Independent project (degree project), 15 credits, for the degree of Master of Science in Computer Science with Specialization in Embedded Systems Spring Semester 2018

Hardware Root of Trust for Linux Based Edge Gateway

Mohamad Al-Galby
Arezou Madani

Faculty of Natural Sciences

Högskolan Kristianstad | www.hkr.se

Författare/Author

Mohamed Al-Galby, Arezou Madani

Titel/Title

Hardware Root of Trust for Linux Based Edge Gateway

Handledare/Supervisors

Professor Qinghua Wang, Kristianstad University

Jens Jakobsen, Labs Development Manager, HMS Industrial Network AB

Johan Ederönn, HMS Industrial Network AB

Examinator/Examiner

Professor Eric Chen, Kristianstad University

Abstract
Linux-based Edge Gateways that connects hundreds or maybe thousands of IoT devices, are exposed to various threats and cyber-attacks from the internet. These attacks form a considerable risk targeting the privacy and confidentiality of IoT devices throughout their gateways. Many researches and studies have been conducted to alleviate such a problem. One of the solutions can be achieved by building a root of trust based on a hardware module such as Trusted Platform Module (TPM) or software like Trusted Execution Environment (TEE). In this work, we provide a solution to the problem by enabling Hardware Root of Trust (HRoT) using TPM on a product from HMS Industrial Network AB known as GWen board, a Linux-based embedded system, used as gateway to connect IoT devices. We describe a method that uses the processor of the GWen (i.e. Zynq-7020 FPGA SoC) to enable secure boot. Besides, we provide a method to enable the TPM chip mounted on the GWen (i.e. SLB 9670 TPM 2.0) using TPM Software Stack TSS 2.0. We demonstrated, in detail, various use-cases using the TPM on GWen including cryptographic keys generation, secure key storage and key usage for different cryptographic operations. Furthermore, we conducted an analysis to the adopted solution by inspecting the latency of TPM commands on the GWen gateway. According to the high restrictions of TPM 2.0 specifications and based on our results, adding the TPM 2.0 to the IoT gateway GWen will enhance the security of its Linux distribution and will makes it possible to securely identify and authenticate the gateway on the network based on its secret keys that are stored securely inside its TPM.

Ämnesord/Keywords
HRoT, TPM 2.0, Zynq, TSS, RSA, Secure Boot, TPM Simulator

Contents

Table of Figures

Table of Tables

Acronyms

AIK	Attestation Identity Key
API	Application Programming Interface
ECC	Elliptic-curve cryptography
EK	Endorsement Key
EPS	Endorsement Primary Seed
ESAPI	Enhanced System Application Programming Interface
FAPI	Feature Application Programming Interface
FPGA	Field Programmable Gate Arrays
HMAC	Hash-based Message Authentication Code
HRoT	Hardware Root of Trust
HSM	Hardware Security Module
IoT	Internet of Things
MAC	Mandatory Access Control
MLS	Multi-Level Security
OCM	On-Chip Memory
PCR	Platform Configuration Registers
PUFs	Physical Unclonable Functions
RNG	Random Number Generation
RoT	Root of Trust
RSA	Rivest–Shamir–Adleman
RTM	Root of Trust for Measurement

RTR	Root of Trust for Reporting
RTS	Root of Trust for Storage
SAPI	System Application Programming Interface
SEE	Secure Execution Environment
SOC	System On-Chip
SRAM	Static Random-Access Memory
SRK	Storage Root Key
TAB	TPM Access Broker
TCB	Trusted Computing Base
TCTI	TPM Command Transmission Interface
TEE	Trusted Execution Environment
TPM	Trusted Platform Module
TSS	TCG Software Stack

1. Introduction

1.1 Background

In 4th industrial revolution it is expected that more than 13 billion things will be connected to the internet by 2020. While it brings many advantages for the human but always there are trade-offs between opportunities and threats. While users are satisfied of these advantages, they became also worried about their data security and privacy. Users want to be confident when they power up their deployed embedded systems, ensuring that the software on their system can be trusted. Trusted, in this sense, means that the system will run only the software that the system integrator considers as guanine, and that no other code, malicious or otherwise, has been added to it [1].

Edge gateways are a critical part of the Internet of Things (IoT) infrastructure where they control the information flow between the sensors and other IoT devices and the cloud where data collected from the sensors are stored [2]. To ensure security, the edge gateways should implement solutions that provide Root of Trust (RoT) which in turn ensures trustworthiness.

The existing studies show that RoT can be realized using Trusted Platform Module (TPM) or Trusted Execution Environment TEE (referred to as TrustZone). Roots of Trust (RoTs) are security primitives composed of hardware, firmware and/or software that provide a set of trusted, security-critical functions. RoTs provide security services including cryptographic support, secure key storage, secure signature storage, and secure access to trusted functions, therefore RoTs need to be secured by their design.

Hardware RoTs are preferred over software RoTs due to their immutability and more reliable behavior. A good example of Hardware RoT is the Trusted Platform Module (TPM), a hardware security microchip for both personal computers and embedded systems that can be used for trusted boot at the system startup, to make sure that the computer boots a valid operating system. TPMs can checks the integrity of the running OS by scanning for signs of changes and checks whether or not the software meets its security requirements before the boot loader is executed. This makes advanced malware such as Bootkits detectable, and it can reduce the risk of data compromises [3].

Furthermore, TPM is used to protect critical assets such as cryptographic keys and authentication secrets inside a tamper-evident hardware module.

Trusted Execution Environment (TEE) is a virtual machine that is dedicated to run as secure processing environment that is tampering-resistant but also as integrity-protector. TEE consists of memory and storage capabilities that runs on a separate kernel and is isolated from the rest of the platform. The TEE separates the environment into secure and normal world. The secure world is protected from tampering and observation by

unauthorized parties by using hardware memory protection and cryptographic protection of storage.

TEE ensures that the executing code is authenticated and is confidential during the runtime. It controls the state of the CPU, memory and I/O in regard to their integrity and save their log in a permanent memory. Furthermore, it also prevents the software and physical attacks to the main memory [4]. In order to enable the Root of Trust (RoT) in the gateway, TEE can be implemented as well as the TPM.

This work attempts to analyze enabling Hardware Root of Trust (HRoT) on one of HMS[1] Linux-based edge gateways. The goal is to enhance the security by counter measuring the increasing threads of cyber-attacks by implementing root of trust inside the gateway to store credentials and allow authentication securely. One of the solutions to realize HRoT can be TPM or TEE. In our thesis we will highlight the two approaches with more focus in implementing HRoT using the TPM.

1.2 Aim and Purpose

The main purpose of the project is to find out how to enable HRoT on one of HMS's[1] products, that's a Linux-based Edge Gateways (Gwen). The goal is to enhance the security of Linux distribution of the gateway to take advantage of the services offered by HRoT. This will accelerate industry efforts to implement security capabilities that can provide a higher degree of assurance of the trustworthiness of a device.

In this work, an SLB9670 TPM 2.0 chip from Infineon Corporation is mounted on the board and thus will be used as the root of trust for the system. The following step is to utilize the TPM to generate and store cryptographic keys securely and privately. These secrets can be used to encrypt/decrypt some credentials that will later be used for different purposes like identify and authenticate the gateway over the network to which the GWen is connected.

1.3 Systematic Literature Study

The systematic literature review was conducted for accomplishing this thesis. So, after searching for related literature, regarding to the targeted problem (enabling HRoT on a Linux base Edge Gateway), we narrowed down and focused on utilizing the trusted platform module (TPM) on the hosted platform. Afterwards, we developed our research in the means of creating and storing cryptography keys.

[1] HMS Industrial Networks AB is a Swedish company develops products and solutions that enable industrial hardware to get connected to IoT software.

We used the following search-string to combine similar keywords: (enabling OR utilizing OR realizing) AND (Hardware Root of Trust) AND (Linux base) AND (Gateway OR Network equipment OR Host Platform) AND (TEE and TrustZone). Using the exact string limited our choices, so, in some cases, we had to use more general strings, e.g. "Secure Linux base Network equipment". By generalizing our searching keywords, we gained many more relevant results, but other more results were irrelevant too. In order to search related literatures, the 6 main area keywords used for searching: (Trusted computing, Hardware Root of Trust, Secure Boot, Trusted Platform module, TPM Software Stack, Cryptography operation)

Trust Computing: "Trusted System", "Secure System", "Edge Computing", "Trusted Platform" this search resulted in 83 Articles.

Hardware Root of Trust: "Secure Boot", "Root of Trust", "Trustworthy System", "HRoT on Platform". For this search 54 articles were founded.

Secure Boot: "Linux Secure Boot", "Zynq Secure Boot", "TPM Secure Boot", "Network Booting", "U-Boot", "Boot Sequence on SoC". This research result in 43 articles.

Trusted Platform Module: "Utilizing TPM", "Enabling TPM", "TPM Architecture", "TPM Specification", "TPM Fundamental Element", "TPM Structure". This research result in 30 articles.

TPM Software Stack: "TPM Communication", "Running TPM", "TPM Development Tools", "TPM Data Transmission", "TPM Command", "TPM Command Template". This research result in 8 articles.

Cryptography Operation: "decryption and encryption by TPM", "TPM Cryptography measurement", "TPM key generating", "TPM Key Storage", "TPM Crypto Algorithm" This research result in 12 articles.

The total sum of all researches resulted in 230 articles, but they are narrowed down to 41 through finding most relevant key words, considering published time, the number of referencing and citation and if they are peer-reviewed. We used another screening method by searching some of the aforementioned keywords inside the selected articles. Many of the resources were related to TPM 1.2 and TPM 1.0 which are the earlier version of the TPM, but our focus was on TPM 2.0 in particular. We tried to omit old articles and publications and select only resources that were published after 2006. Only one old article belongs to 1998 was used for the performance analysis and it is the only reliable resource for our evaluation.

The most frequently used databases were ACM Digital Library, Summon@HKR, IEEExplore, Microsoft Developer Network, SpringerLink and Google Scholar. The TCG homepage was valuable resource for us as it includes all TPM 2.0 specifications that were rarely explained and existed somewhere else. Moreover, we found other resources that provide adequate piece of information about TPM 2.0 and its internal structure and

entities; “A practical Guide to TPM2.0” [11], “Trusted Platform Module Library. Part 1: Architecture” [16], “TPM Provisioning” [34], “Trusted Platform Module Library Part 3: Commands, Family 2.0” [40].

1.4 Sustainability and Ethics

All the organization around the world are responsible for their customer’s information. In the aspect of individual privacy, protecting customer information is not an additional luxury, but an ethical issue for the organization. It is obvious that the assurance of information security comes through secure equipments, that are tamper resistant and impervious.

In regard to sustainability, the concept is a huge, but it can be narrowed down to particular terms which are related to emissions management, ecological economic, human rights, gender diversity, etc. [5] In this regard, we will address human rights and economical aspects. These two fields can be discussed generally or in associate to the organization. In our case developing discussion for human right in the globe and economy in the organization makes more sense. In general, improving the security of the network equipment leads to better protection of people’s private information and one of the major factors of human rights is the secure feeling about the privacy. So, each effort in this regard is towards the respect of human rights, which is a part of social policy for the sustainability.

In the economic view, cyber security has a huge impact on business and this can be measured as millions of dollars of company value [5], because investor won’t invest in distrusted companies and loosing investors consequently means disaster to the economics of these companies. Therefore environmental, social and governance (ESG) performance (i.e. a measure for sustainability) is directly connected to the financial performance which is influenced by the secure IT. Preventing attacks by improving the security, can bring high performance for organization’s IT. As it is mentioned previously, improving security can be gained by the secure network and network equipment.

1.5 Acknowledgement

We would like to thank everyone contributed in accomplishing this thesis. Our special thanks go to our respected families who gave us the support and help we needed to reach the goal of this work. We would also like to show our appreciation to our advisors Professor Qinghua Wang as well as to all our teachers at Kristianstad University, the place where we gained the required knowledge needed for this work.
We would also like to thank our supervisors at HMS Industrial Networks AB: Dr. Jens Jakobsen and Mr. Johan Ederönn who gave us valuable feedbacks and helped us in building the TSS stack into the GWen board, which was used to realize this thesis work.

2. Related Work

With regard to HRoT, there are few widespread applications that rely on the functionality provided by the TPM or TEE. Some studies provide examples of existing pieces of software, and others only propose use cases. A well-known TPM applications from Microsoft are the BitLocker for full-disk encryption, and Virtual Smart Cards for TPM-based authentication instead of physical smart cards [42]. Some more specialized studies that focus particularly on security enhancement using the TPM and TEE are provided in the following subsections.

2.1 HRoT using TPM

A study by [43] attempts to alleviate some vulnerabilities of the Basic Input/Output System (BIOS) that uses TPM for trusted boot which is different from secure boot. The trusted boot process does not prohibit booting into an insecure OS or using an insecure boot loader. Typical BIOSs use a password for pre-boot authentication, which is stored in CMOS memory before reading the Master Boot Record (MBR). If no authentication is performed, an attacker can simply use function key F7 to change the booting sequence priority to load insecure OS into the device. The BIOS-password uses simple encryption like CRC-16-IMB, which can easily be broken using a tool such as CmosPwd. Yet, the entire CMOS memory can be flushed including the BIOS-password, by only removing its battery for about 2 minutes.

The work in [43] proposes a novel method to countermeasure these shortcomings by adopting HRoT using the TPM to improve boot security at the BIOS layer. The study focuses on verifying the integrity of the full MBR before passing control to the OS boot-loader, using TMP Hashing function, and storing the valid digest into TPM's Non-Volatile RAM (NVRAM), such that the boot process will not proceed in the case of an altered or invalid MBR. This solution is augmented with some more techniques to enhance the BIOS security including; (i) RSA authentication for the BIOS-password using the TPM instead of conventional encryption techniques. (ii) storing the encrypted password into TPM's NVRAM to prohibit an attacker from clearing the password by removing the battery. (iii) activating always BIOS-password authentication before passing control to the OS boot-loader. (iv) verifying data integrity of the full MBR using TPM SHA-1 engine, and if verification failed, break the booting sequence.

