HELLENIC MEDITERRANEAN UNIVERSITY

SCHOOL OF ENGINEERING

DEPARTMENT OF ELECTRICAL & COMPUTER ENGINEERING



Bachelor's Thesis

Measured Boot and Runtime Integrity analysis in ARM-based devices.

Author: Michael Lazarou TP4446 **Supervisor:** Prof. George Kornaros

Heraklion

September 2024

Περιληψη

Η επαλήθευση της ακεραιότητας των συσκευών μέσω απομακρυσμένων σχημάτων πιστοποίησης γίνεται κρίσιμη πρόκληση όσο τα συστήματα γίνονται πιο περίπλοκα. Παρόλο που το hardware-based Trusted Platform Module (TPM) είναι ένα ουσιαστικό στοιχείο για το Measured Boot και την απομακρυσμένη πιστοποίηση, οι περιορισμοί κόστους και συσκευής το καθιστούν συχνά μη πρακτικό. Αξιοποιώντας το Trusted Firmware A (TF-A), ARM TrustZone, firmware TPM, and Integrity Measurement Architecture (IMA), αυτή η έρευνα στοχεύει στη δημιουργία ασφαλούς συλλογής αποδεικτικών στοιχείων για τα συστήματα απομακρυσμένης βεβαίωσης. Μέσω λεπτομερούς ανάλυσης της διαδικασίας εκκίνησης και των IMA/EVM kernel modules η έρευνα αναδεικνύει την πρακτική αποτελεσματικότητα αυτών των στοιχείων στην ενίσχυση των συσκευών που βασίζονται σε ARM έναντι απειλών ασφάλεια, συμβάλοντας σε πολύτιμες πληροφορίες για τον τομέα των ασφαλών υπολογισμών και ακεραιότητας σε IoT και ενσωματωμένα συστήματα.

Abstract

Verifying the integrity of devices through remote attestation schemes is becoming a crucial challenge as systems get more advanced. Although hardware-based Trusted Platform Module (TPM) is an essential component for Measured Boot and Remote Attestation, the cost and device constraints often make it impractical. By leveraging Trusted Firmware A (TF-A), ARM TrustZone, firmware TPM, and Integrity Measurement Architecture (IMA), this research aims to establish secure evidence collection for Remote Attestation schemes. Through detailed analysis of the boot process and the IMA/EVM kernel modules the research demonstrates the practical effectiveness of these components to in fortifying ARM-based devices against security threats, contributing valuable insights to the field of secure computing and integrity for IoT and embedded systems.

Table of Content

Table of Contents

Περιληψη.....4
Abstract.....4
Table of Content.....5
Table of Contents.....5
Figures.....7
Tables.....7
1. Introduction.....8
1.1 Motivation.....9
1.2 Research Goal.....11
1.3 Outline.....12
2. Background.....13
2.1 ARM TrustZone.....13
2.1.1 OPTEE Reference.....13
2.1.2 TEE Alternatives.....14
2.2 Trusted Firmware A.....15
2.3 Measured Boot.....16
2.4 Secure Boot.....16
2.5 Event Log.....16
2.6 Trusted Platform Module (TPM).....16
2.6.1 Types of TPM.....17
2.6.2 fTPM Reference.....18
2.6.3 Platform Configuration Registers (PCRs).....19
2.6.4 Tpm Tools.....20
2.7 Linux Integrity Subsystem.....21
2.8 Attestation.....22
3. Implementation details and Proof of Concept.....24
3.1 Experiments throughout the research.....24
3.2 Measured Boot Backends.....25
3.3 Boot Stages.....25

3.4 Critical Data 26
3.5 Measured Boot functions and Platform Interface 27
3.6 Measured Boot Proof of Concept with TF-A and fTPM 28
3.7 IMA configuration and details 35
3.8 Trusted Substrate Firmware overview 39
4. Literature Review 43
Conclusion 45
Bibliography 46
Terminology 52

Figures

Figure 1 Global Internel of Things (IoT) Market Share, By end-use Industry, 2023 [63] *8*
Figure 2 ARM Processor Families *9*
Figure 3 Architectural overview of Attestation protocol *10*
Figure 4 OPTEE Architecture diagram *14*
Figure 5 ARM Trusted Firmware Architecture *15*
Figure 6 Trusted Platform Module (TPM) Architecture diagram *17*
Figure 7 Different TPM types, *18*
Figure 8 Typical Synchronous Operation of the fTPM from Trustzone-based integrity measurements and verification using a software-based trusted platform module [23] *19*
Figure 9 Platform Configuration Register (PCR). PCR Usage *20*
Figure 10 Attestation diagram between Attestor and Verifier with the use of TPM *23*
Figure 11 Bootchain overview of the different components participate on during boot *26*
Figure 12 Trusted Firmware Platform Interface and Measured Boot functions *27*
Figure 13 After building and running the proof of concept, Linux and OPTEE logs in terminal_0 and terminal_1 respectively *29*
Figure 14 Userspace Linux terminal (left), OPTEE logs terminal (right) *30*
Figure 15 Example of ima_policy=tcb - IMA policy file *37*
Figure 16 Grub configuration to pass on kernel command-line parameters the IMA tcb policy (ima_policy=tcb) *37*
Figure 17 Securityfs path after IMA/EVM enabled *38*
Figure 18 IMA measurement list. *38*
Figure 19 IMA failure to find a running TPM device early on boot *39*
Figure 20 Trusted Reference Stack (TRS) Overview of key components *40*
Figure 21 Supported Boards for Trusted Substrate firmware. *41*
Figure 22 Measured Boot according to TRS project *42*
Figure 23 Overview of different component interacting with OPTEE in TRS *43*

Table 1 Measured Boot init function *29*
Table 2 Measured Boot measure image function *29*
Table 3 Measured Boot critical data measure function and finish *29*

Table 4 Event Log sample *32*
Table 5 Event Log sample *33*
Table 6 Event Log sample *34*
Table 7 Event Log sample *35*
Table 8 OPTEE terminal showing the tpm extend command logs *35*
Table 9 tpm2_pcrread function results in Linux userspace *36*
Table 10 IMA/EVM Kernel configuration options *37*

1. Introduction

The increasing prevalence of IoT or Internet of Things and the various network-connected devices that participate in everyday scenarios and critical operations, have increased the demand for heightened responsibility and awareness concerning security challenges. In recent years, an increasing number of small, interconnected devices have been joining and forming networks that support critical functions in industries such as healthcare, transportation, telecommunications, etc. with the global IoT market size to be valued at around
$715 billion in 2024, estimated to grow to $4,062.34 billion by 2032 [2].

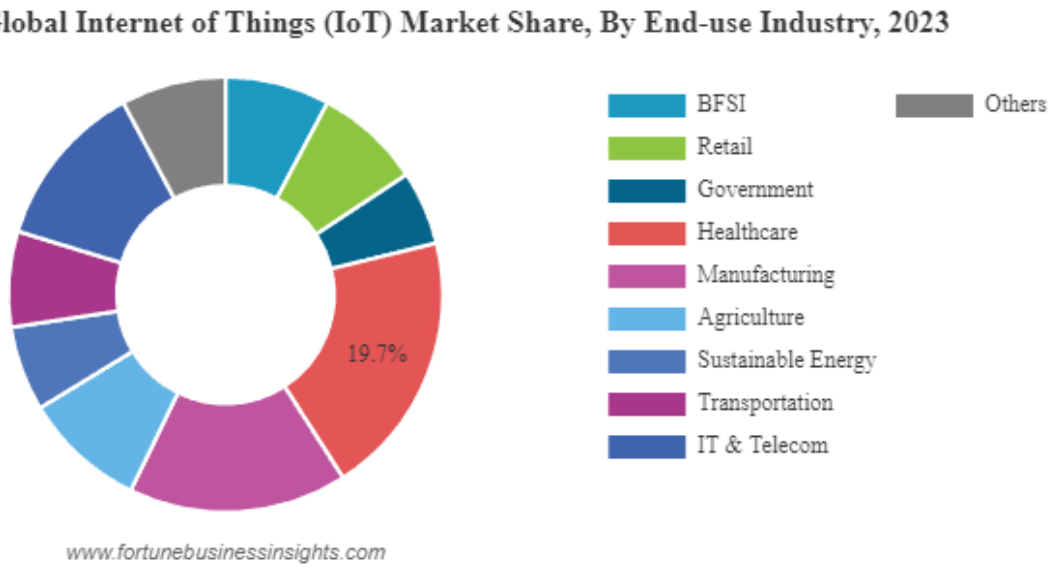


Figure 1 Global Internel of Things (IoT) Market Share, By end-use Industry, 2023 [63]

Ensuring the integrity of computing devices is crucial, particularly those employing ARM architecture. ARM processors and microcontrollers share the majority of mobile and embedded markets powering over 60% of all embedded devices, and 4.5 billion mobile phones with estimations to reach nearly 1 trillion by 2035 [1][2].
The proliferation of the IoT and small embedded devices operating even on the edge network analyzing data and taking part in critical operations, has underscored the necessity of safeguarding the integrity of the device firmware and runtime environment. This is crucial for mitigating unauthorized access, preventing data breaches, and firmware tampering, countering numerous cyber threats.
Due to the low resources, the vast majority of IoT devices lack embedded security measures, and the lessons learned from previous years emphasize their potential to instigate large-scale attacks, thereby compromising networks and organizations. Notable examples of such attacks include the Mirai botnet, Persirai, Torii, and BrickerBot. These incidents highlighted the evolution of botnets from networks of compromised PCs to networks of compromised IoT devices capable of launching extensive DDoS attacks [3]. Additionally, incidents such as the Verkada hack, where surveillance devices were compromised, or SCADA systems (e.g., Stuxnet attack) further

underscored the vulnerabilities of IoT devices and the severity of potential security breaches. Recent research also has been focused on the exponential increasing of security attacks in automotive industry and proposals have been made regarding secure local and remote communications between vehicles [63][64].
Healthcare and agriculture are also critical domains that small, interconnected devices participate in various operations; thus, the focus of recent years has been on developing security solutions to guarantee and attest their authenticity and their tamper-proof status.
The goal of implementing security mechanisms to a device is to ensure the integrity and the detection of an unauthorized tamper of the firmware, and the filesystem providing an overall perception of the device's condition.

1.1 Motivation

Computational devices, running on ARM architecture are met on various applications from small embedded devices to large servers, expanding to a variety of ARM-based processors as in Figure 2.

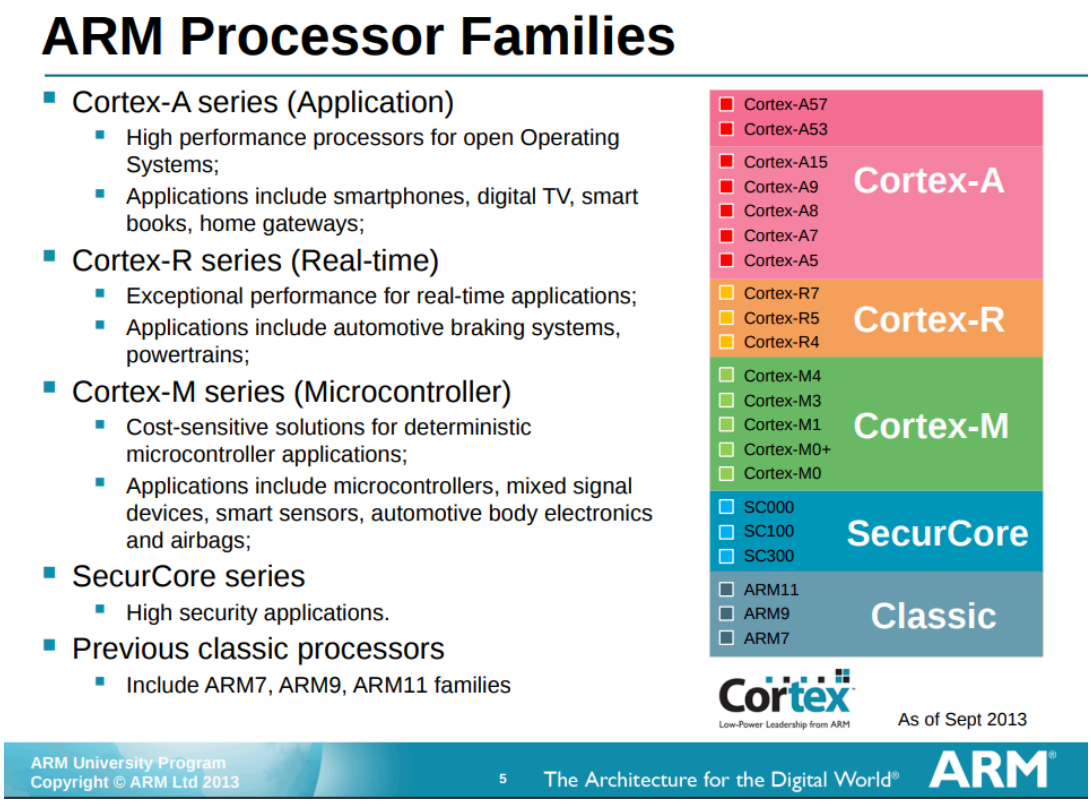


Figure 2 ARM Processor Families

Each device on a network needs to operate within its designated scope and under specific security policies to guarantee that the principles of Confidentiality, Integrity and Authentication are maintained.

In general, recent research has been focused on developing and discussing security mechanisms around IoT devices, including different software and hardware techniques and components respecting the limitations and capabilities of the targeted systems.
To be able to check the integrity of a device, various remote attestation mechanisms have been developed and researched. Remote Attestation is a security mechanism to remotely detect adversarial presence on a device and the process involves a Remote Verifier or a Remote Attestation Server asking an untrusted system or Prover to prove its authenticity. The Prover reports cryptographically signed measurements, or evidence through a secure communication protocol with the Verifier as the attestation architectural overview provided in Figure 3.

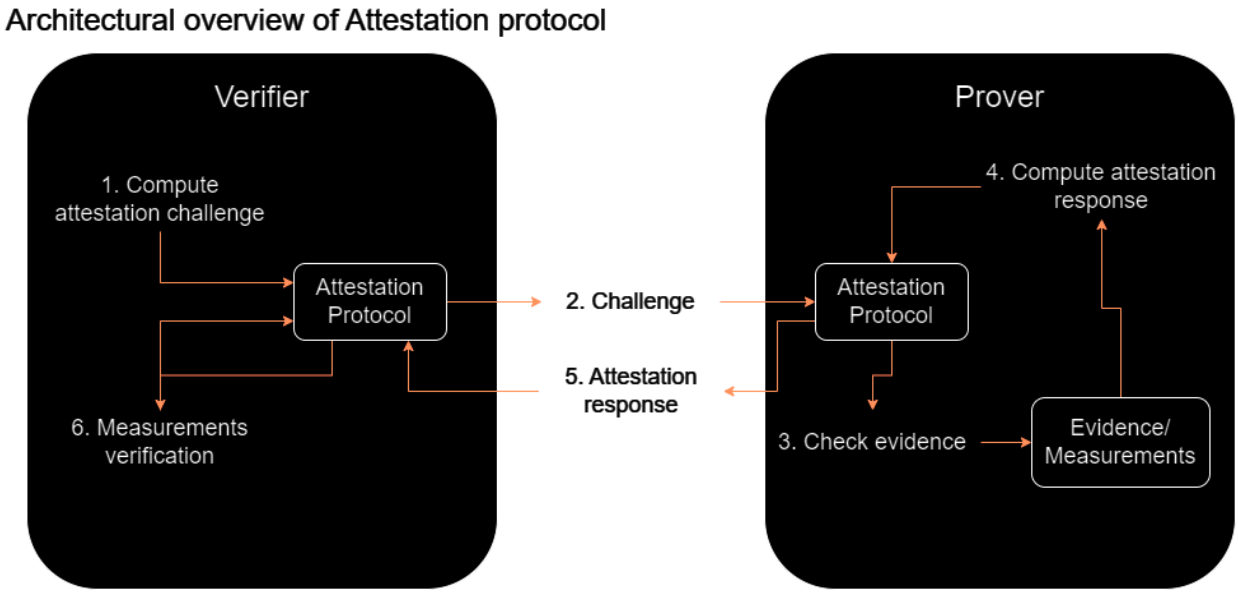


Figure 3 Architectural overview of Attestation protocol

Attestation mechanisms have been developed with different approaches depending on the scenario involved and the device's resources. Typically, these schemes rely on authentic cryptographic measurements (hashes) calculated on prover's boot process or measurements on critical data during run-time requested by a remote verifier aiming to compare them with known `good` values. The device or a swarm of devices usually leverage security components such as Trusted Platform Modules (TPM) or Trusted Execution Environments (TEE) to store the measurements securely and quote these cryptographic hashes solving the remote verifier's challenge and proving that have not been tampered with or compromised.
Integrity check have been applied to many different solutions encompassing embedded devices running microcontrollers seen in Wireless Sensor Networks (WSN), Edge computing devices running an operating system, receiving and analyzing data, and even virtual environments or containerized images running various services. Modern, and more advanced systems use ARM processors that boot from ROM, and subsequently load several firmware images such as Trusted Firmware, U-Boot and Trusted OS (e.g. OPTEE) until reach the Linux kernel and userspace filesystem, offering extensive option flexibility and configurations but also expanding the scope

of the security challenges. To take attestation measurements throughout the boot process and help prevent unauthorized firmware changes, Measured Boot mechanisms can be implemented. Measured Boot is the process of securely recording hashes of critical data and code at each stage in the boot chain before the code/data is used. In contrary with Secure Boot, Measured Boot does not authenticate the firmware or critical data loaded, but simply records their hashes to provide their state during boot and prevent or detect unauthorized tampering.
In a Measured Boot scenario, the system measures each boot component before its loading and extends the hash records into the TPM Platform Configuration Registers (PCR), establishing a Chain of Trust (COT). As the TPM stores these measurements securely, a third-party Remote Attestation scheme can request and compare them against the expected values. Measured Boot process covers the integrity measurements until OS kernel, from then the Linux Kernel Integrity Subsystem extends the ability to take important measurements of critical data in the filesystem during run-time. The Linux Kernel Integrity Subsystem aims to detect if files with high privileges have been maliciously altered and consists of two major components, that are based on the Trusted Computing Group's (TCG) standards, the Integrity Measurement Architecture (IMA) and the Extended Verification Module (EVM) that also use the TPM or TEE to ensure authentic measurements.

