



White Paper by Intel Corporation and IBM Corporation

Trusted Platforms

UEFI, PI and TCG-based firmware

Vincent J. Zimmer
Intel Corporation

Shiva R. Dasari
Sean P. Brogan
IBM

September 2009

Executive Summary

This document provides an overview of the interactions of the Trusted Computing Group (TCG) [TCG Overview], the firmware standards work within the Unified Extensible Firmware Interface (UEFI) Forum, and implementation practices of UEFI PI-based [UEFI Book][UEFI Shell Book][UEFI Overview] implementations. In addition, this paper will provide some use-cases and implementation examples of this technology in addition to the industry threats that motivate the design of this class of technology.

This paper is mainly intended for Hardware, firmware, software, and BIOS engineers. But beyond this audience, some of the information in this paper will be valuable for IT decision makers, marketing, and other parties.

The goal of the paper is to take away an understanding of the motivations behind trusted platform design, the terminology of trust, how to navigate the Trusted Computing Group set of specifications and technology that relate to platform, impact on platform firmware and UEFI, instances of deployment in the market, and some future possible directions for hardware and firmware.

Table of Contents

Overview	2
Problems to solve	3
Security architecture and trust	5
Trusted Computing Elements	10
What is platform firmware/BIOS	15
Firmware and trusted computing	19
Operating System usage of SRTM	26
Platform Perspective	28
Best Practices for Construction	30
Evolution of platform trust	36
Conclusion	44
References	45

Overview

This section will describe an overview of the ensuing chapters.

Description of each chapter

The section on “Problems to Solve” will give an overview of the attacks and industry concerns that motivate the design and deployment of trusted computing.

After the problem statement has been described, “Trusted Computing Elements” will provide taxonomy of the technology elements and the goals that are used to address many of the concerns brought up in the “Problems to Solve” chapter.

The next chapter provides an overview of platform firmware, including conventional PC/AT BIOS and UEFI-based implementations.

After the description of BIOS, the implementation of platform firmware and trusted computing technology is treated in the chapter “Platform firmware and trusted computing technology.”

Following the trusted platform firmware discussions, some operating system use-cases of the preceding technology will be described in “OS Usages of SRTM.”

Given the above technology descriptions and use-cases, a chapter on an actual platform that implements many of these ingredients can be found in “Platform Perspective – Putting it all together.”

After the platform example is shown, the topic of best practices and assurance around platform and firmware design will be discussed in “Some best practices for robust platform construction.” This will provide insight on ‘how’ to implement the various features and standards discussed earlier.

Once the problems, technology, implementation instances and guidelines of today’s practices have been reviewed, the following chapter will provide some glimpse into the future via “Evolution of platform trust.” This will include futures in both the trusted computing and UEFI space.

The conclusion will provide a recap of some key points and summary of the items treated in the preceding chapters.

Summary

This chapter has provided a roadmap to reviewing the successive sections of this paper.

Problems to solve

In this initial chapter on “Problems to Solve,” some motivation for the development of platform trust is covered. Specifically, the many concerns found in the industry around security are discussed.

Problem background

In the past, with non-networked machines and trusted operators, platform trust was only an issue for perhaps high-security government installations. Even the advent of the Arpanet (predecessor of the internet) assumed friendly, trusted end-points. In the 21st century with the planet interconnected and the movement of so much business to computers and computer networks, though, this safe world of computing cannot be assumed.

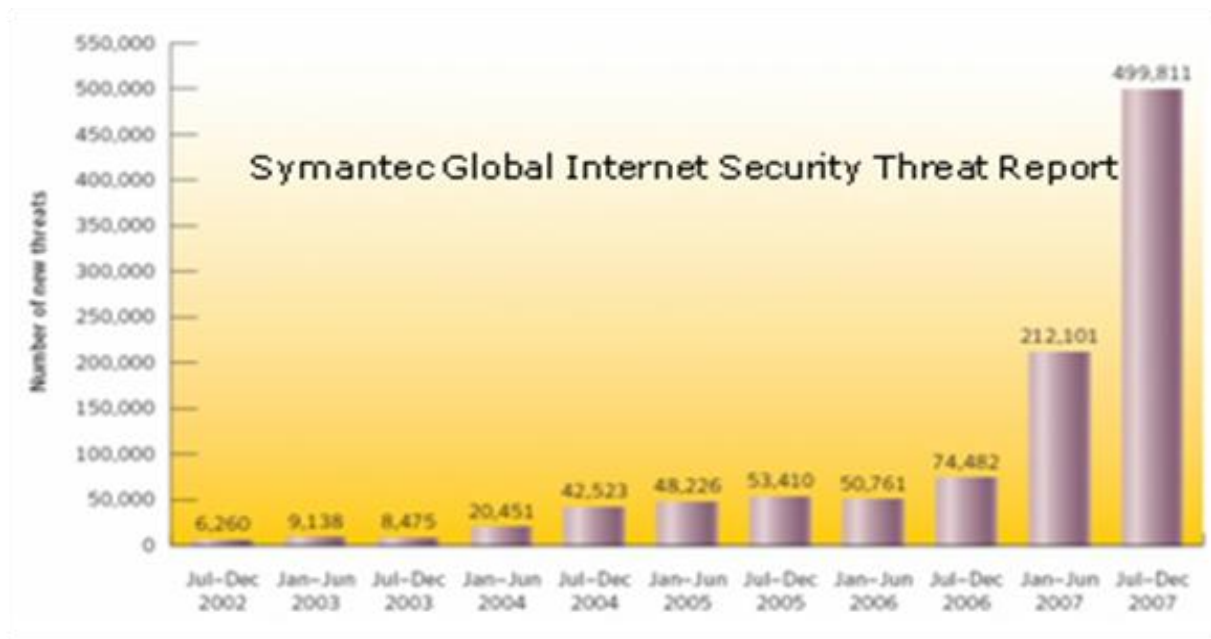
As such, some problems to solve in the computing platform include:

- **Protection Against Malicious Code**
 - This includes concerns for network-based Worms, patching to address flaws and vulnerabilities
- **Business Process Compliance**
 - Regulatory requirements from EU Privacy, Sarbanes-Oxley, Basel II, HIPAA [HIPAA], GLB, Payment Card Industry (PCI) [PCI]
This includes cases where end-user information, corporate data that could imperil a company (i.e., shareholder), HIPAA for medical documents
- **Internal/External Access and Data Protection**
 - Secure provisioning of Infrastructure/Users
 - Managing access/identity across disparate applications

So this taxonomy above describes the new legislation and an ideal list of concerns.

These issues are making security a growing concern for IT. The cost of attack is significant and can include downtime, regulatory, lost customers, and damage to the business’s image. As shown in the figure below, the industry is going through a shift in type of attackers and attacks, namely from social notoriety to targeted criminal activity with focus on stealth.

Finally, virtualization and cloud computing have inherent security requirements, including abstraction of physical hardware and a multi-tenancy movement that implicitly requires audit and security.



Source: Symantec

Figure 1 Symantec data on threats

Key point

Security isn't hype, but a real market need.

Summary

This chapter has provided a brief introduction to some of the business problems and threats that will motivate the development of trusted platforms.

Security architecture and trust

This section of the paper will provide an overview of trust, a security architecture, and a description of a series of goals.

Trust overview

We begin the discussion of trusted platforms with some background on “Trust.” Specifically, the definition of Trust



Figure 2 Dictionary definition of 'Trust'

- **Trust**
 - An entity can be trusted if it always behaves in the expected manner for the intended purpose
- **Measurement**
 - The process of obtaining the identity of an entity
- **Security**
 - “. . . maintenance that ensure a state of inviolability from hostile acts or influences”

In fact, trust is an amalgam of several elements of the platform that span the enterprise to consumer, including reliability, safety, confidentiality, integrity and availability.

Elements of TRUST



Figure 3 The elements of "Trust"

Where should the solution reside, given the “Problems to be solved”, and some of the capabilities like security, trust, and measurement to help effect the solution?

In fact, the implementation of trust and security entail a security architecture that spans the entire system, starting at the base with hardware and spanning all of the way to the end-user application.

The figure below shows all of the layers of a security architecture. The “network” layer is broken out with a few examples, such as protocols (SSL, IPsec); this paper will not delve too deeply into this layer. The “firmware” layer is highlighted to show that a single-layer of security is not sufficient.

One goal of a security architecture is to provide trust.

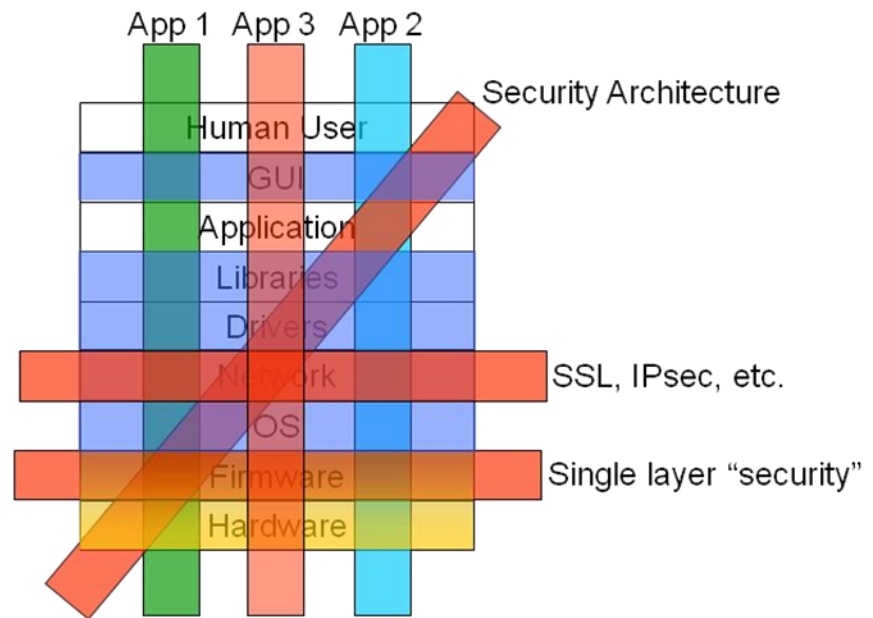


Figure 4 All layers of a security architecture

In fact, the scope of this paper largely treats firmware. Some description of hardware elements and interaction will be provided. As such, the figure below highlights the area that this paper discusses in more depth.

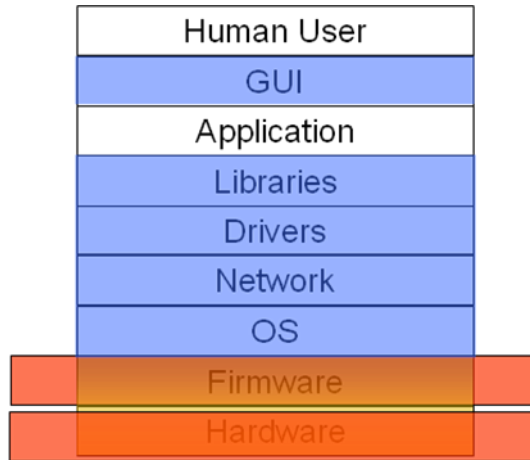


Figure 5 Layers examined in this paper

As seen in the figure above, all layers are important, but if don't have firmware/hardware assurance, you cannot have a security architecture. As the logicians would say, it's 'necessary but not sufficient' to have appropriate design in these layers. And as will be described later, the layer of hardware and firmware provide a 'root of trust' for the rest of the security architecture.

So now that we have trust, security, measurement, and a layered picture of the security architecture, the goals of the security architecture and assets that are protected are as follows.

The first security goal is 'integrity,' and this entails the protection of content and information from unauthorized modification. The next goal is 'Authenticity', and this provides guarantee or assurance in the source of the code or data. Another important goal is "Availability," or the ability to ensure behavior and the liveness of system. Availability also protects from destruction or denial of access. And finally, another goal is "Confidentiality," or the protection of information from unauthorized access.

Through the subsequent discussion of trusted platforms and UEFI, some of these integrity, authenticity and availability goals will be discussed in more details.