However, the study focuses mainly on the design and proposes an architecture for BIOS boot security enhancement and doesn't provide a method for enabling the TPM or how TPM drivers and software are used to set the interrupt calls at the BIOS layer.

2.2 Performance Evaluation of HRoT using TPMs

Another work in [4] provides a method for HRoT in multicore environments using TPM and studied the performance impact of concurrent access to TPM services by multiple applications. The proposed solution enabled HRoT using Trusted Software Stack TSS 1.2

informally known as TrouSerS and applies simulation toolset to test set of major TPM services provided by the TSS. Using the TSS will make it possible for the TPM to serialize commands i.e. perform multiple commands simultaneously. The study focuses mainly on using: (i) multiple TPMs, and (ii) single TPM with different scheduling techniques such as Shortest Job First (SJF) and First Come First Served (FCFS) scheduling. The results indicate that both techniques decrease the TPM response time, with 35% performance improvement with simple request reordering for concurrent applications that uses the same TPM, and a factor of 2.7X performance improvement when adding two more TPMs.

This study, however, doesn't address the secure boot and only utilizes the TSS 1.2 with TPM simulation toolset. Although this study uses an old version of the TSS and the TPM 2.0 on our platform is not backward compatible with such old versions of the TSS, but this study gives a good insight for TPM analysis and which metrics, services and benchmarks to be taken into account as well as how the performance evaluation can be conducted in similar environments.

2.3 HRoT using TEE

The TEE is another way to construct the HRoT and is widely adopted in almost every smartphone and tablet today. The TEE is a secure integrity-protected processing environment runs in parallel to the operating system, to allow secure execution of critical processes isolated from other counterparts. Processes that run in TEE can gain access to full or part of the main processor and memory of the platform, whereas hardware isolation protects these processes from user installed applications running in the main operating system. TEE ecosystems can be implemented if hardware support, such as TrustZone form ARM or SGX from Intel exist on the hosted platform. [45]

In [46], a trusted embedded operating system architecture for mobile devices is realized based on the ARM TrustZone processor which implements the TEE. The TrustZone, which is a hardware isolation mechanism, is used to separate the environment into "secure" and "insecure" worlds, and support executing the processes in either world, where both worlds are running independently in separated isolated execution environments. The study proposes other aspects of HRoT such as secure boot using ARM-TrustZone platforms for future work.

Another study [44] presented a research prototype that provides the root of trust for platforms that uses the TrustZone-enabled. The work leverages the SRAM PUF which is commonly available on mobile devices to provide foundations for the HRoT by building root of trust running in the secure world of the TrustZone. This study uses ALTERA Cyclone development board equipped with Xilinx's Zynq-7020 AP SoC, whereas our GWen board uses Zynq-7020 FPGA SoC. Despite that both belong to the same family, but each processor has different architecture, and yet, different implementation requirements.

To maintain our limitation within the scope of the project, we will specifically address enabling the HRoT using the TPM which is already mounted on GWen board, and we will investigate the possibility of performing secure boot using either the TPM or using the Zynq-7020 FPGA SoC, the main processor on GWen board.

3. Theoretical Background

Upon the development of the first TPM by group of computer engineers, they established what is known as Trusted Computing Group (TCG). TCG came up with some specifications to create a hardware basement of security on which secure systems can be built. As a consequence, the development of a hardware TPM chip took place and it physically can be mounted on the motherboard of a PCs or embedded systems.

Network equipment such as routers, gateways and switches are such a significant kind of devices for keeping the privacy of network's users. According to TCG recommendation, the privacy can be maintained by embedding TPMs in these devices.

In this section the necessity of securing network equipment and the possible solutions of realizing the security according to the Trusted Computing (TC) is presented. Although, regard to lack of information in this area, there are deficiency in implementing TC on network equipment. We tried to present a comprehensive analysis associate to the project scope. So, the importance of the privacy for the edge gateway and the possible threats is explained in section 3.1. The building blocks for the HRoT are described in section 3.2. In addition to that, the role of TPM is provided in section 3.3.

3.1 Privacy and Edge Gateway

The privacy of the network users is provided by a secure networking equipment (gateway, router, access point, etc.). These devices limit data transfer to exclusively authorized destinations or users. For instance, Gateway, in which authorization is necessary for the network connection, are typically the Policy Enforcement Points and unauthorized users cannot access the login information of members. Moreover, unauthorized neighbors cannot read any routing information that belongs to router's peers. The traffic must be decrypted or encrypted consistently after configuration, even as ID protection (Identity of device). Privacy protection is due to the functions employed in every single layer of the networking device including hardware and software [7].

The administrator of the Equipment has the significant role in security of the network user's privacy in order to eliminate any doubt about performing any configured function of each device. It is done by identification of the Network Equipment, its boot configuration and measured device state, such as PCR value, to the administrator so that the user and peer privacy guarantees are secured [7].

It can be concluded that the security of the Gateway is essential for the host network and its users and the equipment should be trusted by users and verified by the administrator.

3.1.1 Threats to the Edge Gateway

Although Gateway is a networking equipment to secure the Network, there are some risks that can threaten its safety. These threads are due to accessibility of unauthorized devices

to the network information, interference of unauthorized codes and hostile hidden firmware implants [8].

3.2 Hardware Root of Trust Building Block

Hardware Root of Trust is formed based on two main concepts, Trusted Computing concept and Chain of Trust. The HRoT can be realized by implementing a hardware module like TPM in the device. In this section Trust computing and chain of the trust are explained and the architecture of the TPM is described in detail. Chain of Trust is utilized whenever Secure boot is required. Although Secure boot is not achieved by the TPM in our case, we think that explaining Chain of trust can give a better overview of the whole concept of HRoT and how it can be realized using secure boot.

3.2.1 Trusted Computing

The network security is achieved based on the trust between components of the system. TC is the method of creating a systems and infrastructures due to the trust as a predefined behaviour which cannot be violated by any components [9]. The TC components must be able to ascertain their identity with public key cryptography, which is bounded to platform with a private key. They use mechanisms, such cryptographic hashes of object code, to link the ongoing configuring software and let the user to decide about the level of trust.

Concept of Trusted Computing is considered as a set of technologies that through them hardware and software operate in a secure manner. Supporting trusted computing requires hardware infrastructure. Set of hardware and software component which are providing trusted security function is a trusted platform [10]. Three roots of trust lie at the core of a trusted platform:

- **Root of Trust for Measurement** (**RTM**), which is responsible for taking platform integrity measurements.
- **Root of Trust for Storage** (**RTS**), which securely stores different integrity measurements.
- **Root of Trust for Reporting** (**RTR**), which is responsible for reliably reporting values stored in the RTS.

3.2.2 Chain of Trust

The running software on the Network Equipment Device should maintain its integrity in order to provide the security of the system. This will be obtained using a chain of trust model. Each step of the model since start of the algorithm, checks the next step before it executes, and it can process as many steps as required. Figure 1 demonstrates this model. The first step of the model is called the Root of Trust which starts the chain of the continuous checking. The code of the Root of Trust should be trusted separately and directly [7]. There are several functions associated with the Root of Trust. Initiation of

the trusted platform, performing the first measurement and confirming the authorizations are the most important of them [7].

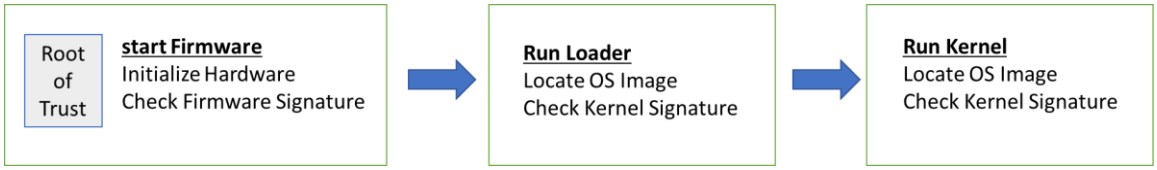

Figure 1 Chain of Trust in a Linux-based embedded environment

3.2.3 TPM Architecture

The latest versions of the TPM are TPM 1.2 and TPM 2.0. The TPM 1.2 specification was the Trusted Computing Group's (TCG's) first introduction and was aimed at addressing the following major issues in the industry [11]:

- Identification of devices
- Secure generation of keys
- Secure storage of keys
- Non-Volatile RAM storage
- Device health attestation

The TPM 2.0 starts and expands based on the legacy of TPM 1.2, but it presents new features such as algorithm agility, enhanced authorization, quick key loading and so on. Our implementation and analysis are based on the TPM 2.0 specifications. The main components of TPM Chip are shown in Figure 2:

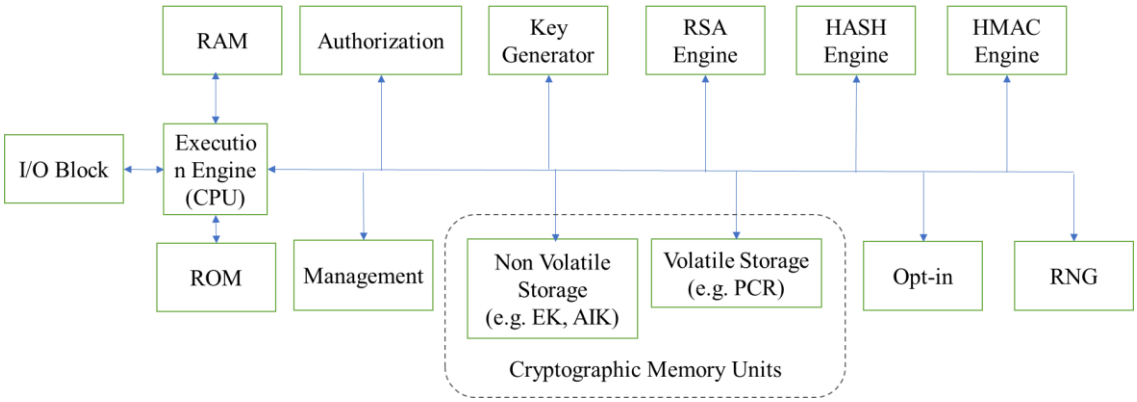

Figure 2 TPM Architecture [12]

I/O Block is a communication bus for managing data flow and communication between components. TPM acts as a slave device through this interface which means it can respond only to the received commands from TPM Device Driver. This is not a general I/O block and it is not accessible by the operating system.

Execution Engine is responsible to run commands from I/O and program code which is located in the TPM. The program code that resides on the TPM is the actual root of trust for integrity measurements [13].

Crypto Engine is used for utilizing the embedded crypto algorithms. There are three main engines: RSA, HASH and HMAC. It also includes a key generator for generating RSA 1024 or 2048-bits keys.

Platform Configuration Registers (PCRs) are used for integrity measurement by storing integrity metrics. These integrity metrics stored in these registers are the measurement of the integrity of any code, from BIOS to applications, typically before the code is executed [13].

Volatile Storage (RAM) is a temporary memory which depends on the power of the device and will lose its content upon TPM reboot. Temporary data such as PCR values and the state of the TPM can be stored in it.

Non-volatile Storage (NVRAM, flash) is used for storing long term keys and objects. Critical objects and privacy-sensitive secrets like Endorsement Key (EK) and Storage Root Key (SRK) can be stored in the NVRAM. It is an adequate to store other secrets like owner authorization data (passwords and policies).

Random Number Generator (RNG) is used to generate random bit stream, which is used as a seed for random number generator and cryptographic keys. Typically, RNGs mix random data with some extra source of entropy such as TPM's internal clock [35].

Key Generator (KG) is based on TPM's own RNG and doesn't rely on any external sources of randomness. Therefore, it decreases the possibility of weaknesses of key security based on software with an insufficient source of entropy or software with weak random number generators. In TPM 2.0, keys can be generated from a seed, using TPM's RNG or imported from another TPM. Generating TPM keys is the most time-consuming cryptographic calculation in most cases.

3.3 Importance Role of TPM

Securing network has some primary requirements to remote verification of each piece of the network. The TPM can play a significant role to secure a gateway. Two main categories are considered for identification: Manufacturer identity and Owner identity. Device Manufacturers characteristically define and configure the Manufacturer identity for each device to be used by the device owner. On the other hand, the device Administrator is the only one who can establish and use the Owner identity [7].

The uniqueness of the Manufacturer identity is critical among all products of that certain manufacturer including a model number and serial number. The name spaces have to be identified exclusively by either the manufacturer, administrator or both, despite the weakness of these kinds of identifiers in cryptography. In some cases, the unique

identifier can be defined as the public key, which is associated specifically to its private part setting up the device identity. The owner identity; however, should not been defined more than once in the specific domain of the Administrator such as an asset number.

IEEE 802.1AR categorizes device identities into an Initial Device ID and Local Device ID [14]. According to IEEE standard, Device ID is described as a safe device identifier to enable secure device authentication due to several industry standards and protocols such as Extensible Authentication Protocol (EAP) [14]. A private key proves the device identity, which is associated to a specific public key and a certificate. Certification Authority (CA) is a trusted authority that can validate a key and generate the corresponding certificate [7].

3.3.1 Gateway Authentication

Authenticity of the network is maintained through guaranteeing the identity of the connected devices to be trusted to assure that the network is not counterfeit. As mentioned, device manufacturer establishes a signed identity in the device, which is a key that is impossible to copy or instantiate in other devices. TPM can generate unique secure keys and store them persistently inside the TPM and hence inside the Gateway.

