Fachhochschule Hannover
University of Applied Sciences and Arts

Faculty IV - Business and Computer Science
Department of Computer Science

TPM 2.0, UEFI and their Impact on Security and Users' Freedom

A thesis submitted in partial fulfilment
of the requirements for the degree of
Master of Science

Thomas Rossow

August 2013

Thesis Committee

First Supervisor

Mr. Prof. Dr. rer. nat. Josef von Helden
Hochschule Hannover (HsH)
Faculty IV
Department of Computer Science
Ricklinger Stadtweg 120
30459 Hannover
E-Mail: josef.vonhelden@hs-hannover.de

Second Supervisor

Mr. Bastian Hellmann, MSc.
Hochschule Hannover (HsH)
Faculty IV
Department of Computer Science
Ricklinger Stadtweg 120
30459 Hannover
E-Mail: bastian.hellmann@hs-hannover.de

Author

Mr. Thomas Rossow, BSc.
Limmerstraße 46
30451 Hannover
E-Mail: rosso@parttimegeeks.net

Statement of Authorship

I hereby declare on oath that this master thesis has been composed by myself, and describes my own work, unless otherwise acknowledged in the text. All references and verbatim extracts have been quoted, and all sources of information have been specifically acknowledged. It has not been accepted in any previous application for a degree.

Hannover, August 15, 2013

Thomas Rossow

Acknowledgements

First and foremost I offer my sincerest gratitude to the supervisor of this thesis, Prof. Dr. rer. nat. Josef von Helden, who supported me with great patience, trust, flexibility and important insights throughout this thesis and beyond and for whom working has been a demanding, educational and yet pleasant experience. To no degree less I would like to thank Bastian Hellmann, second supervisor of this thesis, for his encouragement and useful comments.

Important impulses for looking directions I might have not looked all by myself came from Matthias Kirschner and Dr. Dietmar Wippig.

Lunch, coffee breaks and conversations with my fellow students Leonard Renners and Sebastian Stipkovic were the best recreation I could have imagined during the intense and exhausting phase of reading the TPM, UEFI and other specifications and helped me greatly to stay focused and motivated.

Finally, I would like to thank my dearest friends and family for having been truly inspirational, caring and supportive people for so many years now: Arne Diekmann, Miriam Fitjer, Helge Leinemann, Hubert Lipowski and Dorothee Rossow.

Contents

List of Figures

As a member of the Walkman generation, I have made peace with the fact that I will require a hearing aid long before I die, and of course, it won't be a hearing aid, it will be a computer I put in my body. So when I get into a car – a computer I put my body into – with my hearing aid – a computer I put inside my body – I want to know that these technologies are not designed to keep secrets from me, and to prevent me from terminating processes on them that work against my interests.

– Cory Doctorow

1 Introduction

Almost exactly a year ago Cory Doctorow gave a talk with the title "The Coming Civil War over General Purpose Computing" [32] where Doctorow argued that the same Trusted Computing (TC) technology, a Trusted Platform Module (TPM) in his example, could be used to either give a device owner the certainty that she is in control of her device or to completely take that control away from her. This talk sort of was a sequel to "The Coming War on General Purpose Computing" [31], that Doctorow gave a year before on the 28C3 and in which he underlined the importance of the ability of monitoring the computing devices one owns and having control over the policies being enforced on them. To some extent this discussion is inherent to technologies that enable security, because after all implementing security means implementing restrictions and nothing else. Who is targeted by those restrictions is merely a question of threat modeling.

It is not by coincidence that the question of the owner's control over their device – a question not new to computer science – has been discussed quite frequently over the course of the last very few years by many software activists as well as security researchers; the growing spread of UEFI compliant firmware, mostly promoted by Microsoft's Windows 8 hardware certification requirements, and the publication of the TPM 2.0 specification for public review have pushed this issue back into the conscience of the public. Both the TPM 2.0 and the UEFI specification have potential increasing the security of computing devices, but also bring potential for imposing restrictions on device owners and limiting them in their freedom and rights.

1.1 Contribution

This thesis provides an overview of the key features of the TPM 2.0 specification and a summary of the relevant changes to the TPM 1.2 specification – to the best knowledge of the author, no scientific works covering that topic exist at the time of the publication (August 2013).

The existing work covering UEFI known to the author either is limited almost completely to aspects of Secure Boot or have (firmware) developers as a target group and therefore often focus on details of the implementation rather than examining architectural and design aspects. No previous work seems to provide a comprehensive, abstract overview of the UEFI specification – the work in hand aims to close this gap. Further, this thesis shows what impact UEFI and TPM 2.0 have on secure boot

procedures on their own and how the specifications can be combined in order to implement or enhance secure boot procedures.

This work also discusses the social and economic implications that UEFI and TPM 2.0 or TC technologies in general have or might have if deployed on a broad basis and elaborates recommendations for a responsible deployment of TC technologies. Finally, this thesis provides an analysis and evaluation of the current implementation of UEFI and TPM 2.0 in Windows 8 with regards to the previously defined requirements.

1.2 A Note on Terminology

As the reader will clearly notice, terminology in the field of TC and secure boot procedures is mostly not self-explanatory or intuitive and to make matters worse often used ambiguously across different publications. Although great care was taken to use a consistent terminology throughout this thesis, there might be single cases where terminology is used ambiguously – the reader may excuse this.

The title of this thesis is one example of where terminology is not being used quite aptly; not the freedom of a *user* of a device is subject of this thesis, but the freedom of the *owner* of the device. While in the private sector the owner is usually the (primary) user, this is not necessarily so in e.g. a business environment. To complicate the matter, there is not only an owner of a device in terms of the proprietor of the physical device, but also a owner in terms of what is known as platform ownership in TC. Throughout this thesis the term "owner" means proprietor while owner in a TC context is expressed by using the term "platform owner".

1.3 Outline

The remainder of this work is organized as follows: After an introduction to the most important concepts and notions of TC and secure boot procedures in Section 2, the technical fundamentals of the TPM and UEFI specifications and their significance for secure boot procedures are outlined in Section 3. Section 4 provides a discussion of the effects that the deployment of TC technologies can have on the freedom of device owners and as a result yields a list of requirements for the responsible deployment of TC technologies that is thereupon used to examine and evaluate the current usage of the TPM and UEFI in Windows 8. Section 5 closes this work with a short summary of the findings, recommendations and proposals for future work.

2 Trusted Computing and Secure Bootstrap Procedures

The following Section provides an introduction to the most fundamental concepts and terminology of TC and its applications. Further, the very basics of "regular" and secure computer bootstrap are covered.

2.1 Trusted Computing in a Nutshell

The notion of TC *usually*[1] is used for a set of technologies specified by the Trusted Computing Group (TCG). The TCG, the successor organization of the Trusted Computing Platform Alliance (TCPA), is an industry consortium proclaiming itself an international standards group. It was founded in 2003 and incorporates many of the big players of hardware and software development such as Microsoft, AMD, Intel, IBM and Cisco to name just a few. By their own words the TCG is "a not-for-profit organization formed to develop, define and promote open, vendor-neutral, global industry standards [...] for interoperable trusted computing platforms" [1].

The following paragraphs will provide an overview of the cornerstones of TC. This overview is by no means complete and can, due to the complexity of the TCG specifications, only touch some of their topics very briefly. The TCG architecture overview [41] as well as Grawrock [38] and Lioy [61] be highly recommended as starting points for further research to the interested reader.

The Notion of Trust

In very simple words the TCG specifications aim to enable people or organizations to *trust* computing devices that they interact with. So what does trust mean in the context of TC? The TCG puts it this way:

> Trust is the expectation that a device will behave in a particular manner for a specific purpose. [2]

In fact we trust our devices (or software they run) all the time; we trust our email program to not send our emails to the wrong address, we trust our browser to do

[1] Often other technologies like UEFI Secure Boot are labeled as TC technologies as well.

a correct validation of our bank's Secure Sockets Layer (SSL) certificate. We pretty much have no choice but to trust them, because we have no means of asserting whether our devices are behaving correctly or not. If one were to make an educated decision to trust a device it had to fulfill the following requirements [67]:

- The device must be identified unambiguously
- The device operates unhindered
- There is either first-hand experience of the expected behavior of the device or another trusted device reports such a behavior

The last point is quite interesting since it implicitly states that trust relationships are transitive. Indeed the notion of *transitive trust* (or *inductive trust*) is key to understanding many aspects of TC. If A trusts B and B trusts C, then A automatically trusts C, or expressed more formally: $Trust(A, B) \wedge Trust(B, C) \Rightarrow Trust(A, C)$. Trust relationships can form a whole Chain of Trust (COT), that starts with one trusted component at the bottom, the *trust anchor*, to establish trust to components that no direct relationship exists to.

When speaking about transitive trust on a device level, we first of course need a device that we can trust to report the behavior of other devices correctly. That means we must be able to assert that this device fulfills the requirements listed above. Gathering information about the state of the Operating System (OS) or other software running on a device can only be done from "inside" of the device, i.e. by the software running on the device. Since it is yet undecided whether that information can be trusted or not, we are stuck in a dilemma – the dilemma of the *lying endpoint*, which has been discussed across various publications (e.g. [18, 71]) dealing with computer and/or network security. This is where the concept of a *trusted platform* comes into play.

Trusted Platforms and the Trusted Platform Module

The TCG offers a very short and straightforward definition of a Trusted Platform (TP):

> A Trusted Computing Platform is a computing platform that can be trusted to report its properties. [2]

This definition implies a device to have some special properties not common to "standard" computing platforms. Specifically, a platform must have the following three minimum capabilities in order to be trusted to report its properties and thus to be a TP [41]:

- *Protected capabilities*: Protected capabilities are closely related to the definition of *shielded locations*. Shielded locations are (physical) places on a platform where operations on sensitive data can be carried out securely. The set of commands that has exclusive permissions to operate on those shielded locations constitutes the protected capabilities of the TP.

- *Integrity measurement*: Integrity measurement uses protected capabilities to obtain and store metrics of platform characteristics that affect the trustworthiness of a platform – the measurement can be done in a "cumulative" fashion so that the result is a single value that merely allows a binary trustworthy/not trustworthy decision. Optional *integrity logging* can be used to record the single metrics for later use to e.g. identify single components of a platform failing to be trustworthy. The starting point of measurement is called the Root of Trust for Measurement (RTM). In this work we will be mostly concerned with the Static Root of Trust for Measurement (S-RTM) that begins measuring from a well-known starting state – usually at power-on. An alternative approach is that of a Dynamic Root of Trust for Measurement (D-RTM) where a device can switch from an un-trusted state to a trusted state at any point of its runtime or more precisely can instantiate a new RTM anytime.

- *Integrity reporting*: When the metrics obtained through integrity measurement are transferred to a (remote) third party we speak of integrity reporting. Usually the third party will use the data for *attestation* of the reporting device, i.e. to authenticate the reporting device and make a decision about its integrity/trustworthiness.

Additionally there must be a set of Roots of Trust (ROTs) present on the TP. A ROT is a computing engine responsible for bootstrapping trust; a ROT has to be trusted, because misbehavior of that component is impossible to detect. Typically a TP has three ROTs [41]:

- *Root of Trust for Measurement*: Responsible for measuring the platform's integrity state and storing it into shielded locations. A RTM typically consists of "the normal platform computing engine, controlled by the core root of trust for measurement" [41, p.6]. Whether the RTM is static or dynamic therefore is dependent on the nature of the Core Root of Trust for Measurement (CRTM). A S-RTM is usually implemented as part of the platform's basic input/output system (BIOS) while a D-RTM requires the Central Processing Unit (CPU) to support special instructions to perform partition/core reset [61].

- *Root of Trust for Storage*: The Root of Trust for Storage (RTS) is responsible for providing storage to hold values of integrity measurements (or more precisely a summary of those values) and the sequence of their execution.

- *Root of Trust for Reporting*: The Root of Trust for Reporting (RTR) finally is responsible for reliably reporting information that resides in the RTS.[2]

The definition of the ROTs is tightly coupled with the capabilities of TPs described above in that they enable to establish trust in the data measured, logged and reported. Essentially, a TP can enter a state that is not desired, but is not able to lie about its state. TPs are therefore a solution to the lying endpoint problem.

The bulk of the above TP requirements have been condensed into a single chip, the TPM. The TPM provides protected capabilities and shielded locations and a RTS and RTR [2]. Since the RTM is usually platform dependent[3], a generic chip like the TPM cannot provide one by itself. In combination with a CRTM, however, the TPM provides a complete base for a TP. We will discuss the TPM specifications in detail in Section 3.1.

