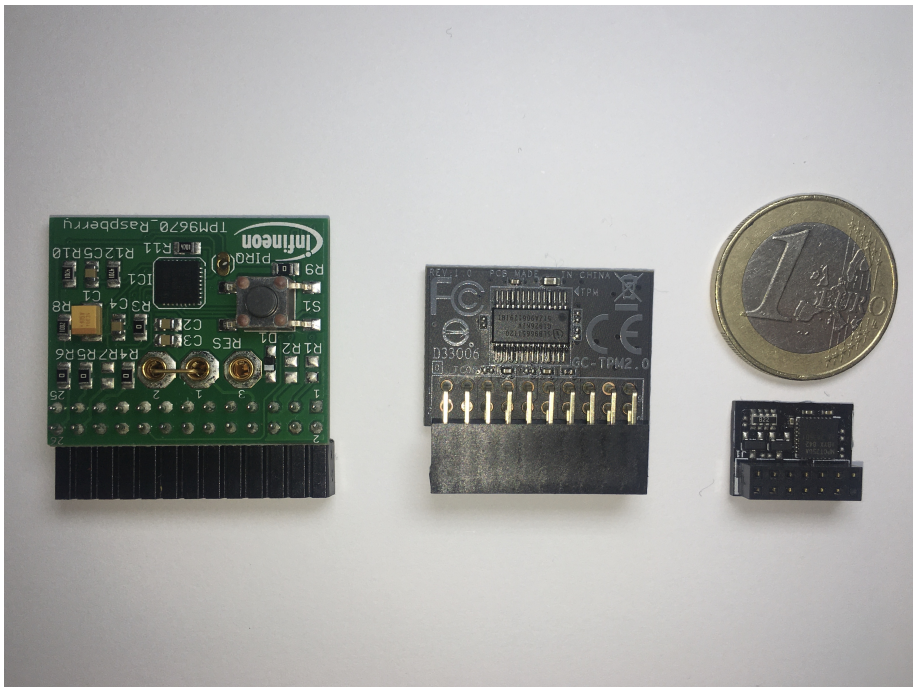

Figure 2.1: Different types of discrete TPM.

ment and share the CPU with untrusted applications. There exist commercial implementations [13] for ARM processors that run inside ARM TrustZone [14].

Intel started incorporating a special processor core to their chipsets called Intel Management Engine (ME) [15]. The role of this processor is to provide remote management capabilities and the root of trust. It is also the place, where Intel implements the firmware TPM [16]. AMD offers similar technology called AMD GuardMI, which provides a firmware TPM [17]. It is implemented inside AMD Secure Processor, which plays a role analogous to the Intel ME.

2.3 Software TPM

Software TPM includes simulator and hypervisor TPM implementations. Trusted Computing Group (TCG) has provided reference implementation, which was packaged into a simulator by Microsoft [18].

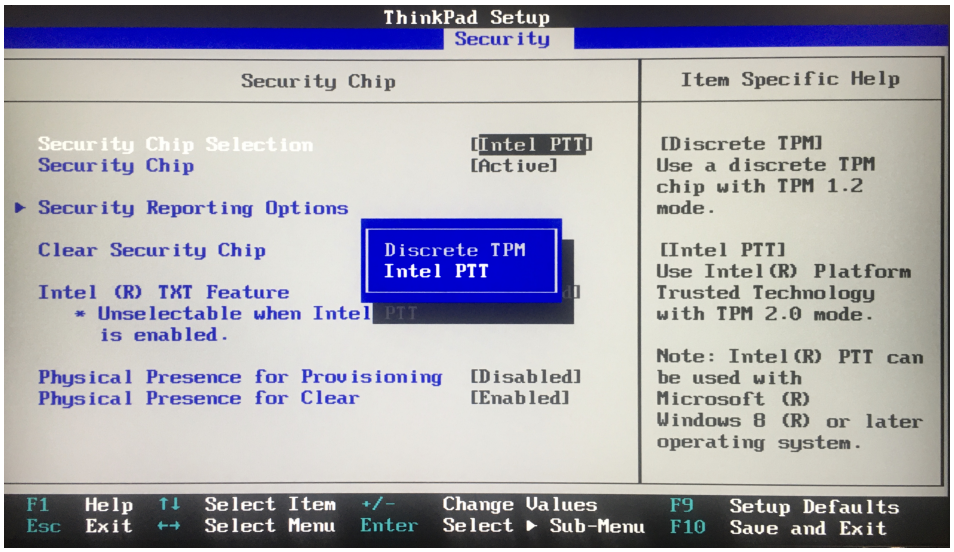

Figure 2.2: Thinkpad X240 setup with an option to turn on firmware TPM.

The simulator is useful during the development of a new TPM application. Google has created an online version with study texts intended to be an educational tool [19]. Some virtualization hypervisors implement virtual TPM to increase security in the virtual guests. Such implementation is present in VMware vSphere [20], Microsoft Hyper-V [21], and Xen [22].

The `swtpm` tool can be used on Linux to create a virtual TPM character device. This device can be used as a regular physical TPM with standard tools. QEMU supports TPM pass-through of a real TPM, or virtual TPM created using `swtpm`.

Xen hypervisor enables virtual TPM support via a distinct vTPM management virtual machine (VM) which emulates the TPMs used by other VMs. It uses a special design to link the virtual TPM certificates to the hardware TPM root of trust [23]. Google provides virtual TPMs in its Shielded VM product. The vTPM is used for measured boot, and it is based on `swtpm` implementation [24].

3 Vulnerabilities

As with any security device, TPM is also not without vulnerabilities. The attacks can be divided into two categories. The first one includes hardware attacks which require physical access. The other one comprises logical flaws in the firmware behaviour.

3.1 Man in the middle attack

This attack requires the attacker to be able to eavesdrop and manipulate the communication between a host and a TPM. The concrete methodology used to accomplish this goal varies according to the TPM type and the bus it uses for interaction.

As described in section 2.1, standard options for the bus are an LPC bus on PCs and I^2C or SPI on embedded devices. Using a firmware TPM or a discrete TPM in the same chip package as the main CPU might significantly complicate the attack. In such cases the bus is not exposed externally and attaching to it would require advanced techniques and costly equipment.

Intercepting a discrete TPM, external to the CPU, might be relatively easy. In the case of TPM soldered to the motherboard, it is necessary to solder conductors directly to the chip pins. This is the case with most of the laptop TPMs. On the other hand, desktop TPMs often come on daughterboards making it sufficient to just disconnect the TPM and insert a malicious device between the host and the daughterboard. It has been shown that cheap off-the-shelf components are sufficient to construct the interception device [25, 26]. All that is necessary is a microcontroller capable of communicating over I^2C or SPI or a low-end FPGA for LPC bus transceiving.

After the attacker has gained control over the communication channel, he can directly read the secret (e.g. disk encryption key) after the TPM unseals it [26]. Moreover, if the host uses the TPM as a source of entropy, the attacker may modify random data returned by the TPM. In practice, this gives him the ability to temper with cryptography operations, like secure key generation, on the host.