To generate such keys, TPM manufacturer has embedded a primary seed from which an endorsement seed in the TPM can be derived. Based on the embedded endorsement seed, several endorsement primary keys can be generated by the device manufacturer. The primary key endorsement generation procedure includes applying standard cryptography algorithms such as RSA or ECC and a well-known template. RSA 2048-bit is the default algorithm for endorsement key. The procedure of generating an EK and its associated AIK is discussed in section 6.4 of this report.

3.3.2 Device ID

Prerequisite of the secure network is the authenticated equipment and the ensuring of counterfeit devices prevention. Authentication is based on Device Identification (Device ID). There are two kinds of Device ID. One of them is the manufacturer identity for a device which is a unique ID and the device will be identified by this whole of the lifetime for the manufacturer. This one is known as the initial Device ID. The other one is the owner identity for a device which is established by the device administrator and it is unique in the administration domain, and it is known as the local Device ID [7]. Device ID is produced based on a private key, which is generated by the TPM in the device and is kept secret there.

3.3.3 Securing Secret

A gateway is exposed to three types of attack; physical attack, network attack and software compromise while storing important data, such as traffic logs and cryptographic

keys. Access to the information is a threat to the confidentiality of the network. Data protection is based on two solutions according to the extent of information and the securing operation [7].

In the first solution, the TPM is responsible of both protection and application of keys. The TPM provides extensive loss protection. The key type, and the low-power crypto engine of the TPM control the signing throughput.

In the second solution, the TPM is responsible of protection of an outside-TPM key. A high-throughput crypto engine might empower this. The “at rest’’ TPM is a secure protection in this case; however, a key-release is necessary to use it again and this jeopardizes the key security.

3.3.4 Key Management

The secure device in the network which could be authenticated by the Device ID and is able to secure secret, should have the capabilities of generation, distribution, backup and destruction of keys.

Keys should be generated randomly and based on a strong random number otherwise they can be regenerated by the attacker easily. Also, key material should be confidential. In the key distributing process key should be kept secure. TPM is designed to generate strong keys by its strong random number generator and supporting secure distribution. Secure distribution is done through keeping records of the keys and devices and keeping public key of central devices in the provisioning time for validating the signing key of them [11].

Keys can be created and recreated many times from a single seed in the TPM. It provides key template which are kept by the devices. In the time of activation by using the same template, the key will be regenerated. In some case after using the key, it should be vanished. TPM is able to destroy the key if needed.

3.3.5 Out of Service Gateway

Network equipment consists of sensitive data such as traffic log, user information and routing policies. In the case of halting the use of the gateway, the content of it should not be accessible any more. Closed data such as TPM keys and encrypted cipher text should be removed from the equipment. The data should be out of access in the TPM procedure for deprovision gateway after taking the device out of service.

4. Methodology

Enabling Hardware Root of Trust on the GWen gateway can be realized using TPM, TEE or both. In this work, we first started by investigating the feasibility to enable the currently existing hardware module i.e. SLB 9670 TPM 2.0 chip by conducting a literature review to gather the required knowledge in this area. Our systematic literature review is provided in section *1.3 Systematic Literature Study*. Our founding shows that TPM can be used with different software and source codes. However, developing new software which constructs TPM commands according to the template provided by TPM 2.0 specification would be very difficult and time-consuming. Instead, we found two open sources compatible with TPM 2.0, we download and tested them, and we also used TPM simulator, a software tool from IBM to validate the adopted solution. The detailed description about these solutions and how they are used is provided in section **5.2 TPM Accessibility**.

In the following sections, we provide brief description about TPM 2.0's most important issues that should be understood prior to starting the implementation. In section 4.1 a definition of TPM 2.0 specifications is provided, and in section 4.2 a short summary on the development of TPM 2.0 compared to legacy TPMs. Section 4.3 describes some of the most important foundational elements of the TPM 2.0.

4.1 TPM 2.0 Specifications

The TCG provides a set of specifications that must be maintained by TPM manufacturers including architecture, structure, commands and supporting routines [15]. TCG published revisions of the specification openly for public review and these specifications are available for download at [15] as four different parts. All these parts are required by TPM manufactured in order to build a complete standard.

Part 1- Architecture: It clarifies the terms used in the specification with diagrams and descriptions. It shows how TPM can react with different actions.

Part 2 - Structure which describes definition of constants, structures, flags and union definition in tables and shows how entities can be converted into C codes by marshal/unmarshal command and response byte streams. The defined values in this part are used by TPM commands and by its supporting routines.

Part 3 - Commands which contain definitions of TPM commands that utilize different entities defined at and represented by part 2 structures. This part includes command implementation in C. The code is separated into part 3 Commands and part 3 Commands and Code that defines the behaviour of the TPM. Commands are used for TPM internal tasks e.g. initialization and self-testing but also for cryptographic operations e.g. generate cryptographic keys and random numbers, hash and sign messages, measure boot sequences and attest measurements.

Part 4 - Supporting routines: This part consists only of C code that represents the algorithms and methods used by part 3 Commands. This part is separated into two parts; part 4 Supporting routines and part 4 Supporting routines code [15].

The TPM commands was developed to provide all functions necessary for TPM security use cases, but to keep the cost of TPM down, some of the unnecessary components was moved out the chip to the software domain. Consequently, the specifications were hard to understand, as it was ambiguous how commands should be used when lots of logic was left to software part and hence to the developer. On the other hand, hardware security was not an easy topic to start off. Therefore, it was required to go back through some earlier version of TPM to understand this technology and the history of its development.

4.2 History of TPM

The first deployed TPM was TPM 1.1b, in 2003 with the capability of RSA key generation, storage, secure authorization, and device-health attestation as well as the TPM 1.1b includes Platform Configuration Registers (PCRs), were introduced for measurements of the integrity of a system's boot-sequence. The TPM 1.1b was incompatible at the hardware level, which leads to different interfaces that require different drivers. TPM 1.2 emerges on 2005 with several releases that improves TPM with a standard software interface and specifications that protect against dictionary attacks and new commands for direct anonymous attestation (DAA) and a method for authorization and administrative functions. Furthermore, TPM 1.2 has a small amount of NVRAM (about 2KB) and migratable keys can be copied from one TPM to another. Due to cryptographic weaknesses of SHA-1, TPM 2.0 specification was released including enhancement in the former version including: algorithm agility with support for new algorithms (e.g. SHA-256, ECC, ECDH), support for more PCR registers, support for enhanced authorization and complex policies. TPM 2.0 also has some unique features that has not been introduced in previous specifications for instance, it has three primary key seeds from which different key hierarchies (platform, storage and endorsement) can be derived as well as endless number of cryptographic keys can be generated from one single seed under one of the hierarchies.

4.3 Foundational Elements

Enabling the TPM begins with provisioning it on the GWen board. The main parts in this procedure are Hierarchies, Entities, Primary Seeds, Endorsement key and Attestation Identity Key. We provide here a brief description of each part in the following subsections.

4.3.1 TPM 2.0 Hierarchies

A hierarchy is a collection of objects that occupies TPM memory and can be referenced directly. Objects in a hierarchy can be primary, permanent and temporary. Primary ones are the root of a tree that its leaves are the temporary objects such as keys. TPM 2.0 has four hierarchies: Platform, Storage, Endorsement and Null hierarchy, see Figure 3 below. Each of the hierarchies has a primary seed (primary object, root of the hierarchy) which is embedded in the TPM by the TPM manufacturer.

Each hierarchy is targeted at specific use cases, following is the explanation for hierarchies.

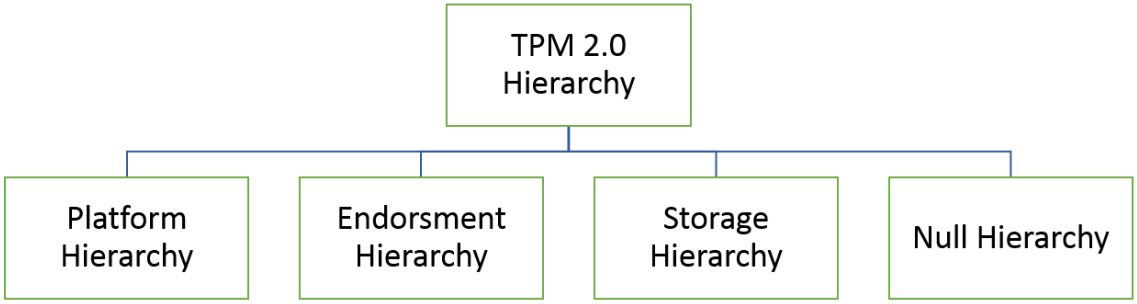

Figure 3 TPM 2.0 Hierarchies

4.3.1.1 Platform Hierarchy

Platform hierarchy is used by platform Original Equipment Manufacturer (OEM) to protect the update mechanism of the root of trust of the platform. The platform hierarchy is enabled by default but TPM provides the ability to enable/disable it through *phEnable* flag using `TPM2_HierarchyControl` command.

This hierarchy needs to be provisioned in order to meets required security standard for data storage and safety development environment. This hierarchy also plays roles in supporting the integrity of the Platform Root of Trust. Therefore, the integrity of the platform with root of trust is the consequence of this supporting.

4.3.1.2 Endorsement Hierarchy

Endorsement hierarchy is the privacy-sensitive tree and is under the control of a privacy administrator, who might be the end user. It is the hierarchy of choice when the user has privacy concerns, but the owner can disable the Endorsement hierarchy while still using the Storage hierarchy for TPM applications.

Similar to Platform hierarchies, the Endorsement hierarchy is enabled by default and can be disabled by clearing the *ehEnable* flag using `TPM2_HierarchyControl` command.

By using Endorsement hierarchy, TPM and platform vendors can prove that primary objects in this hierarchy are trustable because they are constrained to authentic TPM, which in turn is deployed on an authentic platform. Therefore, controlling endorsement hierarchy will prevent using the TPM for attestation without the approval of the device's owner. [16]

4.3.1.3 Storage Hierarchy

This hierarchy is used by the platform owner and it is analogous to the only existing hierarchy in legacy TPM (i.e. TPM 1.2 storage hierarchy). The Storage hierarchy is enabled by default but TPM provides the ability to enable or disable it through *shEnable* flag using `TPM2_HierarchyControl` command. This hierarchy is intended for non-privacy-sensitive operations unlike Endorsement hierarchy which typically addresses the privacy-sensitive credentials [17].

4.3.1.4 Null Hierarchy

Null hierarchy (or Ephemeral hierarchy) is similar to other hierarchies, however, it cannot be disabled. The seed of the Null hierarchy is not persistent and upon each TPM reboot, a new seed (under Null hierarchy) with different value is generated. So, new primary objects can be created from this seed. Keys that have public and private parts should typically be loaded under Null hierarchy for two reasons; first it is always enabled, and second the authorization is always zero-length password.

4.3.2 Primary Seed

Primary seed is a large random number which is generated by the TPM and can never be used outside of the TPM. It is used as a unique source of hardware entropy for generating symmetric and asymmetric keys. In order to encrypt private data, sign or perform an authentication, creating symmetric or asymmetric keys is required. Such keys can be derived from primary objects, which in turn are created from primary seed by using Key Derivation Function (KDF). There are three persistent primary seeds for platform, endorsement and storage hierarchies, as well as, one non-persistent primary seed for the Null hierarchy. The seeds are embedded inside the TPM during manufacturing.

4.3.3 Endorsement Key (EK)

The EK is a cryptographic key which allows particular TPM device to prove its legitimacy. This key is a fundamental component of the TPM and it is derived from the Endorsement seed that is embedded under the Endorsement hierarchy in the tamper resistant non-volatile memory by TPM manufacturer.

EK is a 2048-bit RSA key pair unique for each TPM and mainly is used to decrypt certificates and pass passwords into the TPM during provisioning. The private part of EK will never be revealed outside the TPM [13]. The TCG Infrastructure work group has

defined several templates that can be used when generating endorsement primary keys. For instance, an RSA template, which uses RSA 2048, SHA-256, and AES-128. Another template is the ECC template, which uses ECC with the NIST P256 curve, SHA-256, and AES-128. The EK should have the following attributes: *fixtTPM* and *fixedParent* set to true to ensure that the EK will never be duplicated. The EK should also have restricted and decrypt attributes to indicate a storage key as well as *userWithAuth* and *adminWithPolicy* attributes denoting that a policy will be used instead of a password. The EK has an EK certificate to distinguish and authenticate device identity for an individual TPM.

4.3.4 Attestation Identity Key (AIK)

The AIK is a special purpose TPM key, which is used to provide platform authentication based on the attestation capability of the TPM. AIK is an RSA 2048-bit key size, generated under the Endorsement hierarchy and signed by the EK of the TPM. The AIK is designed to sign data that is exclusively generated by the TPM. Therefore, AIK is used to sign PCR contents[2] and report these signatures to a remote authority. The main purpose of the AIK is to be used in a TPM Attestation which is part of the secure boot process, but it can also be used in authentication of a platform that hosts a TPM to a remote server.

[2] PCRs store signatures of OS's different software components. These signatures are used in secure boot.

5. Implementation

In this section, we describe the specification of the target device i.e. Edge Gateway GWen board, its booting sequence and a brief description of the secure boot on Zynq-devices. We provide also detailed description of the adopted method used to communicate with the SLB 9670 TPM 2.0 chip on GWen board and how commands are used to achieve key generation, encryption/decryption, signing messages, create certificates, etc.

5.1 The Edge Gateway GWen Board

5.1.1 GWen Specifications

Th GWen is an Ethernet edge gateway assembled and developed by HMS Industrial Network AB, **Figure 4**. The device runs a Linux distribution for embedded devices and as shown in **Figure 4** below has a USB-micro cable and USB-serial adapter connected to the PC. The GWen is also connected to the power supply with 12V or 24V (<1A).