Trusted Computing Applications

The above definitions do not exist as an end in itself – a wide range of applications can be built upon them. In basically any situation where it is necessary or desired to be able to identify a device unambiguously or where one has to be able to make provable assertions about the integrity state of a device, TC technologies are worth considering. Identification and integrity assertion can be used across networks to provide secure remote access or locally to bind encryption and authentication keys to specific platforms and/or specific integrity states of a platform. This is by far not limited to applications for Personal Computer (PC) platforms but can be applied to embedded, automotive or virtually any computing platform.

One of the most interesting and complex applications for trusted computing is that of Trusted Network Connect (TNC). TNC uses the abstract Network Accesss Control (NAC) architecture to enable a network infrastructure where the network operators can enforce policy based access control. Policies can relate to e.g. the integrity state of endpoints, the observed behavior of the endpoints and to specific users authenticated on a device.

In short TNC aims to provide [43]:

- Platform-authentication
- Endpoint policy compliance (authorization)
- Access policy
- Assessment, isolation and remediation

[2]Terminology is used ambiguously here, even within the TCG documents. There are differing definitions of what constitutes the RTS in [41] and [42] – the definition used in this document matches the one given in the former document.

[3]Actually it is the CRTM, which itself is part of the RTM, that is platform dependent

Particularly interesting about TNC is that it (a) includes the concept of TPs for Platform-Authentication and Endpoint Policy Compliance but also (b) integrates information from other devices on the network such as switches or arbitrary hard- and software components using a dedicated protocol called Interface Metadata Access Points (IF-MAP). All together this allows for a cooperative and distributed security infrastructure in computer networks that is based on open specifications – something that is much desired and necessary in the complex and dynamic environments that are common today, especially in a corporate context.

It also has to be said that critics often associate trusted computing directly with Digital Restrictions Management (DRM), giving the impression DRM was key motivation and key use case for trusted computing technologies [77]. The case of DRM will be discussed more closely in Section 4.

2.2 An Introduction to Secure Bootstrap Procedures

Secure bootstrapping is one of the major use cases for TC technology and tightly coupled with the concept of a TP. It also is the technological basis for more complex applications such as TNC.

Computer Bootstrap 101

The boot-sequence in traditional PC platforms, that is pre-UEFI platforms, starts with the CPU executing code located at address 0xFFFF0 of the device's BIOS. The BIOS performs basic hardware checks (Power-On Self Test (POST)) and hardware enumeration, locates bootable devices, chooses the "correct" device by checking the boot preferences and finally loads and executes the first-stage boot loader code from the selected boot device [54].

The first-stage boot loader's size is limited to the size of a single sector (512 bytes) and contains executable code (446 bytes) as well as the partition table (64 bytes). Since 446 bytes of code is typically not enough to bootstrap an operating system, a second-stage boot loader is loaded and executed. The position of the second-stage boot loader is determined by scanning the partition table for an active record [54].

The second-stage boot loader's job is it to load the kernel. This often involves loading of various components such as file-system drivers, an `initrd` or such. Examples for second-stage boot loaders are GRand Unified Bootloader (GRUB), LInux LOader (LILO) for linux or BOOTMGR for Windows. After the kernel has been loaded and execution handed over to it, user space programs and services are started – the OS is completely booted (see Figure 2.1).

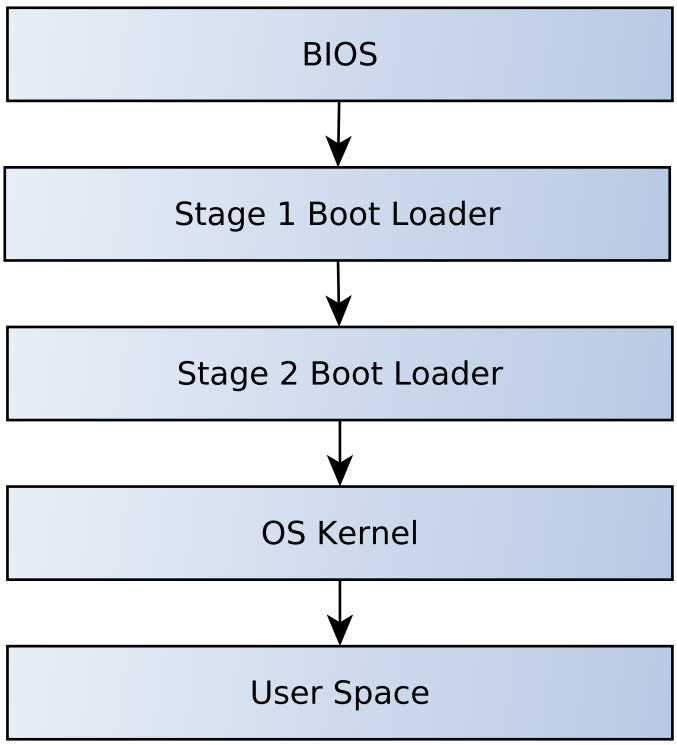

Figure 2.1: Booting a standard PC

Bootstrap procedures for other platforms such as legacy or embedded platforms (or recent PC platforms using UEFI, which will be discussed in Section 3.2) of course differ from this in the nitty details. Roughly they involve the same major steps, however:

1. Hardware initialization
2. Execution of boot loader
3. Passing control to OS kernel

Secure Computer Bootstrap 101

The first serious approaches to secure bootstrap procedures were presented in the early 1990s. Yee [80, 83] proposed the use of cryptographic co-processors that take control over the bootstrapping of the system and ensure the integrity of the system components by comparing the components hash values with known-good values. Other early proposals such as Gasser [36], Lampson [55] and Hartig [45] (and later Arbaugh

[15]) used trusted Read-only Memory (ROM) as ROT and public key cryptography to verify the integrity of the system components.

The basic ideas have not changed very much until today, the TCG, however, has put serious effort into melting the previous efforts into standards. Speaking about "cryptographic co-processors" in the context of secure bootstrap procedures today means speaking about the TPM. The vocabulary of the TCG differentiates between two ways of bootstrapping a platform securely [61, 63]:

- *Authenticated boot*: In an authenticated boot, all system components from the RTM upward are measured and the values stored securely during startup. Those values can later be used by a remote party to verify the system's integrity. The state of the components can, however not be determined on the device itself, meaning that the device can boot into a non-desired state.

- *Secure boot*: In secure boot procedures each component is measured and a trust decision is made on the device itself. If a component fails to be trustworthy, the boot procedure is interrupted. This of course requires additional data on the device. Either known good hash values for all components or key material that can be used to verify signed components.

Or as Challener [22] puts it: Secure boot only allows booting into a trusted state while authenticated boot will merely securely report the state of the boot. Well, instead of "authenticated boot", he uses the term "trusted boot", which is used synonymously in some publications. The TCG, however, is referring to the Trusted Boot (tboot)[4] pre-kernel module when using the term "Trusted Boot" [3].

Additionally to the terms above some vendors have established proprietary names for their implementations of secure bootstrap procedures. So despite the efforts to create a consistent vocabulary by the TCG, there are differing and/or conflicting terms used across publications dealing with secure bootstrap procedures. Throughout this work, we stick closely to the above definitions of the TCG when speaking about secure bootstrap[5] procedures in general and use vendors' proprietary names such as "Verified Boot" (Google) or "Measured Boot" (Microsoft) where appropriate/needed.

If we step back and take a look at the requirements for TPs, it becomes obvious that authenticated boot is built upon integrity measurement, protected capabilities and integrity reporting – exactly the requirements for a TP. So a device capable of performing an authenticated boot in fact is a TP by definition. Secure boot merely requires integrity measurement and therefore not necessarily constitutes a TP. If a device, however, supports secure boot and additionally supports protected capabilities and integrity reporting (usually by means of the TPM), it can be called a trusted

[4] http://sourceforge.net/projects/tboot/

[5] Note that "secure bootstrap" does not mean "secure boot" but is used as an umbrella term for all kinds of integrity measurement enabled boot procedures

platform. Throughout this work, we will refer to secure boot implementations that constitute a TP as *enhanced* secure boot.

In the end both authenticated boot and secure boot can secure the bootstrap process equally well. Which approach is suitable in a given scenario depends on whom is to be assured of the platform's integrity. Authenticated boot enables remote parties to verify a platform's integrity. This is often desired in corporate environments where NAC like TNC are used to control access to sensitive segments of the corporate network. The local user on the other hand does not know whether her device is in an untampered state if she relies on authenticated boot solely, which makes authenticated boot somewhat cumbersome for the average user.

Secure boot on the contrary only allows local verification, which is absolutely sufficient for the average user, but might not be suitable for corporate environments. The trust anchor for devices that perform a secure boot is key material on the device, that usually (but not necessarily) can only be replaced locally, which again is absolutely appropriate for a single user, but time-consuming and expensive in a corporate environment where large quantities of devices need to be updated. If the definition of trustworthy changes for devices that use authenticated boot, updates need to be applied to the remote party verifying the checksums only – most likely a centralized entity.

Apart from the theoretical differences between secure and authenticated boot, there is one huge practical difference: manageability. A "pure" authenticated boot never has been deployed on a broad basis yet. The reason is the complexity of the verification. To establish a complete COT from firmware to application level, huge amounts of binary blobs need to be measured on the device and even greater amounts of known-good values (all possible combinations that constitute a "healthy" device) managed on the remote verifier – these manageability and scalability issues have long been known and pointed out by various publications (see [23, 16, 70]).

Secure Bootstrap Procedures in Practice

An example for a secure boot approach based on digital signatures already in use is Apple's iOS boot. Devices that are intended to run iOS ship with a "Boot ROM" that serves as trust anchor and contains Apple's Root Certificate Authority (CA) public key. Components in the boot path are verified to be digitally signed by Apple before their execution. In case signature validation fails, the system halts. [6]

Google has implemented an enhanced secure boot approach on their Chromebooks called "Verified Boot". It uses a custom firmware as CRTM that ships with key material and performs validation of the components in the boot path using digital signatures. Verified Boot additionally stores information about the state of the device in a TPM. [17]

Also a variety of special purpose devices such as televisions, gaming consoles, etc. use secure boot implementations to restrict the user from booting custom firmware

images. Popular examples are Sony Playstation [29] or Microsoft Xbox [47].

One of the most sophisticated and most developed secure bootstrap procedures currently deployed is the so called “Measured Boot” used in Microsoft Windows 8.

3 Analysis of the TPM 2.0 and UEFI Specifications

The following Section summarizes the technical fundamentals of the TPM 2.0 and UEFI specifications and examines their significance for secure boot procedures.

3.1 TPM 2.0 in a Nutshell

Today the TPM is the most successful cryptographic co-processor available and the most important tool for implementing TPs with roughly 600 million TPM equipped PCs as of December 2012 [9]. Next to providing all functionality needed for implementing TPs (protected capabilities, integrity measurement, integrity reporting and the corresponding ROTs) the TPM was required to be inexpensive in production in order to be accessible for the mass market and support multiple users on a system while still preserving security among them [22].

The first version of the TPM specification was released by the TCPA in 2001 [11]. The TCG began publishing the TPM specification with a new title from version 1.2 in late 2003 [42]. Version 1.2 is still officially relevant, version 2.0 as of now is (June 2013) not officially approved but published as a draft specification for public review. Potential inconsistencies between the contents of the specification described in this work and the final version of the 2.0 specification are likely to be subject of changes applied after the publication of this work.

The next paragraphs will briefly outline the main functions of the TPM 2.0. Functionality of an exceptional relevance for this work will be discussed in a dedicated paragraph each. If not stated otherwise all information is directly extracted from the TPM 2.0 specification.

TPM Architecture and Basic Functionality

Conceptually the TPM is made up of eleven main components as shown in Figure 3.1. These components interact to provide the following services:

- Roots of Trust: The TPM provides shielded locations in its memory and can serve as RTS. An important part of the shielded locations are the Platform

Configuration Registers (PCRs). They are used as storage for integrity measurements performed by the RTM. PCRs are usually[1] only initialized on reset of a device. Values once written into a PCR can never be overwritten or cleared during runtime, instead the old value is *extended* as follows: $\text{PCR}_{new} := \mathbf{H}_{HashAlgorithm}(\text{PCR}_{old} \mid\mid data)$. A TPM can contain one or more Endorsement Keys (EKs) that can only be used inside the shielded locations of the TPM. The EKs (or keys in a EK hierarchy) are used to sign[2] PCR data that leaves the TPM. All EKs are derived from a common seed, the Endorsement Primary Seed (EPS) that serves as RTR.