1.2 Research Goal

The primary objective of this thesis is to analyze and understand the boot states of ARM-based devices with Trusted Firmware – A (TF-A), Open Portable Trusted Execution Environment (OPTEE) and U-Boot until reaching the Linux operating system, alongside the IMA/EVM kernel modules. These components contribute to the collection of cryptographic evidence during the boot process and at runtime. TPMs with their cryptographic functions, hash extensions mechanisms to PCRs, TPM tools, and isolation features based on TCG standards, play a crucial role in Measured Boot and Remote Attestation procedures.
In this thesis, will be discussed the process of Measured Boot, leveraging ARM TrustZone as the key component that provides isolation, combined with Microsoft's firmware TPM (fTPM). While hardware TPMs are vital for Measured Boot and Remote Attestation due to their standards compliance and TCG's specification, they are not always feasible due to cost or device limitations. Authentic measurements for attestation schemes depend on a Root of Trust (RoT) for isolation and secure storage. Thus, TPMs and TEEs are used for such operations. Microsoft has patented the functionality of TPM as a Trusted Application running inside the TEE, combing hardware isolation of ARM TrustZone with the TPM features. Essentially, the fTPM, combines TCG's TPM specification with ARM TrustZone, supports most functions of a hardware-based TPM. TF-A, which acts as the primary bootloader loaded after ROM, can be configured to measure itself and each subsequent firmware until reaching the OS kernel. TF-A along with OPTEE can be configured to take measurements during boot process and populate the Event Log stored in OPTEE's secure storage for later use by the fTPM for extending the hashes to PCRs to ensure authentic records. Experiments and examples were conducted compiling and running the

proof of concept from Trusted Firmware documentation (Interaction between Measured Boot and an fTPM (PoC)) and Linaro's Trusted Substrate (TRS) project that have implemented security solutions for measured boot and runtime integrity. Additionally, efforts were made to establish a Chain of Trust (COT) for the STM32MP157-DK2 and STM32MP157-EV1 boards to explore the possibilities and limitations of building secure images for developing boards.

1.3 Outline

This thesis is structured to provide a comprehensive analysis and understanding of the security mechanisms in ARM-based systems, particularly focusing on the boot process, runtime integrity, and remote attestation schemes. Chapter 1 introduces the topic, outlining the motivation behind the research and the goals to be achieved. Chapter 2 explores the necessary background for understanding the core concepts and the components needed. It starts with the examination of the ARM TrustZone technology, including OPTEE OS as a reference implementation and further mentioned alternative TEEs. It continues with a detailed discussion regarding Trusted Firmware A (TF-A) and its role in the boot process followed by sections of Measured and Secured Boot methodologies. This chapter also covers the event log mechanism and provides an in-depth look at the Trusted Platform Modules (TPMs) with mentions to PCRs, TPM tools, and different types of TPM. Following this chapter there is an introduction to the Linux Integrity Subsystem and the concept of attestation with focus on the Trusted Computing Group (TCG) standards. Chapter 3 presents the implementation details and the proof of concept starting with configuration details of the images (TF-A, OPTEE, U-Boot, Linux kernel), followed by the measured boot implementation as provided from TF-A documentation. Additionally, kernel modification details were provided to enable the IMA/EVM modules to showcase the related filesystem measurements. In chapter 3, the Trusted Substrate project is also introduced as an overall solution that exists offering various security extensions to build and experiment. This thesis concluding includes a literature review to contextualize the research on existing studies for attestation schemes in for various devices and scenarios, a review of the results and a conclusion to summarize the findings and the potential future work exists. Additionally, a bibliography and a terminology section can be found to clarify the studies helped for this thesis and the key terms used throughout the document.

2. Background

The security of ARM-based devices consists of various security components and mechanisms. Recent ARM microcontrollers and microprocessors can leverage their dedicated ARM TrustZone (TEE) technology or a discrete TPM to take advantage of the hardware isolation. Trusted Firmware is an open-source reference implementation work as the initial bootloader and supports TEE in both Cortex-M and Cortex-A processors implementing trusted boot mechanisms. In complex systems, security and boot process rely on components such as OPTEE as trusted OS in TrustZone and U-Boot as the bootloader responsible of loading UEFI, initramfs, and Linux kernel. Upon reaching the operating system, security extensions like IMA and EVM kernel modules can provide further integrity measurements. When appropriately combined and configured, these security components offer security measurements and assurances for the system (e.g. Measured Boot) that remote attestation procedures can take advantage of and assess the overall status of a device.

2.1 ARM TrustZone

ARM TrustZone constitutes a comprehensive security extension for ARM-based processors and microcontrollers designed to create a secure isolated environment for protecting valuable assets and for executing sensitive code on the device. It divides the processor into two distinct execution environments the secure world and the normal world. The Secure world is used to run trusted code and trusted OS creating a Trusted Execution Environment (TEE) isolated from the normal world or Rich Execution Environment (REE) that hosts the main operating system and applications. The secure world has its own dedicated resources and has the capabilities of cryptographic operations, secure storage, and execution of trusted applications.
Secure and Normal worlds operate concurrently but independently with the Normal World being the default execution environment. The switching between two worlds is done by a specific instruction that transfers synchronous control between the secure and non-secure world. This instruction is called the Secure Monitor Call (SMC) and is implemented as part of the Security Extensions making the processor enter Monitor mode to support transitions between secure and non-secure state [8][9]. Numerous recent ARM processors, including Cortex A7, A8, A9, and A15, incorporate ARM TrustZone technology. Different components leverage this technology as a trusted entity for trusted boot, seal secrets, and device attestation procedures.

2.1.1 OPTEE Reference

OP-TEE (Open Portable Trusted Execution Environment) is an open-source implementation of a TEE that is compatible with ARM TrustZone technology. It provides a secure execution environment within the TrustZone, allowing trusted applications to run securely alongside the normal operating system. OP-TEE supports the GlobalPlatform API specification that includes the TEE Internal Core API which is the API exposed to Trusted Applications and the TEE Client

API [10] which describes the communication between the TEE and REE. OPTEE is conducted by several components such as a Linux kernel TEE framework and driver, a user space supplicant daemon (tee-supplicant) responsible for remote services expected by the TEE OS, and a privileged layer, executing at ARM secure PL-1 (v7-A) or EL-1 (v8-A) level. Using the GlobalPlarform API developers can implement various trusted applications for running critical functions isolated in OPTEE, generate cryptographic keys, and seal secrets to secure storage.

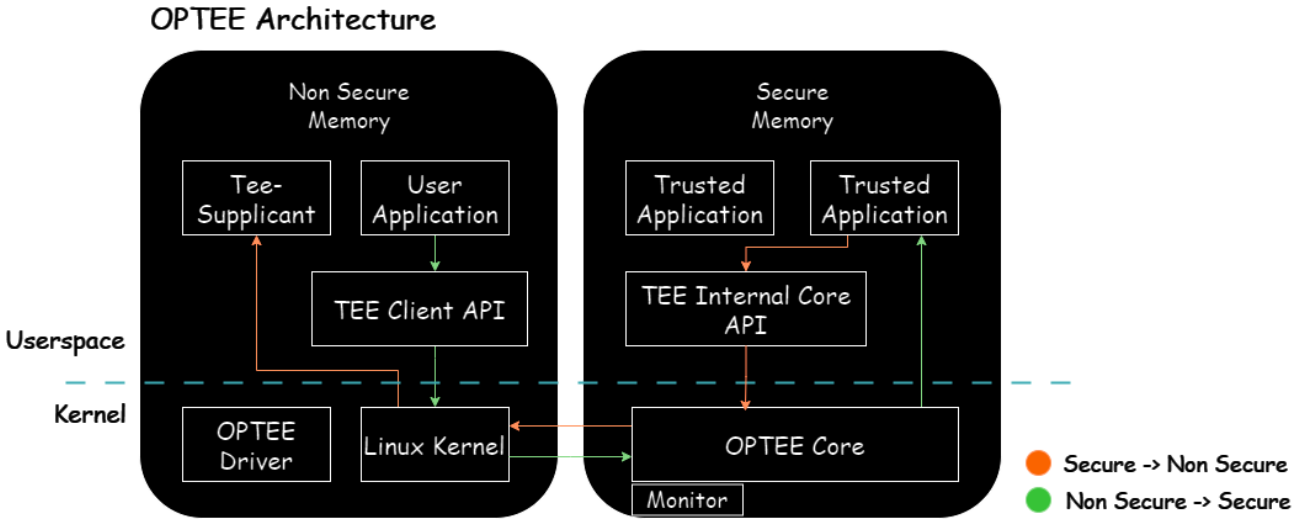


Figure 4 OPTEE Architecture diagram

Numerous recent ARM processors, including Cortex A7, A8, A9, and A15, incorporate ARM TrustZone technology. Different components leverage this technology as a trusted entity for secure storage, and cryptographic operations. This integration facilitates trusted boot and the execution of trusted code, creating a highly secure isolation environment. This environment ensures the security of peripherals, firmware, and data. Within the Trusted Execution Environment, various security operations come to fruition, encompassing cryptography, authentication, attestation mechanisms, and more.

2.1.2 TEE Alternatives

Different companies have been involved in the evolution and research on the security of the devices and the implementation of TEEs to provide system resilience and isolated execution of critical operations. Apart from OPTEE, which leverages ARM TrustZone several alternatives exist in the market and the industry. Intel's Software Guard Extensions (SGX) is one example that is a set of instruction codes in Intel's CPUs that can implement a TEE using protected private regions of the CPU's memory, called enclaves. Samsung Knox [17] is another implementation particular for Samsung Mobile devices, offering a secure environment for applications and sensitive data, and Qualcomm's QSEE (Qualcomm Secure Execution Environment) which also applies to mobile devices. Depending on the security needs and the requirements of the platform, these TEEs provide diverse features and various implementations.

2.2 Trusted Firmware A

Trusted Firmware-A (TF-A) is an open-source implementation of secure world software designed to provide a trusted foundation for ARM-based devices. TF-A is useful in implementing measured and secured boot as it ensures the integrity and authenticity of the firmware for boot stages on a device. TF-A initializes and manages the next boot process components, leveraging ARM's TrustZone technology with the key components and features including a secure bootloader or FSBL, which verifies the integrity of the firmware (SSBL) before loading. Furthermore, includes runtime services such as Secure Monitor Calls (SMCs), which achieve secure communication between the secure world and the normal world. Additionally, it includes support for securely initializing various platform-specific hardware components like device tree blobs. By providing a standardized and well-audited codebase, it helps manufacturers implement security features, for firmware integrity and attestation mechanisms making it an essential component of ARM-based devices.

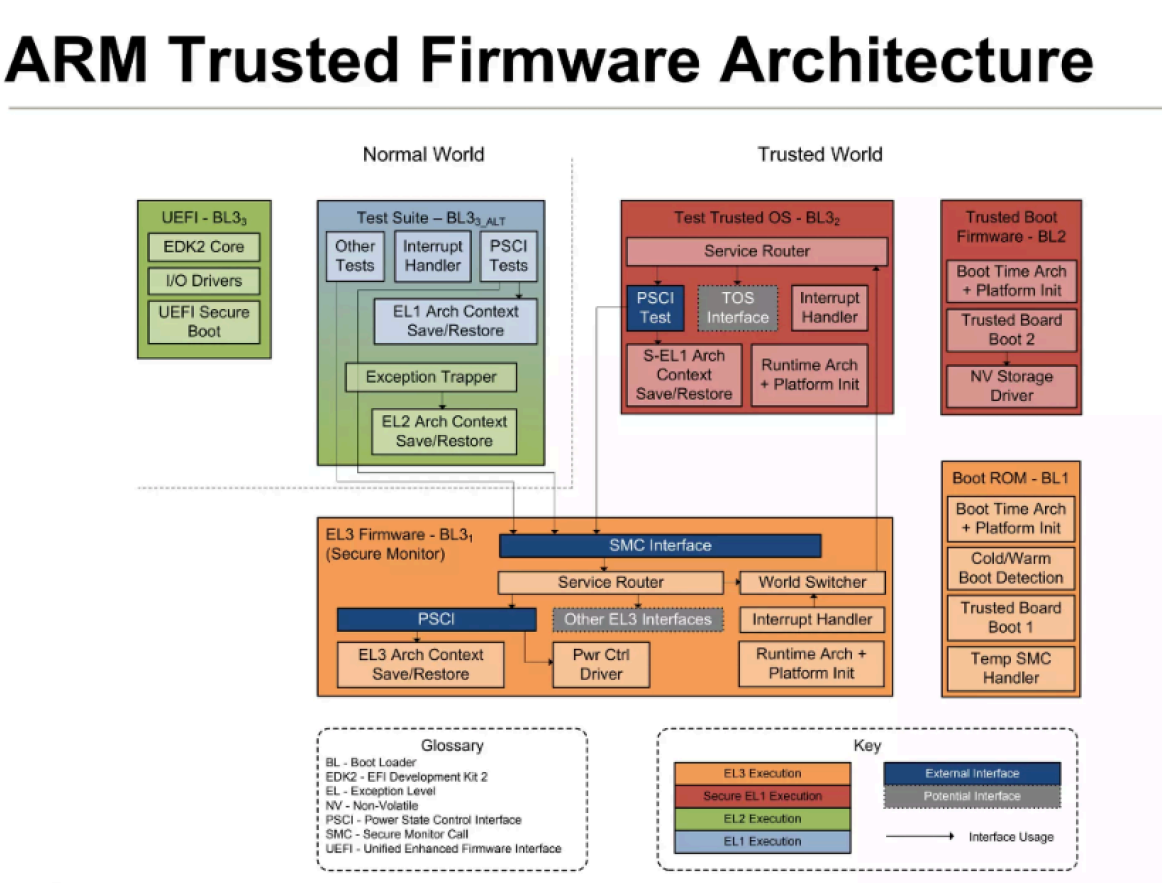


Figure 5 ARM Trusted Firmware Architecture [66]

2.3 Measured Boot

Measured boot is a security mechanism designed to ensure the integrity and authenticity of a device's boot process by taking cryptographic measurements of each component loaded during the boot sequence. Leveraging of TF-A as the FSBL, OPTEE and U-Boot as the SSBL a Boot chain of trust can be achieved having each firmware component checked before loaded into RAM and executed. Essentially TF-A takes initial measurements of its own code and configuration, then loads OP-TEE which includes the fTPM as a trusted application and U-Boot into external RAM and jumps to them. The measurements are consumed by the Event log which is later used by fTPM to extend these measurements to the PCRs creating a secure record of the boot process. The Event log also consumes the hashes of the Linux kernel and initrd. This chain of trust provides cryptographic proof of the device’s state from the moment of power-up to the loading of the operating system and using attestation procedures the trustworthiness of the system can be verified against unauthorized changes.

2.4 Secure Boot

Secure boot is a process that ensures a device boot and operates using software that the OEM trusts. It uses cryptographic techniques like public keys, signatures, and hashes to verify the system's software components. Each step in the boot process verifies the next step before continuing, creating a chain of trust.

2.5 Event Log

The Event Log is an essential component that consumes cryptographic measurements of all boot stages and important events during the boot process. Each entry in the Event Log includes a description of the event, a hash of the component or data involved, and metadata providing context. With these logs it is ensured that every step from the initial firmware load up to the operating system boot is documented, creating a verifiable chain of trust. Event Log serves as a source of trust for the state of the device, allowing the detection of unauthorized modifications. When the trusted environment Trusted Firmware-A (TF-A) is initialized after ROM, it begins the logging process, capturing measurements for components like OP-TEE, U-Boot, and the kernel. These logs are essential for attestation, as they provide the data needed to verify the integrity of the system against known good values.

2.6 Trusted Platform Module (TPM)

A Trusted Platform Module is a security component whose key attribute is to provide methods for collecting and reporting hardware and software identities ensuring that they operate in a desired/trusted way. (TPM is also known as ISO/IEC 11889 an international standard for a secure cryptoprocessor). TPMs provide security functionality that expands in cryptographic operations,

platform identity, secure storing credentials, attestation mechanisms, hashing, random number generation, etc., and is used with open-source APIs to enable these features or for custom software development.