It is outside the scope of this paper to describe confidentiality since this is typically a concern of higher-level applications, but errors in lower layers of the trusted platform may imperil this goal. Specifically, this relates to the introduction of vulnerability via a flaw in integrity or authenticity implementations of a layer that wants to provide Confidentiality (say an 'application') when the hardware or firmware or network underneath is errant.

A final item that will be discussed in this paper includes reads on a final goal that spans all of the above, namely 'assurance.' By 'assurance' we mean having some guarantee on the correctness of an implementation. And for this study, assurance will be treated in detail for the case when platform firmware and trusted computing hardware elements are the embodiment of the platform.

And given the 'trust' definition above, we see that these features are especially important in the enterprise, such as a high-end server, where reliability and safety goals are co-equal to the other concerns like integrity and confidentiality.

Summary

This chapter reviewed what is meant by trust, sample security architecture, and various security goals.

Trusted Computing Elements

This section of the paper will provide taxonomy of the terms and elements used in trusted computing.

TPM and Trust

In building out of the hardware layer of the security architecture, one problem with open platforms is that there hasn't been a location on the system to have a 'root of trust.' The Trusted Platform Module (TPM) and the infrastructure around this component are an industry attempt to build a series of 'roots of trust' in the platform.

The maintenance and evolution of the capabilities of the TPM are managed through an industry standards body known as the Trusted Computing Group (TCG) [TCG]. The TCG members include systems manufacturers, TPM manufacturers, CPU & chipset vendors, operating system vendors, and other parties that contribute hardware and software elements into a trusted platform. IBM includes one vendor that spans many of these categories. Intel also participates in the TCG as CPU & chipset vendor.

To begin, what is a TPM?

A TPM is a "Trusted Platform Module". It features a series of protected regions and capabilities. Typically, a TPM is built as a microcontroller with integrated flash/storage that is attached to the LPC bus on PC, but it can also be a virtual device or more deeply integrated in the platform chipset complex. The TPM interacts with system through a host interface. The TIS in TCG PC [TCG TIS] Client working group describes the memory-mapped I/O interfaces; the TIS is just one such interface. The TPM Main specification [TPM Main Specification] describes the "ordinals" or the byte stream of commands that are sent into the TPM. These commands are the required actions that a TPM must carry out in service of the host.

Below is a picture of some of the specifications that describe the TPM and its integration into the platform.

TCG Doc Roadmap, with EFI

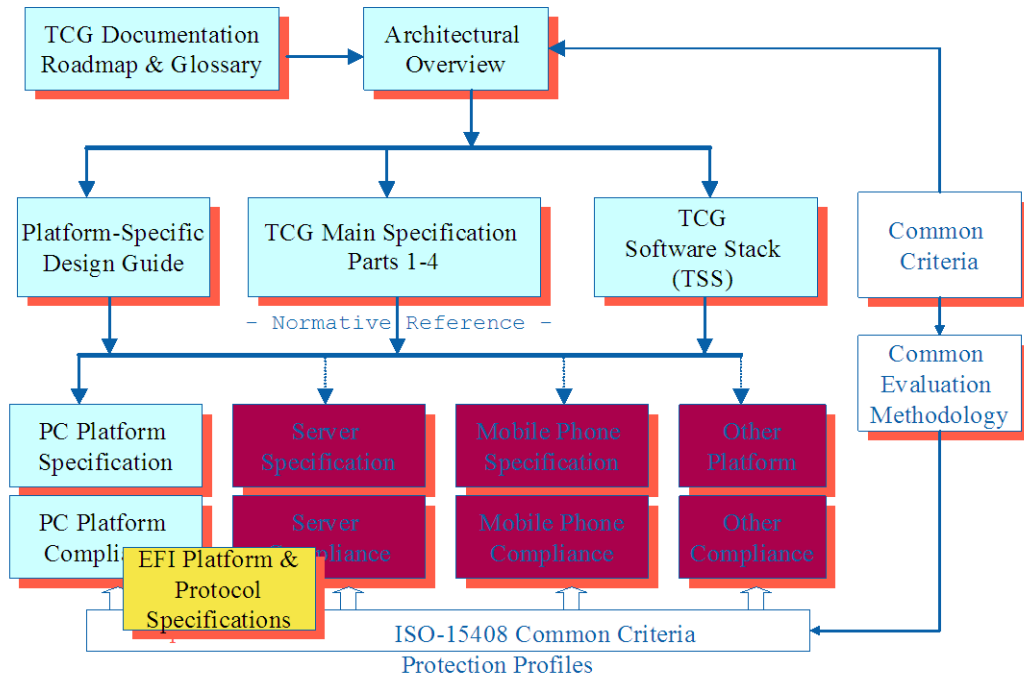


Figure 6 Specification roadmap

The interoperability of the Trusted Computing elements is managed through the Trusted Computing Group (TCG) and a series of specifications. For purposes of this review, the TPM main specification, platform design guides, protection profiles, and the EFI collateral will be of interest, as highlighted above.

- TCG defines TPM's functionality
 - Protected capabilities
 - Shielded locations



- Not the implementation
 - Vendors are free to differentiate the TPM implementation
 - Must still meet the protected capabilities and shielded locations requirements

Figure 7 TPM overview

The figure above shows an instance of a TPM diagrammatically. Given the existence of the specifications mentioned earlier, multiple vendors can provide conformant instances of this technology with the ability to differentiate their implementations.

Below is a picture of the elements that are typically found within a TPM. The protected execution and storage of the TPM allow for hosting the RSA asymmetric key pairs, such as the endorsement key (EK), the AIK (Attestation Identity Key), and SRK's (Storage Root Keys). Recall that in RSA cryptography, the public key can be known to all, but the private key must be shrouded from users. The TPM and its isolated execution can both host the key-pairs and keep the private RSA keys away from attacks/errant agents on the host. In today's platforms without a TPM, only a custom Hardware Security Module (HSM) or other additional hardware can be used for hosting key-pairs. But in these latter solutions, there is no guarantee on construction of platform, surety of the host interface, etc. The Trusted Computing Group via design guides, protection profiles, conformance tests, etc., vies to both describe the requirements on the TPM and the binding into the platform in order to have a trusted building block (TBB).

Basic TPM Block Diagram

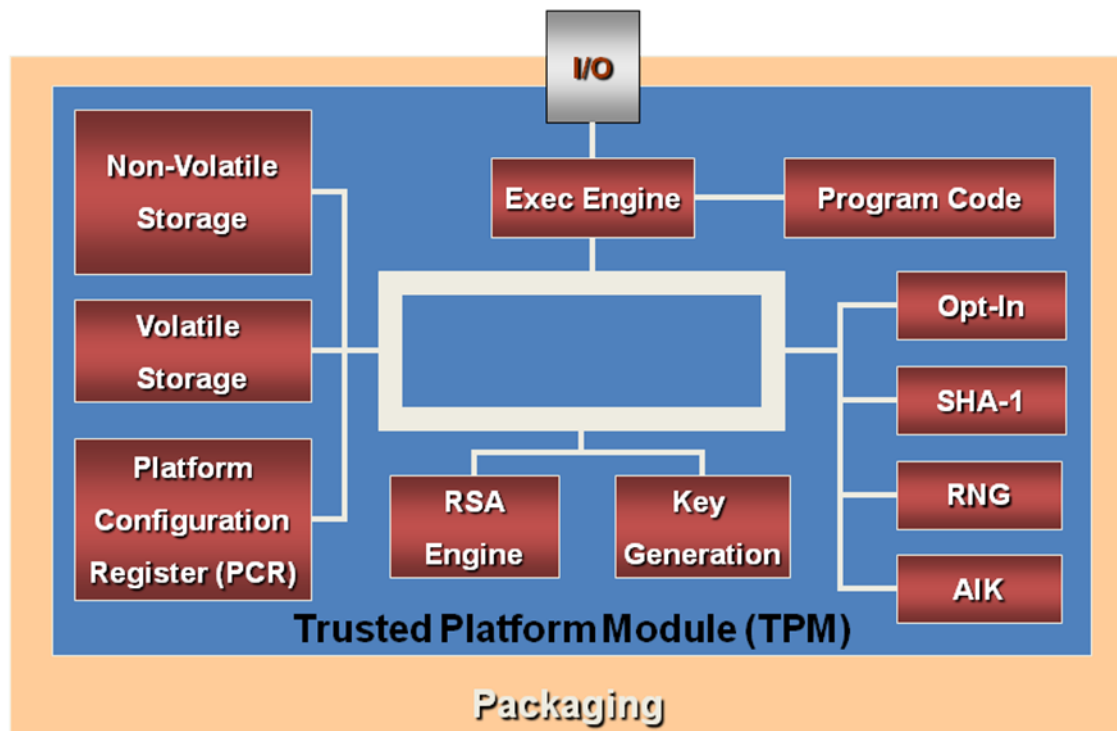


Figure 8 TPM block diagram

What is a Trusted Building Block (TBB)?

The TBB includes the components that make up the platform. These can include the TPM, how the TPM is bound to the platform, flash with the system board firmware, portions of the firmware that must be trusted. The TBB goes beyond TPM ordinals. It leads into prescriptions on the construction of the physical platform. As such, it is not just an issue at one layer of the stack.

Summary

This chapter reviewed a 'hardware' layer element called a "TPM" and associated technology. The latter will provide one building block for that hardware layer to help meet larger platform security and protection goals.

What is platform firmware/BIOS

This section will describe one layer of the security architecture mentioned above: the ‘firmware.’ In fact, the trusted hardware elements listed above cannot be leveraged without firmware.

What is a BIOS

The term *BIOS* stands for ‘basic input/output system’. BIOS is a class of firmware that runs on the in-band processors or CPU’s of a system in order to initialize the platform hardware complex and pass control to an operating system. For purposes of the security architecture mentioned in the earlier chapter, it is the ‘firmware’ layer.

The original PC/XT had an 8kbyte BIOS that that initialized the system and passed control to DOS®. This was 1982. Since that time, the BIOS domain has evolved significantly, including the transition of the industry to the Unified Extensible Firmware Interface (UEFI).

We will refer to the original BIOS as the “Conventional BIOS” and the UEFI-based boot code as “UEFI.” Conventional BIOS and UEFI [UEFI Specification] based-systems must carry out several roles. First, each has the concept of platform initialization. This phase is the code that commences execution immediately after a platform restart (S3, S5, etc). In a Conventional BIOS, this is a vendor-specific flow and construction, but is sometimes referred to as the ‘stackless assembly’ or ‘boot-block code.’ In UEFI, there is a separate set of standards referred to as the Platform Initialization (PI) standards [UEFI PI Specification]. In a PI-based platform initialization, the SEC and PEI phases commence this early execution.

The temporal evolution of a UEFI PI-based boot is shown below.

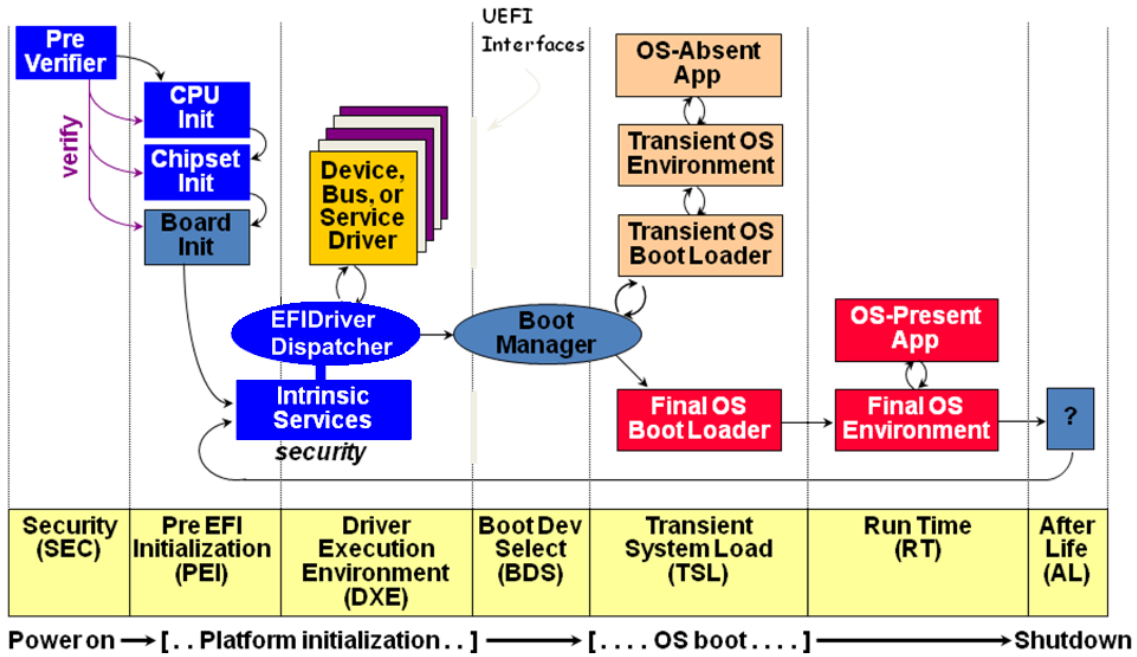


Figure 9 UEFI PI boot flow

Afterward, each platform needs to discover I/O buses and dispatch option ROM's from host-bus adapter cards, etc. In conventional BIOS, this I/O enumeration happens in "POST" (power-on-self test). There is no real standard for POST on a Conventional BIOS. For UEFI PI-based firmware, though, this phase of execution occurs in DXE (Driver Execution Environment). DXE also serves as the UEFI core for purposes of supporting UEFI-based operating systems.