- Protected Storage: The TPM can contain one or multiple Storage Root Keys (SRKs) that can only be used inside the shielded locations of the TPM. The SRKs or keys derived from a SRK can be used to encrypt data to be stored outside of the TPM e.g. on the hard disk. The only way to decrypt said data is to load it into the TPM since the key material used for encryption can only be used there. By this mechanism the shielded locations of the TPM are extended so to speak – providing means to encrypt bulk data.

- Attestation: The TPM can be used to sign different types of data. Most importantly PCR data (`TPM2_Quote()`)to allow for verification of the platform's integrity. Other special cases are the attestation of the platform's clock and time data (`TPM2_GetTime()`) or command auditing, which allows to create a log of the commands executed on the TPM (`TPM2_GetCommandAuditDigest()` and `TPM2_GetSessionAuditDigest()`). Basically any TPM object (in most cases a key) can be signed (`TPM2_Certify()`).

- Dictionary Attack Protection: The TPM provides protection from dictionary attacks when authorization is needed to access a certain object inside of the TPM such as a loaded key.

- Monotonic Counters: The TPM offers a set of variables that can serve as monotonic counters, that is numbers that can only be incremented or read, but never decremented. Google Chrome OS uses this functionality to prevent rollback attacks of validation keys used for their Verified Boot procedure [17].

- Clock: The clock works analogous to monotonic counters in that its values can only progress but never regress. Possible uses are the prevention of replay or rollback attacks.

[1] In some TPM implementations, those that support a D-RTM, PCRs might be reset during runtime in a "system-specific way" [44, p.190]. In rare cases a platform might even allow certain authorized users to call `TPM2_PCR_Reset()` and thereby initialize a PCR [44, p.79].

[2] For those familiar with the TPM 1.2 specifications: Yes, the EKs are *really* used for signing in TPM 2.0 – see further below for an explaination.

- Random Numbers Generator (RNG): A lot of applications, especially those related to security and/or cryptography need a source of randomness. For example as seeds for the generation of cryptographic keys or as sequence numbers in protocols such as Transmission Control Protocol (TCP). Pseudo-randomness often is predictable enough to create serious security problems. The TPM offers a truly random RNG to be used by applications that need a reliable source for random numbers.

Sealing

Sealing, also known as *sealed-binding*, ties the decryption of data (often a key) to the state of the platform. Whatever data is bound to the platform's state can only be decrypted if a defined set of PCRs contain specific values. Since the keys being used for sealing must be derived from the SRK, decryption of the data is bound both to the platform and the state of the platform.

A typical scenario for sealing is the protection of sensitive data against pre-boot malware located in the BIOS or boot loader. Microsoft's BitLocker e.g. uses sealing to tie the decryption of keys used for hard drive encryption to the state of BIOS, Master Boot Record (MBR), New Technology File System (NTFS) boot sector and boot loader, effectively protecting the data on the drive against espionage by early malware [64].

Ownership Hierarchies

The TPM 2.0 specification introduces three roles to divide control over the TPM: the *platform firmware*, the *platform Owner* and the *Privacy Administrator*. Associated with each role is a *hierarchy* that consists of an *authorization*, a *policy* and *Primary Seed* that is used to generate keys.

Each role has different privileges and controls different resources of the TPM. The platform firmware has full control over the TPM that also means that the platform firmware controls the platform owner's and Privacy Administrator's access to the TPM. In detail the platform firmware can perform the following operations not available to ordinary TPM users:

- Allocation, eviction, modification of *any* Non-Volatile (NV) memory.
- Changing of the Primary Seeds, authorization values, policies and the key hierarchies for all roles.
- Transition of the device into Field Upgrade Mode (FUM) to start the process of updating TPM software such as algorithms.

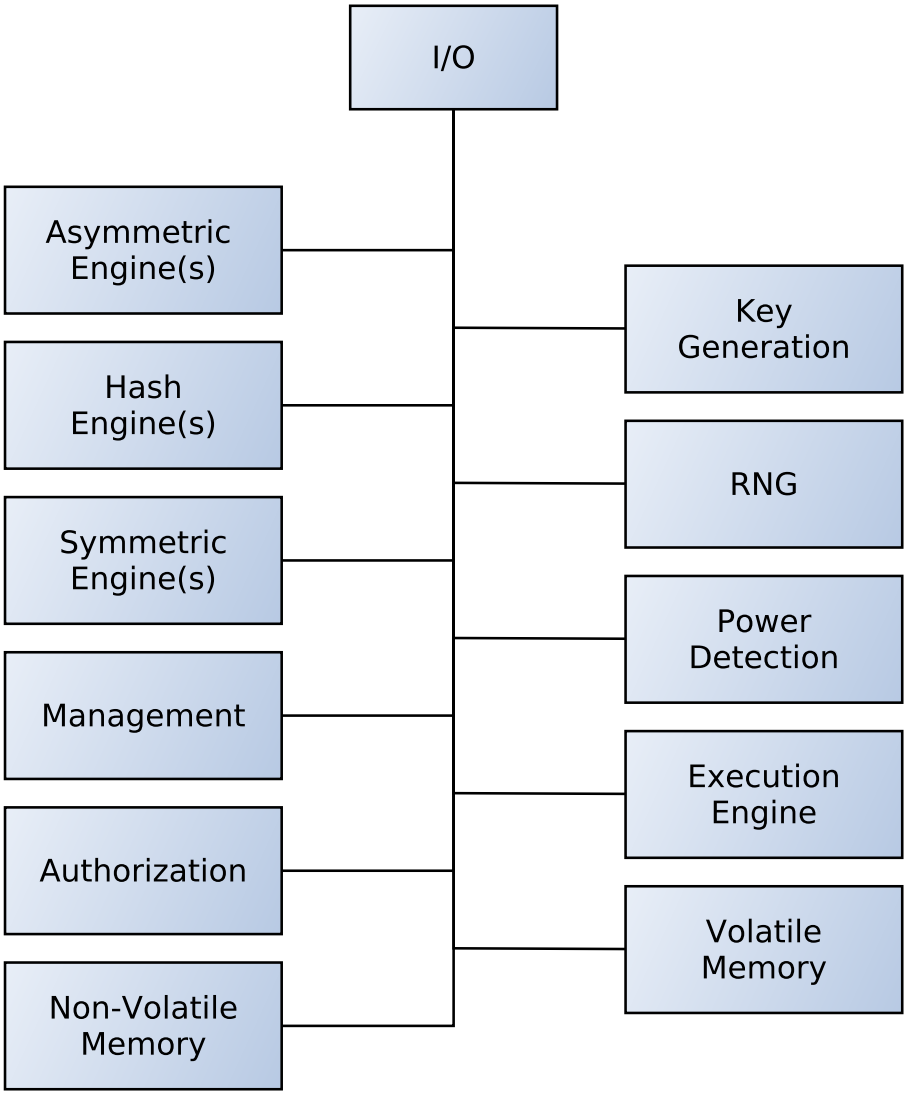

Figure 3.1: TPM architecture [44]

- Configuration of algorithms used by the TPM commands (e.g. of the hash algorithm used for extending the PCRs), of various PCR settings and of the definition of commands that require Physical Presence (PP).

The platform Owner can also control the allocation of TPM NV memory. Further, she is in control of the storage hierarchy – that is all keys derived from the Storage Primary Seed (SPS) or the SRKs for that matter. The Privacy Administrator controls the endorsement hierarchy and privacy aspects during reporting.

The details on how authorization and enforcement of the roles' capabilities is carried out and on the domains of control are complex and not crucial for this work, but it is important to understand the basic concept that the firmware, which is under control of the Platform Manufacturer (PM), is the single most privileged entity in the TPM and that control over storage and reporting are divided between platform Owner and Privacy Administrator.

Enhanced Authorization

Enhanced authorization allows to associate *authorization policies* with TPM objects. An example are the policies for the platform firmware, Owner or Privacy Administrator mentioned above. Authorization policies can also be applied to keys or sealed data blobs. An authorization policy consists of a set of assertions that are combined with the boolean operators `AND` or `OR`. An assertion might for example require selected PCRs or NV variables to contain specific values, demand physical presence of the user or the time and date to be no later than a given value. Assertions can also limit the use of keys to specific operations such as signing only PCR values but no keys or bulk data.

If the evaluation of the authorization policy yields a negative value, the TPM will refuse to carry out the operation requested on or with the objects associated.

Noteable Changes from TPM 1.2 to TPM 2.0

Many of the concepts discussed above were already part of the TPM specification version 1.2 or earlier. Yet, there are a couple of changes or novelties that have been introduced with the 2.0 specification:

- No Opt-in/Opt-out: The 1.2 version and earlier versions explicitly demanded the TPM to be disabled when the device was shipped to the customer. The user had to enable the chip in his computer's BIOS and perform a take ownership of the TPM before it could be used. The 2.0 specification allows it to ship already operating TPMs with a device. Even more the user might not be able to deactivate the TPM since the specification says that "The platform manufacturer decides whether or not the Owner can disable the platform's use of the TPM." [44, p. 67].

- Seeds and Keys: The 1.2 specification defined two keys to be stored inside of the TPM, the SRK and the EK. The 2.0 specification replaces those with *seeds*, large numbers created by the RNG of the TPM, to allow a greater flexibility in choice of algorithms. The EPS and SPS respectively are used in conjunction with a Key Derivation Function (KDF) to create one or more EKs and SRKs. The 1.2 specification allowed only one EK and one SRK. Also a third persistent seed was added, the Platform Primary Seed (PPS)[3]. The EPS, SPS and PPS are under the control domain of the endorsementAuth, ownerAuth and platformAuth respectively. TPM 2.0 further introduces *ephemeral keys*, keys that are generated inside of the TPM and can be used only once. They are used in Elliptic Curve Cryptography (ECC) based Direct Anonymous Attestation (DAA).

- Algorithm Flexibility: In previous versions of the specifications all algorithms to be used for specific functions of the TPM were fixed. TPM 2.0 allows greater flexibility and even "field upgrades" to the algorithms that the TPM supports.

- Remote Attestation (RA): A TPM compliant with the 1.2 specification had a single, non-erasable EK. This EK was linked to a platform certificate that could be used to prove the validity of the TPM to a remote party. Since the EK was bound to the platform, PCR data was never signed with the EK directly but with an Attestation Identity Key (AIK) – a key provided and signed by a trusted third party that vouched for the authenticity of the TPM. This way the identity of the platform was no longer contained in the quoted data, which is crucial for privacy. The TPM 2.0 specifications now go a different way. EKs are not obligatorily linked to the platform certificate anymore. That means that EKs do not necessarily contain the platform identity. The 2.0 specification, however, strongly promotes ECC-based Direct Anonymous Attestation (ECDAA) as an alternative, a method that allows strong privacy.

- Ownership Hierarchies: Prior to the 2.0 specifications there was but a single owner of the TPM or the platform for that matter. Whoever took ownership of the TPM had full control over the platform. Since TPMs were shipped witout an owner (Opt-in), the purchaser was the exclusive owner of the platform. Now, since the platform firmware has authority to add, update or exchange algorithms (that means executable code) on the TPM, great care must be taken no unauthorized person can take control over this role. Otherwise trust in TPM supplied data would be shattered fundamentally. As a consequence the firmware role will always be outside of the domain of the control of the owner. With the firmware hierarchy having other extensive rights, platform ownership now is somewhat divided between the PM and the owner.

[3]The author was not able to find any clues what the PPS is actually used for. It seems it only exists to create a consistent layout of the three hierarchies.

- Enhanced Authorization (EA): TPM 2.0 unitizes the authorization methods for the use, delegated use and migration of objects. Authorization in TPM 2.0 allows for a much greater flexibility through a larger number of authorization methods, such as multi-factor authorization or arbitrary complex authorization policies.

- Reference Implementation: The TPM 2.0 specification contains a reference implementation in C code [4]. This enables developers of TPMs of applications that use the TPM to conduct much more reliable testing than with a written specification only, that leaves room for interpretation.

3.2 UEFI in a Nutshell

The UEFI specification brings fundamental change to the architecture of PC firmware and to computer bootstrap. Not only does UEFI introduce new layers of abstraction that make implementation of firmware drivers much easier, it also creates a foundation to enable secure bootstrap for virtually all notebooks or desktop computers. The following paragraphs describe the UEFI architecture and introduce the most important features of UEFI – especially the ones related to computer security or secure bootstrap.