Figure 4 Linux-based Edge Gateway GWen from HMS AB

The board of the gateway has the following specifications:

- Xilinx Zynq 7020 SoC (dual core ARM Cortex A9 + FPGA)
- 128 MB DDR3
- 512 MB NAND flash (main storage and boot medium)
- Infineon SLB 9670 TPM (TPM 2.0 on SPI interface)
- Micro SD card slot
- 4 Ethernet ports
- Internal 3.3V UART for console access

5.1.2 GWen Setup and Configurations

The board has a static IP address: 192.168.200.200, and once the board is connected to the PC, a USB Ethernet Gadget driver will be installed. The connection between the PC and GWen board can be established using Linux terminal with the root role without password. The communication takes place via USB-micro cable and using the Secure Shell protocol (SSH) via port: 22.

5.1.3 Booting Sequence

The Booting process of the Zynq-7020 FPGA SoC can be illustrated passing through the following stages:

- The BootROM (i.e. non-accessible internal hardwired code)
- The First Stage BootLoader (FSBL)
- The FPGA configuration (bitstream)
- The Second stage bootloader (known as U-boot)
- The Operating System (i.e. Linux)

The booting sequence process is depicted in **Figure 5** passing through these five stages.

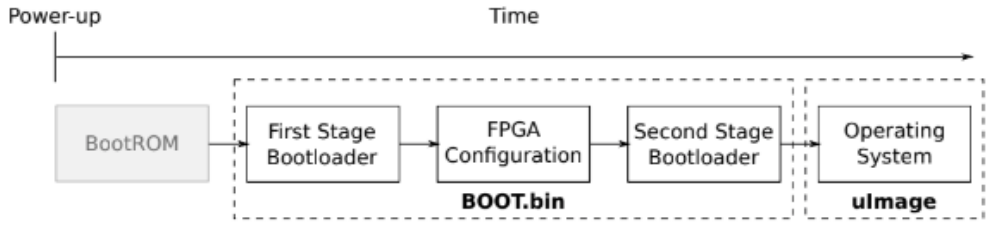

Figure 5 Booting Sequence overview in Xilinx Zynq-7020[19]

Figure 6 shows the building blocks used for booting Linux on Zynq-7020 from FSBL and U-Boot that reside in an external flash.

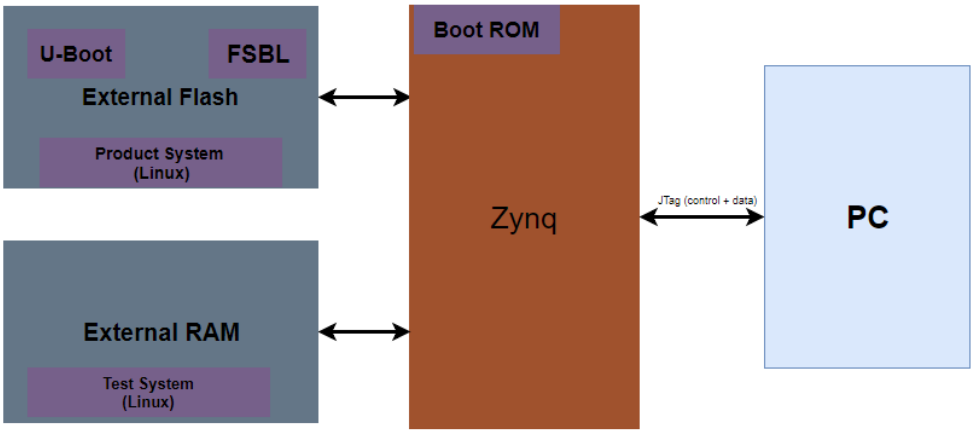

Figure 6 Building blocks used for booting Zynq-7020 SoC

5.1.4 Secure Boot of the Zynq-7020 FPGA SoC

It is recommended to secure boot Zynq devices using the method described in [18], as the effort and cost for such an operation is trivial. The process is based on public and private cryptographic algorithms and does not require programmable logic resources. According to [18], the booting time with a secure Linux system, and with a non-secure system is approximately the same.

The following steps summarize the secure boot process on Zynq-7020 FPGA SoC [19]:

1. CPU executes instruction in BootROM and configure the peripherals to fetch FSBL from NVM (i.e. NAND flash).
2. BootROM copies FSBL from NVM to OCM. During the copying, FSBL is decrypted and authenticated using AES/HMAC core.
3. If authentication is successful, the control is handed to the FSBL and executed internally to initialize GPIOs, DDR controller and the FPGA.
4. FSBL will control the decryption and authentication of the bitstream and second stage bootloader.
5. The bitstream is loaded next to the FPGA configuration memory and is also decrypted and authenticated while copying.
6. If authentication is success, the second stage bootloader (i.e. U-boot) is decrypted and authenticated, otherwise the FPGA configuration will not be activated.
7. The decrypted U-boot will be stored on the external DDR memory and if authenticated, the control will be handed off to the U-boot.
8. The U-boot then decrypts and authenticates the kernel image and reads kernel image headers and jumps to these header addresses and loads the OS.

Throughout the above steps the secure chain-of-trust can be established starting from the BootROM until the OS is up and running. During the process, if any partition is not authenticated and verified successfully, the system will run into a secure lockdown mode. On [18] Xilinx has provided extensive description in detail about the realization of building and booting a secure system and creating secure boot images that can be used to boot Zynq-7020 FPGA SoC build-in devices securely.

5.2 TPM Accessibility

There are many ways and different API levels to access and communicate with TPM 1.2 in Windows and Linux environments like TPM 1.2 TrouserS from IBM. Although software that support TPM 2.0 specifications is still under development, we found on [20] a software release by Infineon called ELTT2 for the SLB 9670 TPM 2.0 chip. Another open source for TPM 2.0 Stack and Tools from Intel has been found on Github on [25]. In this section, we will cover these two software highlighting some of their pros and cons with more details on the adopted software.

5.2.1 Embedded Linux TPM Toolbox 2 (ELTT2)

The GWen board includes a TPM 2.0 chip from Infineon referred to as SLB 9670 TPM 2.0. The communication between the GWen and the TPM uses SPI interface to send commands and receive responses. The driver of the TPM through which the communication takes place is already installed on GWen's Linux image by HMS under the path */dev/tpm0*. To utilize this communication link to pass in commands to the TPM, we used an Embedded Linux TPM Toolbox 2 (ELTT2), an open source implementation from Infineon can be downloaded from [20] and is used to communicate with the TPM on Linux-based embedded devices. The ELTT2 is an executable program written in C consists of one single file and its header. The intention is to perform testing, diagnosis and some TPM basic operations using commands that are provided by TCG under TPM 2.0 specifications **part 3 - Commands**. To realize this, we compiled the ELTT2 source file using GWen's Software Development Kit (SDK), that was provided to us by HMS together with the GWen Board. After compilation, we transferred the generated executable output file to the GWen through SSH interface under the root role. Using the ELTT2 program, we could execute some of the basic TPM 2.0 commands which do neither require any authorization nor any additional command line parameters. The ELTT2 has list of TPM commands to perform for instance: getting random numbers, generating sequences of SHA-1 or SHA-256, reading the TPM internal clock, reading PCR register values, etc.

The ELTT2 communicates with the TPM device through TPM driver layer directly, which means all commands must first be converted into low-level format i.e. hexadecimal or binary. As mentioned previously, ELTT2 program was intended for testing the TPM with some simple commands but does not support TPM's more advanced and fundamental functions such as keys generation, storing credentials, taking ownership of the TPM, performing attestation and cryptographic operations like encryption/decryption, signing messages, generating endorsement key and certificates etc. The ELTT2 requires further development in order to support the complete list of TPM commands and due to this shortcoming, the ELTT2 was only used to ensure that our TPM receive commands manipulate them and sends back responses accordingly.

5.2.2 TPM 2.0 Software Stack (TSS 2.0)

The TPM 2.0 TSS is specifications set by TCG implemented to provide a standard API for accessing and utilizing functions of the TPM. This TSS stack is the main specification that defines a TPM with protected storage and protected capabilities to facilitate developing interoperable tamper-resistant client applications and programs independent of the low-level interface details.

The TSS can be built on both Windows and Linux. Using the TSS stack, makes it possible for developers to write applications that can communicate with the TPM on different API levels without a comprehensive knowledge of the low-level details of the TPM structure.

Moreover, the TSS provides an abstraction of the differences in hardware so that applications written by developers can work with TPMs regardless of the hardware or the Operating System.

Another goal for developing the TSS is to make it possible for applications to communicate with the TPM either locally or remotely. The TSS consists of the following TSS layers, as shown in **Figure 7**, from the lowest level of abstraction to the highest:

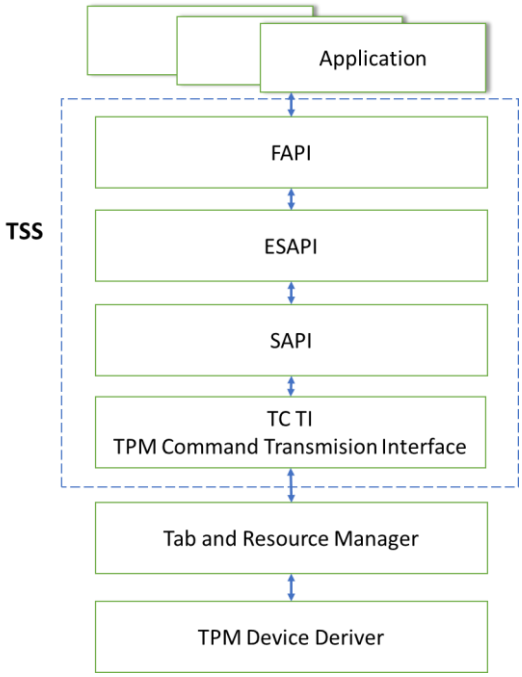

Figure 7 TCG Software Stack TSS 2.0

TPM Device Driver: this is the lowest layer that handles the physical data transmission to and from the TPM. Developing application to this layer is possible, however, it would be hard as it is similar to programming in binary.

TPM Access Broker (TAB) and Resource Manager (RM): TAB and RM are two different components combined in one single layer. The RM swaps TPM objects and sessions in and out of TPM due to the limited amount of TPM memory. The TAB controls access to the TPM by synchronizing multi processes. This allows multiple applications from accessing and utilizing the same TPM chip concurrently in a slice time fashion similar to a PC's virtual memory manager. Both the TAB and RM are optional components, so both might not exist in some embedded systems that don't have multiprocessing units.

TPM Command Transmission Interface (TCTI): an API that provides a standard interface to transmit TPM 2.0 commands and receive responses. Two TCTI implementations are provided currently in this stack: a) libtss2-tcti-device, which is used for direct access to the TPM through the Linux kernel driver. b) libtss2-tcti-mssim, which implements the protocol that are used by the software TPM2 simulator from Microsoft.

Application written to TCTI layer can send byte streams and receive byte responses. TCTI maybe the interface between TAB &RM layer and the device driver layer and will then be considered the interface of multiple layers in the stack. Writing applications to this layer is possible but is similar to writing in assembly language.

System API (SAPI): This API maps TPM2 commands that are documented in TPM2 specification part 3 -Commands. This layer provides the access to all TPM functionalities but requires a high level of knowledge and experience to use them. Writing applications to this layer similar to writing in C language.

Enhanced System API (ESAPI): The ESAPI layer intended to reduce the programming complexity of applications that aim at sending commands to the TPM which require cryptographic operations to be performed on the passed data to and from the TPM. In particular, ESAPI provides support for applications to perform HMAC operations, encryption/decryption, TPM command audit and TPM policy operations. In addition to that, ESAPI provides context and object management as well. The ESAPI is similar to SAPI with less complexity, however, it still requires in-depth of the TPM 2.0 knowledge. Writing application to this layer is similar to programming in C++. This layer is still under development [21].

Feature API (FAPI): This layer is a very high-level API, intended to allow 80% of programmers to write easily programs that talk to the TPM without lots of knowledge of the low-level command templates and TPM structure. The other 20% of programmers might need to supplement this set of APIs with the ESAPI or SAPI layers. FAPI implementations contains profiles, which are pre-created configuration files that define the most common choices like algorithms, key size, mode of encryption, etc. For instance: P_RSA2048SHA256 profile uses the RSA 2048-bit asymmetric keys using PKCS#1 version 1.5 for signing scheme, SHA-256 for the hash algorithm, and the AES-128 with CFB mode for asymmetric encryption.

Writing to this layer is similar to writing in high level languages like Java or C#. This layer API is also still under development [22].

5.2.3 Building and Installing TSS

An open source software implementation of the TSS is already existed on GitHub on [22] with a repository named **Linux TPM2 & TSS2 Software** including the TSS stack and two other packages as follows:

- **tpm2-tss:** An OSS implementation of the TCG TPM2 Software Stack (TSS2)
- **tpm2-tools:** The source repository for the TPM 2 tools including all TPM 2.0 commands.
- **tpm2-abrmd:** TPM2 Access Broker & Resource Management Daemon implementing the TCG specifications.

Infineon has published an Application Note on [23] to demonstrate how Infineon OPTIGA™ TPM 2.0 can be used on a Raspberry Pi® 3. The method gives an idea about how to patch, configure and install Linux kernel on a Raspberry Pi® 3 and inject the TSS into the kernel or compile and install it separately. However, the method doesn't use the **tpm2-abrmd** project as it has not been developed yet when the document was published. We downloaded and installed the TPM 2 & TSS Software repository including the dependencies and the three aforementioned projects. We could successfully compile the projects using ***gcc tool*** according to the instructions given on the **INSTALL.md** file in each project. The projects are associated and should be build one inside the other in the following order: **1) tpm2-tss** 2) **tpm2-abrmd** 3) **tpm2-tools.** These steps are described in detail in the Appendix.