Driver authors often do not consider the possibility of the TPM being malicious. This premise does not hold when the attacker seizes

control over TPM communication. Linux kernel and major bootloaders including *Coreboot*, *EDK II*, *U-Boot* and *tboot* contained vulnerabilities which the attacker could exploit by sending malformed responses from the TPM [25].

Possible mitigations include using authorization sessions. The problem is that this is made mandatory by the TCG specification only for specific commands. However, in practice, developers often do not choose to implement the optional authorization for critical commands like `GetRandom` and `PcrExtend`. In some cases, even when the software authors have used authorization session, they have not used it correctly. For example, in tboot, the HMAC was generated on outgoing messages but was not verified on incoming messages, Linux kernel made the nonce based on random data supplied by the TPM or tboot even disclosed uninitialized memory as the nonce [25].

3.2 RAM readout

While this vulnerability is not directly a TPM vulnerability, it still significantly affects environments in which TPMs are deployed. Some secrets can be sealed to PCRs and automatically released on every reboot after the system gets into the allowed state. For example, BitLocker seals the volume key to a known-good PCR state. These secrets may then be stored in RAM. Some attacks like cold boot or DMA, make it possible to read arbitrary memory, including the TPM-released secret data.

This way, the attacker can also read non-TPM stored secrets if the computer is put to sleep. However, in most cases, the attacker has only one attempt to extract the data. If he fails and the memory content is cleared, he can not repeat the attack. In the case of PCR-sealed data, it can be released from the TPM and loaded into RAM on every reboot (given that the system measurements haven't changed). This gives the attacker a virtually unlimited number of attempts to repeat the attack.

The first way an attacker can read RAM is using a cold boot attack. The cold boot attack is based on the fact that RAM keeps its content for a brief moment even after reset or disconnection. This moment can be further prolonged by exploiting physical properties of storage medium (e.g. by freezing the memory module). During this period attacker

may quickly reset the PC and boot into a small bootloader capable of dumping the RAM contents left from the previously running OS on a persistent disk [27]. To mitigate this, TCG required all complaint systems to overwrite all the memory when an unclean shutdown is detected [28]. Although this is an improvement, it still does not solve the problem when the RAM module is cooled and transferred to a different system where the TCG rules are not followed.

A second way to obtain RAM content is by using a rogue device which misuses DMA capability of PCIe bus. A full form of PCIe on the desktop or mini PCIe on laptops can be used for connecting the offensive device. These connectors are available only after disassembling the case. However, some laptops have them exposed externally in the form of a Thunderbolt port or ExpressCard. It has been demonstrated that a low-end mini PCIe FPGA can be used for the construction of such a device [29]. Using conversion adapters similar tool can be connected to a Thunderbolt connector [30]. To mitigate DMA attacks, IOMMU and Intel VT-d technologies [31] can be used to restrict access to memory by PCI devices.

Several solutions have been proposed to mitigate generic RAM readout attacks. One of them is a series of patches to Linux kernel which keep cryptographic keys solely in microprocessor registers [32]. Microsoft recommends turning on pre-boot authentication (e.g. using a PIN) in addition to sealing the encryption key to PCR measurements to prevent the attacker from having multiple attempts to perform the attack [4].

3.3 Side-channel attack

This class includes attacks in which private information may be disclosed via an unintended channel like duration, power consumption, heat dissipation or electromagnetic radiation while performing a particular operation. A common cause for such vulnerabilities may include improper physical shielding to stop electromagnetic radiation, power consumption pattern during a performance of critical computations or secret-dependent duration of cryptographic primitives like signature creation or verification and data encryption or decryption. Such vulnerability was found in STMicroelectronics discrete TPMs

and Intel fTPMs [33]. It enabled the attacker to recover ECDSA and ECSchnorr private keys after observing 1300 operations for Intel fTPM or 40000 operations for STM TPM. It was fixed by releasing a firmware update for STM TPM and update for Intel ME inside which Intel fTPM runs.

3.4 Logic vulnerabilities

TPM implementations may also include erroneous behaviour, or there might be grey areas in the TCG specifications. A bug was found in the handling of sleep mode on Trusted Platform Modules manufactured by multiple vendors [34]. The TPM relies on the host to tell it that it is entering S3 sleep mode, and the TPM should save its state. Upon restoring from the sleep state, the host notifies the TPM to restore the saved state. The vulnerability consisted of leaving out the command for the TPM to save state, then enter S3 sleep state which cuts off power to the TPM and after restoring from the sleep send a command to the TPM asking to restore state. In TPMs from specific vendors, this caused the TPM (including the PCRs) to reset. It has been demonstrated [34] that this can be combined with a modified bootloader and OS, which save the chain of measurements leading to a good PCR state. The attacker may then be able to replay this chain to set PCRs to the trusted state.

Another algorithmic flaw was discovered using black-box testing of devices manufactured by Infineon Technologies AG. The bug was found in the RSA key generation by an RSALib software component used in the firmware of multiple devices from various domains (including TPMs and smartcards) [35]. The attack simplified private key factorization based on the structure of the generated keypairs, which was limited to a specific form. When utilizing this attack, it was possible to derive the private key just using the knowledge of the public key. The attack takes approximately 3 CPU-months for 1024 and 2048-bit. Infineon probably introduced the vulnerability in an attempt to speed up the key generation.

4 TPM software

The TPM is on purpose as simple as possible and leaves most of the logic to application software. A stack has to exist on the host which performs low-level operations. E.g. sharing access to the TPM among applications or swapping objects in-and-out of the TPM (due to the space constraints).

4.1 TPM Software Stack

TPM Software Stack (TSS) is a TCG developed specification of APIs for communication with the TPM [36]. The APIs are designed to be OS and hardware independent allowing development of portable software. The stack consists of multiple layers.

4.1.1 TCTI

TPM Command Transmission Interface (TCTI) is the layer responsible for direct communication with the TPM. It takes a byte stream from the upper layer and transmits it to the TPM. The TCTI's `receive` and `transmit` functions depend on the way the host communicates with the TPM. There may be different TCTIs for a remote TPM and a local TPM. Apart from the transceiving functionality, the TCTI can cancel the issued command or terminate the connection.

The TCTI layer may occur multiple times in the software stack, when raw TPM commands need to be transmitted [3, p. 94]. For example, a resource manager may run on the host and use a TCTI to communicate with the TPM. Applications then use another TCTI to talk to the resource manager.

4.1.2 SAPI

System API (SAPI) provides a low-level interface to the TPM. It gives direct access to all TPM features. Low memory footprint and no need to allocate memory (except for the caller allocated structures) make it an ideal choice for embedded uses. There are functions to allocate and prepare command context, to execute the command and to finish it.