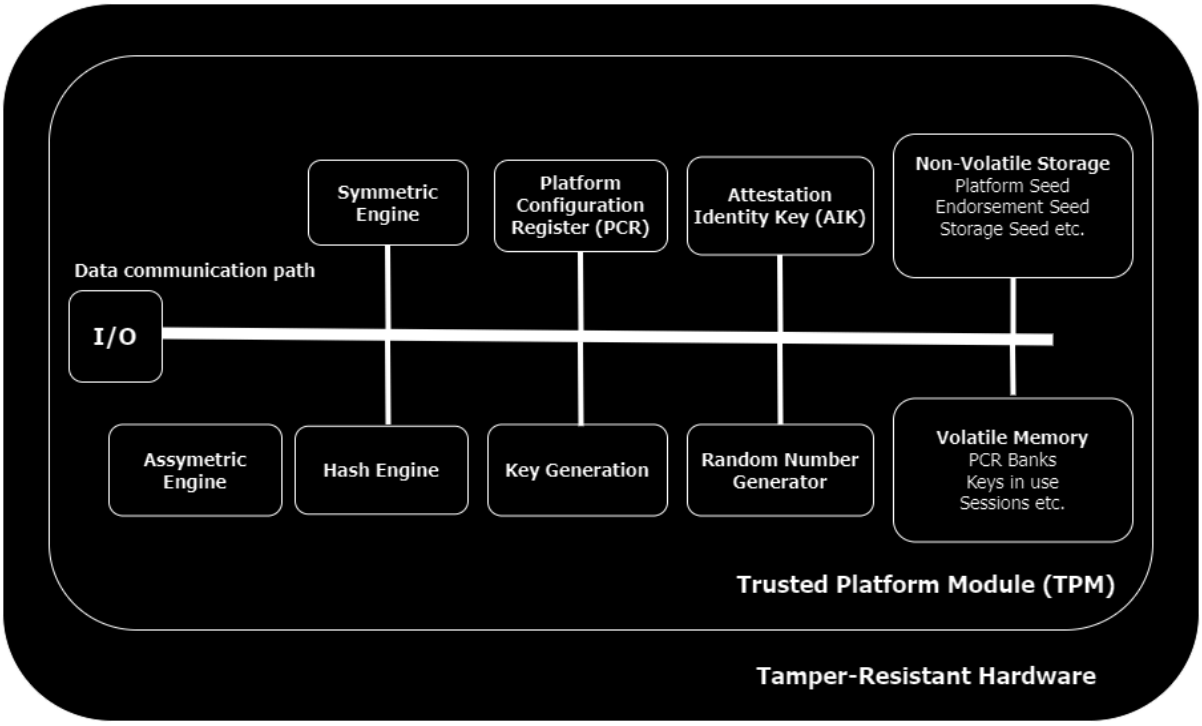


Figure 6 Trusted Platform Module (TPM) Architecture diagram

2.6.1 Types of TPM

Commonly a TPM component is a physical chip isolated from the rest of the platform, a hardware separate processor, that defines a hardware root of trust (HRoT) and offers security benefits in PCs, servers, networking gear, and embedded systems (for more than a decade). There are different types of TPMs and traditionally discrete TPM is implemented as a separate chip or integrated TPM inside the chipset. When virtualization exists in a cloud environment Virtual TPM or Hypervisor TPM can use commands from the hypervisor's physical TPM to each virtual machine. Firmware TPM (fTPM) is implemented in the TEE on the same CPU separated from the rest of the platform's operating system, taking advantage of the ARM Trustzone on ARM-based devices. AMD and Intel are using such technologies as AMD fTPM and Intel PTT (Platform Trusted Technology) to enhance the security of critical operations. In addition, Software TPM is a TPM emulator used basically for testing. The highest level of security is achieved with discrete TPM and is used in critical solutions offering tamper resistance with the chip from physical attacks unlike the rest types but can be a problem with low power consumption or small devices. Firmware TPM or fTPM besides the lack of tamper resistance is dependent on more security aspects such as the TEE operating system and is the type that is used

in this thesis as it is the most suitable for low-resource devices.

TRUST ELEMENT	SECURITY LEVEL	SECURITY FEATURES	RELATIVE COST	TYPICAL APPLICATION
DISCRETE TPM	HIGHEST	TAMPER RESISTANT HARDWARE	$$$	CRITICAL SYSTEMS
INTEGRATED TPM	HIGHER	HARDWARE	$$	GATEWAYS
FIRMWARE TPM	HIGH	TEE	$	ENTERTAINMENT SYSTEMS
SOFTWARE TPM	NA	NA	cc	TESTING & PROTOTYPING
VIRTUAL TPM	HIGH	HYPERVISOR	c	CLOUD ENVIRONMENT

Figure 7 Different TPM types,

2.6.2 fTPM Reference

Microsoft has conducted significant research into integrating the TPMs functionalities into ARM TrustZone, aiming to enhance the authenticity of devices by leveraging security features without a separate hardware piece. This implementation, detailed in U.S. Patent No. US20160048678A1 outlines the fTPM run as a trusted application within the TrustZone environment, aiming to provide a secure execution environment isolated from the normal operating system, safeguarding sensitive activities such as cryptographic functions, attestation mechanisms, and trusted or measured boot procedures. Firmware TPM plays an important role in ensuring the integrity of the device using functionalities of a discrete TPM executed in the Secure World of the processor and TPM commands can be called with the tpm2-tools. It is installed into the Secure World of the device before the processor switches to Normal World during system initialization, the the TPM driver on Linux kernel (CONFIG_TCG_FTPM_TEE) [22], allows programs to interact with the fTPM, as a hardware-based TPM and is used for sending and receiving commands through **/dev/tpmX** character device. Communication with the ARM TrustZone to execute the different commands is achieved via SMC instruction to enter the Monitor. The Monitor passes the command to the fTPM trusted application instance in the Secure World and executes it, writes the response to shared memory, and returns to the Monitor and the Normal World.

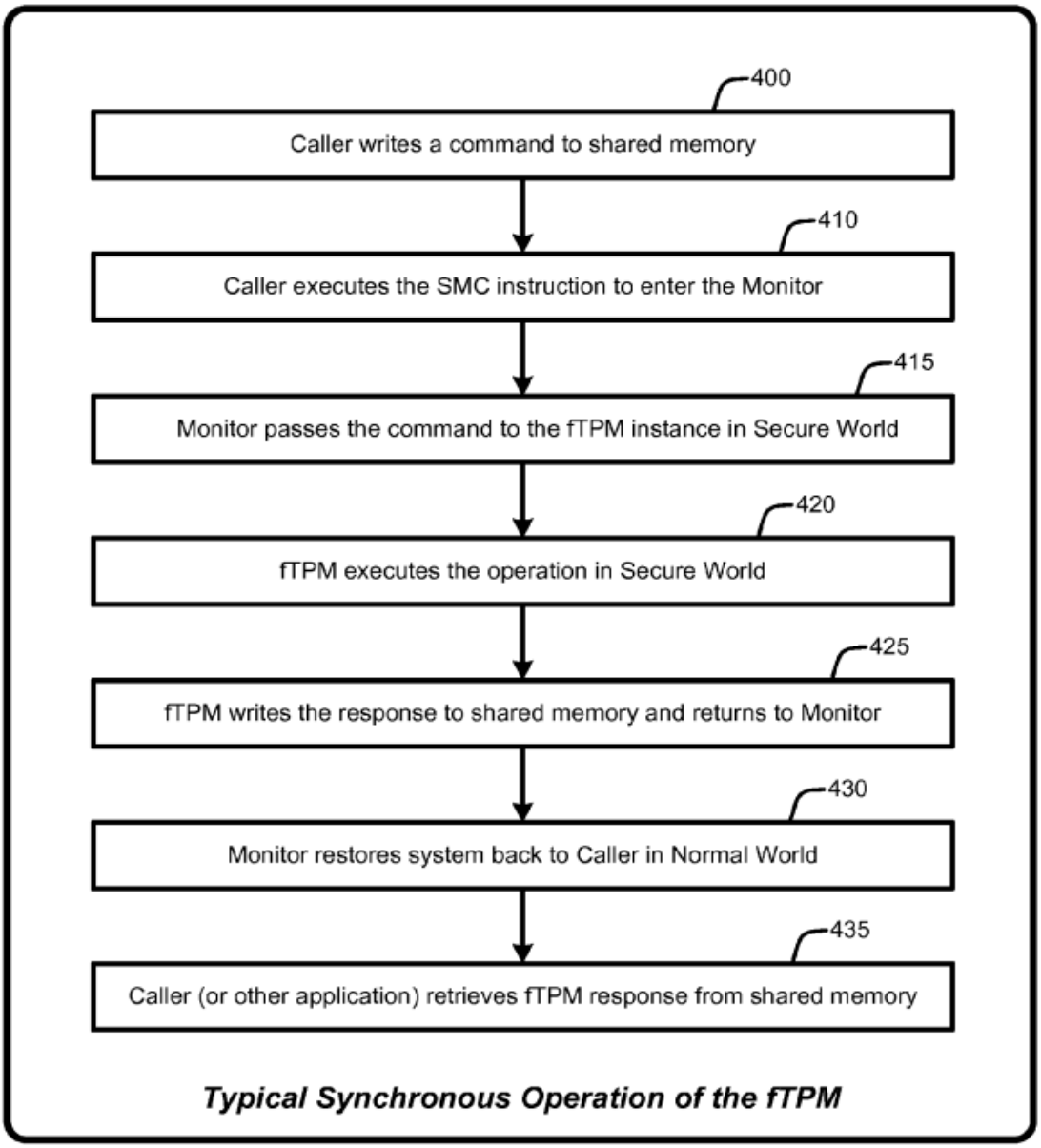


Figure 8 Typical Synchronous Operation of the fTPM from Trustzone-based integrity measurements and verification using a software-based trusted platform module [23]

Platform Configuration Registers (PCR), an essential feature of a hardware TPM, are also implemented in fTPM, providing a safe place for measurements and hashed values of critical system data. Measured Boot can use the PCRs to extend the measurements to the event log and create reference values for attestation protocols. Additional TPM use cases can include, sealing secrets, disk encryption, and secure key generation offering significant.

2.6.3 Platform Configuration Registers (PCRs)

Platform Configuration Registers (PCRs) are specialized registers usually within a TPM that cryptographically store measurements securely and represent a specific aspect or stage of the

boot process, such as the initial firmware, bootloader, kernel, and other critical components. Typically, these registers can be read to report their state by tpm2-tools and can only be written by the extension operation. This extension operation depends on the last value of the register that is stored and extending the new one creates a cryptographic blockchain as it follows:

$$PCR\ new\ value\ =\ Digest\ of\ (PCR\ old\ value\ ||\ data\ to\ extend)$$

In other words, the old PCR value is concatenated with digest data to be extended. The TPM digests the result and stores the new digest as the new PCR value.

TPM implements 24 of these special registers usually organized in banks, each designed for a specific hash algorithm such as SHA256, or SHA512. Essentially PCRs are used to provide a platform's firmware state and the history of critical software storing measurements to offer them in an authorization protocol when asked for a signed attestation quote of the values to compare them with known good ones.

Table 1 PCR Usage

PCR Index	PCR Usage
0	SRTM, BIOS, Host Platform Extensions, Embedded Option ROMs and PI Drivers
1	Host Platform Configuration
2	UEFI driver and application Code
3	UEFI driver and application Configuration and Data
4	UEFI Boot Manager Code (usually the MBR) and Boot Attempts
5	Boot Manager Code Configuration and Data (for use by the Boot Manager Code) and GPT/Partition Table
6	Host Platform Manufacturer Specific
7	Secure Boot Policy
8-15	Defined for use by the Static OS
16	Debug
23	Application Support

Figure 9 Platform Configuration Register (PCR). PCR Usage

2.6.4 Tpm Tools

The tpm2-tools are a collection of programs that provide communication between TPM and the operating system through the /dev/tpmX path. These tools are essential for managing and using

TPM's security and cryptographic functionalities and include a wide range of commands for different operations such as:

- **tpm2_getcap**: Show TPM capabilities and properties.
- **tpm2_create**: Creates various TPM objects, like keys and sealed data.
- **tpm2_pcrextend**: Extends PCRs with specified values.
- **tpm2_pcrread**: Displays PCR values.
- **tpm2_quote**: Provides quote and signature for given PCRs.

The tpm2-tools are significant for leveraging TPM for secure boot, key management, encryption, and other security-related tasks on Linux systems.

2.7 Linux Integrity Subsystem

The Linux Integrity Subsystem is a vital component of the Linux kernel designed to maintain the overall integrity of the system data and prevent unauthorized tampering with critical files.
It aims to detect if files accidentally or maliciously changed, appraise files against good measurements stored in extended attributes, and enforce local file integrity. The are also more security extensions on kernel such as Mandatory Access Control (MAC) protections provided by Linux Security Modules (LSM) that contain SELinux and SMACK and help with the file integrity of system. Kernel integrity subsystem complement these modules and provide several modules such as:

- **Collect** - measure a file before it is accessed.
- **Store** - add the measurement to a kernel resident list and, if a hardware Trusted Platform Module (TPM) is present, extend the IMA PCR
- **Attest** -if present, use the TPM to sign the IMA PCR value, to allow a remote validation of the measurement list.
- **Appraise** - enforce local validation of a measurement against a 'good' value stored in an extended attribute of the file.
- **Protect** - protect a file's security extended attributes

Specifically:

- **Integrity Measurement Architecture (IMA)**: This Linux kernel module helps the maintenance of security in the system by verifying the authenticity of the files and critical system components. The key point of this module is introducing hooks within the Linux kernel for creating and collecting hashes of files when opened before their contents are accessed for reading or execution. IMA maintains a runtime measurement list over the Linux filesystem based on a measurement policy and can be used for auditing and attestation purposes. By comparing the hash list of the filesystem with known good values in a remote attestation protocol the authenticity of the device operation can be assessed. Furthermore, it supports IMA-Appraisal that locally validates the current state

of a file comparing its previous value stored in the measurement list and blocks the operation if the values do not match. IMA has been included in the Linux kernel since the 2.6.30 version while in some distros is enabled by default (Fedora/RHEL) and in others.

- **Extended Verification Module (EVM)**: This is a Linux kernel module that is designed to protect files' extended attributes (also known as xattr). Extended attributes in files are value pairs associated permanently with files and directories and like IMA and SELinux, EVM cryptographically hashes the corresponding xattr values. Essentially, EVM monitors changes in extended attributes and different security components like IMA can store critical data in them. EVM uses Trusted Keys that are initialized on boot to hash the extended attributes of files and assess the integrity of a running system.

The Linux Integrity Subsystem has a crucial role in ensuring the security and the integrity of a device and is part of an Integrity Architecture based on Trusted Computing standards, such as those set by the Trusted Computing Group (TCG). These standards include the Trusted Platform Module (TPM), Trusted Boot, Trusted Software Stack (TSS), Trusted Network Connect (TNC), and Platform Trust Services (PTS). IMA and EVM modules, integral parts of the integrity subsystem, can operate on platforms without a hardware TPM, although without the hardware guarantee of compromised detection.

2.8 Attestation

Attestation is a security mechanism by which appropriate evidence is requested and supplied to verify the trustworthiness of a device. Essentially, a target device known as the Prover, which is considered untrusted, generates cryptographic proof such as hashes, checksums, or computational results linked to hardware, firmware, filesystem, or software. These proofs or measurements are then securely transmitted to the Verifier, who is responsible for assessing the values against good ones and determining the device's state. The reliability of the attestation process depends on the presence of a root of trust on the Prover's side, for the secure parsing and storing of the related measurements, such as TPM, TEE, SGX, or Secure Element (SE). The components are designed to ensure that evidence has not been tampered with. Attestation of a device or a swarm of devices can be performed locally or remotely based on the platform requirements and limitations. It is a crucial mechanism for maintaining the integrity of the systems, for example, in IoT deployments where verifying the state of the participating nodes ensures the network's overall security.
Remote Verifiers need to be capable of providing attestation mechanisms to multiple devices and enough resources to manage integrity responses along the devices. A necessity also is the tracking of attested devices from the verifier server for mapping the connected nodes' state and for further analysis such as forensics or a blockchain-based log system. Remote attestation can be performed in different approaches:
Static Remote Attestation includes measurements of the firmware or software in a device at a specific operation time such as during a measured boot process and the attestation consists of the

comparison of predefined values. This assessment can be used to authenticate the system's integrity and state, although is a static validation as cannot detect component changes after measurement occurs.

Dynamic Remote Attestation or runtime attestation refers to continuously monitoring the device throughout its operation, periodically or on specific event triggering. This dynamic validation of the current state of the device's critical components can detect unauthorized modifications, malware, or other security breaches in real-time.
The Trusted Computing Group (TCG) is a not-for-profit organization formed to develop, define, and promote open, vendor-neutral, global industry specifications and standards, supportive of a hardware-based root of trust, for interoperable trusted computing platforms [25].
TCG's has conducted significant effort developing the TPM which as a security co-processor can be leveraged as the Root of Trust (ROT). TPMs can compute integrity checksums of the loaded memory at boot time or protect the system's integrity by sealing the operating system with an encryption key, taking boot measurements, extending them to PCRs for each component loaded on the boot process, and assessing the measurements through remote attestation procedures [5].
TCG has introduced a hardware-based remote attestation scheme as can be found in TCG Trusted Attestation Protocol (TAP) [24]. Essentially, the TCG specification outlines roles and message flows for remote attestation, with the Verifier and Attester being the primary roles. The Verifier sends a challenge to the Attester, which then collects and provides attestation evidence, typically including the boot state, configuration, and identity. The Endorsement Key (EK) certificate can provision attestation identities, while the AK, which may be pre-provisioned as an Initial Attestation Key (IAK), signs the attestation evidence. The Verifier then checks this evidence against its trust policy to confirm device authenticity and manufacturer approval. This remote attestation procedure with the TPM as ROT is supported and implemented by the tpm2 tools as mentioned in Remote Attestation with Tpm2 Tools [26].