The UEFI specification [53] stretches over more than 2200 A4 pages. Obviously, not every aspect of the specification will be covered in this work. Any information related to the UEFI specification provided in this section is directly obtained from the UEFI specifications if not denoted otherwise.

UEFI, PI and the Framework – The Big Picture

The history of the UEFI specification dates back to the year 1999 when Intel released their first version of the Extensible Firmware Interface (EFI) specification, that was originally designed to enable booting Itanium-based[4] systems. Additionally to enabling different processor modes, abstracting from hardware coupling that traditional BIOS suffered from and increasing addressable space during system startup, EFI was intended to create a more generic boot interface. After Intel had put serious effort into the EFI 1.0 specification, 11 big players of the computer industry including Intel, Microsoft, IBM, AMD and HP joined to form the UEFI forum in 2005. The UEFI Specification Work Group (USWG) of that organization published the first version of the UEFI specification in the same year. It was almost identical to the initial EFI documents except for a couple of minor reviews and additions. [84]

The UEFI main specification is a pure interface specification that defines data tables containing platform-related information as well as services at boot and runtime to be

[4] A family of 64-bit microprocessors by Intel.

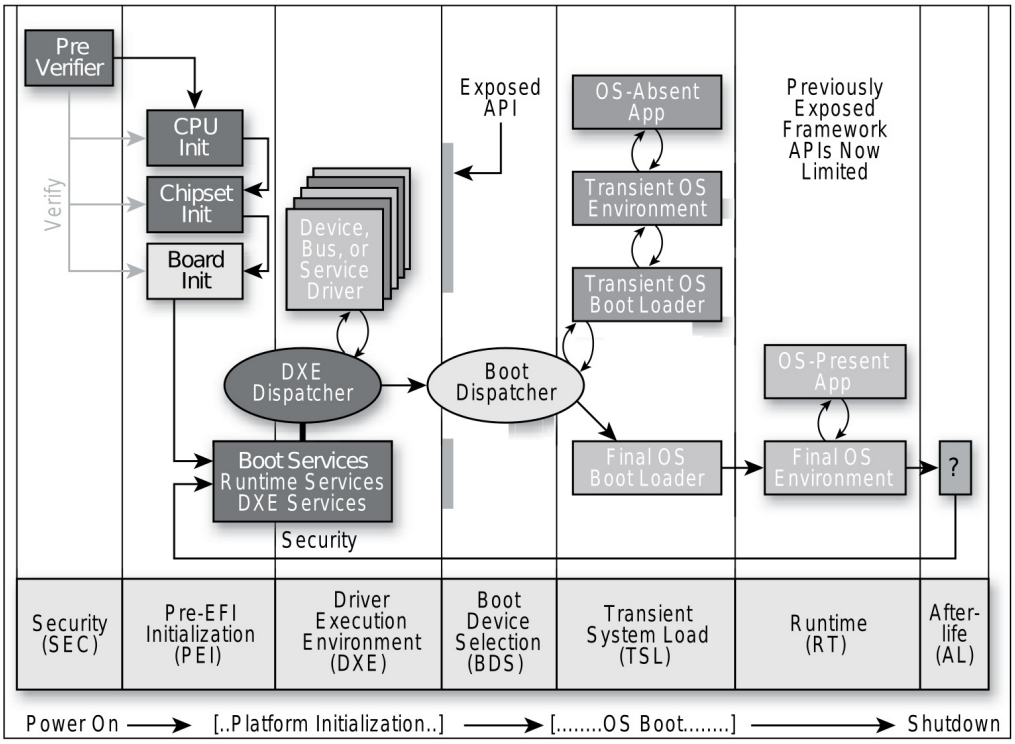

Figure 3.2: Boot flow as specified by the Framework [84]

called by OS loaders or the OS to construct a boot environment and to provide a firmware-to-OS interface. How UEFI is implemented in the firmware of a specific platform and how platform initialization is carried out is completely left to the PM.

Today, UEFI is embedded into a larger design that provides an abstract architecture describing the components/phases necessary to bootstrap (Intel Architecture (IA) based) computing devices that also covers the UEFI Platform Initialization (PI) phase omitted by the UEFI main specification. This high-level architecture is called the Intel Platform Innovation Framework [24] also known as “the Framework”.

The Framework covers the full life cycle of a device’s runtime from system reset to shutdown. Another important piece in the Framework are the UEFI Platform Initialization Specifications [48, 49, 50, 51, 52] published by the UEFI forum’s Platform Initialization Work Group (PIWG). UEFI can not be fully understood without understanding the Framework and the PI specifications at least to some degree. Still neither the Framework nor the PI specifications can be covered in great detail in this work – it will therefore limit itself to the parts indispensable for the understanding of UEFI.

The most important conceptual difference between the UEFI main and the PI spec-

ifications is that the main specification describes interfaces to clients above firmware level (like an OS loader) while the PI specifications describe interfaces to be consumed by clients on firmware level.

Figure 3.2 gives a high level overview of the 7 phases the Framework describes. In the following the phases will be described in greater detail [24]:

- *Security (SEC)*: The SEC phase begins immediately after power-on of a device. Code executed during this phase resides in the CRTM and its job is to authenticate the code executed in the next phase and thereby making sure that any firmware is trustworthy. The SEC phase is also responsible for providing an interface that enables the Pre-Efi Initialization Environment (PEI) to validate further components and thereby continue building a COT.

- *Pre-Efi Initialization Environment (PEI)*: PEI is responsible for initializing a minimum amount of memory and the processor, chip set and motherboard subsystems and to locate and hand over control to the Driver Execution Environment (DXE) code. Typically PEI code consists of a PEI Foundation for a particular processor that is relatively constant across platforms and PEI Modules (PEIMs) that PMs add to meet the needs of their specific hardware configuration. The PEI can perform validations through the interface that SEC exposes and offer a recovery mode in case corrupt firmware is detected.

- *Driver Execution Environment (DXE)*: The DXE phase is meant to not depend on services exposed from earlier phases and to be independent from platform, processor or chip set and may not contain hard-coded addresses. All information about the system's state is passed to DXE in form of Hand-Off Blocks (HOBs) (position-independent data structures) by the PEI phase. A great deal of the system's initialization is done during the DXE phase. The boot devices are identified and appropriate drivers are loaded and executed to read from those devices. Also a set of Boot Services, Runtime Services, and DXE Services are produced and DXE drivers are loaded. DXE exposes an UEFI compliant interface to the later phases, so DXE code is in fact an *implementation* of the (abstract) UEFI main specification.

- *Boot Device Selection (BDS)*: The BDS phase is tightly bound to the DXE phase in that they work together to create system consoles and boot an OS. The phase starts with DXE handing over control to the boot manager (or boot dispatcher). When errors occur during the BDS phase, control can be handed back to the DXE phase to e.g. load additional drivers or start additional services that the boot manager might require to successfully boot an OS.

- *Transient System Load (TSL)*: The TSL phase allows service interfaces to be available to OS loaders before the platform is taken over completely by the OS kernel.

- *Runtime (RT)*: The RT phase begins when the OS calls a special function exposed by the DXE phase that causes all pre-boot services to be terminated. The OS now has full control over the device, while the firmware only offers a set of runtime services via the UEFI interface.

- *Afterlife (AL)*: The AL phase occurs when control is handed back to the firmware by the OS (or by the platform hardware) when e.g. a system reset or an Advanced Configuration and Power Interface (ACPI) sleep state is invoked. AL ends either with the system being reset/shutdown or with reentering the RT phase.

The SEC and PEI phases are covered by the PI specifications. The DXE phase is covered by the UEFI main specifications in that the interfaces exposed by this phase are the ones defined by the specification. The implementation of (a possible) DXE, however, is covered by the PI specifications. So, when speaking about UEFI in terms of the Framework, this means the DXE and BDS phases. This said, it should not be forgotten that DXE and therefore UEFI offers services that can have a life cycle spanning past boot-time.

UEFI Components and Flow of Operations

Now that it is clear how UEFI fits into the life cycle of a device's runtime, one should take a look at the architecture of UEFI itself and its inner workings. In the rough, UEFI consists of the following components [53]:

- *Boot manager*: The boot manager essentially is a piece of firmware implemented by the PM that is responsible for implementing a platform's boot policy and to handle the flow of events in the BDS phase. In detail the boot manager has the following responsibilities:
  - Read the device's boot configuration from Non-Volatile Random-Access Memory (NVRAM) variables
  - Load and execute UEFI images in the appropriate order as specified in the device's boot configuration

- *UEFI images*: UEFI images are executables in Portable Executable / Common Object File Format (PE/COFF) format. UEFI images are loaded and executed in the BDS phase by the boot manager and sometimes also in the late DXE phase by the DXE dispatcher. UEFI images can be subdivided into UEFI *drivers* and UEFI *applications* as shown in Figure 3.3. Applications and drivers differentiate from each other by the way their memory and state are reclaimed. An application's life cycle ends immediately after its execution while drivers can persist across multiple calls – they can be used to construct UEFI services.

- *UEFI services*: UEFI services constitute the UEFI interface and provide an abstraction of the specific platform to the UEFI images invoked by the boot manager. UEFI defines a set of *boot services*, that do not persist the whole runtime of the device but are terminated once the OS kernel takes over control, as well as a set of *runtime services* that can be invoked at any time of the device's runtime. Runtime services provide a slim interface to services such as time and NVRAM access. Boot services make graphical and text consoles and hardware such as block devices and busses available in the early boot environment. A special kind of boot services are *protocol services*. Protocol services are used to extend the functionality of a platform and are specified outside of UEFI. An example for a protocol service is the `EFI_TCG` Protocol specified by [40] that exposes TPM functionality through UEFI and that will be subject in Section 3.3.
- *UEFI OS loaders*: UEFI OS loaders are a special case of UEFI applications. While an application would call the `Exit()` function and return control to the boot manager when it has done its job, an OS loader would instead call the `ExitBootServices()` function and hand over control to the OS or its second stage boot loader. Calling `ExitBootServices()` causes the boot services to be terminated and their resources to be freed; it marks the entry point of the device's RT phase. In case an OS loader fails to terminate successfully, it can call the `Exit()` function to return control to the boot manager.

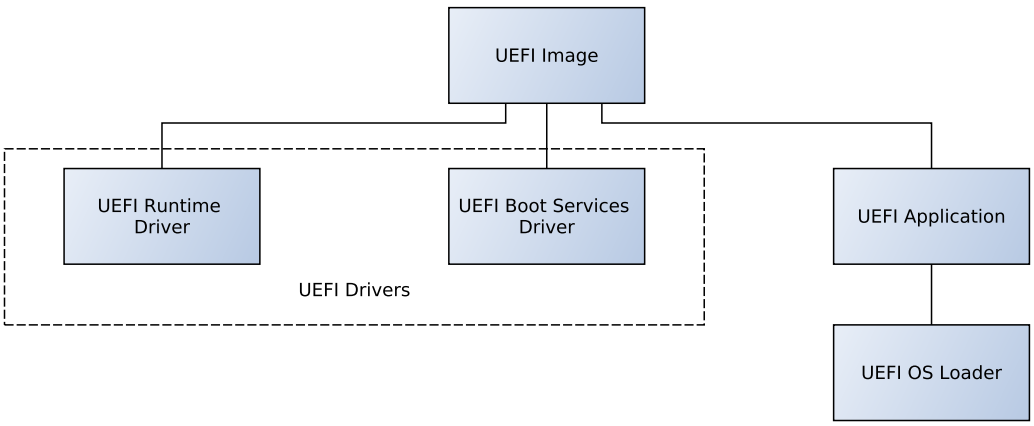

Figure 3.3: UEFI images

Figure 3.4 and 3.5 depict how the UEFI related components specified by the Framework and the PI specifications and the components specified by UEFI interact during system startup and how the platform hardware, the PI firmware, UEFI and the OS are layered.