5.2.4 Using TPM 2.0 Simulator

The TPM library specification includes reference code to implement a software TPM 2.0 simulator. Microsoft has provided the binary download for that implementation as simulator program for Windows environment. IBM has re-packaged that code with some Makefiles so the Microsoft code can be built and executed on Linux systems. The simulator is based on the source code donated by Microsoft and according to TPM specifications **parts 3 - Commands** and **part 4 - Supporting routine codes**, with additional files to complete the implementation.

In order to test the TSS stack and execute some of the TPM commands, we used one of IBM simulators called IBM Software TPM 2.0 **ibmtpm119**, and can be downloaded from [22]. After downloading, compiling and building the **ibmtpm119**, we started the simulator by navigating into the **src** folder inside the simulator and execute the following command:

```
./tpm_server &
```

The simulator will start and by default, accept connections on the localhost, TCP ports 2321 and 2322. To start Rsource Manager RM and TPM Access Broker TAB, we execute the following command:

```
sudo -u tss tpm2-abrmd --tcti="mssim:host=localhost,port=2321"
```

The simulator will accept the client and to execute any TPM 2.0 commands we can start new terminal, for instance we can use TPM's RNG to get random numbers:

```
tpm2_getrandom 4
```

The TPM responses with the following output:

```
0x91 0xBA 0x28 0xAC
```

After testing some of TPM command on the simulator, we tried to use the Linux TPM2 & TSS2 Software and compile it using GWen's SDK environment setup script which was

provided to us by HMS. Unfortunately, the cross-compilation after sourcing our SDK didn't work and results in lots of compilation error. HMS staff fixed this issue by injecting the TSS with its tools and dependencies into new Linux image, and request us to download and deploy the new image on GWen board. Once the new image is successfully installed on GWen, we were able to execute some of TPM 2.0 commands directly as TSS became part of the Linux kernel and existed as a separate module in the new Linux image.

6. Results

Using the TSS 2.0 embedded in the Linux image of the GWen makes it possible to run TPM 2.0 commands directly from command line. In the following sections, we describe key generation and key hierarchies we followed when generating TPM keys, as well as our result for some use-cases.

6.1 Key Generation

Transferring data between two peers securely can be done using asymmetric cryptography, the method upon which a lot of applications are based for secure data exchange. In such cases, the public key can be distributed publicly (everyone can access it including the attacker) and can be used by anyone who wants to send encrypted data to the owner of the private key. In principle, asymmetric cryptographic algorithms, takes significant amount of CPU cycles to do the required mathematical calculations and therefore, only small portions of data are encrypted using public-key cryptographic algorithms.

A small amount of data like a symmetric key can be used in a later step for bulk encryption or configuration data for IoT applications. In this section we will show how a file can be encrypted and decrypted using the SLB 6970 TPM 2.0 on the GWen board. Basically, the encryption can be done on a remote device or on a server that wants to send confidential data to the GWen. The private key is kept private and will never be disclosed to the outside world. Therefore, the encryption (i.e. using TPM's public key) can be done on non-secure hardware as well because, in the encryption process, there is no data involved that can be used to get any information about the private key. For this reason, using the TPM as an encryption engine is possible, but it is not necessary. The aim of demonstrating the encryption/decryption using the TPM is to provide a use-case for later usage.

6.2 Key Hierarchy

In TPM 2.0, each of the persistent and non-persistent hierarchies (platform, storage, endorsement and Null) have parent and child keys. All parent keys are storage keys (i.e encryption or master keys), which can wrap (encrypt) child keys. The main purpose of having parent (storage) keys is to protect their child, offering secrecy and integrity when the child key is migrated outside the secure hardware boundary of the TPM.

Therefore, storage keys are restricted and can’t be used for general decryption. Child keys can become storage keys (parent), if they are used to derive new child keys. The ultimate child key at the bottom is a non-storage key, in which case it is also called a leaf key. [11]

Unlike TPM 1.2 which has only one hierarchy with no seeds, TPM 2.0 has one Primary Seed, from which three primary seeds of each hierarchy will be generated: Endorsement Primary Seed (EPS), Storage Primary Seed (SPS) and Platform Primary Seed (PPS). The

EPS is the root of Endorsement hierarchy which will generate the Endorsement key EK that is used to verify the platform identity. The PPS is the root of Platform hierarchy that will generate the platform primary key which is used to control the firmware of the platform. The SPS is the root of Storage hierarchy, which will generate Platform hierarchy controlled by the platform owners. The SPS also generates keys which serve as storage root key SRK for normal OS and its application [37]. **Figure 8** depicts the key hierarchies in TPM 2.0 and the relation between parent and child keys we generated in our solution.

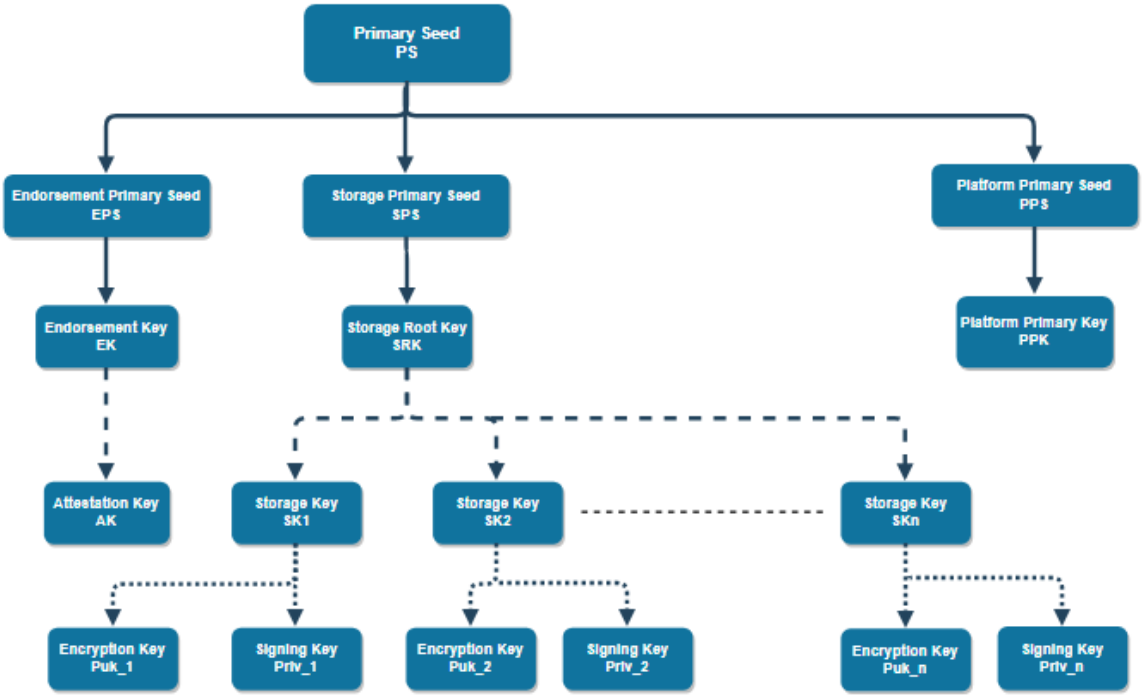

Figure 8 Key hierarchy in TPM 2.0

6.3 TPM 2.0 as Cryptographic Engine

In TPM 2.0, there existed 3 main hierarchies, each of which includes primary seed from which a parent key can be created which in turn can wrap up one or many children keys. Primary keys are the root keys in each hierarchy because they have no parent keys. In this section, we describe, the creation and usage of primary keys and how to derive child keys to be used for encryption/decryption. The key generator is some software that is provisioning the TPM, by sending commands and receiving responses. Typically, it is manufacturer or platform provisioning that performs primary key generation, so, the end user can use primary keys, but theoretically he should not create them. The commands which are needed to generate primary keys and perform the encryption/decryption use-cases are the following:

1- `tpm2_createprimary`
2- `tpm2_create`

3- `tpm2_loadexternal`
4- `tpm2_rsaencrypt`
5- `tpm2_load`
6- `tpm2_rsadecrypt`

The complete template, the required parameters and other details about each command can be found on [24].

6.3.1 Encryption Using RSA-2048 Key

6.3.1.1 Create TPM 2.0 Primary Object

The command `tpm2_createprimary` will create a primary object under the endorsement hierarchy:

```
tpm2_createprimary -H e -g sha256 -G rsa -C primary.ctx
```

The command `tpm2_createprimary` is used to create and load a primary object, which will be created under one of the primary seeds. To specify the seed under which hierarchy, the command takes the hierarchy type by passing ***e, p, o*** or ***n*** after **-H** for *endorsement*, *platform*, *owner* or *null* hierarchies respectively. The command line option -g indicates the hashing algorithm (e.g. SHA-1 or SHA-256) and the option -G defines the algorithm associated with that object (e.g. RSA or ECC). The TPM takes these parameters and returns a blob containing the generated primary object which means the last parameter (-C with a *fileName*) is required to tell the TPM where to save that primary object. See **Figure 9**.

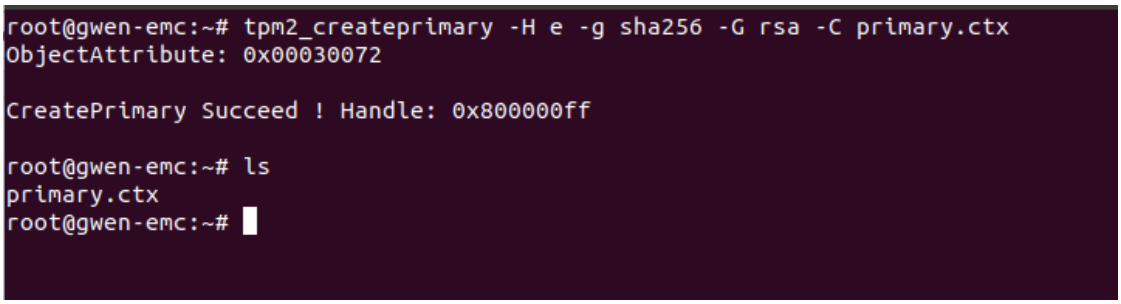

Figure 9 Creating Primary Object Under Endorsement Hierarchy

Table 1 shows the input and output parameters which are used as arguments to the `tpm2_createprimary` command.

Table 1 Parameters of tpm2_createprimary Command

Input parameters		Output parameter	
-H	e	-C	primary.ctx
-g	sha256		
-G	rsa		

6.3.1.2 Create RSA Key Pair

The command `tpm2_create` command will create an object that can later be loaded into the TPM using the command tpm2_load:

```
tpm2_create -g sha256 -G rsa -c primary.ctx -u primary.pub -r primary.priv
```

This command takes the hashing algorithm, encryption algorithm and the primary object as input parameters to create an encryption and decryption key, and returns both the public and private portion of that key. Both parts are stored as files (i.e. *primary.pub* and *primary.priv)* in the folder where the command was executed. The private part (i.e. *primary.priv*) is encrypted with the public part of the parent key and can therefore only be decrypted and used inside the TPM, and thus will be kept secret and will never be revealed to the outside world. **Figure 10** depicts the execution of `tpm2_create` command on GWen with some response from the TPM.

```
root@gwen-emc:~# tpm2_create -g sha256 -G rsa -c primary.ctx -u primary.pub -r p
rimary.priv
algorithm:
  value: sha256
  raw: 0xb
attributes:
  value: fixedtpm|fixedparent|sensitivedataorigin|userwithauth|decrypt|sign
  raw: 0x60072
type:
  value: rsa
  raw: 0x1
  rsa: 9ae0739d5588451e567d12119d3fca7c249b81871d16cbb1a92949c3e3e192709a43dbe39
24b4b943707e1d9ac6eb22e2f7d57fab32cb63aa33809bd5fddb546f931ea9b8368cc969d5eb29ac
dfd81354473ed0a7a4c6f3be30fa83ffb57b9fb7b3b15f1b9ccc9ec551e7ff58886fac5a1f9c0ed4
25312c1b188b5f185df38d3ba334c584293743e60a558a081204d3e73ced85869351f5797744615a
0b9129da223bbadad905e962a53a0dc69b9151f6cd12ad6bcd277433102d8cbb4bea9c1e42eea72d
370c6411d81b40287ef8d5f19f3695ae3b7a440bddf52593652f6fabb52c8566ca9f23607820b840
e47b445e9509e87891b83666d5b629c275027bf
root@gwen-emc:~# ls
primary.ctx   primary.priv  primary.pub
root@gwen-emc:~#
```

Figure 10 Creating RSA Key Pair

It is important to notice that the *primary.ctx* object that was used as an output parameter in the previous command was used as an input parameter in this command. From the output shown in the picture above, the primary object i.e. the parent key, and the public and private portion of the child key are now created and stored inside the GWen board. The input and output parameters of `tpm2_create` command is shown in Table 2.

Table 2 Parameters of tpm2_create Command

Input parameters		**Output Parameter**	
-g	sha256	-u	primary.pub
-G	rsa	-r	primary.priv
-c	primary.ctx		

6.3.1.3 Loading Non-protected Objects into the TPM

The command `tpm2_loadexternal` will load objects indicated as "non-protected objects" such as public part of an RSA key into the TPM:

```
tpm2_loadexternal -H n -u primary.pub -C rsaencrypt_key.ctx
```

Since the TPM has limited amount of NV memory, the keys are stored outside the TPM in GWen's main storage. The public part is unencrypted, but the private part is encrypted with the public part of the parent key. This is one of the new features existed in TPM 2.0, which does not exist in TPM 1.2. That is the ability for TPM 2.0 chips to create and store an unlimited number of keys from single seed with no security drawbacks at all. The only disadvantage is that every key which shall be used for a cryptographic operation must be loaded first into the TPM before the actual operation can take place. For frequently used keys, this mechanism can be simplified by making use of the persistent storage inside the TPM. Since the performance is not considered for this sample use case, the keys are not stored permanently inside the TPM and must therefore be loaded in advance. As seen in **Figure 11**, the null hierarchy is defined and the public key *primary.pub* is loaded into the TPM.

```
root@gwen-emc:~# ls
primary.ctx   primary.priv  primary.pub
root@gwen-emc:~# tpm2_loadexternal -H n -u primary.pub -C rsaencrypt_key.ctx
root@gwen-emc:~# ls
primary.ctx         primary.priv        primary.pub         rsaencrypt_key.ctx
root@gwen-emc:~#
```

Figure 11 Loading Public Key to the TPM

The TPM will return a reference file *rsaencrypt_key.ctx*, which will be used for the actual encryption process in the next command. Note that the null hierarchy has its own unique

seed, from which both primary and child keys can be derived. However, neither the seed nor the primary keys can be persistent in TPM NV memory. Each time the TPM is restarted, a new seed is created, which in turn means new parent and child keys can be re-generated out of that new seed. Therefore, keys in this hierarchy are erased upon each TPM reset. The input and output parameters of `tpm2_loadexteranl` command is shown in Table 3.