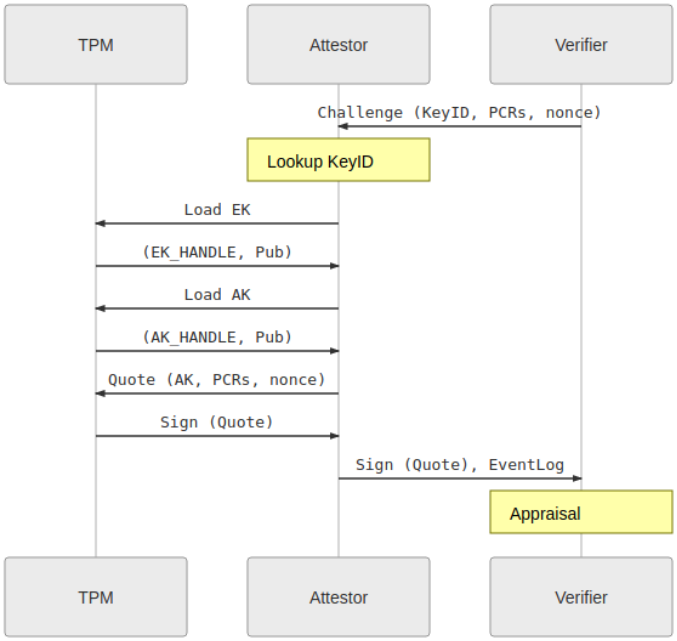


Figure 10 Attestation diagram between Attestor and Verifier with the use of TPM

3. Implementation details and Proof of Concept

3.1 Implementation methodology

After experimenting with security mechanisms for integrity and attestation for this thesis several stages were passed with examples like:

- Developing kernel driver to scrap information of processes running on Linux OS.

Details: Getting to know kernel modules and drivers, made a kernel driver that could scrap different important information from running processes to investigate potential hash measurements in critical data.

- Developing a Trusted Application and enabling IMA kernel module on kernel to scrap hash measurements collected by IMA and secure store them in secure storage in OPTEE.

Details: Successfully a Trusted Application was developed for OPTEE that when executed could take the measurements stored in **/sys/kernel/security/ima/ascii_runtime_measurements** path by IMA and store them into secure storage in OPTEE.

- Experiments with fTPM trying to build it and get it work on STM32MP157C-DK2 and STM32MP157C-EV1 to understand its requirements, the communication with fTPM kernel driver, and the TPM tools.

Details: The fTPM Trusted Application can be built with ARM aarch32 toolchains for STM32MP1 boards and can be ran only after tee-supplicant has been executed. After building the fTPM TA and configuring the Linux kernel with the Firmware TPM Driver a modification in the device tree of the platform is also a requirement to recognize the fTPM as a device.

- Investigate if fTPM can leverage IMA Measurement List early on boot to extend hashes to PCRs

Details: Several implementations were tried such as compiling fTPM as an Early Trusted Application starting at BL32 from OPTEE core or implementing tee-supplicant as a service starting on initramfs to be ready for fTPM to start also as a service.

Although the important experience gained working with all the mentioned examples and security components, there were more complex operations that needed to be done to have an optimal security solution. Getting more into the scope of the thesis the Measured Boot mechanism investigated that can be initiated from Trusted Firmware A, the widely used bootloader that runs on EL3, the most privileged Exception level on ARM. Measured Boot is the mechanism that take cryptographically records of the critical firmware images loaded during the boot process and can be used combined with a TPM or fTPM to later send the measurements to a remote attestation challenger check the device's security status. Measured Boot, in contrast to Secure Boot, does

not authenticate the firmware images loaded, but simply records what critical data was present on the system during boot.

3.2 Measured Boot Backends

Measured Boot implementation supports different Measured Boot Backends for the boot process events to be securely stored such as:

1. **Event Log**: Consumes the boot process chain and holds a record of measurements made into the Measurement Slot or PCR. TCG EFI Protocol Specification [20] provides details on how to measure the different components and specifies the architecture of this log.
    a. PCR0 is used for images measurements.
    b. PCR1 is used for critical data measurements.
2. **Runtime Security Engine (RSE)**: A physical backend to extend the measurements.

This thesis focuses on TCG Event log backend which stores measurements in secure memory. Essentially, during device boot, BL2 (Trusted Firmware) needs to store the Event log in secure storage until fTPM service is ready and operational. When fTPM service is started needs to be able to access the event log to extend the measurements that had consumed. For that reason, the PTA_SYSTEM_GET_TPM_EVENT_LOG syscall was implemented in version 3.9.0 of OP-TEE which allows the fTPM to pass a copy of the Event Log and any TA requesting it.

3.3 Boot Stages

In a Measured Boot implementation all firmware images and their corresponding configuration files, if any, must be measured. The bootchain consists of several components as stated in Figure 11 starting from CPU reset and ROM, up to the Linux OS. ROM, as the first link in the chain of trust, is the first code executed by the processor and it fits in a few Kbytes mapping its data to the embedded RAM, initializing the logic needed to select the boot device from which the FSBL (TF-A) is loaded into the embedded RAM. Typically, trust is established by authenticating the FSBL before starting it, and then each following component authenticates the next one, up to the kernel OS.
The FSBL (TF-A) is responsible for authenticating and then loading and jumping to the SSBL running in a wide RAM, initializing several Measured Boot functions that will be discussed later in this chapter. TF-A except from the SSBL image that is usually U-Boot in Non-Secure World, also uses Secure Monitor exception in Secure World and loads OPTEE as the trusted OS of the system. All the related firmware images are cryptographically measured before each boot stage.

For AArch64, it is divided into five steps (in order of execution):

- Boot Loader stage 1 (BL1) *AP Trusted ROM*
- Boot Loader stage 2 (BL2) *Trusted Boot Firmware*

- Boot Loader stage 3-1 (BL31) *EL3 Runtime Software*
- Boot Loader stage 3-2 (BL32) *Secure-EL1 Payload* (optional)
- Boot Loader stage 3-3 (BL33) *Non-trusted Firmware*

For AArch32, it is divided into four steps (in order of execution):

- Boot Loader stage 1 (BL1) *AP Trusted ROM*
- Boot Loader stage 2 (BL2) *Trusted Boot Firmware*
- Boot Loader stage 3-2 (BL32) *EL3 Runtime Software*
- Boot Loader stage 3-3 (BL33) *Non-trusted Firmware*

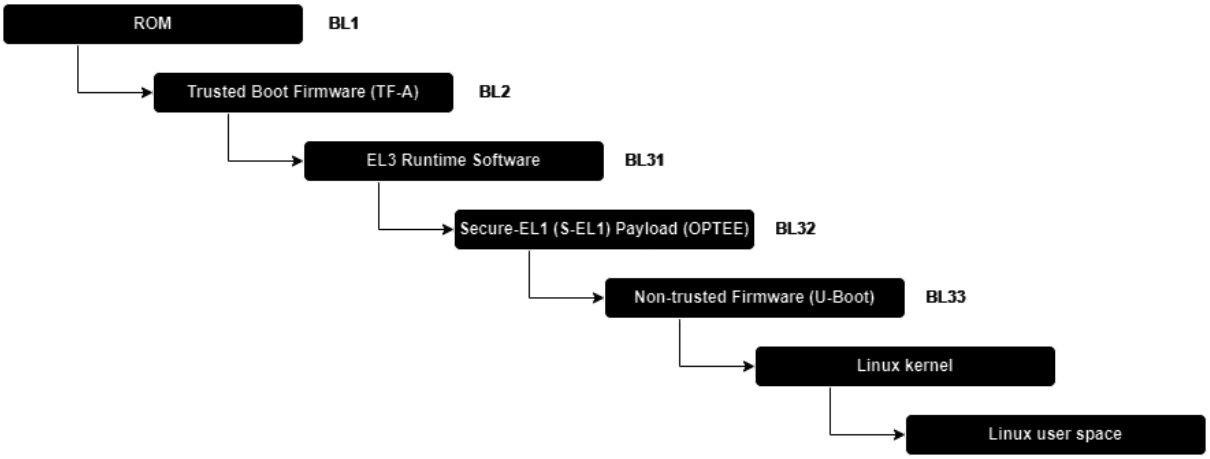


Figure 11 Bootchain overview of the different components participate on during boot

3.4 Critical Data

Furthermore, specific data pieces in each platform might need to be measured. These data are referred to as critical data. From TF-A documentation, critical data for the platform can be determined using the following criteria:

1. Data that influence boot flow behavior such as:
    - Configuration parameters that alter the boot flow path.
    - Parameters that determine which firmware to load from NV-Storage to SRAM/DRAM to pass the boot process successfully.
2. Hardware configuration settings, debug settings, and security policies need to be in a valid state for a device to maintain its security posture during boot and runtime.
3. Security-sensitive data that is being updated by hardware.

Examples of Critical data:

1. The list of errata workarounds being applied at reset.
2. State of fuses such as whether an SoC is in secure mode.
3. NV counters that determine whether firmware is up-to-date and secure.

Measurement slot resides in a trusted module and can be either a secure register or memory and is used to provide a cryptographic record of firmware and critical data on a system. It uses an

extend operation, which is a one-way hash of all previous measurements and the new one as with the PCR extension mechanism, thus captures cannot be removed or overwritten.

3.5 Measured Boot functions and Platform Interface

TF-A as a reference implementation of secure world software for ARM A-profile architectures both Armv8-A and Armv7-A provides a suitable starting point for developing secure world boot and runtime firmware. It incorporates Secure Monitor as Exception Level 3 (EL3) and several functions that include support for Measured and Secure Boot. Essentially, every image, when successfully loaded in memory and authenticated, in the case of Secure Boot, gets cryptographically measured. Additionally, the platform can measure critical data at any point during the boot process as in the diagram provided by TF-A documentation:

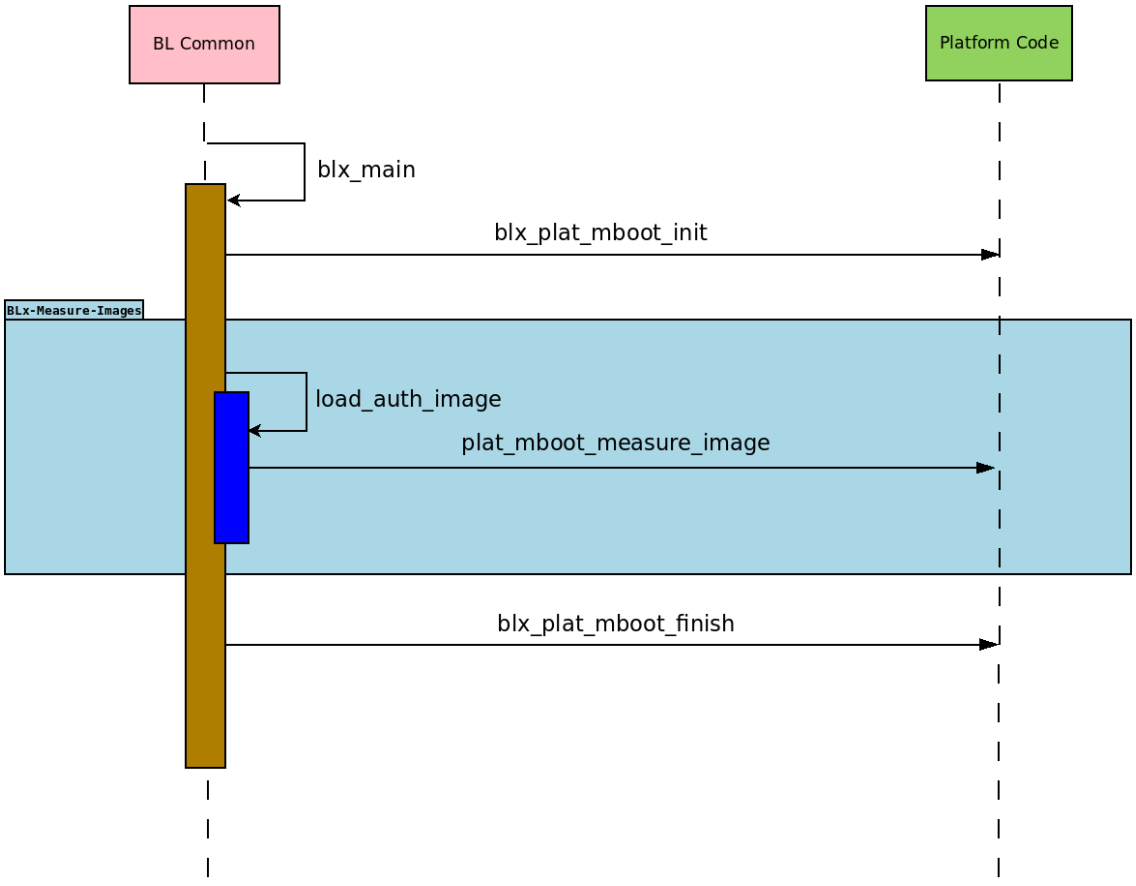


Figure 12 Trusted Firmware Platform Interface and Measured Boot functions

Some of the related functions that implement the boot operation and are used by BL1 and BL2 are declared in **include/plat/common/platform.h** at the TF-A source code [21] and some important of them are mentioned below:

- blx_plat_mboot_init()

```
void bl1_plat_mboot_init(void);
void bl2_plat_mboot_init(void);
```

Table 1 Measured Boot init function

Initialize the supported Measured Boot backends by the platform (e.g. Event Log, RSE). In the PoC that will be discussed later, the BL1 function is used to initialize the Event Log backend driver and to write header information in the Event Log buffer. The BL2 function is used to initialize the Event Log buffer with the received information from the BL1 and result in panic in case of errors.

- plat_mboot_measure_image()

```
int plat_mboot_measure_image(unsigned int image_id, image_info_t *image_data);
```

Table 2 Measured Boot measure image function

Measures the related image using a hash function of the crypto module and records the measurement to the corresponding backend (Event Log or RSE), as it is also given the PCR number as ID. In the PoC the function after measuring the given image records the measurement in the Event Log buffer initialized previously.

- blx_plat_mboot_finish()
- plat_mboot_measure_critical_data()

```
void bl1_plat_mboot_finish(void);
void bl2_plat_mboot_finish(void);
```

Table 3 Measured Boot critical data measure function and finish

These functions pass the related details of the backends such as start address and size to Normal or Secure World, measure all critical data if any exist, and result in panic in case of errors. In the PoC the BL1 function is used to pass the base address of the Event Log buffer and its size to BL2 via **tb_fw_config** to extend the measurements of the images loaded by the BL2. The BL2 function is used to pass the base address and the size of the Event Log buffer to **non-secure (BL33)** and **trusted OS (BL32)** via **nt_fw** and **tos_fw** config respectively.

3.6 Measured Boot Proof of Concept with TF-A and fTPM

The TF-A documentation provides a PoC in the QEMU platform that uses a minimal Linux environment based on the OPTEE toolkit with extensions to enable Measured Boot leveraging ARM TrustZone with a fTPM service as a trusted module. This example uses TF-A as FSBL,

TCG Event Log as a Measured Boot backend, OPTEE and EDK2 or U-Boot as SSBL, and a minimal Linux environment in buildroot. To build the proof of concept at first the repository needs to be initiated, in this thesis repository with tag: 3.20.0 will be used as it was more stable than the newer versions.
The implementation needs the **ARM Architecture Fixed Virtual Platform (FVP)** to be downloaded in the build path of the project.

Fixed Virtual Platforms (FVPs) are complete simulations of an Arm system, including processor, memory and peripherals. These are set out in a "programmer's view", which gives you a comprehensive model on which to build and test your software.

After compiling and running the project two terminal windows appear. **FVP terminal_0** and **FVP terminal_1** for the userspace terminal and the secure world logs respectively as can be seen in Figure 9.

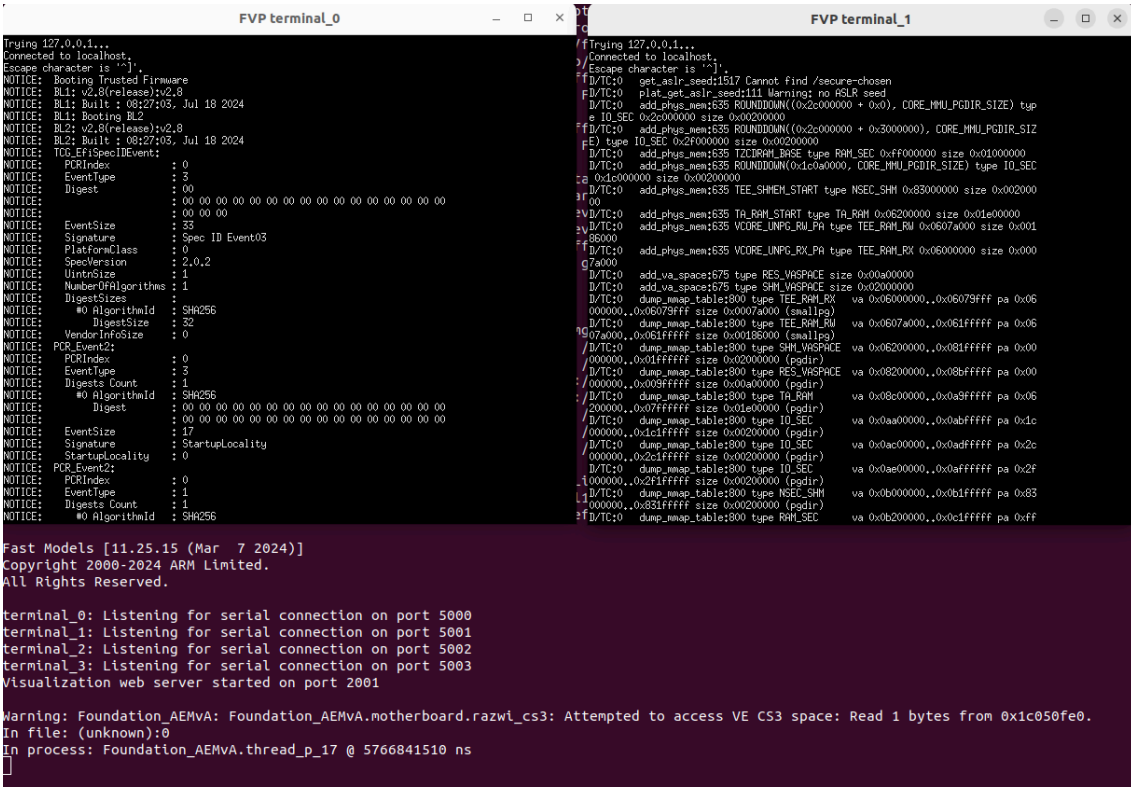


Figure 13 After building and running the proof of concept, Linux and OPTEE logs in terminal_0 and terminal_1 respectively

After the system boots up the “ftpm” command can be issued which is an alias that **loads the fTPM kernel module (tpm_ftpm_tee)** and calling **tpm2_pcrread()** command, to read the recorded PCRs. On Figure 14, can be notices on the left terminal the printed PCR values in index 0 and 1 indicating the Measured Boot measurements and the critical data respectively shown in the Linux userspace. After loading the fTPM kernel module, the fTPM TA is loaded into the

Secure World (OPTEE) and requests a copy of the recorded Event log that was generated during the Measured Boot process.