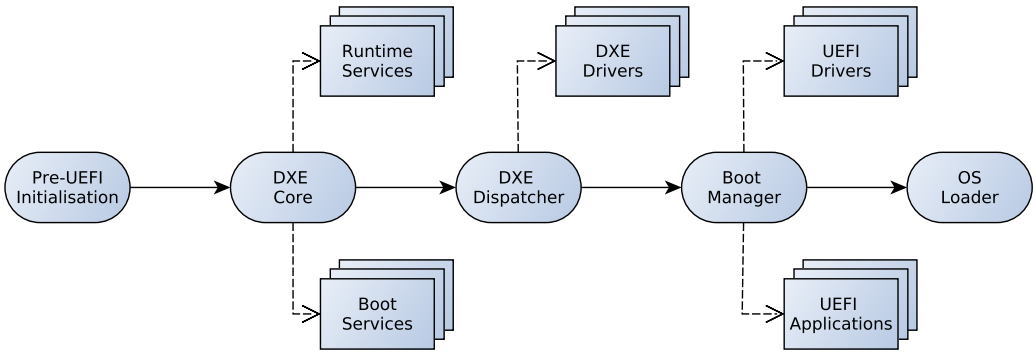

Figure 3.4: UEFI boot flow

UEFI and Computer Security

Additionally to trying to construct a consistent and extensible bootstrap architecture, UEFI was also designed to integrate security measures into the bootstrap process. UEFI defines a set of basic services that can be used to construct complex security solutions. The most important security services UEFI provides are the following [53]:

- *Secure networking*: UEFI implementations completely support the Internet Protocol version 6 (IPv6) protocol. IPv6 support enables the authentication of remote hosts and encryption of the data transfer for network booting using e.g. Internet Small Computer System Interface (ISCSI) or file transfers such as additional UEFI images.

- *User identification*: In contrast to traditional BIOS UEFI allows for user identity management and an authorization scheme in the pre-boot environment. Different authentication mechanisms such as password, smartcards or fingerprint sensors are supported by UEFI, also multi-factor authentication is possible. The access rights of a specific user are defined in the user's access policy record. The access policy can contain rules about the images a user can load, which devices a user can access and which settings (user management, boot priority, etc.) a user can modify.

- *Authenticated variables*: Authenticated variables are NVRAM variables that are enhanced by digital signatures and can therefore be authenticated. They are additionally protected against rollback attacks and can be used to store security-critical system configuration.

- *Driver signing/executable verification*: UEFI images come in PE/COFF format. PE/COFF files can contain embedded digital signatures (see [25] for details) –

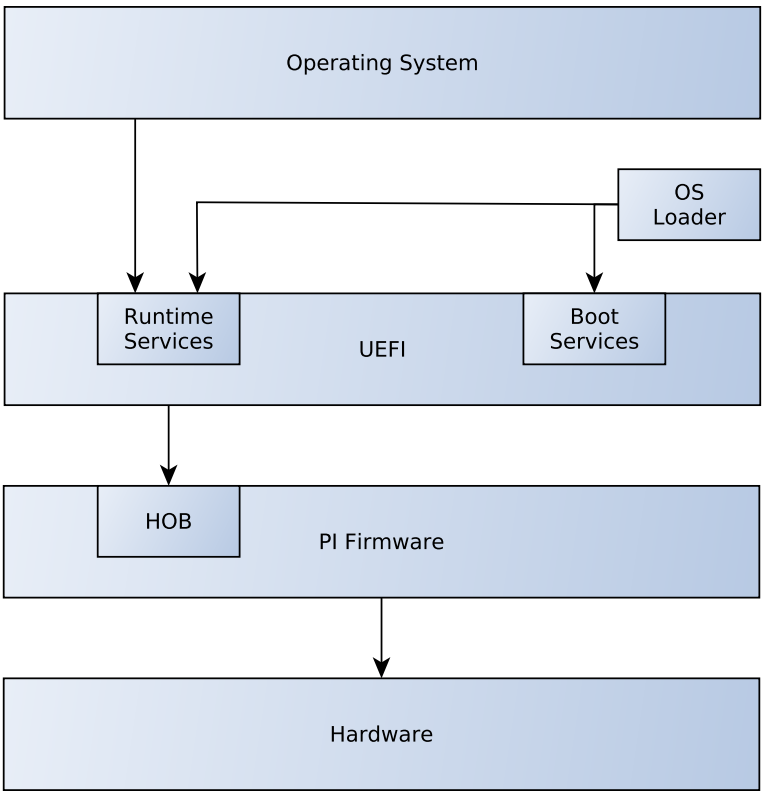

Figure 3.5: UEFI software stack

that is a cryptographic checksum of the binary signed with the private portion of an asymmetric key pair. The embedded signature also contains the public portion of the very key pair for verification purposes. UEFI provides a mechanism to verify the signature and checksum of the binary through its boot services. UEFI also allows for verification of non-signed images by using the images' checksum solely. The authenticated variables `db` and `dbx` are used to store known-good (`db`) and known-bad (`dbx`) checksums for this purpose.

- *Platform key management:* Driver signing alone does not enable a device owner to create any trust relationship to the executables she is verifying. The user needs to be able to depose a set of keys she trusts in the device. If a transitive trust relationship between one of those keys and the key used to sign a UEFI image can be established, the user can trust the image. Those keys are stored in the `db` variable. Changes to this variable (as well as the `dbx` variable) must be authenticated by one of the platform's Key Exchange Keys (KEKs). The platform owner can add or remove KEKs using the Platform Key (PK).

  The public portion of the PK is stored in a protected variable (and so are the

KEKs). If no PK is present (e.g. after assembly or on first shipment), the device is in *setup mode* meaning a (self-signed) PK can be introduced to the system without further authentication. Once a PK has been enrolled, the device enters *user mode* where only the platform owner can modify the set of KEKs or the PK (see Figure 3.6). A device can re-enter setup mode when its PK is cleared. This requires either knowledge PK_{priv} or using a "secure platform-specific method " [53, p. 1452].

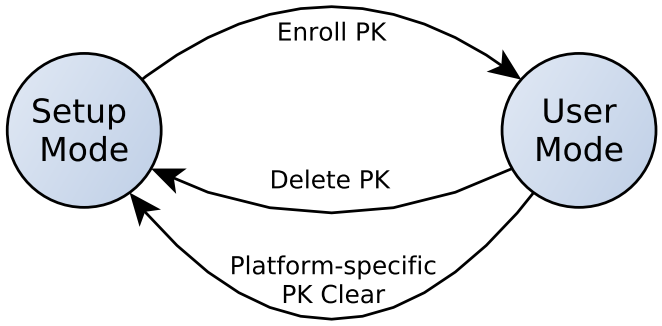

Figure 3.6: Transition between setup and user mode

UEFI Secure Boot

What this work is most concerned with of course is *UEFI Secure Boot*[5] – that is how different security measures in UEFI are combined to implement a secure bootstrap procedure. The fundamental trust anchor in the UEFI architecture is the PK. Knowledge PK_{priv} is required to adjust the device's boot policy which is composed of the authenticated variables listed in table 3.1.

Secure Boot begins with the execution of the boot manager. It is not before this point that necessary UEFI services such as services for image verification are guaranteed to having been started. As soon as the boot manager has control, it reads the `DriverOrder` that contains a sorted list of drivers that have to be loaded before any boot sequence can be initialized. Each driver's image is verified and loaded only if verification is successful.

Verification of an UEFI image works analogous to verification of any digitally signed data. A hash of the data is created, the signature is decrypted with the public key of the signer and the results are compared. If they match, the image has been verified

[5]Note that Secure Boot with capital S and capital B will only be used for the UEFI secure boot implementation throughout this work.

successfully. UEFI additionally performs checks against the white- and blacklist of the signature database.

Once all drivers in this list have been loaded, the boot manager reads the `BootOrder` variable that contains a sorted list of references to *boot options*. Each boot option is a UEFI application that eventually will execute an OS loader in order to hand over control to the OS and is verified just as any UEFI image. When boot options fail to be verified, the next boot option in the `BootOrder` is tried until either an OS is booted successfully or no more entries exist. [53]

Obviously Secure Boot has to rely on secure execution of code before the boot manager takes over control. Most importantly all the services and data required to verify and execute images must be trustworthy. It is up to the PM to implement security below boot manager level. UEFI PI and the Framework provide an architecture to do so, the nitty details of how data or code is authenticated is left to the PM, however.

Name	**Description**
Platform Key	Contains the PK_{pub}, self-signed
Key Exchange Key	Contains the KEK database, signed with PK_{priv}
db	Contains trusted X.509 certificates and a whitelist of valid checksums, signed with KEK_{priv}
dbx	Contains blacklisted X.509 certificates and checksums, signed with KEK_{priv}
Setup Mode	Indicates whether the device is in setup mode or user mode
Secure Boot	Indicates whether image verification is enforced

Table 3.1: Authenticated variables related to Secure Boot [53]

3.3 TPM 2.0, UEFI and Secure Bootstrap

The UEFI Secure Boot feature can be used to verify UEFI applications and OS loaders as soon as the boot manager has taken control, but does not secure the phases prior to the boot manager's execution. This gap is closed by the TCG EFI Platform Specification [39]. The specification defines how pre BDS components are to be measured in order to construct an unbroken COT from the CRTM to the boot manager. In combination with sealing or enhanced authorization policies, the PCR data can also be used to prevent execution of corrupted OS loaders or a corrupted boot manager – providing information to the platform owner whether her device is in a known good state.

A quite promising approach for a TPM based secure boot was recently proposed by Lorenz [62]. In short Lorenz uses TPM NVRAM to store known-good hash-values of the components that constitute the COT and makes their execution conditional

on having the same hash-value during startup. Lorenz's approach seems very fit to verify the pre UEFI environment. If the UEFI Secure Boot variables were stored TPM NVRAM as well, they could be easily integrated in Lorenz's verification scheme.

Tightly coupled with the TCG EFI Platform Specification is the TCG EFI Protocol Specification [40]. It describes the `EFI_TCG` UEFI boot service that exposes TPM functionality to the UEFI environment. Figure 3.7 depicts how `EFI_TCG` and TPM are embedded in the Framework's software stack and the UEFI services. This enables the boot manager to use the TPM in a very simple way and not only verify the images by means of UEFI image verification but to also store the results in a way that can be used for reporting. The same of course applies to applications loaded by the boot manager such as OS loaders.

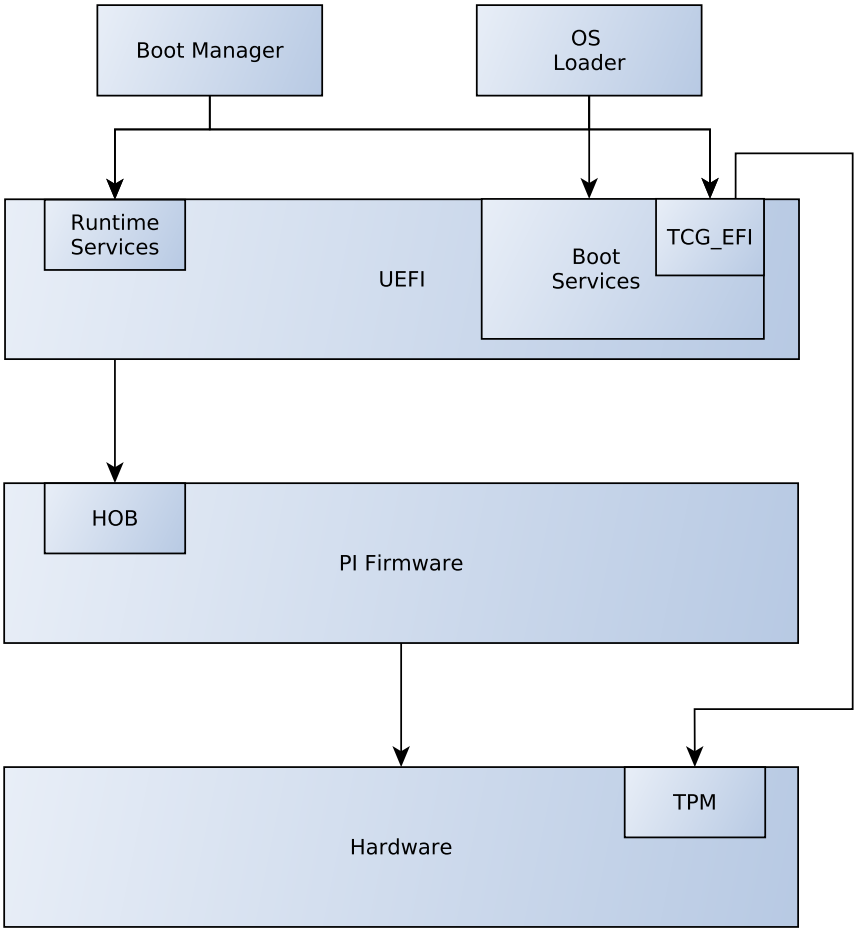

Figure 3.7: TCG EFI

The measurements stored in the PCRs can be used to report the device's state to a remote verifier. The pre UEFI measurements allow a remote verifier to assert

that a complete COT from SEC to OS is intact. If these measurements are fine-grained enough and include e.g. UEFI variables and/or measurements of PEI, DXE and UEFI images, almost every aspect of the platform's configuration can be taken into consideration for trust decisions by the remote party. Of course all components "after" the boot manager can perform additional measurements to provide the remote party with even more information.