Firstly the caller allocates a memory block of size returned by `Tss2_Sys_GetContextSize` and then uses `Tss2_Sys_Initialize` [3, p. 86] to initialize the context structure in the provided memory location with a given TCTI context. Then a command-specific prepare function is called. The command parameters have to be marshalled into a byte stream in the required form. Optionally, it calculates the HMAC and performs encryption if the authorization session is used. Then the command is executed, either synchronously or asynchronously. Finally, a finalization function is called which (analogously to the prepare function) decrypts, verifies HMAC and unmarshalls the received response.

4.1.3 ESAPI

Enhanced System API (ESAPI) is an extension on top of SAPI which aims to reduce programming complexity but still provide the capability to send individual commands to the TPM. It provides 100% of the TPM capabilities, whereas FAPI provides about 80% [37]. It significantly reduces the complexity when sessions and authorizations are used.

4.1.4 RM and TAB

TPM Access Broker (TAB) is a software component which allows multiple processes to use the TPM concurrently. It forbids sending new TPM commands when another process is already communicating with the TPM. TAB prevents a process from accessing another process' TPM sessions and objects.

Resource Manager (RM) loads the objects in-and-out of the TPM due to the limited number of objects which can be loaded at the same time. RM has to virtualize handles to provide an illusion that more free handles are available.

These two layers are usually found in a single package because their functions are closely related [3, p. 95]. They may be omitted completely in single-user and single-threaded systems, like embedded devices.

4.2 Linux TPM software

4.2.1 TSS implementations

Multiple open-source implementations of TSS exist for the Linux platform. An implementation by *tpm2-software community* called `tpm2-tss` exists. The same community develops a set of command-line utilities called `tpm2-tools` [38], built upon `tpm2-tss`, which provide an interface to TPM 2.0 functionality. They support performing all cryptographic operations like signing, encrypting, decrypting or hashing data. Other capabilities include manipulating PCR registers and NV memory, creating policies and loading, generating and exporting cryptographic keys.

The tools by IBM offer similar functionality [39]. They are based on IBM's implementation of TSS, which is not completely TCG-complaint but offers equivalent functionality. Microsoft Research has developed TSS implementations, called `TSS.MSR`, in multiple programming languages including C#, C++, Java and TypeScript [40].

4.2.2 Linux kernel

The Linux kernel contains drivers for multiple TPM 1.2 and 2.0 devices by various vendors including Atmel, Infineon, Nuvoton and STM, but also virtual TPMs created by Xen hypervisor. There exists a special virtual TPM proxy driver which populates a TPM character device and connects it to a user-specified file descriptor [41].

A TPM simulator can be attached to the file descriptor, and software can communicate with it via the virtual device as with regular TPM device. The virtual device and file descriptor pairs are managed through a control device `/dev/vtpmx`. The `swtpm` project [42] aims to provide TPM 1.2 and 2.0 simulator together with a setup utility which creates the virtual TPM and connects it to the simulator. Multiple virtual TPMs can be created and then passed to virtual machines or containers.

Linux kernel has a key retention capability [43]. It supports several types of keys. They are a *user*, *logon* and *keyring*. All of them have a name, and *user* and *logon* keys have a payload. A *user* key can be created, modified and read in userspace, but cannot be used by kernel

services. After a user writes *logon* key, it can be read and used only by kernel services (e.g. as a key for filesystem encryption), and it is impossible to read. A new TPM-backed *trusted* key type was added [44]. The key is generated in the kernel and sealed by a TPM using an RSA key. The key itself never leaves the kernel, and only the TPM-sealed blob is released to userland.

Since version 4.12, the Linux kernel offers in-kernel resource manager. For every TPM character device, it publishes another TPM device. E.g. for `/dev/tpm0` device there is corresponding `/dev/tpmrm0` device. When the TPM is accessed via this proxy device by multiple programs at the same, the kernel manages these accesses and rearranges them so that they do not conflict.

4.2.3 Measured boot in Linux ecosystem

In order to make sealing a secret to a PCR value provide reasonable security guarantees, critical system components have to be measured. This requires cooperation from previous stages of the boot chain. A TPM-aware bootloader is required. Initial attempts included *Trusted-GRUB* [45] based on the now obsoleted *GRUB* version and *Trusted-GRUB2* [46] based on *GRUB 2.02*. Both projects support only TPM 1.2 and do not support EFI. Currently, similar functionality is included in upstream *GRUB* [47], with support for EFI systems only.

5 Practical part

Part of this work was to create a bootable USB image designed for running TPM 2.0 tests and benchmarks. The advantage of this approach is that a user is not required to install any dependencies on his system. The user only writes the disk image to an empty disk device (most likely a flash drive) and boots the system from the written medium. We used an already made tool for collecting TPM 2.0 properties called `tpm2-algtest` [1]. On top of this command-line tool, we made a hybrid user interface capable of running either in the terminal, or in GUI.

5.1 Front-end for `tpm2-algtest`

The live system is intended to be used not only by technically skilled individuals, but also by ordinary users. To make the tool more accessible and to help collect as many test results as possible, we created and packaged a UI front-end for the `tpm2-algtest` utility. As a fail-safe, when the system can not boot to a full graphical environment, the solution is also capable of running in a terminal.

The UI is written in Python 3 and reuses a part of the logic from `run_algtest.py`, which is a part of `tpm2-algtest`. To run in both GUI and terminal mode, we used `libyui` library [1], which is developed as a part of the YaST Linux installer by OpenSuse [48]. The `libyui` library picks the most suitable backend to render the UI. Currently, there are three available: Qt [49], GTK [50] and ncurses [51]. Therefore, the same UI code can run either in the terminal via ncurses, or in a desktop environment via GTK or Qt.

The UI tool lets the user run `tpm2-algtest`, shows the progress and asks the user to enter a contact email address optionally. Upon completion, the tool asks the user whether to upload the results or store them on a persistent partition of the bootable disk. If stored on the disk, the results can be read later by plugging the live USB to for example a Windows or Linux system.

---

1. `https://github.com/libyui/libyui`

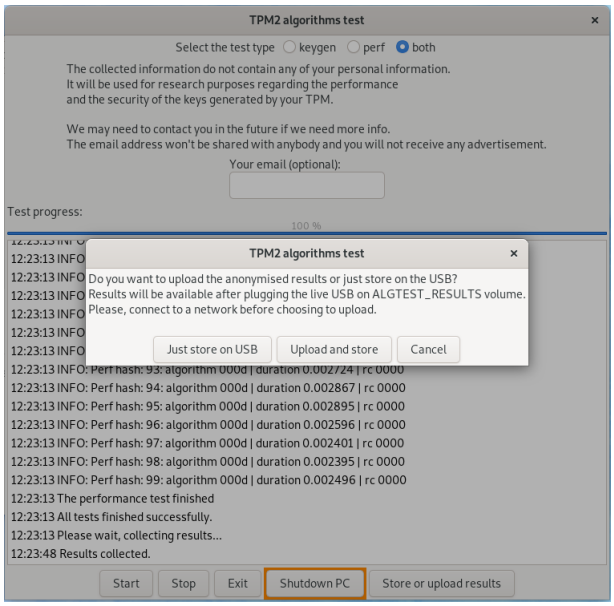

Figure 5.1: Interface for `tpm2-algtest` running with GTK backend.