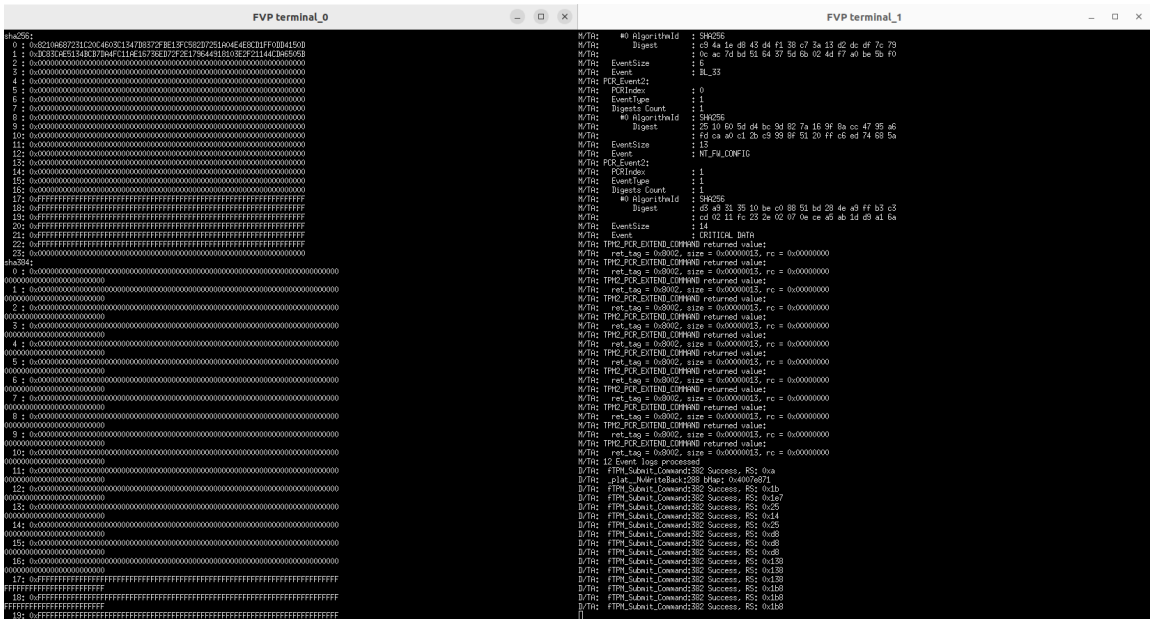


Figure 14 Userspace Linux terminal (left), OPTEE logs terminal (right)

The Event Log can be observed firstly in the logs of Linux userspace terminal when TF-A is loaded in Figure 13. The first event consumed in the event log and can be seen on Secure World Logs in Table 4. is the informational event **TCG_EfiSpecIdEvent** which determines the version of the log, which implies the format of the events, the number, and the size of the recorded digests. Moving to the next one, the **Startup Locality Event (StartupLocality)** or **NO_ACTION event** can be observed that records the locality from which the **tpm2_startup** was sent and is usually not included in the final measure.

```
M/TA: TCG_EfiSpecIDEvent:
M/TA:   PCRIndex           : 0
M/TA:   EventType          : 3
M/TA:   Digest             : 00
M/TA:                      : 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
M/TA:                      : 00 00 00
M/TA:   EventSize          : 33
M/TA:   Signature          : Spec ID Event03
M/TA:   PlatformClass      : 0
M/TA:   SpecVersion        : 2.0.2
M/TA:   UintnSize          : 1
M/TA:   NumberOfAlgorithms : 1
M/TA:   DigestSizes        :
M/TA:     #0 AlgorithmId   : SHA256
M/TA:        DigestSize    : 32
M/TA:   VendorInfoSize     : 0
M/TA: PCR_Event2:
M/TA:   PCRIndex           : 0
```

```
M/TA:   EventType          : 3
M/TA:   Digests Count      : 1
M/TA:     #0 AlgorithmId   : SHA256
M/TA:        Digest        : 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
M/TA:                      : 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
M/TA:   EventSize          : 17
M/TA:   Signature          : StartupLocality
M/TA:   StartupLocality    : 0
```

Table 4 Event Log sample

Next on the event log list seen on Table 5. are the bootloader stages that specify firmware or hardware configuration files as can be seen below:

- **FW_CONFIG**: The firmware configuration file. Holds properties shared across all BLx images. An example is the “dtb-registry” node, which contains information about the other device tree configurations (load-address, size, image_id).
- **TB_FW_CONFIG**: Trusted Boot Firmware configuration file. Shared between BL1 and BL2.
- **BL_2**: BL1 loads and passes control to BL2 at Secure-EL1 (for AArch64) or at Secure SVC mode (for AArch32). BL2 performs the minimal architectural initialization required for subsequent stages of TF-A and normal world software

```
M/TA: PCR_Event2:
M/TA:   PCRIndex           : 0
M/TA:   EventType          : 1
M/TA:   Digests Count      : 1
M/TA:     #0 AlgorithmId   : SHA256
M/TA:        Digest        : d9 09 bd 06 a6 e0 5d 80 2c d0 cd 16 68 ce 0e 8f
M/TA:                      : a6 37 8a fa e4 8a f3 54 6b 30 16 43 f9 66 81 69
M/TA:   EventSize          : 10
M/TA:   Event              : FW_CONFIG
M/TA: PCR_Event2:
M/TA:   PCRIndex           : 0
M/TA:   EventType          : 1
M/TA:   Digests Count      : 1
M/TA:     #0 AlgorithmId   : SHA256
M/TA:        Digest        : dc 45 e2 c1 a9 f5 b1 81 9a 10 8e 8e 0f 0a 5e b2
M/TA:                      : e1 c1 94 00 73 ae 14 83 eb dd e1 f5 fe ac 72 5c
M/TA:   EventSize          : 13
M/TA:   Event              : TB_FW_CONFIG
M/TA: PCR_Event2:
M/TA:   PCRIndex           : 0
M/TA:   EventType          : 1
M/TA:   Digests Count      : 1
M/TA:     #0 AlgorithmId   : SHA256
M/TA:        Digest        : 9c 79 6a 2c 4c c3 08 4b 86 41 89 21 6e c8 24 68
M/TA:                      : 5a f7 b8 0a c2 ae 2d 03 a9 22 0f cb f6 5d f6 24
M/TA:   EventSize          : 5
M/TA:   Event              : BL_2
```

Table 5 Event Log sample

- **SECURE_RT_EL3**: Runtime component. For example, BL31 in Trusted Firmware-A.
- **HW_CONFIG**: The hardware configuration file. All bootloader stages and the Normal World Rich OS can share it.
- **SOC_FW_CONFIG**: It is self-signed with the BL31 key. It contains hashes of the BL31 image and its configuration file. Is loaded by BL2, then its address is passed in arg1 to BL31

Table 6 Event Log sample

```
M/TA: PCR_Event2:
M/TA:   PCRIndex           : 0
M/TA:   EventType          : 1
M/TA:   Digests Count      : 1
M/TA:     #0 AlgorithmId   : SHA256
M/TA:        Digest        : c8 10 6a 8b 24 4b 45 fb 31 2f 11 95 10 15 9c 46
M/TA:                      : 6e 93 cf 6c 6a f8 67 04 8f 73 59 90 b7 b0 e5 a7
M/TA:   EventSize          : 14
M/TA:   Event              : SECURE_RT_EL3
M/TA: PCR_Event2:
M/TA:   PCRIndex           : 0
M/TA:   EventType          : 1
M/TA:   Digests Count      : 1
M/TA:     #0 AlgorithmId   : SHA256
M/TA:        Digest        : 54 10 fe 53 a5 fe 12 75 dc cc 3a 15 8f d8 34 63
M/TA:                      : f3 3e 1e 71 7f e3 4c 39 a1 62 4a 90 5c 0a 84 f3
M/TA:   EventSize          : 10
M/TA:   Event              : HW_CONFIG
M/TA: PCR_Event2:
M/TA:   PCRIndex           : 0
M/TA:   EventType          : 1
M/TA:   Digests Count      : 1
M/TA:     #0 AlgorithmId   : SHA256
M/TA:        Digest        : 4e e4 8e 5a e6 50 ed e0 b5 a3 54 8a 1f d6 0e 8a
M/TA:                      : ea 0e 71 75 0e a4 3f 82 76 ce af cd 7c b0 91 e0
M/TA:   EventSize          : 14
M/TA:   Event              : SOC_FW_CONFIG
```

- **SECURE_RT_EL1_OPTEE**, **SECURE_RT_EL1_OPTEE_EXTRA1**: OPTEE-related firmware images.
- **NT_FW_CONFIG**: Non-Trusted Firmware configuration file. Used by Non-trusted firmware (BL33)
- **BL_33**: BL2 loads the BL33 image (e.g. UEFI or other test or boot software) from platform storage into non-secure memory as defined by the platform.

```
M/TA:     #0 AlgorithmId   : SHA256
```

```
M/TA:        Digest        : 48 e4 ac b6 dc 87 3a 44 82 09 1f 38 eb ad e6 8e
M/TA:                      : 2d 16 20 2d d9 96 05 60 ab 5b 33 6b 97 3c 54 22
M/TA:   EventSize          : 20
M/TA:   Event              : SECURE_RT_EL1_OPTEE
M/TA: PCR_Event2:
M/TA:   PCRIndex           : 0
M/TA:   EventType          : 1
M/TA:   Digests Count      : 1
M/TA:     #0 AlgorithmId   : SHA256
M/TA:        Digest        : ba 49 f0 6a 57 7c 04 19 37 78 cf 1d 34 41 0f 37
M/TA:                      : 29 14 51 e2 27 a0 f0 82 2a 81 30 16 34 57 57 2e
M/TA:   EventSize          : 27
M/TA:   Event              : SECURE_RT_EL1_OPTEE_EXTRA1
M/TA: PCR_Event2:
M/TA:   PCRIndex           : 0
M/TA:   EventType          : 1
M/TA:   Digests Count      : 1
M/TA:     #0 AlgorithmId   : SHA256
M/TA:        Digest        : 9b 68 e8 0b 32 57 26 46 22 9a cd 13 0c 72 8a 2d
M/TA:                      : a2 8a 03 0a df d0 31 6c 93 1f e9 36 bd f6 aa ca
M/TA:   EventSize          : 6
M/TA:   Event              : BL_33
M/TA: PCR_Event2:
M/TA:   PCRIndex           : 0
M/TA:   EventType          : 1
M/TA:   Digests Count      : 1
M/TA:     #0 AlgorithmId   : SHA256
M/TA:        Digest        : 25 10 60 5d d4 bc 9d 82 7a 16 9f 8a cc 47 95 a6
M/TA:                      : fd ca a0 c1 2b c9 99 8f 51 20 ff c6 ed 74 68 5a
M/TA:   EventSize          : 13
M/TA:   Event              : NT_FW_CONFIG
M/TA: PCR_Event2:
M/TA:   PCRIndex           : 1
M/TA:   EventType          : 1
M/TA:   Digests Count      : 1
M/TA:     #0 AlgorithmId   : SHA256
M/TA:        Digest        : d3 a9 31 35 10 be c0 88 51 bd 28 4e a9 ff b3 c3
M/TA:                      : cd 02 11 fc 23 2e 02 07 0e ce a5 ab 1d d9 a1 6a
M/TA:   EventSize          : 14
M/TA:   Event              : CRITICAL DATA
```

Table 7 Event Log sample

The logs correspond to the measurements stored by the TF-A during the Measured Boot process and are extended in the appropriate PCRs as it can be seen on Table 8 the logs of the extend commands in OPTEE initiates from TPM tools (**tpm2_pcrextend()**).

```
M/TA: TPM2_PCR_EXTEND_COMMAND returned value:
M/TA:   ret_tag = 0x8002, size = 0x00000013, rc = 0x00000000
M/TA: TPM2_PCR_EXTEND_COMMAND returned value:
M/TA:   ret_tag = 0x8002, size = 0x00000013, rc = 0x00000000
```

```
M/TA: TPM2_PCR_EXTEND_COMMAND returned value:
M/TA:   ret_tag = 0x8002, size = 0x00000013, rc = 0x00000000
M/TA: TPM2_PCR_EXTEND_COMMAND returned value:
M/TA:   ret_tag = 0x8002, size = 0x00000013, rc = 0x00000000
M/TA: TPM2_PCR_EXTEND_COMMAND returned value:
M/TA:   ret_tag = 0x8002, size = 0x00000013, rc = 0x00000000
M/TA: TPM2_PCR_EXTEND_COMMAND returned value:
M/TA:   ret_tag = 0x8002, size = 0x00000013, rc = 0x00000000
M/TA: TPM2_PCR_EXTEND_COMMAND returned value:
M/TA:   ret_tag = 0x8002, size = 0x00000013, rc = 0x00000000
M/TA: TPM2_PCR_EXTEND_COMMAND returned value:
M/TA:   ret_tag = 0x8002, size = 0x00000013, rc = 0x00000000
M/TA: TPM2_PCR_EXTEND_COMMAND returned value:
M/TA:   ret_tag = 0x8002, size = 0x00000013, rc = 0x00000000
M/TA: TPM2_PCR_EXTEND_COMMAND returned value:
M/TA:   ret_tag = 0x8002, size = 0x00000013, rc = 0x00000000
M/TA: TPM2_PCR_EXTEND_COMMAND returned value:
M/TA:   ret_tag = 0x8002, size = 0x00000013, rc = 0x00000000
M/TA: 12 Event logs processed
```

Table 8 OPTEE terminal showing the tpm extend command logs

In the Linux terminal, after the operations mentioned above conclude and the logs of the Event Log have been extended to the PCRs, the tpm2_pcrread function read and prints the recorded SHA256 hashes as in Table 9.

```
sha256:
    0 : 0xD3158BD77BC00B4DEB41F20FE38EA33E098733FB8605FC01F41FEA57E3B0DC6A
    1 : 0xDC83CAE5134BCB7DA4FC11AE16736ED72F2E179644918103E2F21144CDA6505B
    2 : 0x0000000000000000000000000000000000000000000000000000000000000000
    3 : 0x0000000000000000000000000000000000000000000000000000000000000000
    4 : 0x0000000000000000000000000000000000000000000000000000000000000000
    5 : 0x0000000000000000000000000000000000000000000000000000000000000000
    6 : 0x0000000000000000000000000000000000000000000000000000000000000000
    7 : 0x0000000000000000000000000000000000000000000000000000000000000000
    8 : 0x0000000000000000000000000000000000000000000000000000000000000000
    9 : 0x0000000000000000000000000000000000000000000000000000000000000000
    10: 0x0000000000000000000000000000000000000000000000000000000000000000
    11: 0x0000000000000000000000000000000000000000000000000000000000000000
    12: 0x0000000000000000000000000000000000000000000000000000000000000000
    13: 0x0000000000000000000000000000000000000000000000000000000000000000
    14: 0x0000000000000000000000000000000000000000000000000000000000000000
    15: 0x0000000000000000000000000000000000000000000000000000000000000000
    16: 0x0000000000000000000000000000000000000000000000000000000000000000
    17: 0xFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF
    18: 0xFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF
    19: 0xFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF
    20: 0xFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF
    21: 0xFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF
    22: 0xFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF
    23: 0x0000000000000000000000000000000000000000000000000000000000000000
```