Combining UEFI Secure Boot with TPM capabilities can provide an enhanced secure boot solution, that profits from the better scalability of secure boot approaches, but also enables RA – either in a very simple way by only attesting the pre UEFI environment and relying on Secure Boot to extend the COT to the OS correctly, or in more complex ways where binary measurements are extended up to the OS or application level.

4 Trusted Computing Technologies and Users' Freedoms

While the previous parts of this thesis were of a very technical nature, the following section focuses much more on the social and economic implications that the deployment of TC technologies have or might have. After examining the past and present discussion on those implications and elaborating requirements for the responsible deployment of TC technologies, the current usage of TC technologies in Windows 8 is analyzed and evaluated in regards to said requirements.

4.1 Trust or Treachery? – The Controversy on Trusted Computing

While the following discussion seems to focus on the technologies specified by the TCPA or TCG (the TPM basically), almost all criticism applies to UEFI Secure Boot as well and even more so to the combination of TPM and Secure Boot. When it comes to imposing restrictions on the owner, UEFI and TPM are almost equally powerful, since both support secure bootstrapping. As a matter of fact many recent criticism uses the term "Trusted Computing" for both the TCPA/TCG specifications and UEFI Secure Boot. The only exception are scenarios where reporting the platform's state to a third party is required – this is only possible in conjunction with a TPM for reasons discussed in Section 2.

Since the early days of TC, even before the TCG had been founded, there have been fierce debates whether TC is an overall attempt to significantly increase the security of computing devices or an attempt to restrict the owners in their control over their devices by e.g. implementing DRM. Not only the TC in terms of specifications by the TCPA/TCG was subject to criticism, but also Microsoft's Palladium software architecture that hit the headlines around the same time. Palladium, nowadays known as Next-Generation Secure Computing Base (NGSCB), sort of is Microsoft's own initiative towards the realization of TPs.

Since the advent of those technologies many critics suspected that TCPA and Palladium were inseparable parts of some bigger scheme. Cryptographer and computer security specialist Bruce Schneier put it as follows: "Some say it's related. Some say they do similar things, but are unrelated." [73]. Rumor spread fast that both TCPA

and Palladium were designed to implement DRM on computers. Ross Anderson, Professor in Security Engineering at the University of Cambridge Computer Laboratory, claims that it has been explicitly stated by the TCPA that their outspoken goal was to "to embed digital rights management technology in the PC" [12]. Two patents [34, 33] filed by Microsoft in 2001 describing how to build and attest a "DRM operating system" based on TC technology at least prove that early TC designs were highly desired by Microsoft and that they had put significant effort into designing a DRM solution associated with this particular technology. The patents also show that TC technology *can* be used quite well to implement DRM.

While it can be (and often is) argued that it is a legitimate cause for the content and/or software industry to fight against unauthorized copies of their products, the implementation of DRM is highly problematic from a perspective that incorporates the rights of the owners. Microsoft's patents on a DRM enforcing OS make very good examples. One of the proposals is that content should only be available/downloadable to platforms that will "enforce the limitations the provider places on the content" [33]. The decision whether a content provider can trust a device would have to be "based on its loaded components" [34]. Essentially, this would require complete knowledge of a device's software configuration, which raises most fundamental privacy issues.

In order to prevent "untrusted" software from extracting encrypted or otherwise protected content when in memory unprotected, "the digital rights management operating system refuses to load an untrusted program into memory while the trusted application is executing" [33] and "the DRMOS must prohibit the use of certain types of programs and refrain from performing certain common operating system procedures" [34]. If these proposals were implemented, the consequence would simply be the complete loss of control of the owner over her device. In the early days of TCPA many critics expressed great concern about this being the future of computing in case TC technologies were deployed on a broad basis.

Schneier feared that Palladium and TCPA might lead to a situation where the possessors of computers no longer actually *own* their devices. Instead a "variety of factions and companies" would be the ones controlling the computers. The extend that this reality was facilitated by Palladium would be "bad for society" [73]. In his early work about TCPA and TC Anderson describes scenarios that lead to similar conclusions: TC is likely to create a situation where peoples' computers are no longer controlled by themselves, but by a third party. He concludes that that "Although TCPA is presented as a means of improving PC security and helping owners protect themselves, it is anything but." [12].

Free Software activist Richard Stallman, founder of the Free Software Foundation (FSF)[1] and the GNU's Not Unix! (GNU)[2] project, even went so far suggesting to rather use the term "Treacherous Computing" instead of "Trusted Computing" [77].

[1] `http://www.fsf.org/`
[2] `http://www.gnu.org/`

Both Stallman and Schneier took and still take a very strong position that computers are supposed to be general-purpose devices that need to be under the control of the owner and not suffer restrictions imposed by third parties.

It is likely to be directly related to the harsh critique that the TCPA reconstituted itself as TCG later and that matters of the owners' rights were addressed by the organization on their own behalf. Most importantly the TCG included an *Opt-in* and *Opt-out* in the TPM specification meaning that the TPM needs explicit activation[3] by the owner of the device before any of its functionality can be used by the platform firmware or OS and that the TPM can be deactivated by the owner any time she intends to [42].

The importance of Opt-in has been underlined by several experts on computer security such as Grawrock [38, p.89], Challener [22, p.11] (both co-authors of the TPM specification), Schneier [74] and others. Most curiously the German Federal Government has expressed their views on the issue of TC in a whitepaper [30] shortly after the TPM 2.0 specification had been released for public review. In this whitepaper the German Federal Government demands Opt-in as well as Opt-out capabilities for devices that support TC technologies. For devices in public administration and in the field of national and public security it is further "required that under no circumstances may the owner be forced to give up control, even partial control, over a Trusted Computing security system to other third parties outside the public administration's sphere of influence" [30]. The whitepaper's demands are almost identical with the demands of the Free Software Foundation Europe (FSFE) who had published their analysis on Secure Boot [79] earlier the same year. It should be mentioned that the TPM 1.2 specifications were more or less completely compatible with said demands. The role of the firmware hierarchy in TPM 2.0, however, seems to eliminate this compatibility.

Apart from that the TPM 2.0 as well as UEFI do not necessarily take away control from the owner and hand it over to a third party, but they very well allow it – it mostly depends on the implementation of those standards and the organizational environment. As long as the owner remains in control over the PK and/or can enable/disable Secure Boot at will, UEFI does not impose any restrictions on the owner. In case of the TPM everything boils down to the implementation of the platform firmware. Basically the specification allows for both Opt-in and Opt-out although (in contrast to the 1.2 specification) never addressing those topics specifically.

Why is the owner's control over their devices so important? Basically because the loss of control over their own device clears the way for a range of abuse cases. When software/OS vendors are able to decide which software an owner can run on her device they can easily prevent third-party software from being installed on devices and thereby reinforce their own market position [69, 73] and interfere with the owners'

[3]Strictly speaking the specification requires the TPM to be shipped with no platform owner installed. Activation of the TPM however requires a platform owner [42, p.34]

right to choose. But it is not only software that the owner loses control over, when losing control over their device, it is also their data. This, first of all, is a severe threat to the owners' privacy and opens the door for many abuse cases. Espionage by companies or government agencies in restrictive countries are just one example of many conceivable scenarios, so are censorship mechanisms like banning files that have a certain author or contain unwanted keywords [13, 75].

The example of Amazon's kindle has shown that if given the chance, content providers are likely to implement restrictions significantly more strict than the legal conditions would require. In the case of the kindle the owner had no chance to resell or print an e-book, to make single copies for "fair use" or even read an e-book on a different device [76, 5]. In one incident Amazon even went so far to delete books (ironically "1984" and "Animal Farm" by George Orwell) from devices of their customers remotely without notice because of license issues [78].

Another good example is the case of Sony's Extended Copy Protection (XCP) technology, a copy protection mechanism that installed as a rootkit in the owners' computer and interfered with basic functionality like CDROM drivers and lead to severe security holes in the owners' computers [68, 37]. The kindle and XCP examples (and there are more) are not related to TC admittedly, but they are a warning what little respect of the owners' rights and interests can be expected by content providers if in control of their devices and care must be taken here.

Today that almost every device is connected to the Internet most of the time, control over one's devices is even more important than in the past – otherwise the third party having control over the owners' device could impose additional restrictions remotely almost any time without further notice and totally under the radar of the owner [21]. This is not only a substantial threat to owners' rights, but also a huge threat to the owners' security, since attackers will likely try to mimic the "legitimate" third party to gain control of a device [74].

Conclusion

The discussion above has shown that TC technologies bring along a lengthy list of risks. Those risks include possible vendor lock-in on application and OS level, constraints in customers freedom of choice, loss of privacy, censorship and espionage by restrictive governments and a general threat on the future of general-purpose computing.

Those problems are likely to arise when the trust relationship is ill-defined in the way that not the owner of a device is enabled to trust the device by TC technologies, but a third party is. In order for the owner being able to trust her device, the following requirements have to be met when deploying TC technologies:

R1 TC technology must not be enabled by default, but must require the owner to actively enable it (Opt-in).

R2 The owner must at all times be able to deactivate TC technologies once activated (Opt-out).

R3 Any reporting of the state of the owner's device must happen with the owner's knowledge and (educated) consent.

R4 The policy on the device must be viewable, understandable and editable by the owner.

R5 Updates to the policy by third parties must require authorization by the owner.

R6 Key material used for signing/verification in secure boot procedures must be under the full control of the owner and must either originate from a party that the owner chooses to trust himself or through a trustworthy third party.

R7 When purchasing a device even a technically non-skilled owner must be able to understand which restrictions apply to the device she is purchasing and what implications those restrictions have.

4.2 The Status Quo – Windows 8

The influence of Microsoft's hardware certification requirements on the computer market should not be underestimated. Despite a growing popularity of Mac OS and Linux, Windows is still the most popular OS on the market [7]. Manufacturers hardly have any choice, but to comply with those requirements is they expect their devices to be accepted by their customers. So, while especially UEFI is somewhat agnostic in certain details – e.g. whether Secure Boot is enabled by default or the device ships in Setup or in User mode – it is unlikely that a significant percentage of devices will ever be shipped in Setup mode since Microsoft's hardware certification requirements specify otherwise. With the launch of Windows 8 Microsoft has made a major update to their hardware certification requirements. Hardware to be certified for Windows 8 has to fulfill the following requirements related to UEFI and TC technology [27]:

- The device must implement the UEFI specification.
- Secure Boot must be enabled when the device is shipped, the device must ship in user mode.
- The signature database must be initialized and contain Microsoft's X.509 certificate.
- The KEK database must contain Microsoft's KEK.
- Unsigned/Untrusted images may not be executed.

- PK_{pub} must be set by the Original Equipment Manufacturer (OEM) and must be accessible by the OS.
- PK_{priv} must be protected against un-authorized use or disclosure by the OEM.
- Bypass of Secure Boot failures is forbidden.
- On non-ARM systems Secure Boot must be deactivatable via firmware setup without knowledge of PK_{priv}.
- On non-ARM systems it must be possible to modify the contents of the Secure Boot signature databases and the PK via firmware setup e.g. by providing a mechanism to clear Secure Boot database and put the system into setup mode.
- On ARM systems Secure Boot must not be deactivatable.
- On ARM systems the signature database may not contain *any* certificate or signature but Microsoft's X.509 certificate.
- Secure Boot must be rooted in a protected or ROM-based Public Key.
- The hardware certification requirements for Windows 8.1 [28], that have been released just recently, additionally require Windows 8.1 compatible hardware to ship with a TPM 2.0. and to extend the UEFI Secure Boot variables to PCR [7] compliant to Microsoft's Trusted Execution Environment EFI Protocol [26].

On ARM architectures the situation looks pretty bad for the owner: The device is essentially completely under the control of a third party. Neither can the owner disable Secure Boot nor has she any possibility to place own key material on the device. Policy updates can only be applied by a third party and can be carried out without the owner noticing. Additionally, ARM devices are shipped with Windows RT, a variation of Windows 8, which only allows the installation of signed apps via Microsoft's Windows Store and no installation of native or "untrusted" applications whatsoever [8, 56].

On other architectures the owner has at least the choice to disable Secure Boot and/or to modify the key and signature material on the device. This, however, are non-trivial tasks for the averagely skilled owner, especially because the steps necessary can vary from device to device [58]. At the minimum signatures appropriate to boot the desired OS have to be added to the signature database. Possibly the KEK database has to be adjusted beforehand, in some implementations even a complete reset of the whole signature and key data and a clean start from setup mode might be the only way. All this has to be done in the firmware menu – something many owners are unfamiliar with and that some might perceive as intimidating.