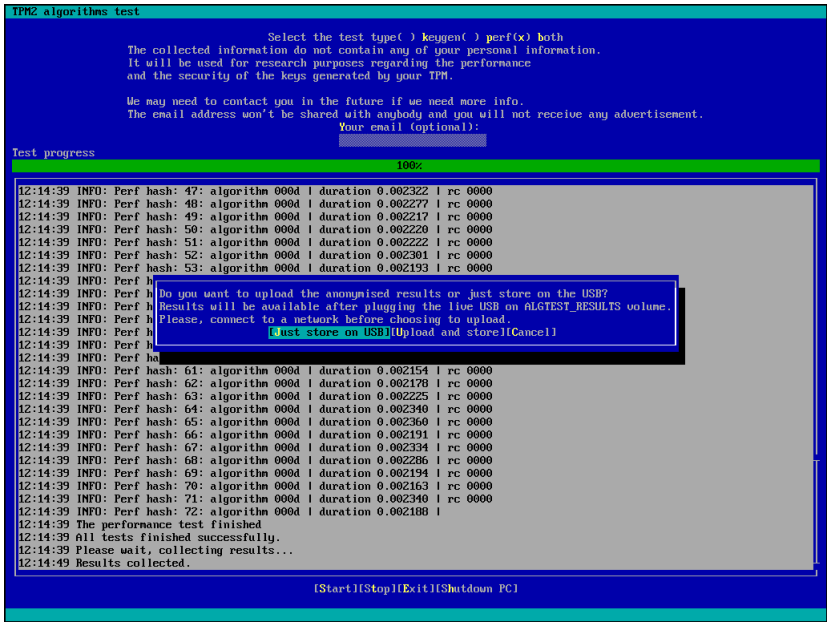

Figure 5.2: Interface for `tpm2-algtest` running with ncurses backend.

5.2 Packaging

The UI along with `tpm2-algtest` were packaged as an RPM package and stored in an online repository to ease versioning, installation and live image build process. For this, we used *Copr* [52] build system run by the *Fedora Project* [53].

To build an RPM package a `.spec` file is required. It contains all the necessary information for building the package from source code and installing it. It includes package metadata like the package name, description, author, license, version, changelog and dependencies. Packages required during the build process are also listed. It contains script to prepare, build and clean up after the package is built, as well as scripts to run before and after installation. Finally, it contains a list of built files with their destination on the system where the package is installed.

When the package is built using `rpkg` [54] (which is the default choice in *Copr*), template `.spec` files can be used (with `.spec.rpkg` extension). The `.spec.rpkg` file can contain specific placeholders which are then substituted for actual values. Builds from remote source code repositories are also supported through an `rpkg` module [55]. When the remote build is used, the module can replace the placeholders with information from the repository like name, author, changelog or version (based on a version control tag).

5.3 Bootable disk image

We chose *Fedora* Linux distribution as a platform for creating the `tpm2-algtest` bootable image. The *Fedora Project* uses a tool called `livemedia-creator` [56] for building a bootable installation ISO file. This utility composes the live system image based on a kickstart file. A kickstart file specifies software repositories, packages to install and scripts to run before and after the installation of the packages. Default kickstart files used to build *Fedora* installation media are shipped in a package `fedora-kickstarts` [2]. By adjusting these kickstart files, a modified *Fedora* distribution, known as a "*Fedora Remix*" [57], can be created.

2. `https://pagure.io/fedora-kickstarts`

In addition to the original *Fedora* repositories, we added the *Copr*-generated repository [3], with the `tpm2-algtest` tool and the UI application. Unnecessary packages were removed from the default kickstart file, to decrease the overall size of the image. We added a startup script to launch the GUI automatically after system boot.

The kickstart file is passed to the `livemedia-creator` program. It creates an empty disk image and starts *Anaconda* [58] installer, to set up the live system on the disk image. The system where *Anaconda* runs should be as similar as possible to the live system, which is to be built (e.g. a live system based on *Fedora 32* should be built on a *Fedora 32* host). To simplify this, `livemedia-creator` offers an option to start *Anaconda* in a virtual system emulated using a *QEMU* [4] emulator. After *Anaconda* finishes, *Lorax* templates [59] are used to transform the disk image into a bootable ISO file.

The template compresses the image by storing it on `squashfs` [60] filesystem. The final ISO image is created from the individual partition images and bootloader binary using `xorrisofs` tool [61]. The tool creates a hybrid partition layout. The hybrid image at the same time contains an ISO 9660 filesystem [62] (for booting from an optical disk), MBR (for booting on legacy BIOS systems) and GPT (for booting on EFI systems).

5.4 Persistent partition

To be able to store the result from the live system offline, we included a persistent partition in the image. We created an empty FAT32 partition image and modified the `xorrisofs` arguments in the default *Lorax* template to add the partition. The produced image was written to a flash drive and tested on Linux and Windows systems. Different positions of the persistent partition with respect to the other partitions were tested. Linux could recognize and mount only some layouts, as well as Windows, but Windows could not mount any.

Therefore, a different solution was chosen. After building the bootable ISO file, it is converted to a partitioned disk image by writing it to a loopback device using `livecd-iso-to-disk` [63]. This process

3. `https://copr.fedorainfracloud.org/coprs/dzatovic/tpm2-algtest/`
4. `https://www.qemu.org/`

removes the ISO 9660 filesystem, which is essential for booting the system from an optical drive. However, a persistent read-write partition can not be used on an optical disc, so it does not present a problem. A shell script is then used to enlarge the disk image, modify the partition table and create a FAT32 partition for storing the test results.

5.5 Uploading results

As the image contains a full Linux system with networking support, we have implemented a method for uploading the results to a remote location. If a wired connection is used, it is set up automatically using DHCP or custom settings can be configured using GNOME settings graphical interface [64]. The same interface can be used for connecting to a wireless network.

For uploading the results, we chose personal depository in the information system of Masaryk University [5]. Every information system user has a depository, readable only by him, where other users can upload files. A user has an option to allow anybody from the Internet to upload files to his depository. The current implementation contains an uploader class, which can upload the result to a depository of the person specified by his university identification number. This class can be potentially replaced, to support different upload destinations.

5. `https://is.muni.cz/`

6 Experiment

The goal of the experiment was to test and verify the implementation on various systems, and identify problems which may arise while performing the data collection. The bootable image has to be built and distributed to the volunteers, along with instructions. If the test is performed remotely, the volunteer is expected to create the bootable USB disk himself.