Table 9 tpm2_pcrread function results in Linux userspace

3.7 IMA configuration and details

To build the IMA and EVM modules the kernel needs to be modified with the appropriate configuration options. Specifically, these modules include the security options mentioned in Table 10. This modification can be done by changing the configuration file on Linux kernel path (.config) or cross compile the kernel with the menuconfig make option to enable the IMA/EVM options from the GUI.

```
# Security options
CONFIG_KEYS=y
CONFIG_TRUSTED_KEYS=y
CONFIG_TRUSTED_KEYS_TPM=y
CONFIG_TRUSTED_KEYS_TEE=y
CONFIG_ENCRYPTED_KEYS=y

CONFIG_SECURITY=y
CONFIG_SECURITYFS=y

CONFIG_INTEGRITY=y
CONFIG_INTEGRITY_SIGNATURE=y
CONFIG_INTEGRITY_ASYMMETRIC_KEYS=y
CONFIG_INTEGRITY_TRUSTED_KEYRING=y
CONFIG_INTEGRITY_AUDIT=y
```

```
CONFIG_IMA=y
CONFIG_IMA_KEXEC=y
CONFIG_IMA_MEASURE_PCR_IDX=10
CONFIG_IMA_NG_TEMPLATE=y
CONFIG_IMA_DEFAULT_TEMPLATE="ima-ng"
CONFIG_IMA_DEFAULT_HASH_SHA256=y
CONFIG_IMA_DEFAULT_HASH="sha256"
CONFIG_IMA_WRITE_POLICY=y
CONFIG_IMA_READ_POLICY=y
CONFIG_IMA_APPRAISE=y
CONFIG_IMA_APPRAISE_BOOTPARAM=y
CONFIG_IMA_TRUSTED_KEYRING=y
CONFIG_IMA_MEASURE_ASYMMETRIC_KEYS=y
CONFIG_IMA_QUEUE_EARLY_BOOT_KEYS=y

CONFIG_EVM=y
CONFIG_EVM_ATTR_FSUUID=y
CONFIG_EVM_ADD_XATTRS=y
```

Table 10 IMA/EVM Kernel configuration options

Measurements are taken according to IMA policies that can be applied to use the built-in ones. Build-in policies can be enforced by parsing them to the kernel command-line parameters. The supported measure values for **ima_policy** are:

- **ima_policy=tcb** - This IMA policy meets the needs of the Trusted Computing Base (TCB), measures all programs directly executed of mmap'd for execution. They measure files opened by root ((euid, uid) == 0) with the read bit set. It measured all kernel modules and firmware loaded.
- **ima_policy=critical data** - Applies the **func=CRITICAL_DATA** rule to the IMA policy.

and the supported appraise values for **ima_policy** are:

- **ima_policy=secure_boot** - Appraises loaded kernel module, firmware, the kexec kernel image and the IMA policy itself, based on a file signature as an extended attribute.
- **ima_policy=appraise_tcb** - Appraises all files owned by root. The policy excludes the "pseudo" filesystems from appraisal.
- **ima_policy=fail_securely** - Affects the appraisal of untrusted mounted filesystem. An example is a FUSE filesystem.

```
dont_measure fsmagic=0x9fa0          # PROC_SUPER_MAGIC
dont_measure fsmagic=0x62656572      # SYSFS_MAGIC
dont_measure fsmagic=0x64626720      # DEBUGFS_MAGIC
dont_measure fsmagic=0x1021994       # TMPFS_MAGIC
dont_measure fsmagic=0x1cd1          # DEVPTS_SUPER_MAGIC
dont_measure fsmagic=0x42494e4d      # BINFMTFS_MAGIC
dont_measure fsmagic=0x73636673      # SECURITYFS_MAGIC
dont_measure fsmagic=0xf97cff8c      # SELINUX_MAGIC
dont_measure fsmagic=0x43415d53      # SMACK_MAGIC
dont_measure fsmagic=0x27e0eb        # CGROUP_SUPER_MAGIC
dont_measure fsmagic=0x63677270      # CGROUP2_SUPER_MAGIC
dont_measure fsmagic=0x6e736673      # NSFS_MAGIC
dont_measure fsmagic=0xde5e81e4      # EFIVARFS_MAGIC
measure func=MMAP_CHECK mask=MAY_EXEC
measure func=BPRM_CHECK mask=MAY_EXEC           # binary executed
measure func=FILE_CHECK mask=^MAY_READ euid=0
measure func=FILE_CHECK mask=^MAY_READ uid=0    # root opened r/o, r/w
measure func=MODULE_CHECK
measure func=FIRMWARE_CHECK
measure func=POLICY_CHECK
```

Figure 15 Example of ima_policy=tcb - IMA policy file

For simplicity the first policy, **ima_policy=tcb**, was chosen and Figure 15 shows an example of how this policy is written. A kernel command-line parameter was added to the project's grub configuration file (fvp/grub/grub.cfg) as can be seen on Figure 16 to show an examples of IMA measurements taken for the filesystem data.

```
set prefix='/EFI/BOOT'

set default="0"
set timeout=10

menuentry 'GNU/Linux (OP-TEE)' {
    linux /Image console=tty0 console=ttyAMA0,115200 earlycon=pl011,0x1c090000 root=/dev/disk/by-partlabel/system rootwait rw ignore_loglevel efi=noruntime ima_tcb
    initrd /initrd.img
    devicetree /fvp.dtb
}
```

Figure 16 Grub configuration to pass on kernel command-line parameters the IMA tcb policy (ima_policy=tcb)

IMA kernel module maintains a runtime measurement list and is reviewed when the securityfs, is mounted. Securityfs is a virtual filesystem in memory for security kernel modules. Policies and other data are in this filesystem and is mounted automatically on **/sys/kernel/security** or manually (**mount -t securityfs security /sys/kernel/security**). If IMA is enabled the securityfs is populated with ima and evm as for Figure 17 paths that contained the measurement files, policies and other files.

```
# ls -lah /sys/kernel/security/
total 0
drwxr-xr-x    3 root     root           0 Jan  1 00:00 .
drwxr-xr-x   13 root     root           0 Jan  1 00:21 ..
lr--r--r--    1 root     root           0 Jan  1 00:00 evm -> integrity/evm/evm
lr--r--r--    1 root     root           0 Jan  1 00:00 ima -> integrity/ima
drwxr-xr-x    4 root     root           0 Jan  1 00:00 integrity
```

Figure 17 Securityfs path after IMA/EVM enabled

The security extensions can be further enhanced enabling the ima_policy=appraise_tcb IMA policy, appraising all the files owned by root, forbidding any operation to be performed if the current measurements do not match the pervious reference measurements. Furthermore, EVM tools can be enabled for protecting files' extended attributes (xattr), as EVM cryptographically hashes security files like IMA measurements, SELinux attributes or other files.
TPM can also be used with the Linux kernel integrity subsystem to provide the hardware guarantees of isolation and the PCRs that the measurements can be extended.
The IMA measurement list can be seen on Figure 18 with the hashes of the files measured according to the tcb IMA policy. The IMA Event log is located on /sys/kernel/security path in two formats:

- **IMA Event Log Binary Format**:/sys/kernel/security/ima/binary_runtime_measurements
- **IMA Event Log Ascii Format**:/sys/kernel/security/ima/ascii_runtime_measurements

The first value on measurement list is the boot aggregate, that is a SHA hash over TPM PCRs 0-7, if a hardware TPM module exists or 0 if TPM does not exist.

```
# cat /sys/kernel/security/integrity/ima/ascii_runtime_measurements
10 0adefe762c149c7cec19da82f0da1297fcfbfff6 ima-ng sha256:0000000000000000000000000000000000000000000000000000000000000000 boot_aggregate
10 8c1eea11049e00ee9f72038f7c2220f858baae18 ima-ng sha256:b2b9f4a5b67f90fece2c83dad0986ae44189fc1b7128c5ed9a38c0ca38e88c90 /init
10 6388896956d4b25a93f0c31ed2fa86e909dc08eef ima-ng sha256:44e59573c3d050b2c530eb495eaa0d7d296d9df3fbdbe5f46ee56f334c45079 /bin/busybox
10 4a8e6df9d5b45b29e2120026407811d10018c6832 ima-ng sha256:912002c930f3096d98fc32e0897466049d6e5182cbb606b456b2ebcc99551366 /lib/ld-linux-aarch64.so.1
10 3fb4c89cf037a5395bf9398d1c1869e6094861d0 ima-ng sha256:2ec220d17b3a34e99d0c186264fdbf37e8da9bce5e7523464b30247f079a8f93 /lib/libresolv-2.31.so
10 b8835e46664ddd0921a616a86981f9aef9b80bdb0 ima-ng sha256:cfc6aa042a080a87e4f0adbfb74b55c9ba9607561905b20f58da96fcb2d3bb86 /lib/libc-2.31.so
10 6dc8338d1c7837c7d0ea76f10c3476b95d1882fa ima-ng sha256:3cc70adf57dfc0704539a2eab6bbd121274725cb3809f2c258f9de73afe67ac4 /etc/inittab
10 22c48b4853594a08a73ad4ae6dbe6f2c2bebc6c5 ima-ng sha256:e3b0c44298fc1c149afbf4c8996fb92427ae41e4649b934ca495991b7852b855 /run/utmp
10 debcf2ad1b9adf686b8d1fe2b9fb60cbf799d243 ima-ng sha256:fa243e77b473fae67ff239904ccbdf7d53e8ffa84de130418eee43fb4e441142 /etc/fstab
10 4141c82cb516ac3c846e0b08abcd6abeee7efa1a ima-ng sha256:b75d7f28772f71715a941c77e07e3922815391dd9cc5718ad21f2231c2da09bb /etc/hostname
10 dfcedd3c7dc3ed42e0921380450448 3ab264e2e3 ima-ng sha256:dc1615df9f2012b20b81ffad8e07e16293033ba7fd837854ca3646d6cfea0c0f /etc/init.d/rcS
10 a20b1960b6a92acfd57536796766cffc39762445 ima-ng sha256:ad6ad88add71db74836b65acf11e3942f3acb62e9be74545a532cd306a9427dd0 /etc/init.d/S01syslogd
10 fcff1ed8ede5d38217e153a96df02e6e609d74 ima-ng sha256:e2f967bf70f912d605e5d60f05e1c2e6b9e8f25a88df1c39109df00eb9564205 /etc/init.d/S02klogd
10 bd448b6c0ce2c89f11ecd61445717ecb6bed6a8879 ima-ng sha256:a89361f308e750ba7411e33459e1bc3011b84ecb51a89273ab4737b9be434311e /etc/init.d/S02sysctl
10 1f84c14d022d5b9bb10e83ba3fc82ff0761f73fa ima-ng sha256:d18b9b4b6e92883f160b5474949 80ffd0f9322778b1e73df42ae7028 69bfbd43 /etc/nsswitch.conf
10 a177bcd21b2e995e679fcaa2e7110c8d873a6f1d ima-ng sha256:543901c538a1050ba2592a09c8a8e642f68ebcfcc3cd6a6bd8ae129ab1234ef7 /lib/libnss_files-2.31.so
10 b055b2e5330c882eea528aaa204485a0887f7d30 ima-ng sha256:ec28cd78c0184cb386774e76ba701e6d4346583d2d7610ba9f029274e58f8a3f /etc/passwd
10 de3e8f01b2a722f0e167bca0fcd7a5f9387650d7 ima-ng sha256:e9c45a92f7fe94f42fbf730ad9fe2990cd7fdaedbb419260c7f1badd04a060d6 /etc/init.d/S20urandom
10 b48220b62209be84e039c139bd13d14d39c650e3e ima-ng sha256:df3f8ce23e93977566f44269a050e1fd5d589cecb79b70d9a6d7f6a2dfb36ebc /etc/init.d/S30optee
10 fe075fa29f45466d85c6858615fc1ddd705d2afc ima-ng sha256:e13f16db9ec77189475b9e3e82c97a390f5e819b9a06a793df01cd2d0419687a /etc/group
10 ed732cf97928693b891a5f74630ca558fc15773e ima-ng sha256:cf34ba9e963727d4697e6919abb8d6d68a9857bbfbbcc3792527babb7116b630 /usr/sbin/tee-supplicant
10 274d4ec474090e7ed493e9d5dcbfcc0c063223a8 ima-ng sha256:2be73493b6852764cf316fea4f13045e01cb91f1478f17e32423c74201ba84b4 /usr/lib/libteec.so.1.0.0
10 4c88377348aaa5008f364723b4fb57cfe40df83d ima-ng sha256:c710e23f6825dea065bc8f254ec3272cbb91f7a8c1b459b9e283cb5d25fa7919 /lib/libdl-2.31.so
10 7f31c72dc1a688314c20c2770152c1bec7097d8b ima-ng sha256:7bbc94a43324f9da05f63936be2422f1ee55ce9432eb37658ae071b59a35a13f3 /lib/libpthread-2.31.so
10 22e2881a387a5c880eb061c4f92099f0107861c2b ima-ng sha256:c9fbad69724d5983e235e75ba30c057ec0e947df0f62005fb2b65b2217be7d69 /usr/lib/tee-supplicant/plugins/f07bfc66-958c-4a15-99c0-260e4e7375dd.plugin.so
10 0757b44af3f067fbad70783ad7876de4a10c3365 ima-ng sha256:74352dd217f4d707d72cb1bc9d4e3ffdb2da501394a610ad6ec9cbc970db8824 /usr/lib/tee-supplicant/plugins/96bcf744-4f72-4866-bf1d-8634fd9c65e5.plugin.so
10 b031a28f6ccef82cdbe68800eaea75f8f0864e56 ima-ng sha256:27eac7a647b151a9688d972b6668f082e1a01b699e0f6435b5fe10693135abf5 /etc/init.d/S40network
10 879fbea24c0045996813231b1da13f3052f5fe6a4 ima-ng sha256:bf24145cc4a9986989955f512359a6ded0dffb789a93dc94ad5b41e7a378b1b2 /etc/network/interfaces
10 ed9d8053a190bf7b4c9d940e8b1a715c06b50bcb ima-ng sha256:b70434ea0e10ab723658230c0b52b687e939009886329ee57e4b23008b64c465 /etc/network/if-pre-up.d/wait_iface
10 9786dd4d192f3ef2ed5f23cd3c24c4063aa33a5f ima-ng sha256:00b298bc83a63b9fbb752f6c869e360267fac20611c083648c3692e20e228f25 /etc/issue
10 f04cbc6ba56d7617741a6edc42ffd06771d7319 ima-ng sha256:3fdca54cd8d819031b4a9d49d841ba848f430108a383323e8f849b89f6415056 /etc/shadow
10 53418efbd26e92708bf2987f19e636fac76d3214 ima-ng sha256:82aaa6819e07689a9ff2f1bf7fdb306e91bc9fac89dd650d214996b937cf49c6 /etc/profile
10 84123d9b1d9663efaa473328034c6239fa32cbe0 ima-ng sha256:e5340e16d58cf5022a8578aece46900c31461255274360ddadb0be03a6a4000 /etc/profile.d/ftpm_alias.sh
10 2793b07462cb1b3155f3032aad25c6ecb2c59364 ima-ng sha256:3b7dae25ad0e172374b7d845a5d3d76e2f62a06b6556fd9c523031419c78d16a /etc/profile.d/umask.sh
10 df28b1ed3d9839e5539008004aed591183d1f573 ima-ng sha256:bbd7c0b7b56855a1d40c9556327e491bcf4a23ae6b7537ad7909a34d2fea69c9 /lib/modules/extra/tpm_ftpm_tee.ko
10 ef3e8de956ddaa83277b7b373c64b812ddc2885c ima-ng sha256:45e3b9a6ae28de44e5673f0c7d0ed0ecef322a182fb6ad72ea7dd4fe302e294 /usr/bin/tpm2_pcrread
10 4501bc95d5dd55deb13adffc34302a7bbf63cb4b ima-ng sha256:ac695dff2dda03b0a93b6fbf1da5b668b953aae60df2bfa7e7076c5be27a6928 /usr/lib/libtss2-esys.so.0.0.0
10 d3b60561ab85515c23bd9a8c441034d9fdb2f5d7 ima-ng sha256:76048e43466f7b48281315ae798bd7b3ec0a00739853b7ac695c806c38e367b4 /usr/lib/libtss2-sys.so.1.0.0
10 f9b225b7a67e7ec3cf57cf486270e80d54709735 ima-ng sha256:428d2842f96f899b55197434f68ecda042e5d293314b917cc7492e5ba6aad4b3 /usr/lib/libtss2-mu.so.0.0.0
10 6fcffce09ed4aeaaa5cb911defabb73d3890b87a ima-ng sha256:357fa73c79015174e886a90911c413ad1837c347bf7967ed7178ea2a2642bbe3 /usr/lib/libcrypto.so.1.1
10 0c5ed7bac8ad80159de4062b12d058ac8c2a6737 ima-ng sha256:104bea65aed18ce308570eefb43b34c733306f8a6e7eac60e5f705d26d22fd3c /usr/lib/libtss2-tctildr.so.0.0.0
10 281f6e50a16fea462350274339c469aee12b21a7 ima-ng sha256:82ad7d39d18450acf2d904570c872c9ad5487b0e65528386ee41180550861f1 /usr/lib/libtss2-rc.so.0.0.0
10 855a51333a918d01e46a6b79e8650656663841ed9 ima-ng sha256:9f84ce96bcf9b3c0c689517a817f1a0b4e51e3aa1dc4defde761b41cd54d04ec /lib/libuuid.so.1.3.0
10 200632df26cd5156eafbeac5c1549b837afd9b47 ima-ng sha256:4e5d7e05d3c9b5d202cd8914731eb566041ffcd7e0faf51aba25962650196bbb1 /lib/libatomic.so.1.2.0
10 2770c23a62fd6093bf74cedd241e142c13d5d7c0 ima-ng sha256:83df00b715301df0e047366a5cc518005b1a29d54f5014ac3f1bc4cf928e9b85 /usr/lib/libtss2-tcti-device.so.0.0.0
# 
```

Figure 18 IMA measurement list.