After BIOS POST and DXE, though, the 'standard' part of the interface to the platform appears. For a PC/AT BIOS, the 'standard' includes the de facto int-callable interface (e.g., Int13h for disk, Int10h for video) that executes in 16-bit real mode on the x86 architecture. For UEFI, this option ROM and loader interoperability includes the UEFI boot services and protocols (e.g., EFI_BLOCK_IO_PROTOCOL as analog to BIOS int13h), and is described by the UEFI specification. It is during this phase of execution where 3rd party content can appear from disk or adapters that did not necessarily ship with the platform manufacturer's (PM) system board.

This taxonomy above is critical because the transition from an execution regime provided by the PM into a space where 3rd party codes can run has implications on the construction of a trusted platform.

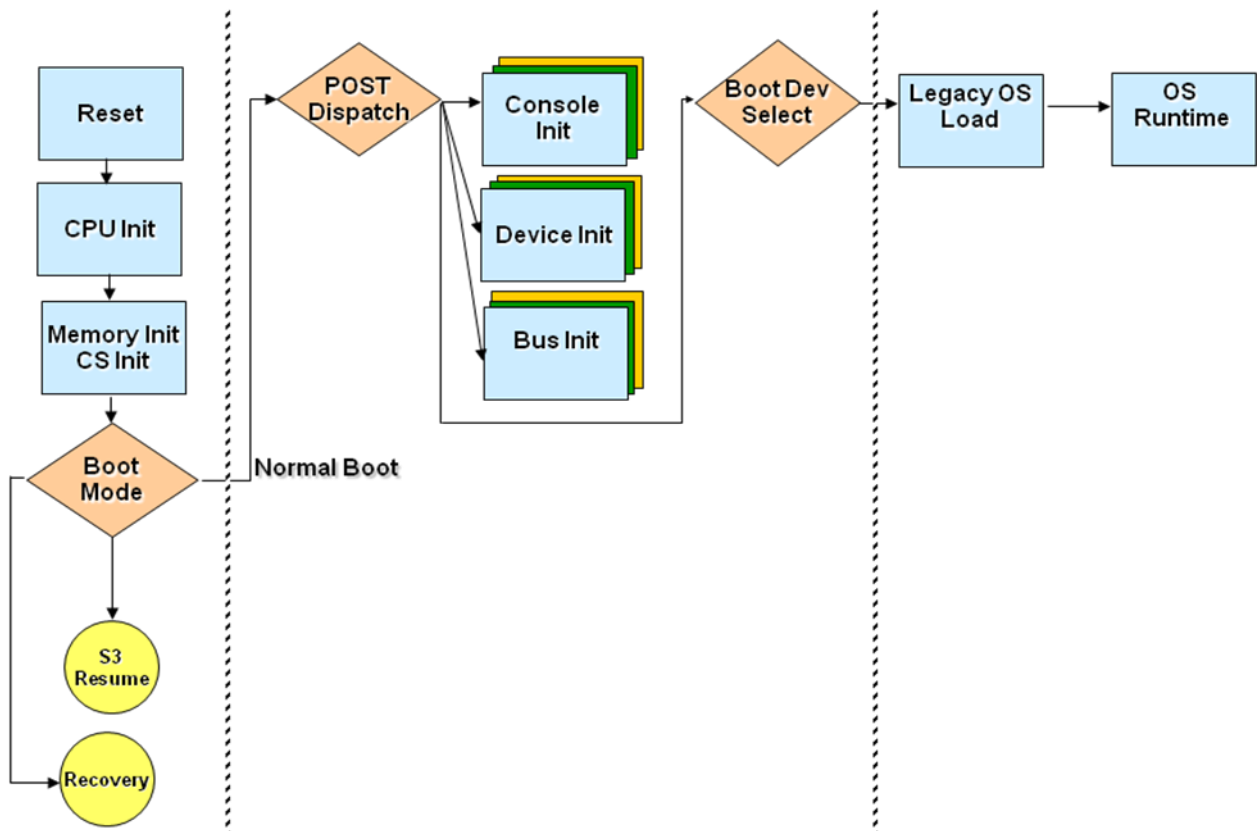


Figure 10 High-level BIOS flow

The figure above shows the generic BIOS initialization flow. This flow includes the initialization of the platform CPU, memory, and I/O devices during POST. The POST flow again is analogous to the DXE flow in the earlier figure.

More details on the UEFI PI flow will be treated in the ‘best-practices’ chapter.

Summary

The platform firmware represents a mandatory layer in the security architecture we introduced earlier. As such, specific activities around interactions with trusted hardware, construction, and deployment of firmware will be covered in the following chapters.

Firmware and trusted computing

The preceding chapter provided a brief description of the platform firmware and its role in initializing the system. The firmware and TPM, which we described individually earlier, come together with the S-CRTM.

What is a S-CRTM

A S-CRTM is a "static core root of trust for measurement." The S-CRTM is the portion of the platform firmware that must be 'implicitly trusted.' The S-CRTM makes the first measurements, starts TPM, and detects physical presence per the TCG privacy model.

And it is where the S-CRTM portion of the TBB intersects with the platform firmware and other roots-of-trust in the platform. "S-CRTM," "CRTM," and "SRTM" will be used interchangeably later in the section.

Following is a quick overview to clarify the roots-of-trust in the platform and which business entity delivers them.

Taxonomy of terms in the platform:

- RTM
 - Generic term for "Root of Trust for Measurement"
 - SRTM is the static root of trust for measurement (SRTM) – CRTM + unbreakable measurement chain to OS
 - DRTM is the dynamic root of trust for measurement (DRTM)
- CRTM
 - Static CRTM (S-CRTM) or "CRTM". Portion of platform firmware that must be implicitly trusted.
- RTR
 - Root of trust for Reporting
 - These are the Platform Configuration Registers (PCR's) in the TPM
 - 20-byte non-resettable registers to store the 'state' or 'measurements' of code + data
 - Typically SHA1 (new info || former PCR value), where "||" is the catenation of data
- RTS
 - Root of trust for storage
 - Endorsement key (EK) – unique per TPM
 - Storage root keys (SRKs) – use by OS and others to build key hierarchies
- TPM Owner
 - Applies the authentication value
 - Several commands are 'owner authorized'
- SRTM
 - Static root of trust for measurement

- CRTM (e.g.-CRTM) + platform firmware measuring all code and data prior to boot
- Records information into non-resettable or “static” PCR’s (0-15); these static PCR’s zeroed only across platform reset
- Described by TCG BIOS and EFI specifications
- DRTM
 - Dynamic root of trust for measurement
 - Initiates the measurement later in boot. Includes ‘resettable’ PCR’s 16 and above; these ‘resettable’ PCR’s zeroed upon initiation of the DRTM launch
- Physical presence
 - Administrative model of the TPM. Assertion by operator of presence in order to perform privacy or administrative activities with the TPM.

In general, a hardware instantiation of the Trusted Platform Module (TPM) is a passive hardware device on the system board. It serves as the Root of Trust for Storage (RTS) and Root of Trust for Reporting (RTR). The former is the use of the Storage Root Key (SRK) and the Platform Configuration Registers (PCR’s).

The figure below shows the synthesis of the various roots in the platform.

Functional TPM Diagram

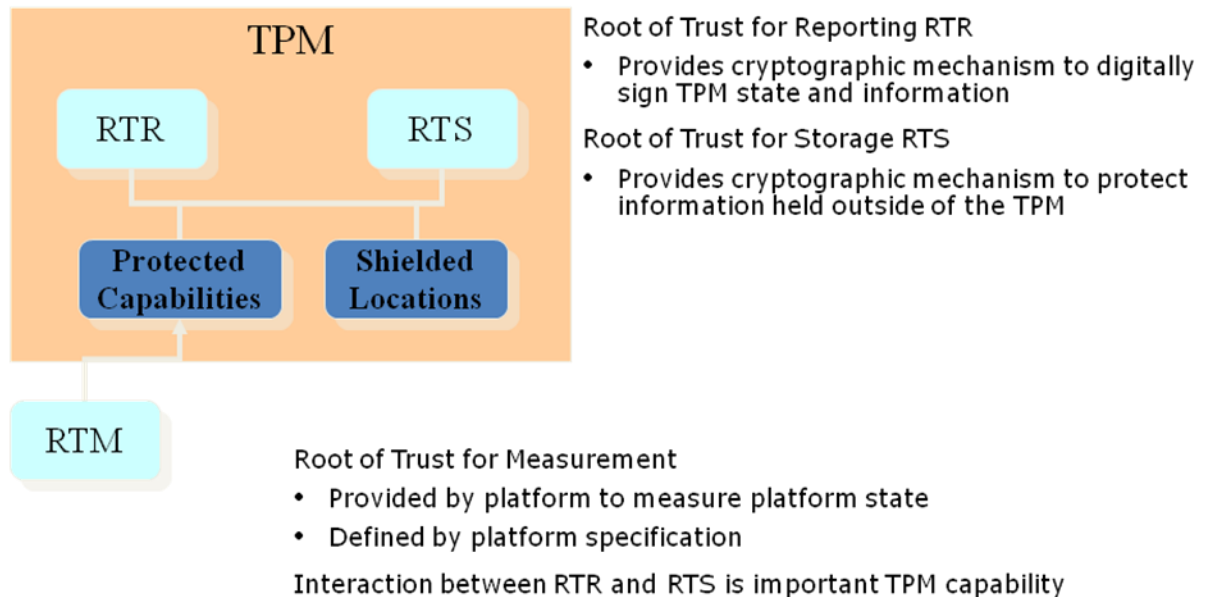


Figure 11 Synthesis of firmware and TPM

The active agent on the platform is the Root of Trust for Measurement (RTM). The RTM can be either Static or Dynamic (SRTM versus DRTM, respectively). The SRTM, on the other hand, entails trust chain from the platform reset vector going forward.

The definition of the SRTM for EFI is defined in [TCG EFI Protocol] and [TCG EFI Platform]. The flow of the SRTM into the operating system is shown in the figure below.