To create the bootable disk, an image writing tool which allows bit-by-bit copy is used. On the Windows platform, a commonly used tool, which also works with our solution, is *Rufus* [65]. On the Linux platform, the *Disks* [66] utility, which is part of the *GNOME* project, can be used. It is shipped on all distributions which use *GNOME* desktop environments like *Fedora* or *Ubuntu*. A command-line utility `dd` [67] can be used, but extra caution must be taken not to confuse the block devices and overwrite user data. Therefore, it is not recommended to be used by regular users.

After the disk is written, the volunteer has to restart the computer and boot from the created disk. He can use the one-time BIOS boot menu to select a temporary boot device. A vendor-specific key has to be pressed on startup to bring up the menu. A list of hotkeys for BIOSes by different vendors can be included in the instructions.

When the system boots, the `tpm2-algtest` UI is shown. It includes information about the test duration, the intended use of collected data and the fact that they do not contain the user's sensitive data. The participant may optionally enter his email address, if he wants to further participate in the research.

After the test finishes, the user is asked if the result shall be uploaded to the remote repository or only stored locally, on a persistent partition of the bootable USB drive. The result is named uniquely, so the same drive can be reused for multiple test runs across different computers. If the user wants to upload the result, he is asked to set up networking.

If the experiment is performed in person, the user may be individually guided, and he can be given a pre-prepared drive. It is possible to hand-out bootable drives and let the volunteer run the test independently, store the result on the USB and return it.

In remote or cloud systems, it might be difficult or impossible to boot an arbitrary image. Instead, the user can choose from a list of Linux distributions. If *Fedora* is available, the user can add a custom repository [1] via a package manager to install the `tpm2-algtest` UI. The same approach can be used if the user already runs *Fedora* distribution.

We have collected results both, by directly booting the image, and by installing the tool to already running remote servers and workstations. All the systems, except one, were based on *x86* architecture. One system (Raspberry Pi with TPM evaluation board [6]) was based on a 64-bit *ARM* architecture. For this system, we used a pre-built *Fedora 31* image and manually installed the testing tool. The *x86* systems ranged from desktop and laptop computers to dual-socket workstations and servers.

6.1 Discovered issues

One system was discovered (Thinkpad X240 with an Intel firmware TPM 2.0), where the test failed to run, although TPM 2.0 was present. The `tpm2-algtest` utility uses an elliptic curve primary key for creating every key used during the benchmark. This specific system, however, does not support any elliptic curve.

The device was initially produced with a discrete TPM 1.2. Later, TPM 2.0 capabilities were added as a chipset update which included a firmware implementation of a TPM. It was tested with the latest available Intel ME update (from July 2018) [2].

During the testing, a discrepancy was discovered between the duration of the test with and without a TPM resource manager and access broker. The measurements acquired during the performance testing phase were affected. The kernel resource manager (`tpmrm0` device) performed the worst, followed by `tpm2-abrmd` and direct device access via `tpm0` was the fastest. Several factors might have introduced the slow-down. There is an additional overhead of marshalling and

1. `https://copr.fedorainfracloud.org/coprs/dzatovic/tpm2-algtest/`
2. `https://pcsupport.lenovo.com/sk/en/products/laptops-and-netbooks/thinkpad-x-series-laptops/thinkpad-x240/downloads/driver-list/component?name=Chipset`

unmarshalling the data, coming to and from resource manager, as well as the need to virtualize handles.

To show this, we have performed the test using all three mentioned TCTIs. The measurements were done on two significantly different devices. One was a low-cost Raspberry Pi 3 Model B+ [68] board with an *ARM* CPU, and the other one was a high-end dual-CPU *x86* HP Z8 G4 workstation [69]. Both devices included the same TPM 2.0 chip (Infineon SLB9670 [7]). The `tpm2-algtest` was the only application on the system accessing a TPM.

The difference among the different TCTIs was present on both systems. It was the most significant when the operation (measured via direct device access) was too quick (under 25 ms). The measurements are presented in Table 6.1 and Table 6.2. It is worth noting that the TPM on Raspberry Pi performed better than the one on HP Z8 G4, although they are the same model and the HP workstation is significantly more performant than the Raspberry Pi.

Table 6.1: Comparison of average operation duration, in *ms*, using direct device access compared to resource managers (Raspberry Pi with SLB9670).

	tpm0	**tpmrm0**	**tpm2-abrmd**
TPM2_GetRandom	4.15	65.24	4.90
TPM2_Create			
ECC (NIST P256)	213.79	275.57	246.30
HMAC	51.07	112.81	83.58
RSA 1024	1906.09	1968.85	2092.06
RSA 2048	9546.34	10589.42	10080.02
AES 128	48.99	110.63	81.48
TPM2_HMAC			
SHA-256	13.49	135.96	75.17
TPM2_Hash			
SHA-1	4.61	65.78	6.80
SHA-256	4.78	65.82	6.83
TPM2_RSA_Decrypt			
RSA 1024	88.44	210.82	150.26
RSA 2048	185.58	309.06	247.79
TPM2_RSA_Encrypt			
RSA 1024	6.45	128.89	68.37
RSA 2048	9.74	132.51	71.44
TPM2_Sign			
ECC (NIST P256) ECDSA	70.89	193.08	132.12
RSA 1024 RSAPSS	94.26	217.28	155.33
RSA 2048 RSAPSS	192.84	314.99	254.52
TPM2_VerifySignature			
ECC (NIST P256) ECDSA	106.16	229.08	167.67
RSA 1024 RSAPSS	14.39	136.35	75.63
RSA 2048 RSAPSS	22.93	143.36	82.28

Table 6.2: Comparison of average operation duration, in *ms*, using direct device access compared to resource managers (HP Z8 G4 Workstation with SLB9670).

	tpm0	**tpmrm0**	**tpm2-abrmd**
TPM2_GetRandom	4.19	120.88	3.98
TPM2_Create			
ECC (NIST P256)	156.82	272.90	214.78
HMAC	107.55	223.95	165.44
RSA 1024	787.75	1040.78	902.22
RSA 2048	7079.44	6670.52	6885.91
AES 128	102.56	219.02	160.11
TPM2_HMAC			
SHA-256	25.06	257.38	142.91
TPM2_Hash			
SHA-1	6.26	123.30	7.59
SHA-256	6.02	122.43	7.31
TPM2_RSA_Decrypt			
RSA 1024	81.21	314.76	198.07
RSA 2048	439.25	672.93	555.96
TPM2_RSA_Encrypt			
RSA 1024	7.00	240.26	125.25
RSA 2048	11.26	243.87	128.46
TPM2_Sign			
ECC (NIST P256) ECDSA	71.12	304.66	188.98
RSA 1024 RSAPSS	112.13	345.67	229.49
RSA 2048 RSAPSS	502.18	734.64	618.85
TPM2_VerifySignature			
ECC (NIST P256) ECDSA	107.61	340.04	224.96
RSA 1024 RSAPSS	37.10	269.51	154.14
RSA 2048 RSAPSS	70.65	305.72	186.90

7 Conclusion

The goal of this work was to provide an overview of the TPM 2.0 technology and compare it to its predecessor TPM 1.2. We have described possible use cases, where TPM usage might be beneficial. Survey of Linux TPM software stacks by multiple authors was provided along with the tools which can be used for managing and performing operations on the TPM. Linux kernel and its services associated with TPM, like drivers, support for virtual TPMs and built-in resource manager, were presented. We also discussed the support for measured boot in the context of GNU/Linux ecosystem.