In this proof of concept, the fTPM is not loaded when the IMA starts looking for a TPM, as would be with a hardware TPM module, so is not populated with any hashes but zeros. In Figure

19 can be observed in the boot logs the failure of IMA to find a running TPM to aggregate measurements early on boot. Thus, it is interesting in feature work to find a way of interaction between IMA and EVM modules with fTPM. Taking advantage of the isolation mechanisms of the fTPM, IMA measurements could be extended PCRs along with the Measured Boot values taken during the start of the device and provide to an attestation scheme an overall status of the system.

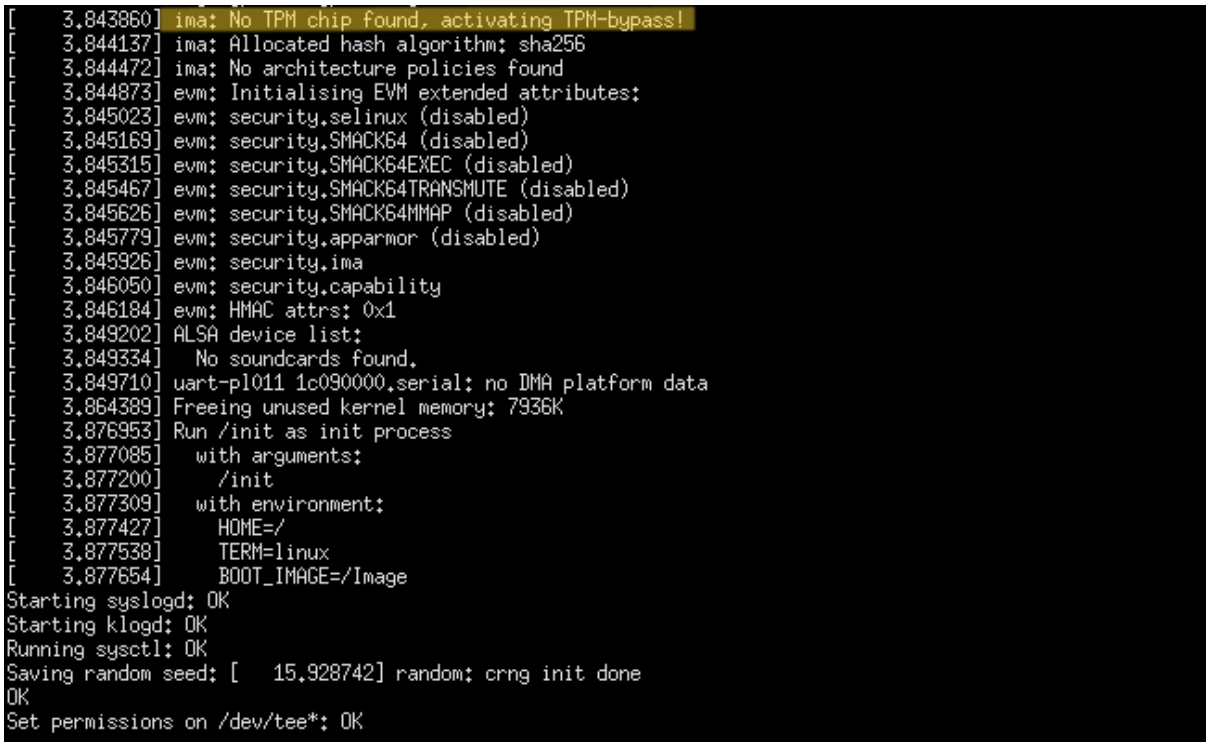


Figure 19 IMA failure to find a running TPM device early on boot

Typically, the first reader of the IMA event log should be an attestation application, which is responsible of sending the values along with a quote to a remote verifier. This quote can be a TPM quote which includes a signature over the event log to a remote verifier, this is why getting fTPM work with IMA is essential for extending critical measurements to PCRs and ensuring the integrity of the device attesting those values.

3.8 Trusted Substrate Firmware overview

Although the Trusted Firmware's FVP proof of concept is a comprehensive project for showcase of the Measured Boot operation leveraging fTPM, recording critical data and firmware images, to build such a solution for a specific device board would be a more complex implementation depending on the device's limitations and capabilities. Systems are complicated, with various components that need to be configured and built appropriately to interact with each other and meet the right security standards expanding the threat model from each individual component to the whole system as well. Recently Linaro created and maintains a project called Trusted

Substrate which is under the Trusted Reference Stack (TRS) umbrella and is a software stack that includes well tested software and components to help building systems with several security features for different use cases.
Examples of use cases include:

- Deployment of applications via Docker, Podman and k3s
- Remote updates for software components, to be able to address security issues, fix bugs and add features.
- Measured Boot facilitating sealed keys technologies
- Encrypted filesystem
- Remote attestation to be able to trust and be trusted by remote entities
- Etc.

TRS provides baremetal and virtualization variants to support a variety of different applications and is composed of components in all architecture layers from low-level communicating with the hardware to user-space environment. In Figure 20 an overview of the key components of TRS can be found for baremetal architecture.

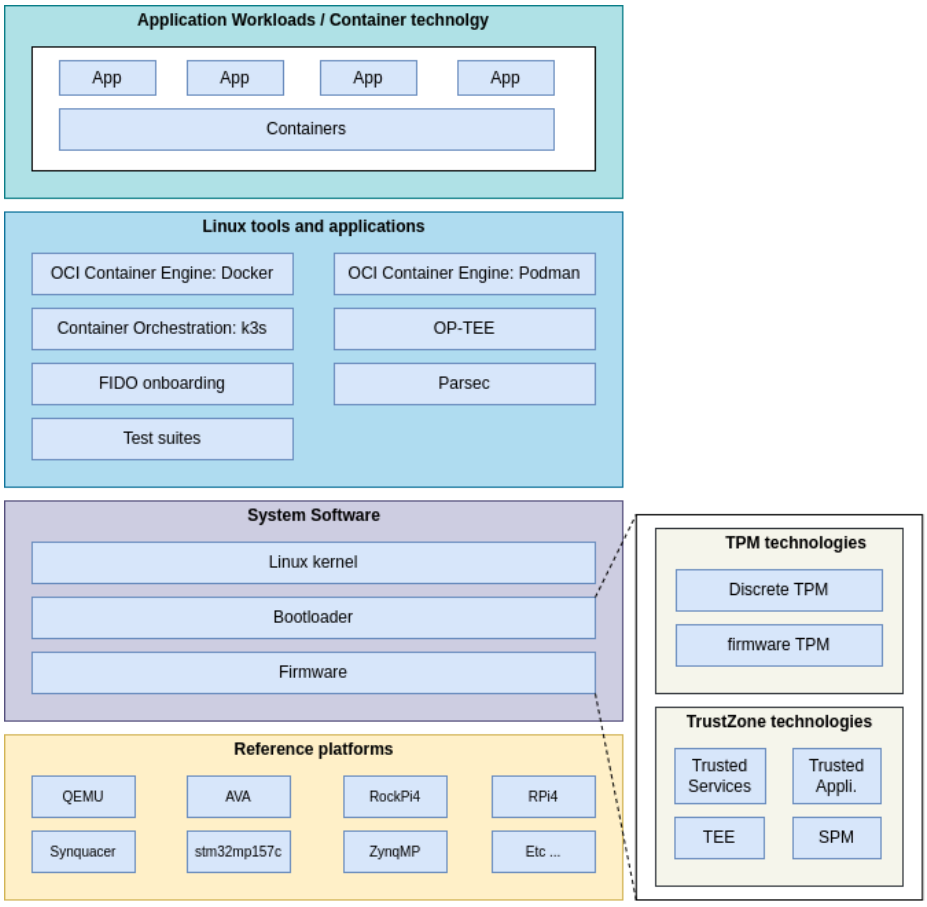


Figure 20 Trusted Reference Stack (TRS) Overview of key components

In the system software section can be observed the system software section which includes the low-level technologies supported such as Linux kernel, Bootloaders and Firmware. For the

support and configuration of the Bootloader and the Firmware components the Trusted Substrate project is responsible. Trusted Substrate is a meta-layer in OpenEmbedded designed with the goal of producing ARM SystemReady compliant firmware based on Embedded Base Boot Requirements (EBBR). This meta-layer ensures device consistency and tamper protection enabling mechanisms to verify the integrity of running software and firmware loaded across different system stages. Essentially, Trusted Substrate is configured to support standards like:

- UEFI secure boot
- Measured Boot. TPM or fTPM
- Dual banked firmware update with rollback and bricking protection.

According to TRS documentation [65], currently there are several platforms that are supported for building their firmware with security extensions configured by TRS and are categorized as can be seen on Figure 21.

Board	FSBL	Secure Boot	Measured Boot	Auth. Capsule Updates	A/B updates
QEMU	TF-A	Yes (Built-in vars)	Yes	No	No
DeveloperBox	SCP + TF-A	Yes (RPMB vars)	Yes [fTPM]	Yes	WIP
stm32mp157c-dk2	TF-A	Yes (Built-in vars)	No	No	WIP
stm32mp157c-ev1	TF-A	Yes (RPMB vars)	No	No	WIP
Rockpi4	U-Boot SPL	Yes (RPMB vars)	Yes [fTPM]	Yes	No
Raspberry Pi4	Proprietary	Yes (Built-in vars)	Yes (needs SPI TPM)	No	No
Xilinx kv260 starter kit	U-Boot SPL	Yes (Built-in vars)	Yes	Yes	WIP
Xilinx kv260 commercial	U-Boot SPL	Yes (Built-in vars)	Yes	Yes	WIP

Figure 21 Supported Boards for Trusted Substrate firmware.

Trusted Substrate firmware supports the EFI TCG Protocol and the TCG PC Client Specific Platform Firmware Profile Specification providing the OS requirements for Measured Boot Figure 22.

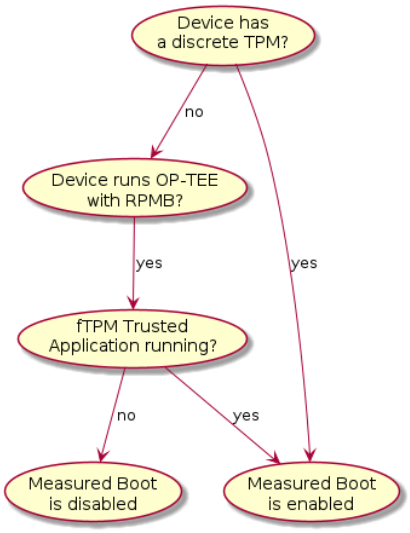


Figure 22 Measured Boot according to TRS project

Another worth mentioned functionality of the TRS firmware is the filesystem encryption that can be used with the help of TPM. TRS also take advantage of the isolation technology of the ARM TrustZone and used OPTEE for various operations such as fTPM, storing EFI variables in case of RPMB partition existence and cryptographic functionalities. Components interacting with OPTEE in different ways and exception levels depending on the scenario as it is summarized in Figure 17 by TRS documentation.

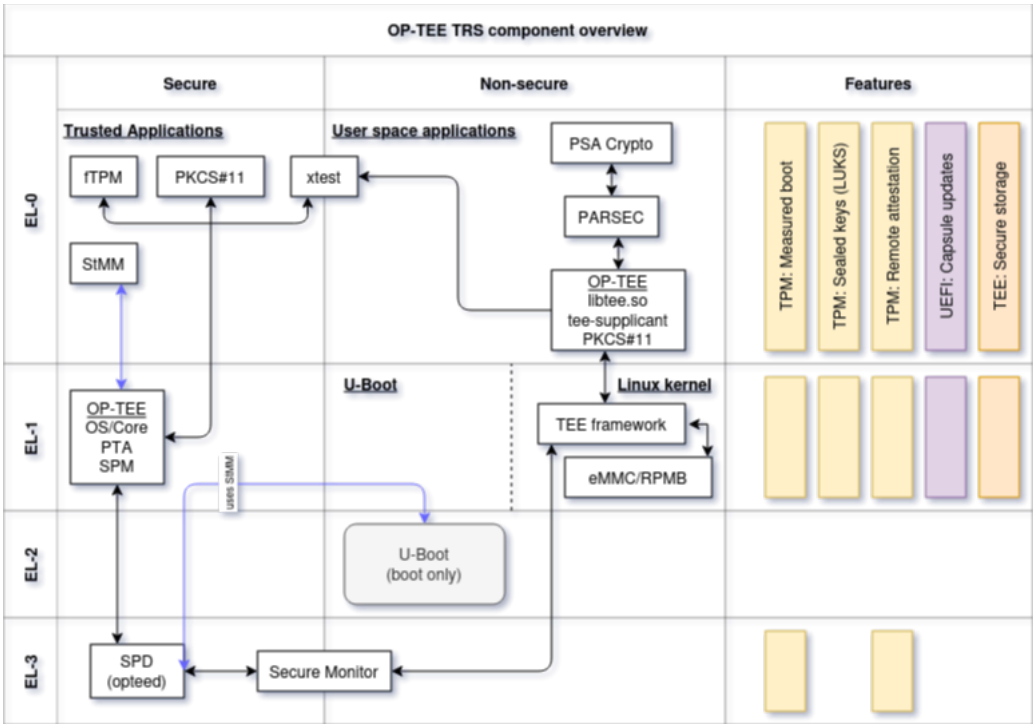


Figure 23 Overview of different component interacting with OPTEE in TRS

4. Literature Review

Over the last two decades, research has increasingly focused on developing attestation mechanisms to ensure the integrity of computational systems. This can be implemented with various methods depending on the security scope and the limitations of the associated platforms, from Wireless sensor networks (WSN) running on microcontrollers to Edge computing systems running on processors with a kernel and operating system (OS). Establishing a chain of trust between boot states and taking measurements of critical data during run time is essential to provide valid values in attestation procedures for assessing a system's overall security. Attestation procedures only operate as intended if the system configuration is measured from the early boot chained to the current running software. Software-based, hardware-based, and hybrid attestation schemes have been developed supporting various devices and solutions with different needs, depending on the system's resources and limitations.

Several Software-based attestation schemes have been proposed over the years including integrity checks on the peripherals, for example in WSNs and the code running on the system [29] - [38]. These schemes depend on local memory check procedures, measurements, or access policies as they can be considered secure only if the verifier communicates directly with the prover according to Francillon et al. (2014) [27]. Furthermore, attestation techniques without a hardware root of trust rely on strong assumptions regarding the adversary's capabilities, concluding that such approaches usually refer to legacy or resource-constrained devices as discussed by Ankergård et al. (2021) [28].

Hardware-based attestation techniques count on secure hardware as the root of trust to ensure tamper-proof measurements of a device. The main focus is on isolation with different approaches using MPUs, PUF, or single-purpose processor solutions [39]-[50]. According to W. A. Johnson et al (2021) [53], although these scheme approaches are designed to provide various benefits over other roots of trust, several of these proposes are unavailable as commodity products and thus are not yet adopted. One additional note is that a hardware root of trust can not be changed in case of an exploited vulnerability in the design [51][52].

Hybrid schemes can combine both software and hardware root of trust and are not as limited in their adversary's assumptions as software-based schemes. Furthermore, hybrid proposes are more compliant with changes and upgrades than hardware-based schemes. Even though TEE itself in practice is a hardware solution, RA schemes that use minimal hardware or security features such as isolation and MPU functionality are classified as hybrid [53]. James. M. et al (2022) discussed different state-of-the-art remote attestation procedures which leverage hardware-assisted TEEs. After comparing schemes such as Intel SGX, ARM Trustzone, AMD SEV, and RISC-V TEE they concluded that TEE-based remote attestation is still an industrial challenge and not yet widely studied [62].

According to TCG as mentioned in **2.8.1** the safety of a device depends on the hardware and software being able to trust each other and operate as intended. The security of a system consists

of cryptographic measurements and attestation against good, expected reference values. Based on this specification several approaches have been developed from commercial companies [55]. Intel Trusted Execution Technology (TXT) offers a measured boot functionality and is designed to robust the security of the platform at the hardware level, from software-based attacks or attacks to firmware and hypervisor. Intel TXT offers measured launch and protected execution through a set of hardware executions to Intel processors and chipsets [54][56].
Microsoft also has its own Measured Boot and Remote Attestation scheme using UEFI for measurements and a TPM module to store the hash of the bootloader, firmware, and other critical data and assess them through a remote attestation server [57].
Additionally, and worth mentioning is that attestation protocols are not limited to a single device but also focused on a swarm of nodes [58]-[61].

Results

The experiments and the analysis conducted in this thesis demonstrated the effectiveness of taking attestation evidence during boot and runtime on ARM devices. Respecting the TCG specification and standards a fTPM used leveraging the isolation features of ARM TrustZone and the functionality of the TF-A to implement a COT between the firmware images loaded during boot and the measurements to critical data during runtime. The results also showed that in some boards such as STM32MP157C-DK2 and STM32MP157C-EV1 this kind of measuring the boot process is not yet supported by TF-A. Although Measured Boot is not yet implemented for specific vendors, TF-A's proof of concept and Linaro's Trusted Substrate proved to be promising projects to work on and have an overall view of how such techniques are implemented. Furthermore, when measurements on critical data consumed with fTPM mechanisms attestations schemes could be implemented with the help of different tools such as Keylime, or Go-Attestation.

Conclusion

Even though various and different methods have been implemented for collecting attestation evidence from devices not every scheme is suitable for appraising the overall status of a running device. This thesis explored the security of ARM-based devices, focusing on the implementation of Trusted Firmware, OPTEE, fTPM, and IMA to enhance boot and runtime integrity. The use of ARM TrustZone in conjunction with fTPM provided a secure environment for executing critical functions and collecting attestation evidence in a trust-worthy way. The implementation of IMA/EVM kernel modules further strengthened the system's integrity by appraising the state of the files in Linux OS. While the experiments emphasized the effectiveness of these security mechanisms, they also revealed certain limitations and challenges in terms of compatibility and support for Measured Boot. Overall, this work contributes to the field of secure computing by offering a practical approach to achieving Measured Boot and filesystem measurements in ARM-based devices that can be provided as evidence to attestation protocols, thereby assessing their overall security status.