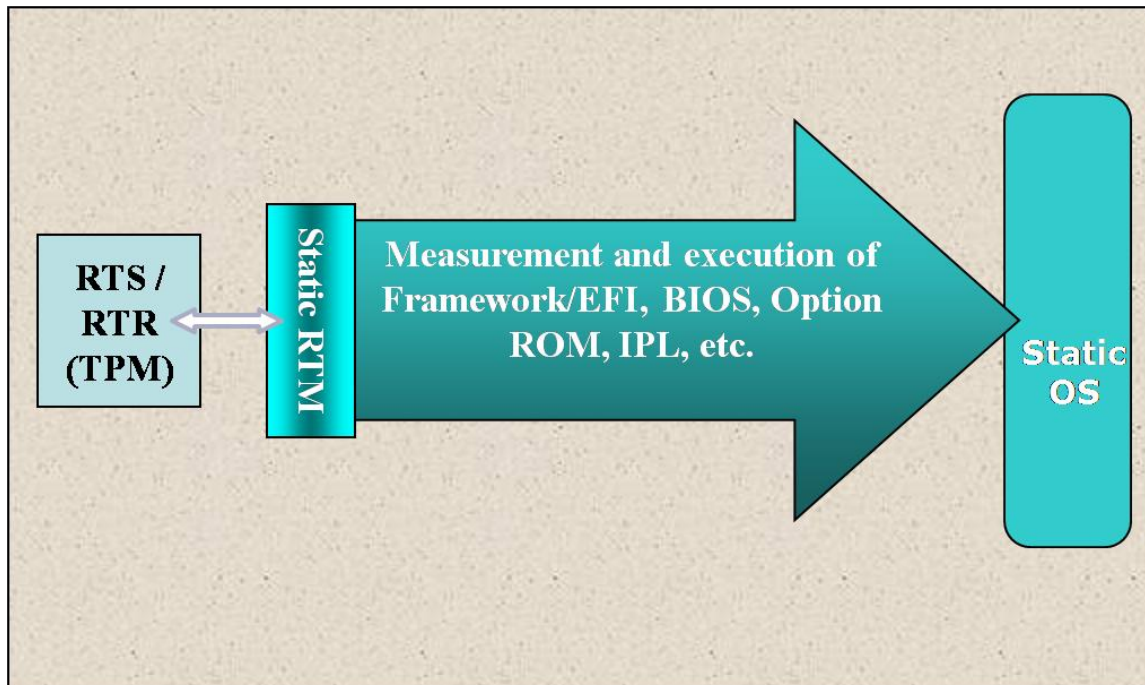


Figure 12 Boot flow

There need to be UEFI API's available so that the UEFI OS loader can continue to measure the operating system kernel, pass commands to the TPM to possibly unseal a secret, and perform other TPM actions prior to the availability of the OS TPM driver. In addition, this API can be installed at the beginning of DXE to enable measurement of the DXE and UEFI images.

The figure below shows where the UEFI TCG API's would appear relative to the other interfaces.

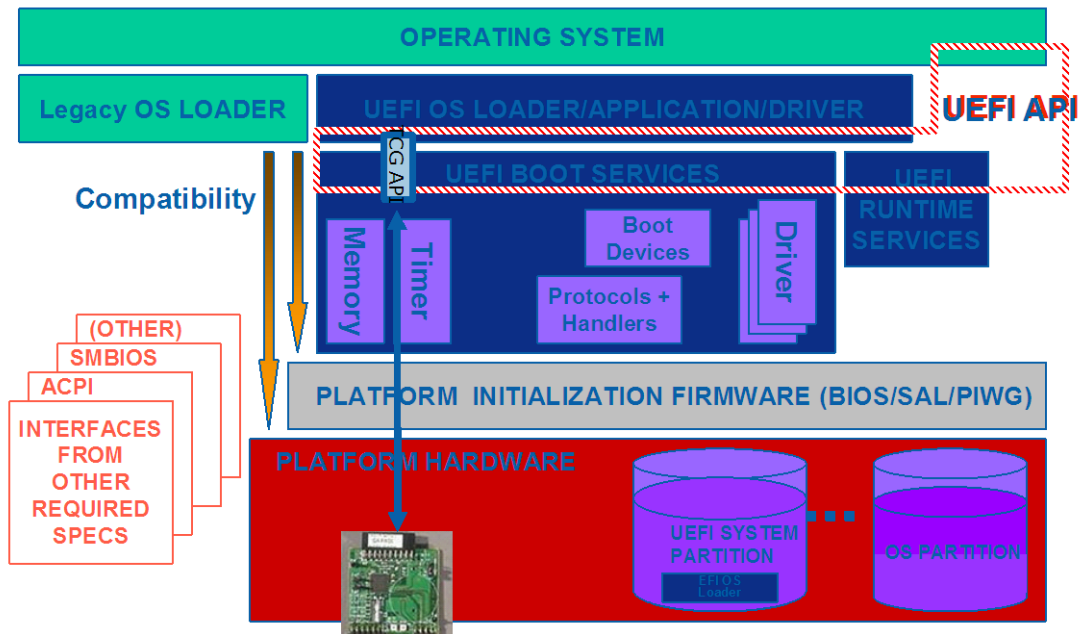


Figure 13 Layering of API's in UEFI system

The UEFI specifications are cross-listed in the TCG PC and Server Working Groups such that both consumer and enterprise-class operating systems can participate in this boot flow behavior.

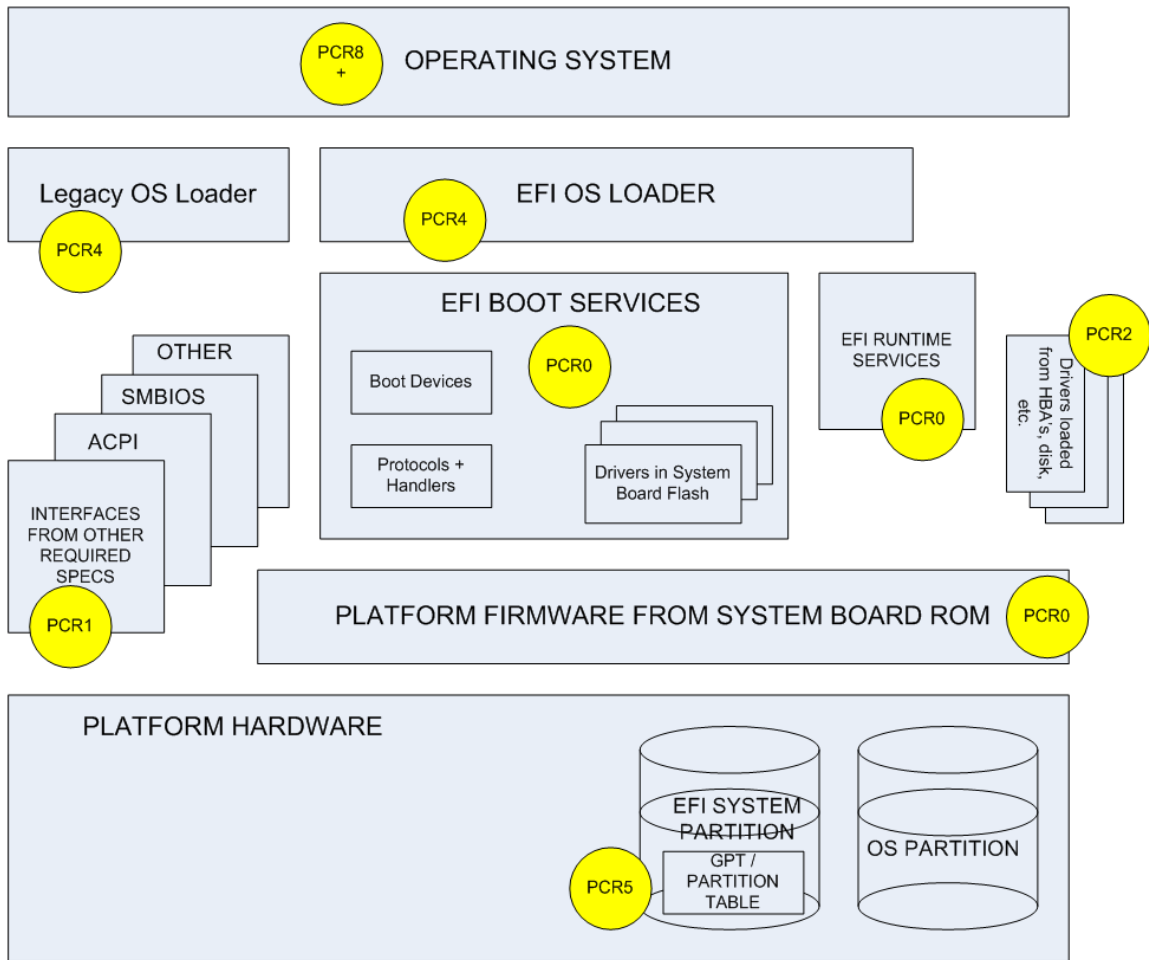


Figure 14 Measured objects in UEFI

The UEFI TCG Platform specification describes which objects to measure in a UEFI System, such as the images, on-disk data structures, UEFI variables, etc. The figure above shows which objects in a UEFI system correspond to measures in PCR's.

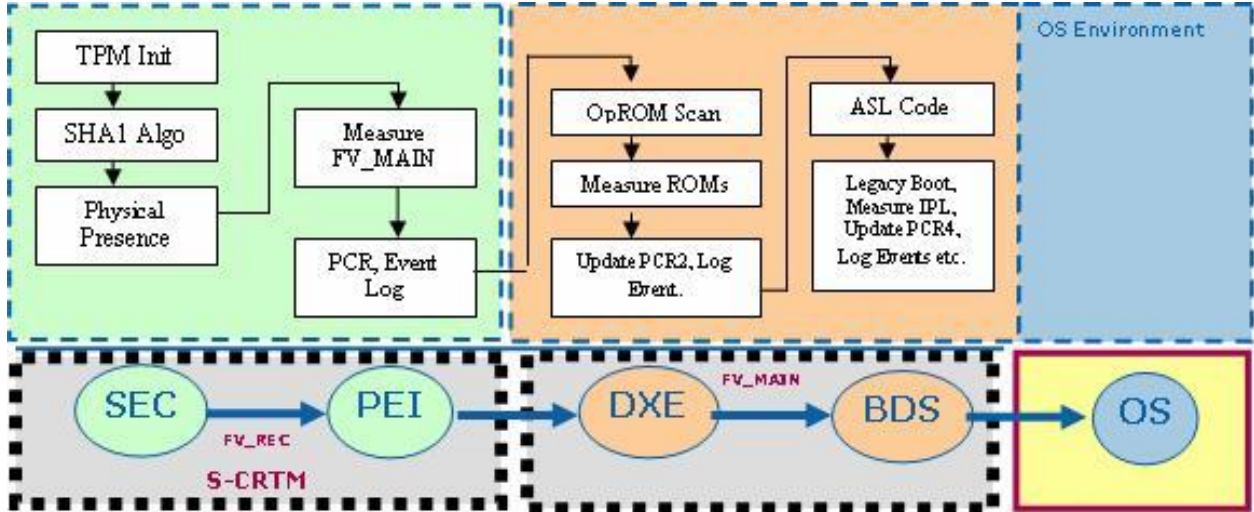


Figure 15 Measurement and temporal boot flow

Prior to the “EFI Phase” of platform execution, the Framework & PIWG describe the PEI and DXE phases. Therein, the CRTM is mapped to the PEI phase and what is thought of as BIOS POST is mapped to DXE. There are interfaces in PEI (namely, the PEIM-to-PEIM interface, or “PPI”) to allow for fine grain measurement in that phase of execution, too.

The figure above shows one possible PEI-based CRTM and the flow into the operating system.

What is the point of measurements?

The process of measurements records the state of the platform, for both executable code and data hashes, into the TPM’s platform configuration registers (PCR’s). These PCR’s are write-only and cleared upon a platform reset (at least the static PCR’s for SRTM). The PCR’s reflect the platform state. They are used such that software, when installed upon the platform, can “Seal” some information to the platform. A seal is like an encryption that also includes PCR’s. There is a corresponding “Unseal” operation, which is a decryption that also uses the PCR’s.

What this practically means is that if the state of the platform changes between the installation of some software (and the Seal operation) and successive invocations of software on later restarts (and the use of Unseal operation), unauthorized changes to the platform in the interim will be detected (i.e., PCR’s changed).

This is sort of the Resurrecting Duckling security model wherein the initial state of the platform (i.e., PCR values upon installing application) is considered safe or acceptable.

UEFI offers an opportunity here. PI and UEFI have specification-based components written in a high-level language (e.g., C). The software development lifecycle (SDL) for

drivers and other system software can be applied, as can static analysis tools (e.g., Klockwork®, Coverity®, etc). Later in the paper we'll talk about additional practices to complement the SDL that address domain-specific issues with platform firmware.