Also, it is unlikely that most owners understand the implications that buying a Windows 8 certified device has. Someone unaware of Secure Boot and UEFI inserting

an e.g. Compact Disc (CD) with a Linux installer into his drive and finds the CD unable to boot, she might rather think of the volume as faulty than to assume the device is refusing to boot because of restrictive settings that the she has not made herself.

This all is why Linux distributions like Fedora and Ubuntu had their boot loaders signed by Microsoft so that their OSs can be booted without the owner having to make adjustments to their UEFI setup [60, 72]. Essentially the Linux distributions are in a situation now, where they have to live with either their (potential) owners having to go through a lot of hassle to run their operating system on devices or of putting themselves at the mercy of their competitor. And in fact signing of their boot loaders does not come for free for the distributions, but is bound to contracts and conditions that Microsoft dictates [58]. Additionally, getting a signature from Microsoft has not been easy from the beginning, starting with the need to use proprietary tools to sign and upload the binary to be signed [81].

One of said conditions it seems might be that the OS kernel needs to be signed and that it may only load signed modules. Since Microsoft only signs PE/COFF images, this would require the Linux kernel to include code to verify those kind of images. Linus Torvalds rejected a patch by Red Hat developer David Howells that aimed to add this functionality, rudely commenting on Microsoft's position in the whole process. [20, 59]

Not complying with Microsoft's conditions can result in Microsoft revoking the signatures [57]. Since Microsoft requires their KEK to be contained in the KEK database and UEFI exposes read and write access to UEFI variables through runtime services, updates of the signature databases are possible during runtime without the consent or knowledge of the owner.

Concern about Microsoft preventing alternative OSs from being installed had already been expressed as early as 2011 by e.g. Matthew Garret [35], Ross Anderson [14], the FSF [66] or the Linux Foundation [19]. At that time, however, Microsoft had denied that this would happen [82]. Hispalinux, a Spanish association of Linux users and developers have filed a complaint with the European Commission describing the status quo of Secure Boot as "absolutely anti-competitive" and "de facto technological jail for computer booting systems" [10, 65]. A similar complaint had been filed by Linux Australia before [46]. The outcome of those complaints is yet to await.

4.3 A Future Darkly? – Windows 8 and Beyond

If one looks at the past and present discussions on the deployment of TC technology it becomes obvious that taking a definite position is not easy. TC technologies *can* be beneficial platform security and *can* be deployed in ways that respect owners' freedom of choice and privacy. Both the TPM 2.0 and the UEFI specifications it seems allow suchlike implementations, since they are pretty agnostic to whom controls the

platform. As the case of Windows 8 shows, however, they both allow implementations that are very restrictive just equally well.

In the case of Windows 8 on ARM platforms almost every single requirement for the responsible deployment of TC as defined in chapter 4.1 is violated (R1-R6). On non-ARM platforms the situation is slightly more complicated. While R1, R2 (for TPM 2.0) and R6 are clearly violated, for some of the requirements a conclusive judgement is not or not yet possible.

Updates to the device's policy (R5), reporting of the device's state (R3) *can* be performed by the OS or the firmware without the owner's consent or knowledge, but so far there is no indication that this is common practice already. Ability to view and edit the device's policy (R4) is partially given since at least the reset of the UEFI signature and key databases must be possible. With the adoption of TPM 2.0 this again might change, because at least the policy associated to the Firmware hierarchy is more than likely outside of the owner's domain of control.

Whether technically non-skilled owners are able to understand the restrictions implemented in order to make an educated purchase decision (R7) is a difficult question for ARM and non-ARM platforms likewise. First, technical skill is hard to quantify and in addition the average skill level might shift drastically even in short time due to the fast evolution of information technology. Second, only half of this requirement can be really fulfilled by the vendor (by e.g. labeling their products appropriately) while to at least some degree the educational system, press, Non-governmental organizations (NGOs) or "society in general" has to enable people to properly orient in the digital world. As of now the only hint for the owner is a "Windows 8 compatible" sticker. Since reading and interpreting Microsoft's certification requirements is clearly not feasible for any non-skilled owner, said requirement is rather not fulfilled than fulfilled as of now.

Essentially the strategy Microsoft uses for ARM is very close to the absolute worst-case scenario in terms of owners' rights. When the current architecture is further enhanced by adding a TPM 2.0 chip that neither provides Opt-in nor Opt-out to verify the firmware and the UEFI databases, the approach seems pretty waterproof also. If Microsoft decides to expand the restrictions they impose on ARM devices to other platforms, the future of general-purpose computers is indeed threatened. After all the owner loses the power to choose their operating system and is strictly limited in her choice of applications she can install and run.

The latter is especially problematic because arbitrary additional restrictions can be imposed by demanding app producers to comply with certain policies such as implementing DRM mechanisms for software that is capable of playing music files, removing writings by certain authors, etc. (see Section 4.1 for more examples).

5 Conclusion

The analysis of the TPM 2.0 and UEFI specification and the discussion of the relationship between TC technologies and the device owners rights and freedoms that this work has provided, have shown that those specifications could (for the most part) be used to implement an enhanced secure boot without violating the device owner's rights or freedoms. A review of the implementation that Microsoft aims at with Windows 8 and that is likely to become the standard if not the only implementation illustrates that UEFI and TPM 2.0 can as well easily be used to enforce restrictions that clearly violate the rights that device owners expect to and are supposed to have. It also has become obvious that additional, much more frightening scenarios can be built on top of what is currently in the making.

What are the lessons here? First, that one should not underestimate the practical constraints that manufacturers suffer from; whatever implementations they might prefer from a technological or ethical standpoint, in the end it is economical pressure that shapes their decisions. Authors of specifications that have potential to create abuse cases should be aware of this and pay more attention to the social and economic environment that they release their specifications to. Releasing specifications that can but need not be implemented in a socially acceptable way, might not be enough.

In case of UEFI and TPM 2.0, specifications in a field where a controversy about the impact on the rights of users have been ongoing for years, one would expect the authors to assure the public of the good nature of their specifications by eliminating threats and abuse cases in the specifications themselves from the start. Now, that the specifications have been released as they are and that a quite threatening implementation is on the way, it is up to the public to demand for a change of course.

5.1 Recommendations

To point a direction how an implementation of UEFI and TPM 2.0 that provides a maximum of control over their devices for the owners and protects their freedoms as good as possible, the following recommendations/proposals be made, some of which had already been proposed by Bottomley [19]:

- Devices must ship in Setup mode.
- Secure Boot must be deactivateable on every device.

- On delivery of a device, the TPM should be deactivated by default (Opt-in).
- Firmware on devices that ship with a TPM 2.0 must enable the owner to deactivate the TPM completely (Opt-out).
- Initial setup of Secure Boot for a platform in Setup mode must be made easy for the user. Ideally, the OS installer would assist the owner with the creation of a PK and with adding the appropriate KEK and signatures.
- A standardized workflow to manage KEK and signature databases in the firmware menus is needed. It should be designed in a way that an enables an averagely skilled user to use it.
- An independent CA must be established, its X.509 key should be included in the KEK database. Firmware vendors and OS developers should be able to obtain their own certificates from this CA in order to sign their software.
- A standardized mechanism for booting OSs from external media whose signatures are not contained in the device's signature database must be established. This could be achieved by allowing the owner to add the appropriate signatures to the signature database "on the fly".
- Firmware on devices that ship with a TPM 2.0 must not modify NV memory or any configuration of the TPM without the owner's knowledge and consent.

5.2 Future Work

The analysis this thesis has provided could be used as a basis for the following future work:

The requirements defined in Section 4.1 could be used to develop a more comprehensive list of recommendations for the responsible implementation of UEFI and TPM 2.0 than the one provided above. The result could very well serve as a basis for a review of the specifications with the goal to elaborate proposals how the specifications had to be adjusted to not only allow but enforce the fulfillment of those requirements.

Although this thesis has discussed certain economic and social aspects, it remains the work of a computer scientist and has a strong focus on the technical dimension of UEFI and TPM 2.0. An interdisciplinary discussion of this many-faceted issue involving researches from social, economic, law and computer sciences would produce a much more comprehensive evaluation.

Bibliography

[1] http://www.trustedcomputinggroup.org/about_tcg. retrieved 16.04.2013.

[2] http://www.trustedcomputinggroup.org/developers/glossary/. retrieved 16.04.2013.

[3] http://www.trustedcomputinggroup.org/resources/trusted_boot/. retrieved 24.04.2013.

[4] https://www.trustedcomputinggroup.org/resources/tpm_20_library_specification_faq. retrieved 10.06.2013.

[5] Amazon's kindle swindle. http://www.defectivebydesign.org/amazon-kindle-swindle. retrieved 25.06.2013.

[6] ios security. http://www.apple.com/ipad/business/docs/iOS_Security_Oct12.pdf, October. retrieved 15.05.2013.

[7] Os platform statistics. http://www.w3schools.com/browsers/browsers_os.asp. retrieved 22.07.2013.

[8] Windows rt: Faq. http://windows.microsoft.com/en-us/windows/windows-rt-faq. retrieved 25.07.2013.

[9] Increasing pc security and data integrity - trusted platform module solution from infineon supports windows 8. http://www.infineon.com/cms/en/corporate/press/news/releases/2012/INFCCS2012011-008.html, December 2012.

[10] Secure boot complaint filed against microsoft. http://www.h-online.com/open/news/item/Secure-Boot-complaint-filed-against-Microsoft-1830714.html, March 2013. retrieved 13.07.2013.

[11] Trusted Computing Platform Alliance. Trusted computing platform alliance (tcpa) main specification version 1.1a. TCG Public Review, Dec 2001.

[12] Ross Anderson. Security in open versus closed systems—the dance of boltzmann, coase and moore. *at Open Source Software Economics*, 2002.

[13] Ross Anderson. 'trusted computing' frequently asked questions. http://www.cl.cam.ac.uk/~rja14/tcpa-faq.html, 2003. retrieved 25.06.2013.

[14] Ross Anderson. Trusted computing 2.0. `http://www.lightbluetouchpaper.org/2011/09/20/trusted-computing-2-0/`, September 2011. retrieved 15.07.2013.

[15] William A Arbaugh, David J Farber, and Jonathan M Smith. A secure and reliable bootstrap architecture. In *Security and Privacy, 1997. Proceedings., 1997 IEEE Symposium on*, pages 65–71. IEEE, 1997.

[16] Ingo Bente, Gabi Dreo, Bastian Hellmann, Stephan Heuser, Joerg Vieweg, Josef von Helden, and Johannes Westhuis. Towards permission-based attestation for the android platform. In *Trust and Trustworthy Computing*, pages 108–115. Springer, 2011.

[17] Ingo Bente, Bastian Hellmann, Thomas Rossow, Joerg Vieweg, and Josef von Helden. On remote attestation for google chrome os. In *Network-Based Information Systems (NBiS), 2012 15th International Conference on*, pages 376–383. IEEE, 2012.

[18] Ingo Bente and Josef von Helden. Towards trusted network access control. In *Future of Trust in Computing*, pages 157–167. Springer, 2009.

[19] James Bottomley and Jonathan Corbet. Making uefi secure boot work with open platforms. Technical report, October 2011.

[20] Jon Brodkin. Linus torvalds: I will not change linux to "deep-throat microsoft". `http://arstechnica.com/information-technology/2013/02/linus-torvalds-i-will-not-change-linux-to-deep-throat-microsoft/`, February 2013. retrieved 12.07.2013.

[21] Deutscher Bundestag. Zehnter zwischenbericht der enquete-kommission "internet und digitale gesellschaft". *Projektgruppe Bildung und Forschung: Handlungsempfehlungen, Ausschussdrucksache*, 2013.

[22] David Challener, Kent Yoder, Ryan Catherman, David Safford, and Leendert Van Doorn. *A practical guide to trusted computing*. IBM press, 2007.

[23] Liqun Chen, Rainer Landfermann, Hans Löhr, Markus Rohe, Ahmad-Reza Sadeghi, and Christian Stüble. A protocol for property-based attestation. In *Proceedings of the first ACM workshop on Scalable trusted computing*, pages 7–16. ACM, 2006.

[24] Intel Corporation. Intel® platform innovation framework for efi architecture specification, September 2003. Specification Version Version 0.9.

[25] Microsoft Corporation. Windows authenticode portable executable signature format, March 2008.