Bibliography

[1] ARM - The route to a trillion devices, the outlook for IoT investment to 2035, Philip Sparks, June 2017

[2] Internet of Things (IoT) Market Size, Share & Industry Analysis, By Component (Platform and Solution & Services), By End-use Industry (BFSI, Retail, Government, Healthcare, Manufacturing, Agriculture, Sustainable Energy, Transportation, IT & Telecom, and Others), and Regional Forecast, 2024-2032 - Source: https://www.fortunebusinessinsights.com/industry-reports/internet-of-things-iot-market-100307

[3] The weaponization of IoT devices Rise of the thingbots IBM X-Force® Research https://www.ibm.com/downloads/cas/6MLEALKV

[4] Trusted Firmware-A documentation. Available online: Trusted Firmware-A Documentation

[5] TPM 2.0 Library | Trusted Computing Group (TCG) - https://trustedcomputinggroup.org/resource/tpm-library-specification/

[6] fTPM: A Software-only Implementation of a TPM Chip - https://www.microsoft.com/en-us/research/publication/ftpm-software-implementation-tpm-chip/

[7] Open Portable Trusted Execution Environment (OP-TEE). Available online: OP-TEE

[8] SMC CALLING CONVENTION System Software on ARM Platforms
System Software on ARMÂ® Platforms - SMC CALLING CONVENTION System Software on ARM Platforms

[9] ARM Architecture Reference Manual ARMv7-A and ARMv7-R edition - ARM Architecture Reference Manual ARMv7-A and ARMv7-R edition

[10] TEE Client API Specification v1.0 Errata and Precisions v2.0 | GPD_EPR_028 - https://globalplatform.org/specs-library/tee-client-api-specification-v1-0-errata-and-precisions-v2/

[11] TPM 2.0 A brief Introduction - https://trustedcomputinggroup.org/wp-content/uploads/2019_TCG_TPM2_BriefOverview_DR02web.pdf

[12] Trusted Platform Module Library Part 1: Architecture - https://trustedcomputinggroup.org/wp-content/uploads/TCG_TPM2_r1p59_Part1_Architecture_pub.pdf

[13] TPM recommendations - https://learn.microsoft.com/en-us/windows/security/hardware-security/tpm/tpm-recommendations

[14] Trusted Firmware-A Release 2.10.0 - https://readthedocs.org/projects/trustedfirmware-a/downloads/pdf/stable/

[15] Interaction between Measured Boot and an fTPM (PoC) - https://trustedfirmware-a.readthedocs.io/en/latest/design_documents/measured_boot_poc.html#interaction-between-measured-boot-and-an-ftpm-poc

[16] "A Practical Guide to TPM 2.0: Using the Trusted Platform Module in the New Age of Security" by Will Arthur and David Challener

[17] Samsung KNOX - https://www.samsung.com/us/business/solutions/samsung-knox/

[18] Chapter 8. Enhancing security with the kernel integrity subsystem - https://access.redhat.com/documentation/en-us/red_hat_enterprise_linux/7/html/kernel_administration_guide/enhancing_security_with_the_kernel_integrity_subsystem

[19] TCG PC Client Platform Firmware Profile Specification - https://trustedcomputinggroup.org/wp-content/uploads/TCG_PCClientSpecPlat_TPM_2p0_1p04_pub.pdf

[20] TCG EFI Protocol Specification - https://trustedcomputinggroup.org/wp-content/uploads/EFI-Protocol-Specification-rev13-160330final.pdf

[21] Trusted Firmware-A GitHub source - https://github.com/ARM-software/arm-trusted-firmware/tree/master

[22] TEE-based fTPM Interface - https://www.kernelconfig.io/config_tcg_ftpm_tee

[23] Thom, Stefan, Jeremiah Cox, David Linsley, Magnus Nystrom, Himanshu Raj, David Robinson, Stefan Saroiu, Rob Spiger, Alastair Wolman, and Microsoft Technology Licensing Llc. 2011. “US20160048678A1 - Trustzone-based integrity measurements and verification using a software-based trusted platform module - Google Patents.” November 08, 2016. https://patents.google.com/patent/US20160048678A1/en.

[24] TCG Trusted Attestation Protocol (TAP) Information Model for TPM Families 1.2 and 2.0 and DICE Family 1.0 - https://trustedcomputinggroup.org/wp-content/uploads/TNC_TAP_Information_Model_v1.00_r0.29A_publicreview.pdf

[25] Trusted Computing Group (TCG) - https://trustedcomputinggroup.org/

[26] Remote Attestation With Tpm2 Tools - https://tpm2-software.github.io/2020/06/12/Remote-Attestation-With-tpm2-tools.html

[27] Francillon, A., Nguyen, Q., Rasmussen, K. B., & Tsudik, G. (2014). A minimalist approach to Remote Attestation. Design, Automation & Test in Europe Conference & Exhibition (DATE), 2014. doi:10.7873/date.2014.25

[28] Ankergård, Sigurd Frej Joel Jørgensen, Edlira Dushku, and Nicola Dragoni. 2021. “State-of-the-Art Software-Based Remote Attestation: Opportunities and Open Issues for Internet of Things.” Sensors 21 (5): 1598. https://doi.org/10.3390/s21051598.

[29] Shaneck, M., Mahadevan, K., Kher, V., Kim, Y. (2005). Remote Software-Based Attestation for Wireless Sensors. In: Molva, R., Tsudik, G., Westhoff, D. (eds) Security and Privacy in Ad-hoc and Sensor Networks. ESAS 2005. Lecture Notes in Computer Science, vol 3813. Springer, Berlin, Heidelberg. https://doi.org/10.1007/11601494_3

[30] Arvind Seshadri, Mark Luk, Elaine Shi, Adrian Perrig, Leendert van Doorn, and Pradeep Khosla. 2005. Pioneer: verifying code integrity and enforcing untampered code execution on legacy systems. In Proceedings of the twentieth ACM symposium on Operating systems principles (SOSP '05). Association for Computing Machinery, New York, NY, USA, 1–16. https://doi.org/10.1145/1095810.1095812

[31] A. Seshadri, M. Luk, A. Perrig, L. V. Doorn, and P. Khosla. SCUBA: Secure Code Update By Attestation in sensor networks. In ACM WiSec, 2006

[32] Y. Yang, X. Wang, S. Zhu and G. Cao, "Distributed Software-based Attestation for Node Compromise Detection in Sensor Networks," 2007 26th IEEE International Symposium on Reliable Distributed Systems (SRDS 2007), Beijing, China, 2007, pp. 219-230, doi: 10.1109/SRDS.2007.31.

[33] A. Seshadri, M. Luk, and A. Perrig. SAKE: Software attestation for key establishment in sensor networks. Ad Hoc Networks, 9(6), 2008

[34] C. Kil, E. C. Sezer, A. M. Azab, P. Ning, and X. Zhang. Remote attestation to dynamic system properties: Towards providing complete system integrity evidence. IEEE/IFIP DSN, 2009

[35] Y. Li, J. M. McCune, and A. Perrig. SBAP: Software-Based Attestation for Peripherals. TRUST, 2010

[36] Y. Li, J. M. McCune, and A. Perrig. VIPER: Verifying the Integrity of PERipherals Firmware. In ACM CCS, 2011.

[37] Francillon, A., Nguyen, Q., Rasmussen, K. B., & Tsudik, G. (2014). A minimalist approach to Remote Attestation. Design, Automation & Test in Europe Conference & Exhibition (DATE), 2014. doi:10.7873/date.2014.25

[38] Yang, X.; He, X.; Yu, W.; Lin, J.; Li, R.; Yang, Q.; Song, H. Towards a Low-Cost Remote Memory Attestation for the Smart Grid. Sensors 2015, 15, 20799-20824. https://doi.org/10.3390/s150820799

[39] Joonho Kong, Farinaz Koushanfar, Praveen K. Pendyala, Ahmad-Reza Sadeghi, and Christian Wachsmann. 2014. PUFatt: Embedded Platform Attestation Based on Novel Processor-Based PUFs. In Proceedings of the 51st Annual Design Automation Conference (DAC '14). Association for Computing Machinery, New York, NY, USA, 1–6. https://doi.org/10.1145/2593069.2593192

[40] Steffen Schulz, Ahmad-Reza Sadeghi, and Christian Wachsmann. 2011. Short paper: lightweight remote attestation using physical functions. In Proceedings of the fourth ACM conference on Wireless network security (WiSec '11). Association for Computing Machinery, New York, NY, USA, 109–114. https://doi.org/10.1145/1998412.1998432

[41] K. Eldefrawy, G. Tsudik, A. Francillon and A. Perito, "SMART: Secure and minimal architecture for (establishing dynamic) root of trust", Proc. Netw. Distrib. Syst. Secur. Symp., vol. 12, pp. 1-15, Feb. 2012.

[42] Patrick Koeberl, Steffen Schulz, Ahmad-Reza Sadeghi, and Vijay Varadharajan. 2014. TrustLite: a security architecture for tiny embedded devices. In Proceedings of the Ninth European Conference on Computer Systems (EuroSys '14). Association for Computing Machinery, New York, NY, USA, Article 10, 1–14. https://doi.org/10.1145/2592798.2592824

[43] Victor Costan, Ilia Lebedev, and Srinivas Devadas. 2016. Sanctum: minimal hardware extensions for strong software isolation. In Proceedings of the 25th USENIX Conference on Security Symposium (SEC'16). USENIX Association, USA, 857–874.

[44] Ferdinand Brasser, Brahim El Mahjoub, Ahmad-Reza Sadeghi, Christian Wachsmann, and Patrick Koeberl. 2015. TyTAN: tiny trust anchor for tiny devices. In Proceedings of the 52nd Annual Design Automation Conference (DAC '15). Association for Computing Machinery, New York, NY, USA, Article 34, 1–6. https://doi.org/10.1145/2744769.2744922

[45] Ghada Dessouky, Shaza Zeitouni, Thomas Nyman, Andrew Paverd, Lucas Davi, Patrick Koeberl, N. Asokan, and Ahmad-Reza Sadeghi. 2017. LO-FAT: Low-Overhead Control Flow ATtestation in Hardware. In Proceedings of the 54th Annual Design Automation Conference 2017 (DAC '17). Association for Computing Machinery, New York, NY, USA, Article 24, 1–6. https://doi.org/10.1145/3061639.3062276

[46] G. Dessouky, T. Abera, A. Ibrahim and A. -R. Sadeghi, "LiteHAX: Lightweight Hardware-Assisted Attestation of Program Execution," 2018 IEEE/ACM International Conference on Computer-Aided Design (ICCAD), San Diego, CA, USA, 2018, pp. 1-8, doi: 10.1145/3240765.3240821.

[47] Noorman, J., Agten, P., Daniels, W., Strackx, R., Van Herrewege, A., Huygens, C., ... & Piessens, F. (2013). Sancus: Low-cost trustworthy extensible networked devices with a zero-software trusted computing base. In 22nd USENIX Security Symposium (USENIX Security 13) (pp. 479-498).

[48] Tigist Abera, N. Asokan, Lucas Davi, Jan-Erik Ekberg, Thomas Nyman, Andrew Paverd, Ahmad-Reza Sadeghi, and Gene Tsudik. 2016. C-FLAT: Control-Flow Attestation for Embedded Systems

Software. In Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security (CCS '16). Association for Computing Machinery, New York, NY, USA, 743–754. https://doi.org/10.1145/2976749.2978358

[49] V. Costan and S. Devadas, "Intel SGX explained", IACR Cryptol. ePrint Arch., vol. 2016, no. 86, pp. 1-118, 2016.

[50] M. Conti, E. Dushku and L. V. Mancini, "RADIS: Remote Attestation of Distributed IoT Services," 2019 Sixth International Conference on Software Defined Systems (SDS), Rome, Italy, 2019, pp. 25-32, doi: 10.1109/SDS.2019.8768670.

[51] Moritz Lipp, Michael Schwarz, Daniel Gruss, Thomas Prescher, Werner Haas, Jann Horn, Stefan Mangard, Paul Kocher, Daniel Genkin, Yuval Yarom, Mike Hamburg, and Raoul Strackx. 2020. Meltdown: reading kernel memory from user space. Commun. ACM 63, 6 (June 2020), 46–56. https://doi.org/10.1145/3357033

[52] Paul Kocher, Jann Horn, Anders Fogh, Daniel Genkin, Daniel Gruss, Werner Haas, Mike Hamburg, Moritz Lipp, Stefan Mangard, Thomas Prescher, Michael Schwarz, and Yuval Yarom. 2020. Spectre attacks: exploiting speculative execution. Commun. ACM 63, 7 (July 2020), 93–101. https://doi.org/10.1145/3399742

review
[53] W. A. Johnson, S. Ghafoor and S. Prowell, "A Taxonomy and Review of Remote Attestation Schemes in Embedded Systems," in IEEE Access, vol. 9, pp. 142390-142410, 2021, doi: 10.1109/ACCESS.2021.3119220

[54] Ruan, Xiaoyu. 2014. “Boot With Integrity, or Don’t Boot.” In Apress eBooks, 143–63. https://doi.org/10.1007/978-1-4302-6572-6_6

[55] Admin, Tcg. 2022. “Remote Platform Integrity Attestation | Trusted Computing Group.” Trusted Computing Group. September 30, 2022. https://trustedcomputinggroup.org/remote-platform-integrity-attestation-2

[56] Intel® Trusted Execution Technology - https://www.intel.co.uk/content/dam/www/public/us/en/documents/technology-briefs/trusted-execution-technology-overview.pdf

[57] Vinaypamnani-Msft. 2023. “Secure the Windows Boot Process - Windows Security.” Microsoft Learn. August 11, 2023. https://learn.microsoft.com/en-us/windows/security/operating-system-security/system-security/secure-the-windows-10-boot-process.

[58] M. Ambrosin, M. Conti, R. Lazzeretti, M. M. Rabbani and S. Ranise, "PADS: Practical Attestation for Highly Dynamic Swarm Topologies," 2018 International Workshop on Secure Internet of Things (SIoT), Barcelona, Spain, 2018, pp. 18-27, doi: 10.1109/SIoT.2018.00009

[59] N. Asokan, Ferdinand Brasser, Ahmad Ibrahim, Ahmad-Reza Sadeghi, Matthias Schunter, Gene Tsudik, and Christian Wachsmann. 2015. SEDA: Scalable Embedded Device Attestation. In Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security (CCS '15). Association for Computing Machinery, New York, NY, USA, 964–975. https://doi.org/10.1145/2810103.2813670

[60] Lee J. Collective Attestation for Manageable IoT Environments. Applied Sciences. 2018; 8(12):2652. https://doi.org/10.3390/app8122652

[61] S. VL, M. Singh Dilip Thakur and R. Ma, "A Novel Multi Verifier Device Attestation Scheme for Swarm of Devices," 2018 32nd International Conference on Advanced Information Networking and Applications Workshops (WAINA), Krakow, Poland, 2018, pp. 124-129, doi: 10.1109/WAINA.2018.00074.

[62] Jämes Ménétrey, Christian Göttel, Anum Khurshid, Marcelo Pasin, Pascal Felber, Valerio Schiavoni, and Shahid Raza. 2022. Attestation Mechanisms for Trusted Execution Environments

[63] Kornaros, G., Tomoutzoglou, O., Mbakoyiannis, D., Karadimitriou, N., Coppola, M., Montanari, E., … Gherardi, G. (2020). *Towards Holistic Secure Networking in Connected Vehicles through Securing CAN-bus Communication and Firmware-over-the-Air Updating. Journal of Systems Architecture, 101761.* doi:10.1016/j.sysarc.2020.101761

[64] G. Kornaros, D. Bakoyiannis, O. Tomoutzoglou, M. Coppola and G. Gherardi, "TrustNet: Ensuring Normal-world and Trusted-world CAN-bus Networking," *2019 IEEE International Conference on Communications, Control, and Computing Technologies for Smart Grids (SmartGridComm)*, Beijing, China, 2019, pp. 1-6, doi: 10.1109/SmartGridComm.2019.8909715.

[65] TRS 0.3.0 Documentation - https://trs.readthedocs.io/.

[66] An Introduction to ARM Trusted Firmware Date: 28-10-2013 Speaker: Andrew Thoelke - https://www.slideshare.net/slideshow/arm-trusted-firmareforarmv8alcu13/32664514#4

Demystified. In Distributed Applications and Interoperable Systems: 22nd IFIP WG 6.1 International Conference, DAIS 2022, Held as Part of the 17th International Federated Conference on Distributed Computing Techniques, DisCoTec 2022, Lucca, Italy, June 13-17, 2022, Proceedings. Springer-Verlag, Berlin, Heidelberg, 95–113. https://doi.org/10.1007/978-3-031-16092-9_7

Terminology

REE: Rich Execution Environment
TEE: Trusted Execution Environment
TPM: Trusted Platform Module
RSE: Runtime Security Engine
PCR: Platform Configuration Register
EL3: Exception Level 3
WSN: Wireless sensor networks
ROT: Root Of Trust
OS: Operating System
IAK: Initial Attestation Key
AIK: Attestation Key
TAP: Trusted Attestation Protocol
PUF: Physical Unclonable Function
MPU: Memory Protection Unit
IMA: Integrity Measurement Architecture
EVM: Extended Verification Module
fTPM: Firmware TPM
EBBR: Embedded Base Boot Requirements