Table 3 Parameters of tpm2_loadexteranl

Input Parameters		**Output Parameter**	
-H	n	-C	rsaencrypt_key.ctx
-u	primary.pub		

6.3.1.4 Asymmetric Encryption

The command `tpm2_rsaencrypt`, performs an RSA encryption operation using the *rsaencrypt_key.ctx* object:

```
tpm2_rsaencrypt -c rsaencrypt_key.ctx data.txt -o cipher.enc
```

To perform the encryption using the TPM, we use a 12-byte size text file as the plaintext input data (*data.txt*) and we pass a filename for the data that shall be encrypted (*cipher.enc*), see **Figure 12**.

```
root@gwen-emc:~# ls
data.txt            primary.priv        rsaencrypt_key.ctx
primary.ctx         primary.pub
root@gwen-emc:~# tpm2_rsaencrypt -c primary.ctx -c rsaencrypt_key.ctx data.txt -o cipher.enc
root@gwen-emc:~# ls
cipher.enc          primary.ctx         primary.pub
data.txt            primary.priv        rsaencrypt_key.ctx
root@gwen-emc:~#
```

Figure 12 Encrypting Operation

The encryption yields *cipher.enc* file, which is an encryption of the plaintext data.txt file. The content of the file *data.txt* can be credentials or any critical secrets that require to be protected by TPM using encryption. Only the owner of the private key would be able to decrypt them, which in this case is the TPM. The input and output parameters of `tpm2_rsaencrypt` command is shown in Table 4 below.

Table 4 Parameters of tpm2_rsaencrypt

Input parameters		**Output Parameter**	
-c	rsaencrypt_key.ctx	-o	cipher.enc
	data.txt		

6.3.2 Decryption Using RSA-2048 Key

6.3.2.1 Loading Protected Objects into the TPM

The command `tpm2_load` is used to load protected objects like the private part of an RSA key, and possibly the public part (optional), into the TPM. It has the following syntax:

```
tpm2_load -c primary.ctx -u primary.pub -r primary.priv -C rsadecrypt_key.ctx
```

As can be seen, the primary object *primary.ctx* which was used for creating public and private key, the public key *primary.pub* and the private key *primary.priv* are passed as command line arguments. These objects will be loaded to the TPM that will return a key context *rsadecrypt_key.ctx*, which is needed for referencing the private key inside the TPM, see **Figure 13** below.

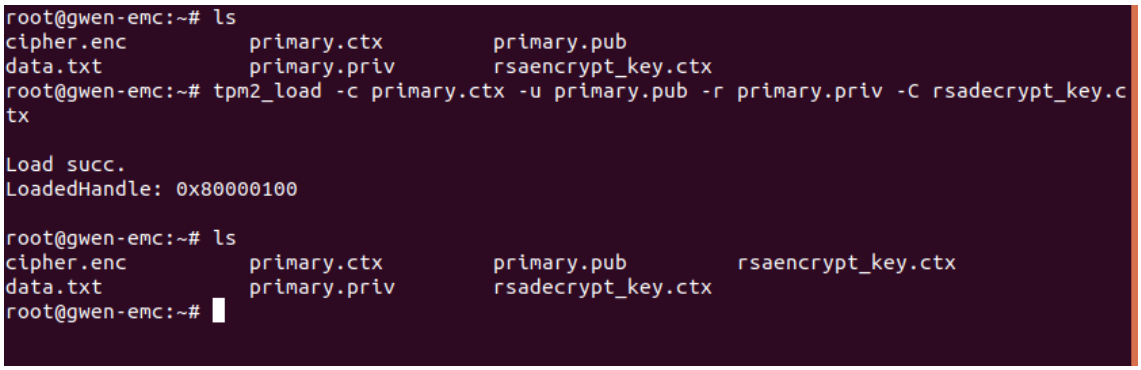

Figure 13 Loading Public and Private Keys into the TPM

The *rsadecrypt_key.ctx* is the object that will be used for the actual encryption process in the subsequent command `tpm2_rsadecrypt`. The input and output parameters of `tpm2_load` command is shown in Table 5.

Table 5 Parameters of tpm2_load Command

Input parameters		**Output Parameter**	
-c	primary.ctx	-C	rsadecrypt_key.ctx
-u	primary.pub		
-r	primary.priv		

6.3.2.2 Asymmetric Decryption

The command `tpm2_rsadecrypt` performs an RSA decryption operation using the TPM:

```
tpm2_rsadecrypt -c rsadecrypt_key.ctx -I cipher.enc -o output.txt
```

The decryption command takes the previously created encrypted file *cipher.enc* and a file descriptor for the output data *output.txt* as command line arguments, see **Figure 14**.

Figure 14 Decryption Operation Using the Private Key

By executing this command TPM will decrypt the *cipher.enc* file and save the output data in *output.txt* file. The content of *output.txt* file is identical to *data.txt* file which indicates that the decryption process has been done successfully. The input and output parameters of `tpm2_rsadecrypt` command is shown in Table 6.

Table 6 Parameters of tpm2_rsadecrypt Command

Input parameters		**Output Parameter**	
-c	rsadecrypt_key.ctx	-o	output.txt
-I	cipher.enc		

Further commands and use-cases with authorization (using passwords) are added to the Appendix section at the bottom of this report including encryption/decryption, signing/verification and making objects persistence in TPM's NV memory.

6.4 Generating EK and AIK keys

Before generating attestation AIK keys, an endorsement EK must be created first in order to sign the AIK. The following command are used to generate the EK:

```
tpm2_getpubek -H 0x81010000 -g 0x01 -f ek.pub
```

This command will generate a new RSA key, store it in the NVRAM of the TPM with a handle number 0x8101000 and export the public portion in a file named ek.pub. Similarly, we can generate an AIK:

```
tpm2_getpubak -E 0x81010000 -k 0x81010010 -f ak.pub -n ak.name
```

This command will create an AIK under the endorsement hierarchy and so it needs to be generated using an EK (which in this case is stored at handle 0x81010000). This new AIK key will be stored in the device NVRAM with handle 0x81010010. The public part is exported in ak.pub. The ak.name contains the cryptographic secure name of the key.

6.5 System Comparison to Related Works

In [47], a study by Aaraj N. et. al. demonstrates software-based TPM (SW-TPM) that implements TPM commands and functions using the TPM Software Stack TSS 1.2 (TrouSerS) and TPM emulator. The platform used for the SW-TPM is a battery-powered handheld embedded device Sharp Zaurus PDA and the TPM services are adapted from TPM emulator which was implemented as Linux kernel module running on x86 Linux PC. The study evaluates the energy overhead and execution time required by each TPM command, including the secure boot of the Linux OS, secure file storage, secure VoIP client, and the secure web browser. The study shows that for most TPM commands, the overhead is due to using 2048-bit RSA operations, and that Elliptic Curve Cryptography (ECC) can give a significantly execution time reduction with an average of 6.51X for individual TPM commands, and an average of 10.25X for applications. Therefore, the study proposes the use of ECC-based SW-TPMs instead especially in resource-constrained embedded systems.

This study, however, underlines that SW-TPM is not completely equivalent to a conventional TPM chip in terms of protection against physical and hardware attacks, and recommends executing SW-TPM within a protected or isolated execution domains that are supported by embedded CPUs such as ARM TrustZone. In contrast to this study, we used the hardware chip SLB 9670 TPM 2.0 module in our work as a secure key generator and key storage, and we utilized the latest version of the TSS 2.0 that was designed specifically according to TCG specification 2.0 with new features such as support for various number of cryptographic algorithms and hash functions that didn’t exist in legacy TSS 1.2. Besides, the study in [47] uses SW-TPM that is equivalent to HW-TPM 1.2 whereas in our work, we used hardware TPM 2.0 module. The following are some differences between TPM 1.2 and TPM 2.0:

- In TPM 1.2, everything is under the control of a single “owner” including authorization, with an RSA 2048b EK for signing and attestation, and a single RSA 2048b SRK for encryption. In other words, the "owner" of the TPM has control for signing, attestation and encryption functions.
- TPM 2.0 has the same functionalities, but with more sophisticated authorizations, policies and access control. TPM 2.0 was specifically designed to make discovery and management less cumbersome than in legacy TPM 1.2.
- In contrast to the “owner” in TPM 1.2 who controls everything, the control in TPM 2.0 is split into three different hierarchies; the Endorsement Hierarchy (EH), the Storage Hierarchy (SH) and Platform Hierarchy (PH), plus an extra hierarchy known as Null Hierarchy, where each hierarchy has its own unique “owner” for authorization, and thus TPM 2.0 supports 4 different authorizations which would be analogous to the single TPM 1.2 hierarchy “owner”.
- TPM 1.2 can generate only limited number of keys due to insufficient space in its NVRAM. This problem is alleviated in TPM 2.0, which is shipped with single primary seed from which 3 different seeds can be derived; Endorsement, Platform and Storage seeds for Endorsement, Platform and Storage hierarchies respectively. Each seed will be used under the corresponding hierarchy to derive the primary keys (parent keys) that, in turn, are used to derive and encrypt other keys (child keys) under the same hierarchy. This makes it possible for TPM 2.0

to generate endless number of cryptographic keys of different types and sizes, but it then hands in the responsibility of key management to the user instead. Hence, TPM 2.0 can store persistently limited number of primary objects and certificates inside its NVRAM.

The SW-TPM in the study in [47] uses a hash-complemented Mersenne Twister (MT) as random number generator, that's to run the output of MT through SHA-1 function. However, the TPM 2.0 we used in our work, has secure key generation using hardware random-number generator instead of poor key generation by some software which might be breakable as the SHA-1 is now considered completely unsafe.

From the one hand, the study [47] adopted the deployment of ECC due to its small key sizes for offering the same security robustness as RSA, as well as because ECC requires less resources such as processor cycles and energy. In our solution, the version of TSS 2.0 we used was under development and includes incomplete list of TPM commands that are provided by TCG specifications. For instance, it doesn't include TPM commands for ECC key generation or encryption/decryption, and therefore, we applied our evaluation in section *7. Performance Evaluation and Analysis* merely for RSA algorithm. Recently, further releases existed in [25] including OpenSSL engine for TPM 2.0 devices and a PKCS#11 interface for TPM 2.0 hardware, which supports ECC as well as ECDSA signatures.

From the other hand, this study uses SW-TPM implemented in the PDA device, which utilizes another TPM emulator with TSS 1.2 TrouSerS on different platform to provide TPM services such as secure boot for the PDA device in early boot when the SW-TPM is still not available. We believe it is redundant to have one system always running in order for the other system to be able to perform secure boot. The use of hardware TPM chip in such scenarios is a key value to resolve this complexity as well to address some of those security problems, which are discussed earlier in section *2. Related Work*. For Zynq-7000 SoC-based platforms, the support of secure boot is already provided, and it is recommended to be always enabled with almost no extra overhead much faster than measured boot that the TPM is capable of.

7. Performance Evaluation and Analysis

In this section, the system performance evaluation and analysis will be presented. The evaluation process at each step is explained starting from the purpose and goal of the evaluation going through a list of metrics and parameters to be evaluated and ending up with the most significant factors that have the major impact on the experiment.

7.1 Goal and Purpose

The goal of the project is to enable a HRoT on the Edge Linux-based Gateway GWen using the SLB 9670 TPM 2.0 module from Infineon. The goal is to store secret keys securely. Although we have achieved this objective and simply tested the system to see whether or not it works well, it is still insufficient analysis to the entire performance of the system. The goal of this system performance analysis is to quantify the impact of the major factors that affect the system and distinguish the significant effects from the insignificant counter parts. The study mainly focuses on measuring the time response and what would consider as important factors that has a major impact in the performance. In this section, we analyzed the system by setting up the following: system services, metrics, parameters, workload parameters, factors, evaluation techniques, workload, experiments and result analysis.

7.2 Services

The services provided by TPM can be concluded as follows:

1. Generate EKs, AIKs and SRKs and store TPM keys securely inside TPM's NV memory with/without authorization.
2. Generate primary (parent) keys and derive child keys.
3. Perform different cryptographic operations using the generated keys.
4. Authenticate the hosting system securely.
5. Attest the status of the system.
6. Prepare keys in early boot when the disk is still not accessible.

7.3 Performance Metrics

Metrics are selected based on the services that the system can provide. Out of the above listed services, we defined metrics of our system as follows:

- Response time of TPM key generation of different types and for both parent and child keys.
- Response time of cryptographic operations such as: encryption, decryption, signing, verify, etc.

The latency overhead when sending commands to the TPM is the most important metric that the HMS Industrial Network AB would consider. The power consumption and the

memory capacity, from one side, are considered less significant since the GWen board has a steady power supply and doesn't rely on a lithium battery or any other resource-constrained power source. The memory size, from other hand, is also not considered since the TPM chip we used can store persistently up to 21 secret keys in NV memory[3] besides some other Non-Volatile (NV) indexes for storing certificates. So, there will be no need to use the GWen's storage memory to store credentials.