With all these elements of security and protections in place how the CRTM is updated becomes critical and much more challenging. Since the CRTM is the root, and is itself inherently trusted, it must be a very controlled and secure process. The TCG describes CRTM maintenance in the Trusted Building Block (TBB) protection profile. Either the CRTM is immutable, or never changed in the field, or appropriate cryptographic techniques need to be employed in order to update the CRTM.

Regarding the cryptographic-based update, the figure below shows a possible implementation where the firmware volume (FV) update is enveloped using an RSA-2048/SHA-256-based update [RSA][SHA]. This is just one possible UEFI PI implementation that leverages the UEFI PI-based firmware volume construct and the WIN_CERT that can be found in the UEFI2.0 specification.

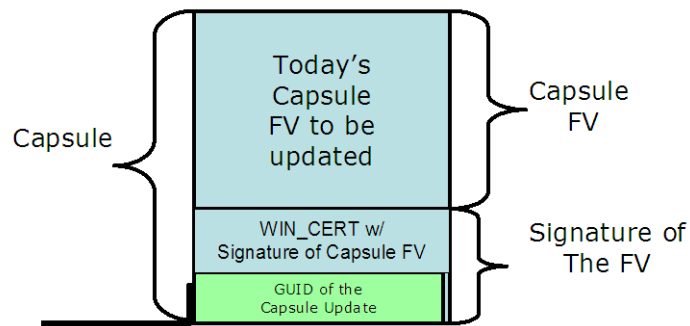


Figure 16 Firmware volume update

As noted above, a signed capsule is one implementation path. The system flash is not directly updated by a flash utility but instead the CRTM update capsule is stored in a staging area. The next time the CRTM gains control of the system (at reset), it will check for any pending updates. If updates are found, they will be validated and then cryptographically verified. If they are valid, the CRTM update can be applied. It's important to note that when validating the update this all must be done by using only CRTM code and data. Code or data outside the CRTM cannot be trusted until verified.

Summary

This chapter describes the various roots-of-trust in the platform, how the platform firmware as the RTM leverages the TPM via measurements and the SRTM flow. And finally, the deployment criteria of a CRTM is reviewed, with due consideration in the construction of the firmware CRTM and updates.

Operating System usage of SRTM

There are various use-cases for a trusted platform. The TCG describes combining network-access control scenarios based upon trusted platforms in their Trusted Network Connection (TNC)-based specifications.

But this portion of the paper will highlight a specific use-case of the SRTM and a shipping operating system feature.

OS usages of SRTM

One of the uses of the static measurement includes the Microsoft Server 2008® BitLocker® Feature. This uses the loader, in tandem with the TPM, and then encrypts the full disk volume.

BitLocker™ Drive Encryption Static Root of Trust Measurement of early boot components

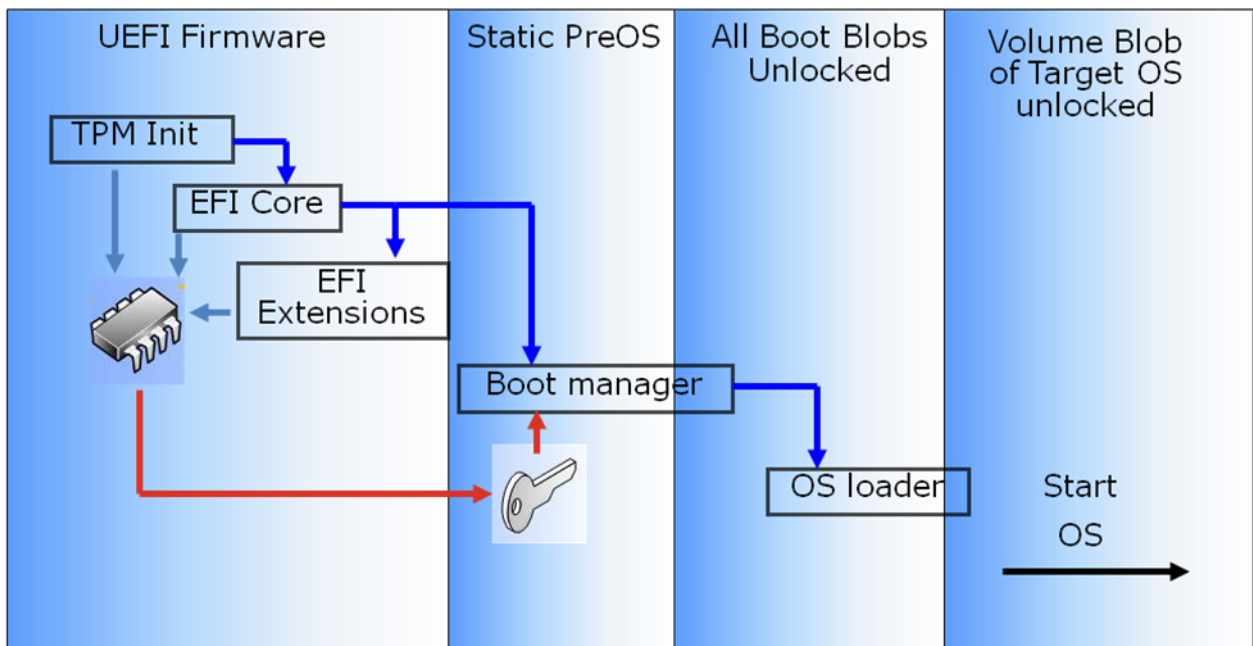


Figure 17 BitLocker EFI SRTM

This capability is a class of behavior called “Full Disk Encryption” (FDE). Concerns around data mobility, such as a server disk array being stolen, all the way down to a laptop abandoned in a Taxi, could entail a data breach. In fact, earlier discussions in the “Problems to solve” section included compliance standards like Sarbanes-Oxley and

HIPAA which are driving enterprises to deploy this type of capability.

The TPM and an SRTM-capable BIOS are enablers of this operating system feature.

Key points

This is a scenario that leverages SRTM capabilities.

Summary

This brief section was intended to highlight the fact that there are commercially available operating system features that leverage trusted platforms.

Platform Perspective

This section will provide an example of a real-world system that supports many of the elements of trusted hardware, UEFI firmware, and the OS use-case described earlier.

IBM System

IBM has integrated many of the features in the preceding section to implement a secure platform.

The IBM firmware adheres to the UEFI Framework/PI design, and an example breakout is shown below.

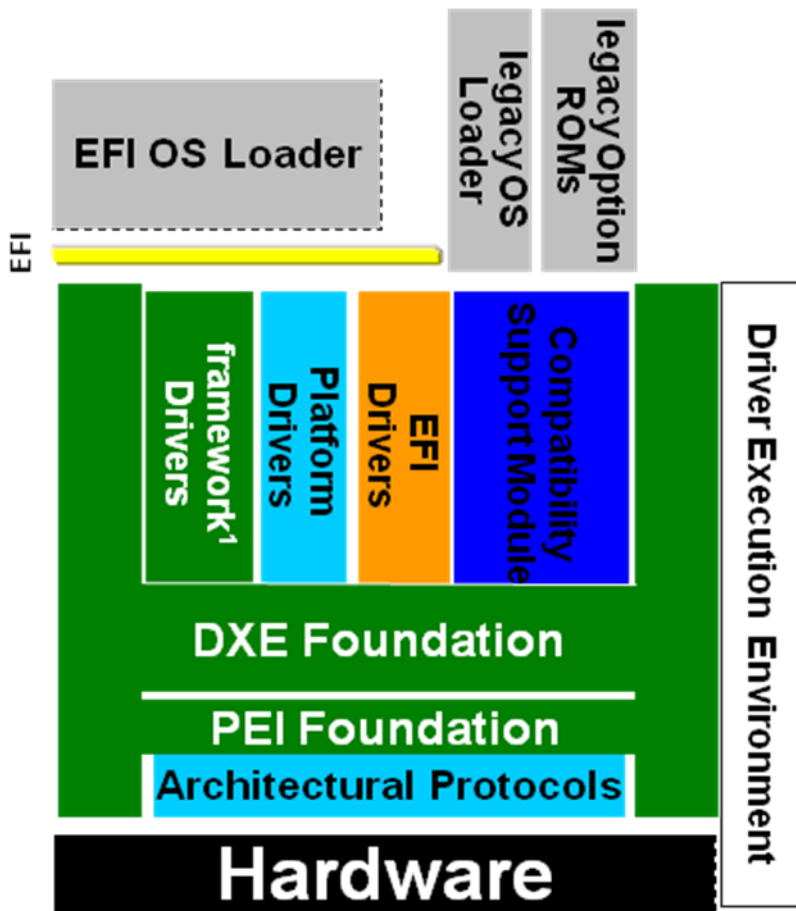


Figure 18 System firmware layering


This layering allows for support of both UEFI operating systems via a DXE core and support of CPU and chipsets modules via PEIM's and DXE driver drivers.

IBM started shipping trusted computing elements several generations ago and is extending the capabilities across their product lines.

The figure below shows a small set of the trusted platforms that includes both UEFI firmware and trusted hardware elements.


IBMSystem x Servers★

- Comprehensive System x portfolio Transition to UEFI based firmware
- UEFI 2.1 PI 1.0 specification compliant
- Improved management and configuration capabilities
- Advanced "Touchless" Compatibility Support Module (CSM)
- **Trusted Platform features: TPM enablement, TCG and Core Root of Trust for Measurement support**




Blade

- HS22



Rack-mount

- x3650 M2
- x3550 M2
- X3250 M3



Tower

- x3500 M2
- x3400 M2
- x3200 M3

Figure 19 IBM trusted platform portfolio

By leveraging UEFI to build trusted platforms, more features can be readily added to the platform versus the space and size limitations of legacy BIOS.

Summary

This chapter shows that the promise of trusted computing is delivered via IBM systems based upon industry standards and Intel hardware.

Best Practices for Construction

This section gives a background on platform hardware and firmware construction practices for a trusted platform. It treats the earlier issue of “assurance” in the overview with more details.

Background of practices

Many of these prescriptions covered below are already treated in various TCG documents and design guides. The intent of this section is to provide a platform and UEFI PI-focused summary of rules and practices.

Recall the earlier section of the security architecture with hardware and firmware at the lowest layer. In addition to the having appropriately specified components and designs, ‘how’ the elements are integrated and build is just as important. A review of some hardware construction guidelines, along with firmware best practices, will be included.

Platform Hardware

Recall that the lowest layer of the security architecture in the early chapter is hardware, as shown in the figure below.

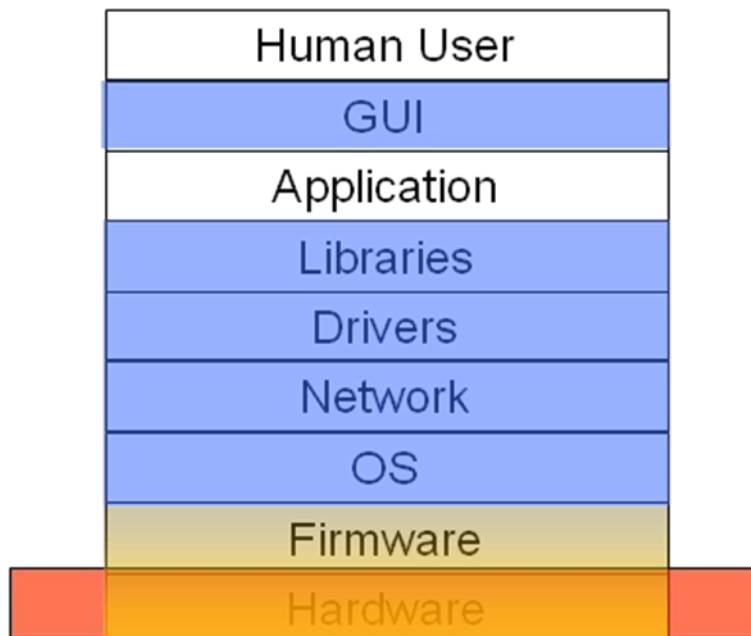


Figure 20 Hardware Layer of security architecture

The hardware best practices treated below are focused at the system board platform manufacturer. Assurance around chipsets, CPU's, I/O devices, and other elements are not treated here.