The implementation task was to provide a framework for creating bootable images with a user interface, for collecting data for research purposes by laypeople. We have provided a hybrid user interface for a `tpm2-algtest` utility, which benchmarks and collects data about TPM 2.0 devices. The tool was packaged into a bootable image, to make it accessible to the users of all OSes. We integrated capabilities to upload the collected results to a researcher-specified repository or store them on a persistent partition of the bootable disk.

Lastly, an experiment was performed to verify the implementation against a heterogeneous set of real-world systems. We demonstrated that the solution is flexible and works either locally, using a GUI, or remotely via a terminal UI. It can operate on an *ARM*-based embedded system, but also on *x86* PCs, workstations and servers.

7.1 Future work

During the experiment phase, we have discovered a difference in performance which depends on which resource manager is used. The framework might be extended to perform benchmark not only a single method, but to measure them all. This would provide an insight into the heterogenous TPM ecosystem on the Linux platform and possibly uncover performance bugs in resource a managers. The `tpm2-algtest` utility can be extended to support different primary keys, to enable benchmarking TPMs which do not support elliptic curve cryptography.

The framework can be extended to detect if the TPM is susceptible to any published attack (like [35, 33, 34]), as there already exist detection tools. This information would then be presented to the user, which could make it more attractive to volunteers.

Bibliography

1. STRUK, Šimon. *A tool for analysis of supported cryptographic properties of TPM 2.0 chips*. 2019. Master's thesis. Masaryk University, Faculty of Informatics, Brno. Supervised by Petr ŠVENDA.
2. ISO Central Secretary. *ISO/IEC 11889-1:2015 – Information technology – Trusted platform module library – Part 1: Architecture*. Geneva, CH, 2015. Standard. International Organization for Standardization.
3. ARTHUR, Will; CHALLENER, David; GOLDMAN, Kenneth. *A Practical Guide to TPM 2.0: Using the New Trusted Platform Module in the New Age of Security*. 1st ed. Apress, 2015. ISBN 978-1-4302-6583-2,978-1-4302-6584-9.
4. Microsoft, Co. *BitLocker Countermeasures* [online]. 2015 [visited on 2020-04-28]. Available from: `https://docs.microsoft.com/en-us/previous-versions/windows/it-pro/windows-8.1-and-8/dn632176(v=ws.11)?redirectedfrom=MSDN#protection-during-pre-boot-pre-boot-authentication`.
5. *Trusted Platform Module (TPM) 2.0: A Brief Introduction*. Trusted Computing Group, 2015. Available also from: `https://trustedcomputinggroup.org/wp-content/uploads/TPM-2.0-A-Brief-Introduction.pdf`. Technical report.
6. Infineon Technologies, AG. *Iridium 9670 Evaluation Board for OPTIGA™ Trusted Platform Module* [online]. 2020 [visited on 2020-04-28]. Available from: `https://www.infineon.com/dgdl/Infineon-Iridium_1-0_9670_HD-AdditionalTechnicalInformation-v01_01-EN.pdf?fileId=5546d46271bf4f920171ef70667e51b4`. Technical report. Rev. 1.1.
7. INFINEON TECHNOLOGIES, AG. *OPTIGA™ TPM SLB 9670 TPM2.0* [online]. 2018 [visited on 2020-04-28]. Available from: `https://www.infineon.com/dgdl/Infineon-SLB%209670VQ2.0-DataSheet-v01_04-EN.pdf?fileId=5546d4626fc1ce0b016fc78270350cd6`. Technical report. Rev. 1.4.

8. GIGA-BYTE Technology Co., Ltd. *GC-TPM2.0* [online]. 2020 [visited on 2020-04-28]. Available from: https://www.gigabyte.com/Motherboard/GC-TPM20#ov.

9. INFINEON TECHNOLOGIES, AG. *OPTIGA^{TM} TPM SLB 9665 TPM2.0* [online]. 2018 [visited on 2020-04-28]. Available from: https://www.infineon.com/dgdl/Infineon-data-sheet-SLB9665_2.0_Rev1.2-DS-v01_02-EN.pdf?fileId=5546d462689a790c016929d1d3054feb. Technical report. Rev. 1.2.

10. ASUSTeK Computer, Inc. *TPM-SPI (14-1 pin)*. 2018. Available also from: https://dlcdnets.asus.com/pub/ASUS/mb/Add-on_card/E15028_TPM-SPI_card_QSG_WEB.pdf. Technical report.

11. Nuvoton Technology, Co. *NPCT7xxTPM 2.0FIPS 140-2 Security Policy*. 2019. Available also from: https://csrc.nist.gov/CSRC/media/projects/cryptographic-module-validation-program/documents/security-policies/140sp3187.pdf. Technical report. Rev. 1.0.9.

12. Trusted Computing Group. *TPM Certified Products* [online]. 2020 [visited on 2020-03-21]. Available from: https://trustedcomputinggroup.org/membership/certification/tpm-certified-products/.

13. RAJ, Himanshu et al. *fTPM: A Firmware-based TPM 2.0 Implementation*. Microsoft Research, 2015. Technical report.

14. ARM, Ltd. *TrustZone for Armv8-A* [online]. 2019 [visited on 2020-04-28]. Available from: https://developer.arm.com/-/media/Arm%20Developer%20Community/PDF/Learn%20the%20Architecture/TrustZone%20for%20Armv8-A.pdf?revision=c3134c8e-f1d0-42ff-869e-0e6a6bab824f.

15. Intel, Co. *What is Intel ® Management Engine?* [online]. 2017 [visited on 2020-04-28]. Available from: https://www.intel.com/content/www/us/en/support/articles/000008927/software/chipset-software.html.

16. BOSCH, Peter. *Intel Management Engine deep dive* [online]. 36th Chaos Communication Congress, 2019 [visited on 2020-04-28]. Available from: https://pbx.sh/intelme_talk.pdf.

17. Advanced Micro Devices, Inc. *AMD GuardMI Technology* [online]. 2019 [visited on 2020-03-21]. Available from: https://www.amd.com/en/system/files?file=documents/guardmi-infographic.pdf.

18. Microsoft, Co. *MS TPM 2.0 Reference Implementation* [online]. 2020 [visited on 2020-04-28]. Available from: https://github.com/microsoft/ms-tpm-20-ref/blob/master/README.md.