[26] Microsoft Corporation. Trusted execution environment efi protocol, 2012. retrieved 26.07.2013.

[27] Microsoft Corporation. Windows hardware certification requirements: Client and server systems, September 2012.

[28] Microsoft Corporation. Windows certification program: Hardware certification taxonomy & requirements - systems, June 2013. Windows 8.1.

[29] Eric DeBusschere and Mike McCambridge. Modern game console exploitation. 2012.

[30] Bundesministerium des Inneren. Federal government white paper on trusted computing and secure boot, Nov 2012.

[31] Cory Doctorow. The coming war on general purpose computation. Talk given at 28C3, the Chaos Computer Congress in Berlin, December 2011. Video recording can be found on `http://boingboing.net/2011/12/27/the-coming-war-on-general-purp.html`, transcript on `https://github.com/jwise/28c3-doctorow/blob/master/transcript.md`.

[32] Cory Doctorow. The coming civil war over general purpose computing. Talk given at Google, August 2012. Video recording can be found on `http://boingboing.net/2012/08/23/civilwar.html`.

[33] Paul England, John D DeTreville, and Butler W Lampson. Digital rights management operating system, December 11 2001. US Patent 6,330,670.

[34] Paul England, John D DeTreville, and Butler W Lampson. Loading and identifying a digital rights management operating system, December 4 2001. US Patent 6,327,652.

[35] Matthew Garret. Uefi secure booting. `http://mjg59.dreamwidth.org/5552.html`, September 2011. retrieved 15.07.2013.

[36] M. Gasser, A. Goldstein, C. Kaufman, and B. Lampson. The digital distributed system security architecture. In *Proceedings of the National Computer Security Conference*, 1989.

[37] Alorie Gilbert. Attack targets sony 'rootkit' fix. `http://news.cnet.com/Attack-targets-Sony-rootkit-fix/2100-7349_3-5956707.html`, November 2005. retrieved 06.07.2013.

[38] David Grawrock. *Dynamics of a Trusted Platform: A building block approach.* Intel Press, 2009.

[39] Trusted Computing Group. Tcg efi platform specification, June 2006. Revision 1.0.

[40] Trusted Computing Group. Tcg efi protocol specification, June 2006. Revision 1.0.

[41] Trusted Computing Group. Tcg specification architecture overview, August 2007. Revision 1.4.

[42] Trusted Computing Group. Tpm main part 1 design principles, March 2011. Revision 116.

[43] Trusted Computing Group. Tnc architecture for interoperability, May 2012. Specification Version 1.5, Revision 3.

[44] Trusted Computing Group. Trusted platform module library part 1: Architecture. TCG Public Review, October 2012. Revision 00.93.

[45] Hermann Hartig, Oliver Kowalski, and Winfried Kuhnhauser. The birlix security architecture. *Journal of Computer Security*, 2:5–21, 1993.

[46] Luke Hopewell. Linux users threaten microsoft with accc. `http://www.zdnet.com/linux-users-threaten-microsoft-with-accc-1339323063/`, September 2011. retrieved 15.07.2013.

[47] Andrew Huang. Keeping secrets in hardware: The microsoft xboxtm case study. In *Cryptographic Hardware and Embedded Systems-CHES 2002*, pages 213–227. Springer, 2003.

[48] Unified EFI Inc. Platform initialization specification volume 1: Pre-efi initialization core interface, October 2012. Version 1.2.1 Errata A.

[49] Unified EFI Inc. Platform initialization specification volume 2: Driver execution environment core interface, October 2012. Version 1.2.1 Errata A.

[50] Unified EFI Inc. Platform initialization specification volume 3: Shared architectural elements, October 2012. Version 1.2.1 Errata A.

[51] Unified EFI Inc. Platform initialization specification volume 4: System management mode core interface, October 2012. Version 1.2.1 Errata A.

[52] Unified EFI Inc. Platform initialization specification volume 5: Standards, October 2012. Version 1.2.1 Errata A.

[53] Unified EFI Inc. Unified extensible firmware interface specification, June 2012. Version 2.3.1, Errata C.

[54] M Tim Jones. Inside the linux boot process. *IBM Developer Works*, 2006.

[55] Butler Lampson, Martín Abadi, Michael Burrows, and Edward Wobber. Authentication in distributed systems: Theory and practice. *ACM Transactions on Computer Systems (TOCS)*, 10(4):265–310, 1992.

[56] Brandon LeBlanc. Announcing the windows 8 editions. `http://blogs.windows.com/windows/b/bloggingwindows/archive/2012/04/16/announcing-the-windows-8-editions.aspx`, April 2012. retrieved 25.07.2013.

[57] Thorsten Leemhuis. Secure-boot-weg von der linux foundation. http://heise.de/-1727468, October 2012.

[58] Thorsten Leemhuis. Gesichtskontrolle. *c't*, 5:170–175, 2013.

[59] Thorsten Leemhuis. Secure boot: Torvalds will keinen support für microsoft-zertifikate im linux-kernel. `http://heise.de/-1810927`, February 2013. retrieved 15.07.2013.

[60] Thorsten Leemhuis. What's new in fedora 18. `http://www.h-online.com/open/features/What-s-new-in-Fedora-18-1783656.html`, March 2013. retrieved 13.07.2013.

[61] Antonio Lioy and Gianluca Ramunno. Trusted computing. In *Handbook of Information and Communication Security*, pages 697–717. Springer, 2010.

[62] Markus Lorenz. TPM-based Secure Boot for embedded Hypervisors. Master's thesis, Technische Universität München, June 2012.

[63] John Marchesini, Sean Smith, Omen Wild, and Rich MacDonald. Experimenting with tcpa/tcg hardware, or: How i learned to stop worrying and love the bear. *Computer Science Technical Report TR2003-476, Dartmouth College*, 2003.

[64] Microsoft Corperation. Bitlocker drive encryption in windows vista. `http://technet.microsoft.com/en-us/library/cc725719(v=ws.10).aspx`, 2010. retrieved 16.06.2013.

[65] Sarah Morris. Exclusive: Linux users file eu complaint against microsoft. `http://www.reuters.com/article/2013/03/26/us-microsoft-eu-idUSBRE92P0E120130326`, March 2013. retrieved 12.07.2013.

[66] Katherine Noyes. Worried about win 8 secure boot? so is the free software foundation. `http://www.pcworld.com/article/242088/worried_about_win_8_secure_boot_so_is_the_free_software_foundation.html`, October 2011. retrieved 15.07.2013.

[67] GJ Proudler. Concepts of trusted computing. *Trusted computing*, 6:11–28, 2005.

[68] Mark Russinovich. Sony, rootkits and digital rights management gone too far. `http://blogs.technet.com/b/markrussinovich/archive/2005/10/31/sony-rootkits-and-digital-rights-management-gone-too-far.aspx`, October 2005. retrieved 06.07.2013.

[69] Mark Dermot Ryan. Trusted computing and ngscb. `http://www.cs.bham.ac.uk/~mdr/teaching/TrustedComputing.html`, 2004. retrieved 12.06.2013.

[70] Ahmad-Reza Sadeghi and Christian Stüble. Property-based attestation for computing platforms: caring about properties, not mechanisms. In *Proceedings of the 2004 workshop on New security paradigms*, pages 67–77. ACM, 2004.

[71] Ravi Sahita, Uday R Savagaonkar, Prashant Dewan, and David Durham. Mitigating the lying-endpoint problem in virtualized network access frameworks. In *Managing Virtualization of Networks and Services*, pages 135–146. Springer, 2007.

[72] Fabian Scherschel. What's new in ubuntu desktop. `http://www.h-online.com/open/features/What-s-new-in-Ubuntu-Desktop-12-10-1730978.html`, October 2012. retrieved 13.07.2013.

[73] Bruce Schneier. Palladium and the tcpa. `http://www.schneier.com/crypto-gram-0208.html`, 2002. retrieved 12.06.2013.

[74] Bruce Schneier. Who owns your computer? `http://www.schneier.com/blog/archives/2006/05/who_owns_your_c.html`, 2006. retrieved 18.04.2013.

[75] Seth David Schoen. EOF: Give TCPA an owner override. *Linux Journal*, 2003(116):14–14, December 2003.

[76] Richard Stallman. E-books must increase our freedom, not decrease it. `http://www.fsf.org/bulletin/2012/spring/e-books-must-increase-our-freedom-not-decrease-it`, 2012. retrieved 25.06.2013.

[77] Richard M Stallman. Free software, free society: Selected essays of richard m. stallman. 2002.

[78] Brad Stone. Amazon erases orwell books from kindle. `http://www.nytimes.com/2009/07/18/technology/companies/18amazon.html?_r=0`, July 2009. retrieved 22.07.2013.

[79] John Sullivan. Free software foundation recommendations for free operating system distributions considering secure boot. Technical report, June 2012.

[80] J. D. Tygar and Bennet Yee. Dyad: A system for using physically secure coprocessors. Technical report, Proceedings of the Joint Harvard-MIT Workshop on Technological Strategies for the Protection of Intellectual Property in the Network Multimedia Environment, 1991.

[81] Steven J. Vaughan-Nichols. Linux foundation uefi secure boot key for windows 8 pcs delays explained. `http://www.zdnet.com/linux-foundation-uefi-secure-boot-key-for-windows-8-pcs-delays-\explained-7000007841/`, November 2012. retrieved 03.07.2013.

[82] Christof Windeck. Linux-community fürchtet windows-"verdongelung". `http://heise.de/-1347168`, September 2011. retrieved 15.07.2013.

[83] Bennet Yee. *Using secure coprocessors*. PhD thesis, IBM, 1994.

[84] V. Zimmer, M. Rothman, and S. Marisetty. *Beyond BIOS: Developing with the Unified Extensible Firmware Interface*. Intel Press, 2010.

Glossary

ACPI Advanced Configuration and Power Interface.

AIK Attestation Identity Key.

AL Afterlife.

BDS Boot Device Selection.

BIOS basic input/output system.

CA Certificate Authority.

CD Compact Disc.

COT Chain of Trust.

CPU Central Processing Unit.

CRTM Core Root of Trust for Measurement.

D-RTM Dynamic Root of Trust for Measurement.

DAA Direct Anonymous Attestation.

DRM Digital Restrictions Management.

DXE Driver Execution Environment.

EA Enhanced Authorization.

ECC Elliptic Curve Cryptography.

ECDAA ECC-based Direct Anonymous Attestation.

EFI Extensible Firmware Interface.

EK Endorsement Key.

EPS Endorsement Primary Seed.

FSF Free Software Foundation.

FSFE Free Software Foundation Europe.

FUM Field Upgrade Mode.

GNU GNU's Not Unix!.

GRUB GRand Unified Bootloader.

HOB Hand-Off Block.

IA Intel Architecture.

IF-MAP Interface Metadata Access Points.

IPv6 Internet Protocol version 6.

ISCSI Internet Small Computer System Interface.

KDF Key Derivation Function.

KEK Key Exchange Key.

LILO LInux LOader.

MBR Master Boot Record.

NAC Network Accesss Control.

NGO Non-governmental organization.

NGSCB Next-Generation Secure Computing Base.

NTFS New Technology File System.

NV Non-Volatile.

NVRAM Non-Volatile Random-Access Memory.

OEM Original Equipment Manufacturer.

OS Operating System.

PC Personal Computer.

PCR Platform Configuration Register.

PE/COFF Portable Executable / Common Object File Format.

PEI Pre-Efi Initialization Environment.

PEIM PEI Module.

PI UEFI Platform Initialization.

PIWG Platform Initialization Work Group.

PK Platform Key.

PM Platform Manufacturer.

POST Power-On Self Test.

PP Physical Presence.

PPS Platform Primary Seed.

RA Remote Attestation.

RNG Random Numbers Generator.

ROM Read-only Memory.

ROT Root of Trust.

RT Runtime.

RTM Root of Trust for Measurement.

RTR Root of Trust for Reporting.

RTS Root of Trust for Storage.

S-RTM Static Root of Trust for Measurement.

SEC Security.

SPS Storage Primary Seed.

SRK Storage Root Key.

SSL Secure Sockets Layer.

TC Trusted Computing.

TCG Trusted Computing Group.

TCP Transmission Control Protocol.

TCPA Trusted Computing Platform Alliance.

TNC Trusted Network Connect.

TP Trusted Platform.

TPM Trusted Platform Module.

TSL Transient System Load.

UEFI Unified Extensible Firmware Interface.

USWG UEFI Specification Work Group.

XCP Extended Copy Protection.