Some of the platform hardware platform best practices include the following:

1. CRTM flash protection: Hardware must be designed to have a robust flash protection mechanism. Locking must not be controlled by any untrusted programmable entities in the system including non-host processor. Also, once locked within CRTM code, it must not be un-lockable without going through a system reset .
2. Physical Presence: Physical Presence (PP) hardware must be designed such a way that this pin is not changeable by any un-trusted programmable entity.
3. Reset: System reset functionality must be designed such a way that TPM gets reset whenever there is any kind of system reset and also hardware needs to make sure that there is no path available to manipulate reset vector in the system .

UEFI Firmware

Above the hardware layer of the security architecture is the firmware, as shown in the figure below.

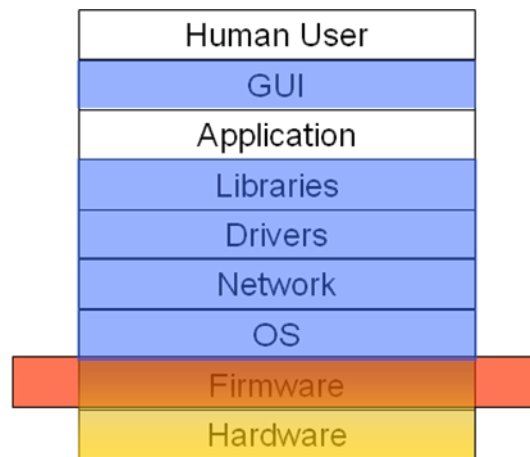


Figure 21 Firmware layer of the security architecture

Since there are several code-base implementations of platform firmware today and possibly in the future, this discussion will be held in the context of the Platform Initialization (PI) standards and some representative codebases thereof.

As a backgrounder, **Platform Initialization Working Group of UEFI Forum** has the following properties:

- Delivering the Platform Initialization Architecture Specifications
- Based on Intel PEI and DXE specs
- Independent of UEFI 2.0 Specification
- PI Architecture platforms can still boot today's OS's

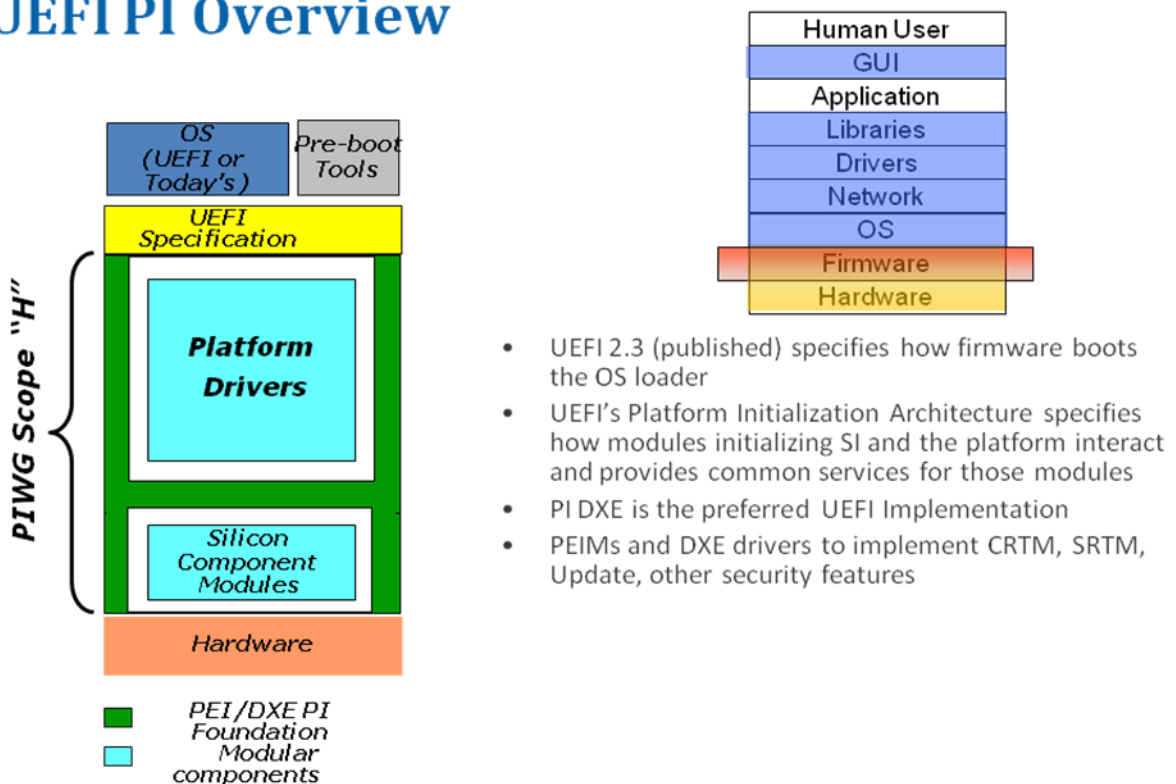
Ownership: AMD, AMI, Apple, Dell, HP, IBM, Insyde, Intel, Lenovo, Microsoft, Phoenix own the specifications

Goal: Allow silicon vendors who create "reference code" today to package this reference code as modules that snap-into PI Architecture firmware implementations

The figure below speaks more to the PI scope.

What About Firmware Practices?

UEFI PI Overview



- UEFI 2.3 (published) specifies how firmware boots the OS loader
- UEFI's Platform Initialization Architecture specifies how modules initializing SI and the platform interact and provides common services for those modules
- PI DXE is the preferred UEFI Implementation
- PEIMs and DXE drivers to implement CRTM, SRTM, Update, other security features

Figure 22 UEFI PI

UEFI & PI specifications only describe normative material around interface definition and mechanism, but No informative content, such as ‘why’, ‘how’. Intent of latter to be purview of design guides.

More information on the UEFI PI Specifications can be found at [UEFI PI Specification].

In contrast to monolithic BIOS’s, the UEFI PI allows hardware agility via software extensibility by exposing a driver model, such as dependency-expression based PEI Modules (PEIM’s) and DXE drivers. Extensibility points can serve as a point of attack for malware. This type of malware is essentially undetectable by OS hosted protection technologies, such as anti-virus since it executes well before the OS. Consequently, the security integrity foundation of the pre-OS boot environment provided by UEFI and other pre-OS extensibility must be solid, while still providing enough flexibility to support hardware agility.

The PI phase is intended to only be extensible by the platform manufacturer (PM), not third party (in contrast to UEFI and its option ROM/loader/driver model). The installation and behavior of code in this early PI flow is said to act under the authority of the Platform Manufacturer; this will be referred to as “PM_AUTH” below.

Preservation of this PI as PM-extensible-only-intent in both construction and survivability in the field is a goal of the system design and required to have appropriate assurance for earlier-described instances of SRTM, CRTM, etc.

Below is a figure describing the UEFI PI boot-flow, including annotation of the PM-extensible-only PI code and the 3rd party extensible UEFI.

Overall View of Boot Time Line

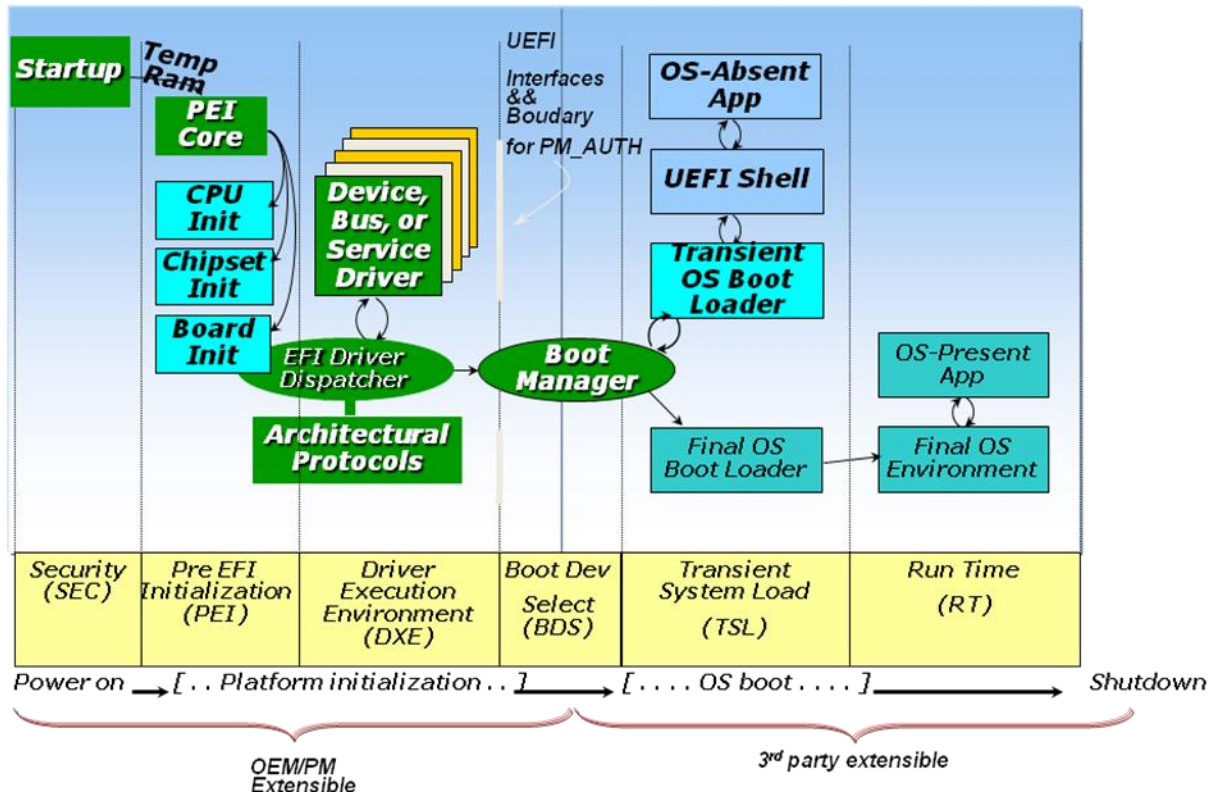


Figure 23 UEFI PI Boot flow

Best practices of PI

Regarding the construction of UEFI PI code, there are a series of best practices that should be followed. The summary of the best practices list is as follows:

1. HW misconfiguration:
 - Appropriate set locks and other hardware configuration should be set by the PM-only PI code prior to running 3rd party code, such as UEFI drivers or operating system loaders
2. Callouts
 - Don't call out from PM_AUTH PI code to non-PM_AUTH code
 - Measure and code before loading
3. Interface correctness
 - Pass compliance tests
 - Check & validate input, especially from non-PI PM_AUTH into PI code
4. Flash protection and update security
 - Appropriate update of PI and CRTM – either immutable or cryptographic update
5. Denial of service
 - Platform recovery/update strategy

These practices do not remove the need for threat-model [Threat] and other activities to ensure robustness of the platform. But the above items will allow for the business intent of the UEFI PI specifications, such as PM_AUTH, to be achieved in deployment of actual systems.

Summary

In addition to industry standards adherence, it is also important to follow some implementation best practices to gain assurance in the construction of the hardware and firmware layers of the security architecture.

Evolution of platform trust

This section will talk about several emergent technologies that build upon the earlier trusted computing elements and firmware assurance.

These technologies will span evolution in both the UEFI firmware and also the root-of-trust for measurement, namely going from a static root-of-trust to a dynamic one, or SRTM going into DRTM.

UEFI Futures

To begin, the UEFI evolution described below appear as elements of the UEFI main specification in version 2.2 [UEFI Specification]. These features entail updates to the boot behavior and the features briefly treated will include image verification, networking enhancements such as IPSec, and user identification.

The figure below shows where in the stack the emergent UEFI features described in this chapter exist, namely in the UEFI Services and boot manager .