19. Google, LLC. *TPM-JS* [online]. 2020 [visited on 2020-04-28]. Available from: https://google.github.io/tpm-js/.

20. VMware, Inc. *Add a Virtual Trusted Platform Module to a Virtual Machine* [online]. 2019 [visited on 2020-04-28]. Available from: https://docs.vmware.com/en/VMware-vSphere/6.7/com.vmware.vsphere.security.doc/GUID-3D39CBA6-E5B2-43E2-A596-B9A69B094558.html.

21. Microsoft, Co. *Generation 2 virtual machine security settings for Hyper-V* [online]. 2016 [visited on 2020-04-28]. Available from: https://docs.microsoft.com/en-us/windows-server/virtualization/hyper-v/learn-more/generation-2-virtual-machine-security-settings-for-hyper-v.

22. Xen Project. *Virtual Trusted Platform Module (vTPM)* [online]. 2020 [visited on 2020-04-28]. Available from: https://wiki.xenproject.org/wiki/Virtual_Trusted_Platform_Module_(vTPM).

23. BERGER, Stefan; CÁCERES, Ramón; GOLDMAN, Kenneth A.; PEREZ, Ronald; SAILER, Reiner; DOORN, Leendert van. VTPM: Virtualizing the Trusted Platform Module. In: *Proceedings of the 15th Conference on USENIX Security Symposium - Volume 15*. Vancouver, B.C., Canada: USENIX Association, 2006. USENIX-SS'06.

24. ZIMMERMAN, Miriam. *Virtual Trusted Platform Module for Shielded VMs: security in plaintext* [online]. 2018 [visited on 2020-04-28]. Available from: https://cloud.google.com/blog/products/gcp/virtual-trusted-platform-module-for-shielded-vms-security-in-plaintext.

25. BOONE, Jeremy. *TPM Genie: Interposer Attacks Against the Trusted Platform Module Serial Bus*. NCC Group, 2018. Available also from: https://github.com/nccgroup/TPMGenie/raw/master/docs/NCC_Group_Jeremy_Boone_TPM_Genie_Whitepaper.pdf. Technical report.

26. ANDZAKOVIC, Denis. *Extracting BitLocker keys from a TPM* [online]. Pulse Security, 2020 [visited on 2020-04-28]. Available from: https://pulsesecurity.co.nz/articles/TPM-sniffing.

27. HALDERMAN, J. Alex; SCHOEN, Seth D.; HENINGER, Nadia; CLARKSON, William; PAUL, William; CALANDRINO, Joseph A.; FELDMAN, Ariel J.; APPELBAUM, Jacob; FELTEN, Edward W. Lest We Remember: Cold Boot Attacks on Encryption Keys. In: OORSCHOT, Paul C. van (ed.). *Proceedings of the 17th USENIX Security Symposium, July 28-August 1, 2008, San Jose, CA, USA*. USENIX Association, 2008, pp. 45–60.

28. *TCG Platform Reset Attack Mitigation Specification*. 2008. Available also from: https://trustedcomputinggroup.org/wp-content/uploads/Platform-Reset-Attack-Mitigation-Specification.pdf. Technical report. Trusted Computing Group, Inc.

29. FRISK, Ulf. *Direct Memory Attack the Kernel* [online]. DEF CON24, 2016 [visited on 2020-04-28]. Available from: https://media.defcon.org/DEF%20CON%2024/DEF%20CON%2024%20presentations/DEF%20CON%2024%20-%20Ulf-Frisk-Direct-Memory-Attack-the-Kernel.pdf.

30. FRISK, Ulf. *DMA attacking over USB-C and Thunderbolt 3* [online]. 2016 [visited on 2020-04-28]. Available from: https://blog.frizk.net/2016/10/dma-attacking-over-usb-c-and.html.

31. Intel, Co. *Intel ® Virtualization Technology for Directed I/O (VT-d)* [online]. 2012 [visited on 2020-04-28]. Available from: https://software.intel.com/content/www/us/en/develop/articles/intel-virtualization-technology-for-directed-io-vt-d-enhancing-intel-platforms-for-efficient-virtualization-of-io-devices.html.

32. MÜLLER, Tilo; FREILING, Felix C.; DEWALD, Andreas. TRESOR Runs Encryption Securely Outside RAM. In: *Proceedings of the 20th USENIX Conference on Security*. San Francisco, CA: USENIX Association, 2011, p. 17. SEC'11.

33. MOGHIMI, Daniel; SUNAR, Berk; EISENBARTH, Thomas; HENINGER, Nadia. TPM-FAIL: TPM meets Timing and Lattice Attacks. In: *29th USENIX Security Symposium* (*USENIX Security 20*). Boston, MA: USENIX Association, 2020.

34. HAN, Seunghun; SHIN, Wook; PARK, Jun-Hyeok; KIM, HyoungChun. A Bad Dream: Subverting Trusted Platform Module While You Are Sleeping. In: *27th USENIX Security Symposium* (*USENIX Security 18*). Baltimore, MD: USENIX Association, 2018, pp. 1229–1246. ISBN 978-1-939133-04-5.

35. NEMEC, Matus; SYS, Marek; SVENDA, Petr; KLINEC, Dusan; MATYAS, Vashek. The Return of Coppersmith's Attack: Practical Factorization of Widely Used RSA Moduli. In: *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*. Dallas, Texas, USA: Association for Computing Machinery, 2017, pp. 1631–1648. CCS '17. ISBN 9781450349468.

36. Trusted Computing Group. *TPM Software Stack* (*TSS*) [online]. 2020 [visited on 2020-04-28]. Available from: `https://trustedcomputinggroup.org/work-groups/software-stack/`.

37. *TCG TSS 2.0 Enhanced System Level API* (*ESAPI*) *Specification*. 2019. Available also from: `https://trustedcomputinggroup.org/wp-content/uploads/TSS_ESAPI_v1p00_r05_published.pdf`. Technical report. Trusted Computing Group, Inc.

38. The *tpm2-software* community. *`tpm2-tools` Wiki* [online]. 2020 [visited on 2020-04-28]. Available from: `https://github.com/tpm2-software/tpm2-tools/wiki`.

39. IBM. *TPM 2.0 TSS Project Page* [online]. 2020 [visited on 2020-04-28]. Available from: `https://sourceforge.net/projects/ibmtpm20tss/`.

40. Microsoft Research. *TSS.MSR* [online]. 2020 [visited on 2020-04-28]. Available from: `https://github.com/microsoft/TSS.MSR/blob/master/README.md`.

41. BERGER, Stefan. *Virtual TPM Proxy Driver for Linux Containers* [online]. 2020 [visited on 2020-03-21]. Available from: https://www.kernel.org/doc/html/latest/security/tpm/tpm_vtpm_proxy.html.

42. BERGER, Stefan. *swtpm Wiki* [online]. 2020 [visited on 2020-03-21]. Available from: https://github.com/stefanberger/swtpm/wiki.