7.4 Factors Selection

Generating keys is the most time-consuming operation on TPM according to [11]. However, since it is possible to generate TPM keys only once and store them persistently inside the TPM, the latency of key generation will be neglected and not selected as a factor in our performance analysis. Instead, we will consider other factors that are time-consuming and will be used frequently, like the type of cryptographic operation; encrypt, decrypt, sign or verify and message size. From the other side, if we perform these operations using persistent or transient objects, this may also affect the response time.

To preserve and maintain the project inside its scope and limitation, we devoted our measurements only to RSA keys, and since encryption keys will not be made persistent because they are public, we will measure the time only with decryption and signing operations. The reason is that only decryption and signing operations use private key, which can made persistent. Thus, we selected three factors as follows:

1. Cryptographic operation type: decryption / signing
2. Message size: a few bytes / 245 bytes[4].
3. Status of the key: transient / persistent.

7.5 Evaluation Techniques

The three techniques for performance evaluation are analytical modeling, simulation, and measurement in a real system [41]. In our experiment, we chose the measurement and analytical modeling techniques to best describe the model. Although the measurement would spend more time and increased the calculations, it becomes more reliable when combine it with analytical modeling. Besides, it is easier to measure the data and evaluate the performance when having a real system compared to a virtual one.

[3] For instance; handles 0x81010000, 0x81010001, 0x81010002, etc. can be used to store persistent objects.
[4] RSA has a maximum message size of 245 bytes. Any message exceeds this size can only be encrypted using symmetric encryption.

7.7 Workload Selection

The workload of measurements can be performing crypto operations on messages of two different sizes using either transient or persistent keys. To vary the workload, the measurement will be performed with repetitions.

7.8 Experiments

A $2^k \times r$ experimental design is used to determine the effect of k factors and r times replications, each of which have two alternatives or levels [41]. It is easy to analyze and help in sorting out factors in the order of impact. In our measurement, we repeat each experiment 5 times. Since we have only 3 factors, k = 3 and r =5, that's $2^3 \times 5 = 40$ experiments.

7.8.1 Measurements

We measured the response time of TPM commands using Linux system command "`time`". **Figure 15** shows the latency of executing `tpm2_rsadecrypt` command with a transient RSA key. The latency is about 1.076 seconds and 1.107 seconds, for decryption of cipher files with sizes 10 and 245 bytes, respectively.

```
root@gwen-emc:~# time tpm2_rsadecrypt -c rsaDecKey.ctx -I cipher10.enc -o recovered_10_bye
.txt

real    0m1.076s
user    0m0.040s
sys     0m0.000s
root@gwen-emc:~# time tpm2_rsadecrypt -c rsaDecKey.ctx -I cipher245.enc -o recovered_245_b
ye.txt

real    0m1.107s
user    0m0.030s
sys     0m0.010s
root@gwen-emc:~#
```

Figure 15 Latency of tpm2_rsadecrypt Command When Decrypting 10- and 245-byte File Size.

Similarly, we did the measurement of response time for signing messages with `tpm2_sign` command, see **Figure 16**.

```
root@gwen-emc:~# time tpm2_sign -c rsaDecKey.ctx -g sha256 -m plain_10_byte.txt -s si
g_10_byte

real    0m1.639s
user    0m0.040s
sys     0m0.000s
root@gwen-emc:~# time tpm2_sign -c rsaDecKey.ctx -g sha256 -m plain_245_byte.txt -s s
ig_245_byte

real    0m1.608s
user    0m0.040s
sys     0m0.000s
root@gwen-emc:~# ls
cipher_10_byte.enc   plain_10_byte.txt    primary.priv         rsaEncKey.ctx
cipher_245_byte.enc  plain_245_byte.txt   primary.pub          sig_10_byte
data.txt             primary.ctx          rsaDecKey.ctx        sig_245_byte
root@gwen-emc:~#
```

Figure 16 Latency of tpm2_sign Command with Files of Sizes 10 and 245 bytes.

We repeated the procedure and measured the time 5 times with only decryption and signing operations and using a transient key on one time and a persistent key another time. To give more accuracy, we applied `tpm2_startup` command between each subsequent command. Table 7 below shows the results of our measurements.

Table 7 Measurement of TPM Response Time Using 3 Factors and with 5 Repetitions

Time Response (*in millisecond*) **using RSA keys**					
		Transient Keys		**Persistent Keys**	
Experiment No.	**Message Size**	*Decryption* *tpm2_rsadecrypt*	*Sign* *tpm2_sign*	*Decryption* *tpm2_rsadecrypt*	*Sign* *tpm2_sign*
1	10 bytes	1076	1639	676	927
	245 bytes	1107	1608	684	911
2	10 bytes	1106	1667	680	925
	245 bytes	1108	1643	683	896
3	10 bytes	1100	1616	687	952
	245 bytes	1104	1589	659	930
4	10 bytes	1103	1635	686	926
	245 bytes	1105	1612	685	902
5	10 bytes	1107	1702	679	922
	245 bytes	1103	1641	668	900

We define three variables: xA, xB and xC as follows:

$$xA = \begin{cases} -1, & if\ operation\ is\ decryption \\ 1, & if\ operation\ is\ signing \end{cases}$$

$$xB = \begin{cases} -1, & if\ message\ size\ is\ 10\ bytes \\ 1, & if\ message\ size\ is\ 245\ bytes \end{cases}$$

$$xC = \begin{cases} -1, & if\ the\ key\ used\ is\ transient \\ 1, & if\ the\ key\ used\ is\ persistent \end{cases}$$

Then the performance y in mA can be regressed on xA (type of cryptographic operation), xB (message size) and xC (status of the key) using a nonlinear regression model:

$$y = \ q0 + qAxA + qBxB + qCxC + qABxAxB + qACxAxC + qBCxBxC + qABCxAxBxC + e$$

Where, e is the experimental error and q's are the effects.

7.8.2 Computation of Effects

The sign table and computation of effects is shown in Table 8, where all results are calculated using Excel.

Table 8 Sign Table for Computation of Effects of Each Factor

	A	B	C	D	E	F	G	H	I	J	K	L	M	N
1	I	A	B	C	AB	AC	BC	ABC	y1	y2	y3	y4	y5	Y_mean
2	1	-1	-1	-1	1	1	1	-1	1076	1106	1100	1103	1107	1098,4
3	1	1	-1	-1	-1	-1	1	1	1639	1667	1616	1635	1702	1651,8
4	1	-1	1	-1	-1	1	-1	1	1107	1108	1104	1105	1103	1105,4
5	1	1	1	-1	1	-1	-1	-1	1608	1643	1589	1612	1641	1618,6
6	1	-1	-1	1	1	-1	-1	1	676	680	687	686	679	681,6
7	1	1	-1	1	-1	1	-1	-1	927	925	952	926	922	930,4
8	1	-1	1	1	-1	-1	1	-1	684	683	659	685	668	675,8
9	1	1	1	1	1	1	1	1	911	896	930	902	900	907,8
10	8669,8	1547,4	-54,6	-2278,6	-57,0	-585,8	-2,2	23,4						Total
11	1083,7	193,4	-6,8	-284,8	-7,1	-73,2	-0,3	2,9						Total/8
12	q0	qA	qB	qC	qAB	qAC	qBC	qABC						
13	1174459,9	37413,2	46,6	81125,3	50,8	5361,9	0,1	8,6						q^2
14	46978395,0	1496529,2	1863,2	3245011,2	2030,6	214476	3,0	342,2	4969508	51947903	9252,4		(q^2*8*5)	SSx
15		30%	0%	65%	0%	4%	0%	0%	100,0	1045,3	0,2			SSx/SST
16	SS0	SSA	SSB	SSC	SSAB	SSAC	SSBC	SSABC	SST	SSY	SSE			
17														
18														
19	1078,7	188,4	-11,8	289,8	12,1	78,2	5,3	2,1						low
20	1088,7	198,4	-1,8	279,8	2,1	68,2	4,7	7,9						high
21	q0	qA	qB	qC	qAB	qAC	qBC	qABC						

Where:

- qA = Average of $A_i \cdot Y_i$_mean,
- qB = Average of $B_i \cdot Y_i$_mean,
- qC = Average of $C_i \cdot Y_i$_mean,
- Similarly, are qAB, qAC, qBC, qABC calculated.
- $SSx = qX^2 \cdot (2^k \cdot (r-1)) = qX^2 \cdot 32$.
- $SSY = \sum_{i=1}^{2^k} \sum_{j=1}^{r} {y_{ij}}^2$
- $SS0 = 2^k \cdot r \cdot q0^2$
- $SSE = SST - \sum_{j=1}^{2^k-1} SS_j$

Table 9 is a supplementary table of **Table 8**, showing residual (errors) from the mean value Y_mean and also the sum square of all deviations as SSE.

Table 9 Residuals and sum square of all deviations.

I	J	K	L	M	N	O	P	Q	R	S	T
y1	y2	y3	y4	y5	Y_mean	e1	e2	e3	e4	e5	e^2(sum)
1076	1106	1100	1103	1107	1098,4	-22,4	7,6	1,6	4,6	8,6	657,2
1639	1667	1616	1635	1702	1651,8	-12,8	15,2	-35,8	-16,8	50,2	4478,8
1107	1108	1104	1105	1103	1105,4	1,6	2,6	-1,4	-0,4	-2,4	17,2
1608	1643	1589	1612	1641	1618,6	-10,6	24,4	-29,6	-6,6	22,4	2129,2
676	680	687	686	679	681,6	-5,6	-1,6	5,4	4,4	-2,6	89,2
927	925	952	926	922	930,4	-3,4	-5,4	21,6	-4,4	-8,4	597,2
684	683	659	685	668	675,8	8,2	7,2	-16,8	9,2	-7,8	546,8
911	896	930	902	900	907,8	3,2	-11,8	22,2	-5,8	-7,8	736,8
											9252,4
											SSE

7.8.3 Allocation of variation

From **Table 8**, we calculated the allocation of variance as follows:

$SSY = SS0 + SSA + SSB + SSC + SSAB + SSAC + SSBC + SSABC + SSE =$ 46978395 +1496529,2+ 1863,2+ 3245011,3+ 2030,7+ 214476 +3,0 + 342,2 + 9254 = 51947903

$SST = SSY - SS0 = 51947903 - 46978395 = 4969507,9 = 100\%$

$SSE = SSY - (SS0 + SSA + SSB + SSC + SSAB + SSAC + SSBC + SSABC) =$ 51947903 $-$ (46978395+ 1496529,2+ 1863,2+ 3245011,3+ 2030,7+ 214476 +3,0 + 342,2) = 9254

The Standard Deviation of Errors:

Se = $\sqrt{SSE/2^k(r-1)} = \sqrt{9254/32}$ = 17,006

The Standard Deviation of Effects:

(STDe) = $Se/\sqrt{2^k(r-1)} = 17{,}006/\sqrt{32}$ = 3,006

Thus, the t-value at 32 degree of freedom and 90% confidence interval is $t_{[0.95,\ 32]}$ = 1,671.

The confidence intervals:

$qi' = \pm(1{,}671)(3{,}006) = \pm 5{,}023$

7.9 Result Analysis

Percentage of y's variation explained by x:th effect is (SSx/SST) × 100% can be interpreted as follows:

- Variation due to factor A (*type of operation*) =1496529,2/4969507,9×100%= (30.11%).

- Variation due to factor B (*message size*) = 1863,22/4969507,9×100% = (0.00037%).
- Variation due to factor C (*status of the key*) = 3245011,2/4969507,9×100% = (65.3%).
- Variation due to interaction of (AB) factors =2030,625/4969507,9×100% = (0.0004%).
- Variation due to interaction of (AC) factors = 214476,025/4969507,9×100% = (4.3%).
- Variation due to interaction of (BC) factors = 3,025/4969507,9×100% = (0.0%).
- Variation due to interaction of (ABC) factors = 342,2/4969507,9×100% = (0.0%).
- Variation due to experimental errors = 9252,4/4969507,9× 100% = (0.19%).

Out of variation results, we can conclude the following:

1. Most of the variation is explained by factors C (status of the key: persistent or transient) with 65.3% variation and factor A (type of cryptographic operation: decryption or signing) with 30.11% variation, and considerably very low effect due to factor B (message size) with only 0.00037% variation.
2. The variation due to experimental errors is small (0,19%), however it is larger than variation of factor B as well as larger than interaction of AB, BC and ABC factors. Some statistically significant effects (factor B, interaction of AB, BC and ABC) explain less than 0.001% of the variation.
3. Only effects A, C, and AC are both statistically and practically significant.
4. All effects don't include zero, and hence are significant.