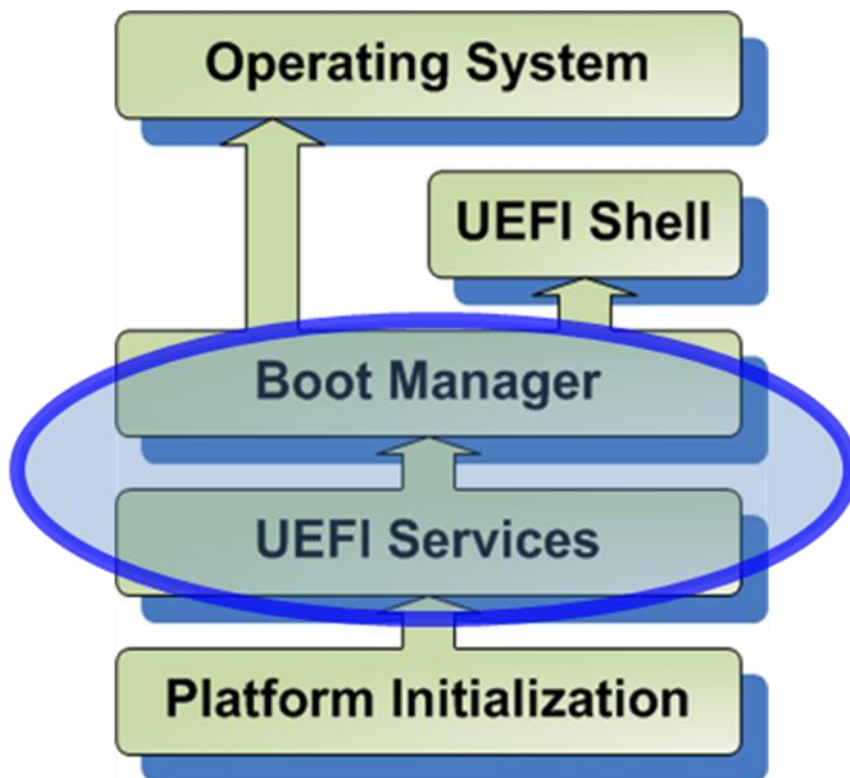


Figure 24 UEFI stack

UEFI2.2 executable verification

The first feature from UEFI2.2 to discuss is driver signing or executable verification.

The figure below provides some details on the capabilities:

Driver Signing

- Expands the types of signatures recognized by UEFI
 - SHA-1, SHA-256, RSA2048/SHA-1, RSA2048/SHA-256 & Authenticode
- Standard method for configuring the “known-good” and “known-bad” signature databases.
- Provides standard behavior when execution is denied to provide policy-based updates to the lists.

Figure 25 UEFI 2.2 Driver signing overview

One evolution beyond the SRTM described in earlier chapters, that UEFI can provide is “verification.” Recall that the SRTM records the state of the code and data in the platform such that a later entity may assess the measurements. For verification, or enforcement, of some policy, the UEFI firmware can act as a ‘root-of-trust-for-enforcement’ (RTE) or ‘root-of-trust-for-verification’ (RTV) wherein the boot process can change as part of policy. This policy can include the UEFI2.2 image verification using Authenticode-signed images [Authenticode], for example.

Below shows the steps necessary for signing of UEFI images. The signing can include RSA asymmetric encryption [RSA] and the hash function a member of the security hash

algorithm [SHA] family.

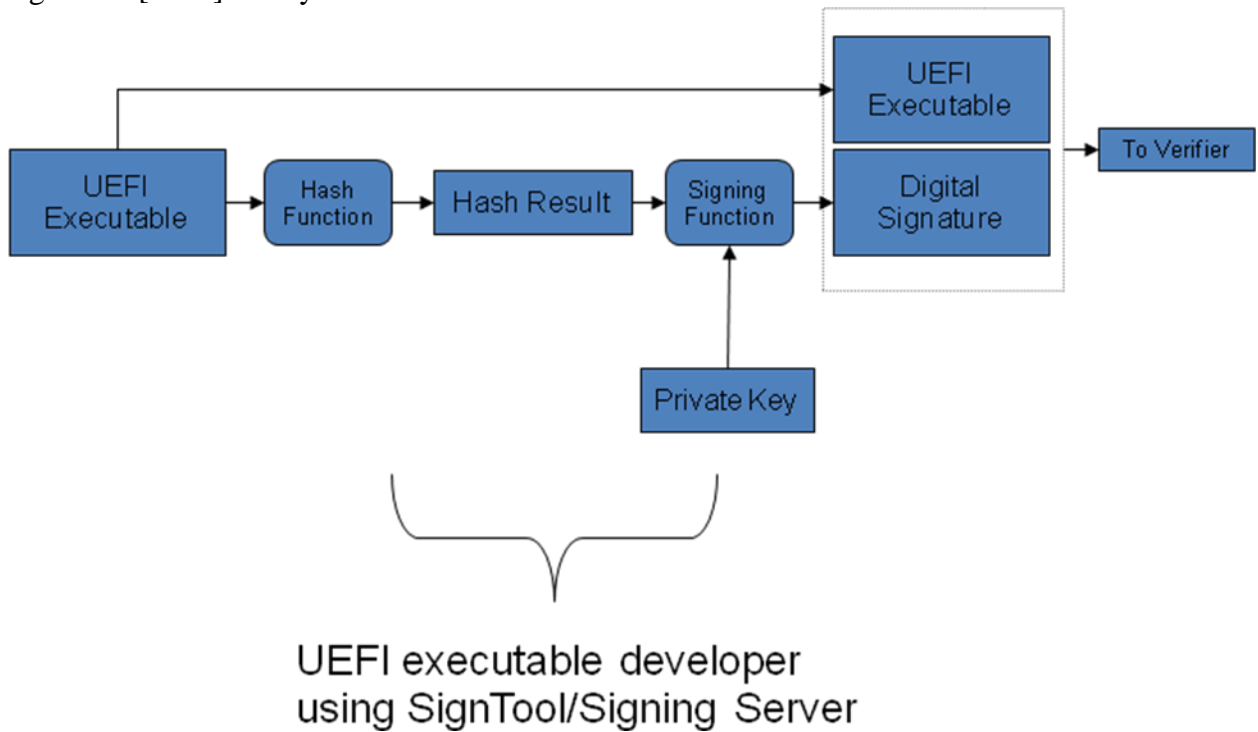


Figure 26 Signing of UEFI images

This preparation would happen at the manufacturer facility or could be facilitated by a 3rd party, such as VeriSign ® Certificate Authority (CA).

Once the signed images are deployed in the field, whether loaded across the network, from a host-bus adapter card, or via the UEFI system partition, below is the behavior of the UEFI 2.2 firmware to verify the image integrity.

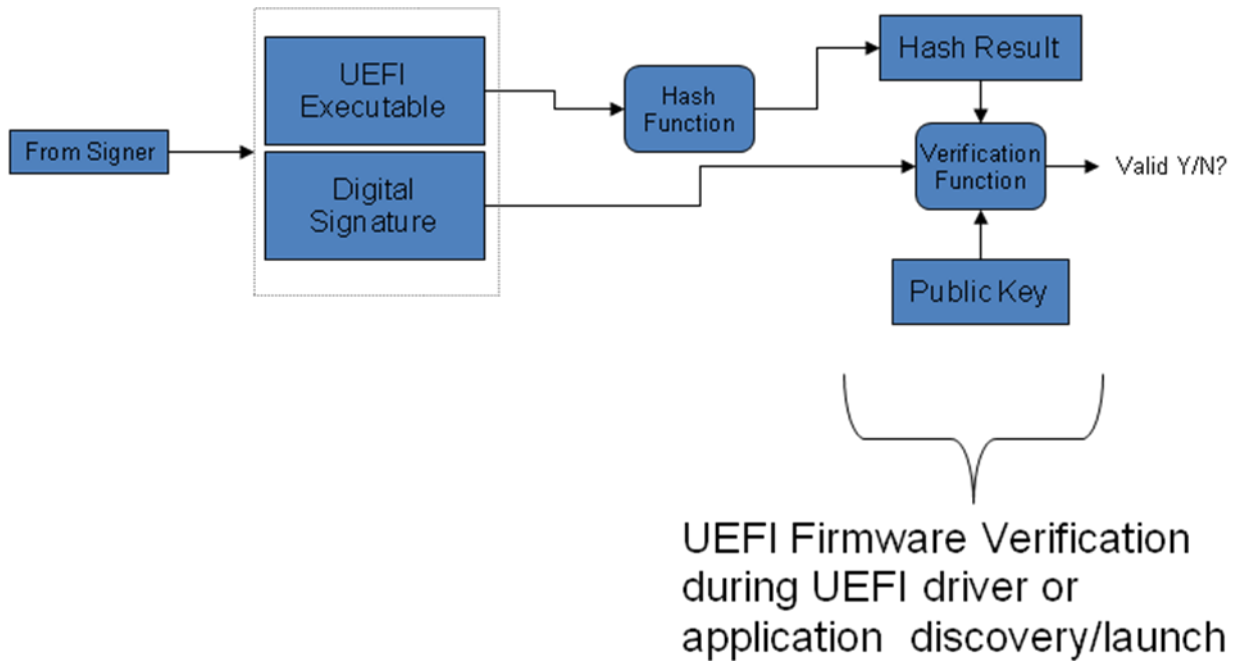


Figure 27 Verification of UEFI images

The combination of robust UEFI implements and interoperable trust infrastructure will allow for evolving the extensibility of UEFI in a safe, robust fashion.

UEFI2.2 IPsec/Network stack

Another element that appears in UEFI2.2 entails additional network security, including IPsec support.

Trusted hardware like the TPM can be used to help store the IPsec credentials, but to the stronger, assurance around the UEFI firmware implementation of the IPsec cryptography and the networking code will need to follow the guidelines in the preceding chapter. IPsec can be used for platform network boot to harden scenarios such as iSCSI-based provisioning.

A figure showing the EFI IPsec implementation using the UEFI2.2 IPsec protocol and IPV6 network stack, including a pre-deployed security association (SA), can be seen

below.

EFI IPsec Impl (Pre-deployed SA)

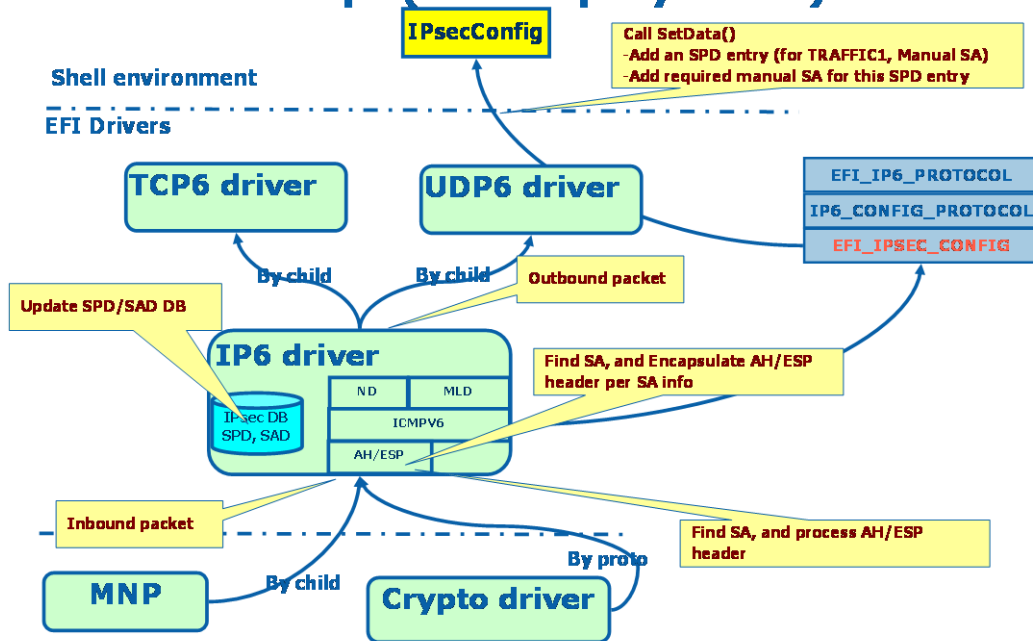


Figure 28 UEFI 2.2 IPsec

IPsec in the platform will allow for performing both an IPV4 and IPV6-based ISCSI boot and provisioning, for example.