43. The Linux kernel development community. *Kernel Key Retention Service* [online]. 2020 [visited on 2020-03-21]. Available from: https://www.kernel.org/doc/html/latest/security/keys/core.html.

44. The Linux kernel development community. *Trusted and Encrypted Keys* [online]. 2020 [visited on 2020-03-21]. Available from: https://www.kernel.org/doc/html/latest/security/keys/trusted-encrypted.html.

45. STUEBLE, C.; SELHORST, Marcel. *TrustedGRUB Wiki* [online] [visited on 2020-03-21]. Available from: https://sourceforge.net/p/trustedgrub/wiki/Home/.

46. Rohde & Schwarz GmbH & Co. KG. *TrustedGRUB2 documentation* [online]. 2017 [visited on 2020-03-21]. Available from: https://github.com/Rohde-Schwarz/TrustedGRUB2/blob/master/README.md.

47. Free Software Foundation, Inc. *GNU GRUB Manual 2.04* [online]. 2019 [visited on 2020-03-21]. Available from: https://www.gnu.org/software/grub/manual/grub/html_node/Measured-Boot.html.

48. SUSE, LLC. *YaST the installation and configuration tool* [online]. 2020 [visited on 2020-04-28]. Available from: https://yast.opensuse.org/documentation.

49. The Qt community. *About Qt* [online]. 2019 [visited on 2020-03-21]. Available from: https://wiki.qt.io/About_Qt.

50. The GTK Team. *Getting Started with GTK* [online]. 2020 [visited on 2020-03-21]. Available from: https://www.gtk.org/docs/getting-started/.

51. PRADEEP PADALA. *What is NCURSES* [online]. 2005 [visited on 2020-03-21]. Available from: https://tldp.org/HOWTO/NCURSES-Programming-HOWTO/intro.html#WHATIS.

52. Fedora Project. *Copr User Documentation* [online]. 2020 [visited on 2020-04-28]. Available from: https://docs.pagure.org/copr.copr/user_documentation.html.

53. Fedora Project. *Fedora's Mission and Foundations* [online]. 2020 [visited on 2020-04-28]. Available from: https://docs.fedoraproject.org/en-US/project/.

54. rpkg team. *rpkg documentation* [online]. 2018 [visited on 2020-04-28]. Available from: https://docs.pagure.org/rpkg/intro.html.

55. rpkg team. *module-build documentation* [online]. 2017 [visited on 2020-04-28]. Available from: https://docs.pagure.org/rpkg2/commands/module-build.html.

56. LANE, Brian C. *livemedia-creator Documentation* [online]. 2019 [visited on 2020-04-28]. Available from: https://weldr.io/lorax/lorax-composer/livemedia-creator.html.

57. Red Hat, Inc et al. *Fedora Project Wiki* [online]. 2020 [visited on 2020-04-28]. Available from: https://fedoraproject.org/wiki/Remix.

58. Red Hat, Inc. *Introduction to Anaconda* [online]. 2015 [visited on 2020-04-28]. Available from: https://anaconda-installer.readthedocs.io/en/latest/intro.html.

59. LANE, Brian C. *Lorax Documentation* [online]. 2018 [visited on 2020-04-28]. Available from: https://weldr.io/lorax/lorax.html#how-it-works.

60. PAVLOV, Artemiy I.; CECCHETTI, Marco. *What is SquashFS* [online]. 2008 [visited on 2020-04-28]. Available from: https://tldp.org/HOWTO/SquashFS-HOWTO/whatis.html.

61. SCHMITT, Thomas. *xorrisofs Manual Page* [online]. 2019 [visited on 2020-04-28]. Available from: https://www.gnu.org/software/xorriso/man_1_xorrisofs.html.

62. ISO Central Secretary. *ISO 9660:1988 – Information processing – Volume and file structure of CD-ROM for information interchange*. Geneva, CH, 1988. Standard. International Organization for Standardization.

63. Fedora Project. *`livecd-iso-to-disk` Manual Page* [online]. 2017 [visited on 2020-04-28]. Available from: `https://github.com/livecd-tools/livecd-tools/blob/master/docs/livecd-iso-to-disk.pod`.

64. GNOME Documentation Project. *GNOME Disks* [online]. 2018 [visited on 2020-03-21]. Available from: `https://wiki.gnome.org/Design/SystemSettings/`.

65. BATARD, Peter. *Rufus project page* [online]. 2020 [visited on 2020-03-21]. Available from: `https://rufus.ie/`.

66. GNOME Documentation Project. *GNOME Disks* [online]. 2018 [visited on 2020-03-21]. Available from: `https://wiki.gnome.org/Apps/Disks`.

67. RUBIN, Paul; MACKENZIE, David; KEMP, Stuart. *`dd` Manual Page* [online]. 2020 [visited on 2020-04-28]. Available from: `https://man7.org/linux/man-pages/man1/dd.1.html`.

68. Raspberry Pi Foundation. *Raspberry Pi 3 Model B+* [online] [visited on 2020-04-28]. Available from: `https://static.raspberrypi.org/files/product-briefs/Raspberry-Pi-Model-Bplus-Product-Brief.pdf`. Technical report.

69. Hewlett-Packard Company. *HP Z8G4 Workstation* [online]. 2020 [visited on 2020-04-28]. Available from: `https://www8.hp.com/h20195/v2/GetPDF.aspx/c05527763.pdf`. Technical report. Version 29.

Acronyms

BIOS Basic Input/Output System.

EA Enhanced Authorization.

EFI Extensible Firmware Interface.

EK Endorsement Key.

ESAPI Enhanced System API.

FAPI Feature API.

fTPM Firmware Trusted Platform Module.

GPT GUID Partition Table.

HMAC Hash-based Message AUthentication Code.

I^2C Inter-Integrated Circuit.

IC Integrated circuit.

IOMMU Input–Output Memory Management Unit.

IoT Internet of Things.

LPC Low Pin Count.

MBR Master Boot Record.

NV Non-Volatile.

PCIE Peripheral Component Interconnect Express.

PCR Platform Configuration Register.

RM Resource Manager.

RNG Random Number Generator.

RPM RPM Package Manager.

RSA Rivest–Shamir–Adleman.

SAPI System API.

SHA-1 Secure Hash Algorithm 1.

SPI Serial Peripheral Interface.

SRK Storage Root Key.

STM STMicroelectronics.

TAB TPM Access Broker.

TCG Trusted Computing Group.

TCTI TPM Command Transmission Interface.

TEE Trusted Execution Environment.

TPM Trusted Platform Module.

TSS TPM Software Stack.

VM Virtual Machine.

vTPM Virtual Trusted Platform Module.

A Digital attachments

- `tpm2-algtest` - modified `tpm2-algtest` to report progress status
- `tpm2_algtest_live` - kickstart files, makefile and description for building the live image
- `tpm2_algtest_ui` - Python 3 user interface for `tpm2-algtest` with a `.spec.rpkg` file for building a package
- `collected_results` - results of TPM tests collected during the experiment