The figure below shows an iSCSI layering on top of the UEFI2.2 network stack, for example.

iSCSI over IP4 & IP6

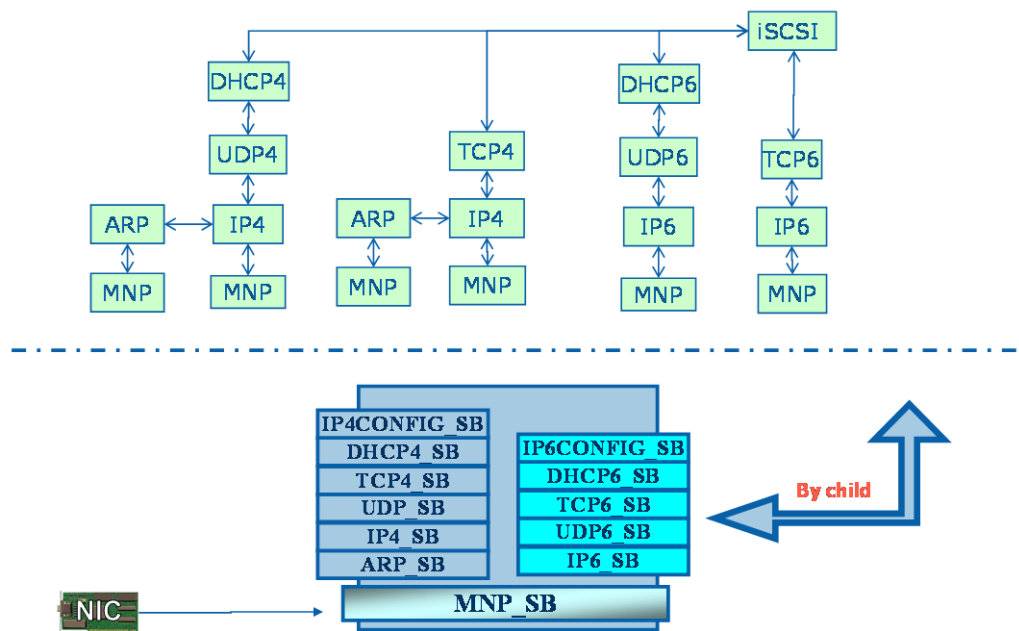


Figure 29 iSCSI application with the UEFI network stack

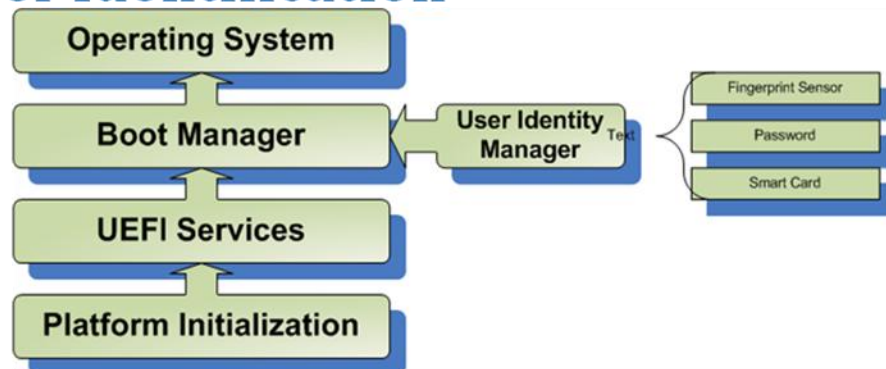
Beyond the IP6 and IPsec UEFI interfaces, the wire-protocol for network booting has commensurate evolution to the UEFI API's. Specifically, in the DHCPv6 extensions for IPV6 network booting [DHCP6 Net Boot], the boot file information is sent as a Uniform Resource Locator (URL). As such, the UEFI client machine and the boot server can negotiate various types of downloads, including TFTP, HTTP, or ISCS. This allows the network capabilities to track the needs of the market and the machine firmware capabilities.

UEFI 2.2 user identification

A final ingredient in UEFI2.2 includes the user identity support. This is infrastructure that allows for loading drivers from token vendors to abstract authentication of the user, including many factors, and a policy engine to assign rights to certain users. This can include limiting service access for certain users.

A figure showing this capability can be seen below.

UEFI User Identification



- Standard framework for user-authentication devices such as smart cards, smart tokens & fingerprint sensors.
- Uses UEFI HII to display information to the user.
- Introduces optional policy controls for connecting to devices, loading images and accessing setup pages.

Figure 30 UEFI 2.2 User identity

Implementation of these UEFI feature would also build upon and require the assurance/best practices in firmware discussed earlier.

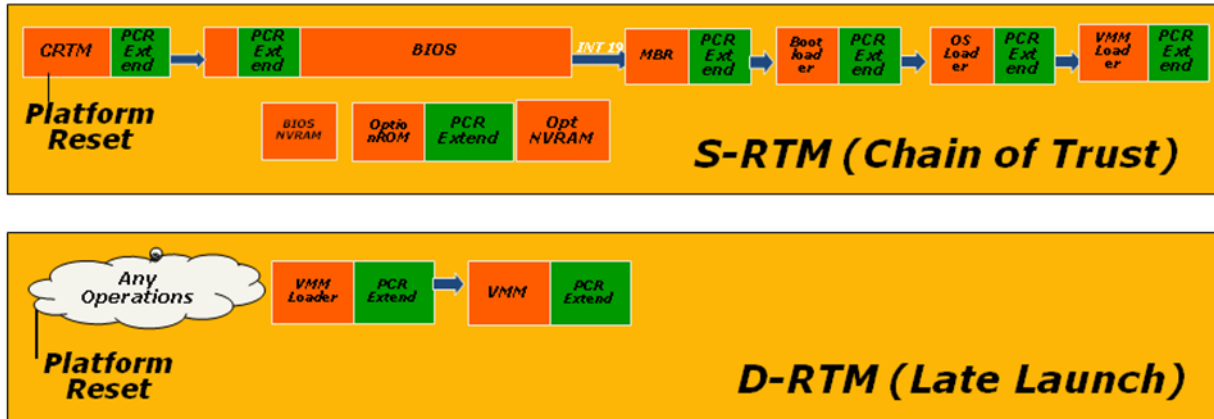
More information on the UEFI-based features can be found at [UEFI Main Specification].

Hardware evolution: SRTM-to-DRTM

As a final element getting introduced into the platform going forward is the dynamic root of trust for measurement, or “D-RTM.” The D-RTM provides platform hardware capabilities to support a MLE or ‘measured launch environment.’ An S-RTM and D-RTM feature set can exist on the same platform, or each feature can exist independently.

The figure below compares the two RTM's and their temporal evolution and features.

Trust Models: S-RTM & D-RTM



- S-RTM measurement chain starts at reset and includes components from various sources
- D-RTM measurement chain starts with a trusted secure event trigger such as SINIT. D-RTM leads to a smaller TCB, reduced attack surface and thus a more secure system
- MLE provider must make assurances that the MLE maintains the TCB. Smaller TCB simplifies MLE design.

Figure 31 DRTM boot flow

A DRTM implementation can also include a root-of-trust for verification (RTV), too.

More information on Intel's D-RTM implementation can be found in the following book [Secure Computing Book].

Summary

This chapter shows that the earlier investment in trusted platforms can be preserved while providing additional capabilities. At the same time, the best practices described earlier are applicable for the build-out of these new features, too.

Conclusion

This paper has introduced a set of security problems in the industry. To address these concerns, a set of trust concepts with associated security architecture were defined. In order to meet the goals of the architecture definition, though, the required roots-of-trust in the hardware and firmware were reviewed. These roots include TCG-based hardware and UEFI-based firmware.

These features can provide additional end-use features, such as more observable behavior, disk encryption, malware resistance, etc. But in order to meet the needs of a trusted robust platform, care and due consideration must be applied during construction of the constituent firmware elements.

The combination of UEFI and TCG-based technologies offers a technology platform to deploy richer interoperability and greater trust in the boot.

In addition to the having these various technologies in hand, the importance of how they are built and deployed was also reviewed. IBM System-X technology, including but not limited to support for Intel hardware and industry standards, is one such vehicle to deliver the security architecture.

Beyond delivering end-user value today, investments in trusted computing platforms and design practice also allows for evolution of platform trust.

References

- [Authenticode] Windows Authenticode Portable Executable Support
http://www.microsoft.com/whdc/winlogo/drvsign/Authenticode_PE.msp
- [DHCP6 Net Boot] Huth, Freimann, Zimmer, “DHCP option for network boot,” Internet Draft, Expires February 11, 2010
<http://tools.ietf.org/id/draft-ietf-dhc-dhcpv6-opt-netboot-05.txt>
- [EDK2] EFI Developer Kit www.tianocore.org
- [FRAMEWORK] Intel Framework Specifications www.intel.com/technology/framework
- [HIPAA] Health Insurance Portability and Accountability Act, <http://www.hipaa.org/>
- [IBM TCG] Challenger, et al, *A practical guide to trusted computing*, IBM Press, 2008
- [PCI] Payment Card Industry, <https://www.pcisecuritystandards.org/>
- [RSA] PKCS#1: RSA Cryptography Standard, Version 2.1
<http://www.rsa.com/rsalabs/node.asp?id=2125>
- [Secure Computing Book] David Grawrock, *Dynamics of a Trusted Platform*, 2nd edition, Intel Press, 2009
- [SHA] Secure Hash Algorithm, FIPS 180-2 <http://csrc.nist.gov/publications/fips/fips180-2/fips180-2.pdf>
- [TCG Overview] Trusted Computing Group Benefits,
http://www.trustedcomputinggroup.org/trusted_computing/benefits
- [TCG Main] TCG TPM Main specification, version 1.2 www.trustedcomputinggroup.org
- [TCG PC] TCG PC Client Specification, version 1.2 www.trustedcomputinggroup.org
- [TCG EFI Platform] EFI TCG Platform Specification, version 1.2
www.trustedcomputinggroup.org
- [TCG EFI Protocol] EFI TCG Protocol Specification, version 1.2
www.trustedcomputinggroup.org
- [TCG TIS] TCG TPM Interface Specification www.trustedcomputinggroup.org
- [THREAT] [Frank Swiderski](#), [Window Snyder](#) “Threat Modeling,” Microsoft

[UEFI Book] Zimmer, Rothman, Hale, *Beyond BIOS*, Intel Press, 2006,
http://www.intel.com/intelpress/sum_efi.htm

[UEFI Main Specification] UEFI Specification, Version 2.3 www.uefi.org

[UEFI Shell Book] Rothman, Lewis, Zimmer, Hale, *UEFI Shell*, Intel Press, 2009,
http://www.intel.com/intelpress/sum_eshl.htm

[UEFI Overview] Zimmer, Rothman, Hale, “UEFI: From Reset Vector to Operating System,” Chapter 3 of *Hardware-Dependent Software*, Springer, February 2009

[UEFI PI Specification] UEFI Platform Initialization (PI) Specifications, volumes 1-5, Version 1.2 www.uefi.org

Authors

Vincent J. Zimmer (vincent.zimmer@intel.com) is a Principal Engineer with the Software and Services Group at Intel Corporation.

Shiva R. Dasari (shiva@us.ibm.com) is a Senior Software Engineer with the System X ESW Development at IBM Corporation.

Sean P. Brogan (spbrogan@us.ibm.com) is a Senior Software Engineer with the System X ESW Development at IBM Corporation.

This paper is for informational purposes only. THIS DOCUMENT IS PROVIDED "AS IS" WITH NO WARRANTIES WHATSOEVER, INCLUDING ANY WARRANTY OF MERCHANTABILITY, NONINFRINGEMENT, FITNESS FOR ANY PARTICULAR PURPOSE, OR ANY WARRANTY OTHERWISE ARISING OUT OF ANY PROPOSAL, SPECIFICATION OR SAMPLE. Intel disclaims all liability, including liability for infringement of any proprietary rights, relating to use of information in this specification. No license, express or implied, by estoppel or otherwise, to any intellectual property rights is granted herein.

Intel, the Intel logo, Intel. leap ahead. and Intel. Leap ahead. logo, and other Intel product name are trademarks or registered trademarks of Intel Corporation or its subsidiaries in the United States and other countries.

*Other names and brands may be claimed as the property of others.

Copyright 2009 by Intel and IBM. All rights reserved