7.10 Time measurement using OpenSSL

To achieve fair evaluation, we run the same operations i.e. decryption and signing with the same file sizes using OpenSSL library, which GWen support under its Linux distribution. The time as shown in **Figure 17** and **Figure 18** is roughly 88 milliseconds in both cases.

```
root@gwen-emc:~# time time openssl rsautl -decrypt -inkey private.pem -in cipher
_10_Byte.enc -out out_10.txt

real    0m0.088s
user    0m0.080s
sys     0m0.000s
root@gwen-emc:~# time time openssl rsautl -decrypt -inkey private.pem -in cipher
_245_Byte.enc -out out_245.txt

real    0m0.088s
user    0m0.080s
sys     0m0.000s
root@gwen-emc:~#
```

Figure 17 Latency of Decryption Using OpenSSL

```
root@gwen-emc:~# time openssl rsautl -sign -in plain_10_Byte.txt -inkey private.
pem -out sig_10

real    0m0.088s
user    0m0.080s
sys     0m0.000s
root@gwen-emc:~# time openssl rsautl -sign -in plain_245_Byte.txt -inkey private
.pem -out sig_245

real    0m0.088s
user    0m0.080s
sys     0m0.000s
root@gwen-emc:~#
```

Figure 18 Latency of Signing Using OpenSSL

7.11 Measurement Observation

Signing operation is basically performed by calculating the digest of the message using the private part of the RSA key to yield a valid signature, which can be used to verify the integrity of the message using the public part of the same RSA key. However, asymmetric algorithms such as RSA are computationally expensive, and there is always limitation in the data size that RSA encryption algorithm can operate on, which is a maximum message size of 245 byte when using RSA 2048-bit key size. A common bug is trying to sign or decrypt data that exceeds the capacity of this key size. When hashing using SHA-256 algorithm, the message is divided into blocks with similar size (multiple of 32 bits). For each block that is less in size than the maximum block size, a padding scheme based on OAEP or PKCS#1 will be used to cover this shortage. [11]

From Table 7 above, it was observed that the signing and decrypting of 10-byte message size using RSA persistent key takes sometimes longer time than signing or decrypting 245-byte messages using the same key. This was unexpected as smaller size messages should take less time to process than larger messages. As it is known that SHA-256 algorithm produces outputs that has the same size always regardless of their input size, and in the case of small input size than SHA-256 size, the input will be padded. In our case, 245-byte size messages will be padded with a smaller number of bits compared to 10-byte size messages and this extra round padding operation (with small size messages) can be the reason as to why signing 10-byte messages takes slightly longer time than signing 245-byte message.

To test this hypothesis, we repeated the time measurements many more times and it was always confirmed that RSA signing of 10-byte message size using persistent TPM keys takes slightly longer time compared to signing 245-byte message size using the same key size and the same algorithm. However, this case is not true with the decryption using RSA persistent TPM key. We repeated the time measurements for the decryption too and we found that decrypting 10-byte messages almost always is performed faster than decrypting 245-byte message using the same RSA key and the same hashing function i.e. SHA-256.

The reason for this extra latency when signing small size messages compared to larger messages is that signing operation does need SHA-256 with its additional overhead while decrypting messages does not, and thus the measurements of decryption are more consistent in contrast to the measurements of signing operation.

8. Conclusion

As a result of this project, we were able to answer our research question showing that it is possible to enable Hardware Root of Trust (HRoT) using SLB 9670 TPM 2.0 chip on the Edge Linux-based Gateway (GWen), a product from HMS industrial Network AB. To enable the TPM, we utilized the TSS 2.0, an open-source standard of the TCG software stack implemented by *Developer Community for those implementing APIs and infrastructure from the TCG TSS2 specifications*. Our method is based on literature review and testing and evaluating the open source TPM 2.0 Software Stack (TSS 2.0) using a TPM simulator before injecting the TSS 2.0 into Linux image and then download and install the new image into the GWen board with all TSS 2.0 tools and dependencies. This makes it possible to construct a communication link to the SLB 9670 TPM 2.0 chip and call TPM 2.0 commands directly similar to any Linux build-in command.

We demonstrated how TPM commands can be used to generate TPM keys of different types, perform various cryptographic operation as well as we stored cryptographic keys securely inside the TPM. We described a method to secure boot the GWen board based on Xilinx Zynq-7020 FPGA SoC instead of the TPM chip for three reasons:

1. Enabling secure boot on Zynq-based systems is highly encouraged by Xilinx[5] in [18] with almost no additional latency overhead in systems with secure boot compared to non-secure boot systems.
2. Zynq-7020 SoC is a powerful ARM-based Cortex A9 processor with a hardware programmability of an FPGA that is able to perform, manage and schedule cryptographic operations needed for secure boot much faster than slow TPMs.
3. TPM can only perform either Trusted Boot or Measured Boot. The Trusted Boot will check the integrity of every component of the startup process before loading the entire OS. In Measured Boot, TPM will log the boot process, store them into PCRs and request the system to report the states of PCRs to a trusted server that can objectively assess the system's health and takes the proper action. TPM, however, in either boot type, has no control over the booting sequence in case of non-genuine software is detected during the booting process.

Therefore, performing secure boot using Zynq-7020 FPGA SoC processor instead is recommended. This will certify that the running OS is coming from a valid genuine Linux image and also that any software updates or upgrades comes from a trusted source.

Based on our performance evaluation and analysis of the response time of TPM commands that implements TSS 2.0 and hosted on GWen board, we conclude that TPM key-generation, which is the most time-consuming operation can be resolved if the frequently used keys are stored persistently inside the TPM. We could also observe that

[5] Xilinx is the inventor of the FPGA, hardware programmable SoC Zynq-7020.

the message size, when encrypted with RSA-keys, has trivial or almost no impact on TPM time response. Moreover, by comparing TPM and OpenSSL, we could see that the SLB OpenSSL library can achieve both the decryption and signing operations fairly faster compared to the time response required by 9670 TPM 2.0 chip using the same file sizes, the same key sizes.

Lastly, according to the high restrictions of TPM 2.0 specifications and based on our results, we could say that, adding the TPM 2.0 to the IoT gateway GWen will enhance the security of its Linux distribution and will make it possible to securely identify and authenticate the gateway on the network based on its secret keys that are stored securely inside its TPM.

9. Discussion and Future Work

In section 9.1 we provide some of TPM 2.0 advantages, and in section 9.2, we highlight the Offline Hamming Attack and how TPM 2.0 addresses this problem, and in section 9.3, we discussed key storage and management in TPM 2.0. In section 9.4, we describe some recommendations for TPM 2.0 provisioning and de-provisioning processes, and in section 9.5, we propose an approach for a future work in which TPM can be used to authenticate the hosting platform using Transport Layer Security (TLS) protocol.

9.1 TPM Characteristics and Benefits

TPM 2.0 from the other hand will provide the following advantages for the hosting system:

1. Identification of host devices: instead of using MAC or IP-addresses, which consider not secure identifiers.
2. Secure generation of keys: using hardware random-number generator allows for secure key generation instead of poor key generation by some software which can be breakable.
3. Secure storage of keys: keeping private keys stored securely inside the TPM with absolute isolation from the outside access by whosoever, except the TPM itself. NVRAM storage: makes it possible for the TPM to maintain a certificate store. Device health attestation: instead of using software to attest a system and report that it is healthy while it is not, TPM provide this mechanism more securely and trustworthy.
4. Algorithm agility: support for various cryptographic algorithms, such as: SHA-1 (default), SHA-256, RSA (default), AES, Keyedhash, ECC, Symcipher, etc. Enhanced authorization: new capability in TPM 2.0 defines a way to authorize all TPM entities, as well as, giving the TPM the ability to enable authorization policies that allow multi-user authentication.
5. Quick key loading: using symmetric encryption method to load keys into the TPM rather than asymmetric encryption method used in TPM 1.2 previously which considered relatively slow.

Although TPM can function as a cryptographic coprocessor, but it is considerably slower than a pure software implementation. However, TPM features can be useful for some applications such as:

- In some highly embedded resource-constrained environments, which may not support implementing a complex crypto math or may not have any external persistent storage for cryptographic keys.

- In some low-performance applications, it might be suitable to utilize the TPM than to implement cryptographic software with an extra overhead. In an early boot cycle, when a key is needed before the disk is available.
- In some applications that require a certified implementation, the TPM can be a good choice as a certified cryptographic engine.

9.2 TPM mitigation to Offline Hammering Attack

When a secret is encrypted with a password, the secret is still vulnerable to an offline hammering attack. So, if the attacker could obtain the encrypted key, he could apply a high-speed attack that tries an unlimited number of passwords. If the attack is parallelized with many powerful computers that try different passwords simultaneously, the probability to crack the password becomes very feasible. According to [11], TPM 2.0 has addressed this problem in two aspects:

1. Each key in TPM is wrapped (encrypted) with a strong parent key. The attacker will have to crack a strong key rather than a weak password.
2. Each key loaded in the TPM is protected by the so-called Dictionary Attack Protection Logic (DAPL). The DAPL logs the failure each time an attacker fails to crack the key's authorization. If the number of failures exceeds a specific number, the TPM blocks all further attempts for a certain amount of time. This limits the speed at which an attacker can try new passwords and can even make a weak TPM key password much more time consuming to crack than a strong software key password.

9.3 TPM 2.0 Key Storage and Management

The end user or the administrator of the platform is responsible for key management and not the TPM. It was observed that all TPM keys generated in section 5.3.2 were transients since they belong to Null hierarchy in which its primary seed is regenerated with new values on each TPM reboot. Although primary objects and RSA keys still exist in host device's main storage (i.e. GWen's flash memory) and can be loaded into the TPM, but it will not be possible to decrypt or verify any ciphers or signatures that were previously encrypted or signed when using the Null hierarchy.

TPM 2.0 has the ability to store limited number of secrets and essential primary objects persistently in NV memory inside the TPM. Key generation is perhaps the most time-consuming cryptographic operation in TPM. Essential keys should be moved to persistent storage after creation to avoid the performance penalty of recalculating the key again on each and every boot cycle. It is usually, Storage Root keys (SRKs), primary restricted signing keys like AIKs, and possibly EKs, which are the only entities that are stored persistently inside the TPM. Other objects are volatile and only persistent objects will

remain stored until they are flushed by the administrator or TPM deprovisioning process using `TPM2_Clear` command. [11]

We conducted more tests for key generation and loading under other hierarchies, and we reboot the platform (i.e. GWen and TPM) after each TPM command. Similar to Null hierarchy, it was also not possible to decrypt messages or verify signatures if a reboot was conducted when using other hierarchies. To make an object persistent, a TPM command `tpm2_evictControl` can be used with a specific handle (i.e. index) at which the object will be stored. That persistent object will remain in TPM NV memory and can be protected with an authorization and policy. The full template and complete set of arguments of `tpm2_evictControl` command can be found at [40]. Besides, one use-case is added to the Appendix at the end of this report.

9.4 Provisioning the TPM

TPM provisioning means activating and utilizing the TPM for use in device identification, storage of encryption keys and credentials such as EK, AIK and their associated certificates, for authentication, attestation, etc. The adopted steps for TPM provisioning should be done not repeatedly but once during the TPM usage lifetime. The provisioning process is divided among three parties: TPM Manufacturer Provisioning, Platform Manufacturer Provisioning and End User Provisioning.

9.4.1 TPM Manufacturer Provisioning

At the beginning of this process, it is assumed that the TPM is in a clean state with no EKs, no EK Certificates, or any other objects prior to the provisioning. The TPM manufacturer should ensure that the following criteria are maintained and supported by the TPM:

1. **Cryptographic Algorithm Support** – TPM should support SHA256, AES-128 in CFB mode, and one or more of the following asymmetric algorithms: RSA-2048, ECC-P256.
2. **Security Certification** – Use a scheme that can satisfy the requirements set by the target market to certify the TPM.
3. **Reserved Locations** – Make the reserved locations on the TPM available. These locations are used for the storage of persistent objects in the NV Index and in Persistent Memory like EK, EK certificate, SRK, IDevID key and IDevID key certificate.

For support of an identity and/or attestation, the same additional steps are required:

1. The Endorsement Primary Seed (EPS) should be populated.
2. At least one EK should be created using an approved template and stored in NV Index 0x8101000.
3. Create an EK certificate and store it in NV Index 0x01C00002.

Once the TPM is powered on, it can generate a primary seed in the endorsement hierarchy. The TPM manufacturer then uses `TPM2_CreatePrimary` one or many times to create endorsement primary keys. These primary objects are the ones at the root of the endorsement hierarchy. This command returns the public key, which the manufacturer can use to create a certificate.

9.4.2 Platform Provisioning

The platform provisioning has two main goals: authenticity and control. Same as the TPM manufacturer, platform manufacturer should certify that the hardware is trustable and is not counterfeit. So, it claims that the TPM is embedded in the hardware and platform software operations are according to the TCG rules. The platform provisioning can be done using the following common recommendations that are needed to support Identity, Storage, and/or Attestation:

1. **Boot Firmware Protection** – the installed boot firmware should follow the security guidance published by the TCG and NIST for designing and installing boot firmware.
2. **Functional Requirements** – Installing drivers for pre-boot and OS and some software for functionality that provisions and manages the authorization values and policies for the following administrative roles: Platform Administrator (i.e. TPM Owner), Privacy Administrator, and Lockout Administrator, as well as a functionality that sets the Dictionary Attack Parameters.
3. **Reserved Locations** – Make the reserved locations on the TPM available.

There are some other recommended steps to support identity as follows:

1. **IDevID Key Creation** – Create an IDevID Key Pair using an approved template, store it and encrypt the key pair with the EK public key and store the resulting blob in a known external location so that the Platform software can import it into the TPM and use it later.
2. **IDevID Key Certificate Creation** – Create a certificate of the IDevID using an approved template, store the certificate in some safe location so that the Platform software can load it into the TPM and use it later.
3. **Platform Authorization** – The platform firmware should set *platformAuth* and *platformPolicy* flags as following: *platformAuth* / *platformPolicy* / PPS for platform firmware; *ownerAuth* / *ownerPolicy* / SPS for the Owner; and endorsementAuth / *endorsementPolicy* / EPS for the Privacy Administrator. After each TPM reboot, the TPM sets *platformAuth* and *platformPolicy* to an Empty Buffer by default. Therefore, the platform firmware should reinstate the non-Empty Buffer values of *platformAuth* and *platformPolicy* after each TPM reboot.
4. **Platform Hierarchy** – The Platform Hierarchy can be disabled by clearing *phEnable* flag, which means that subsequent platform firmware, OSs, and

applications cannot use *platformAuth* and *platformPolicy* to authorize any TPM action until the Platform restart or reset.

9.4.3 End User Provisioning

In this case the end user is considered as: the IT organization, platform administrator or supporting personnel who will provision the Endorsement and Storage hierarchies. The IT organization can choose to provision the Endorsement hierarchy and/or the Dictionary Attack Reset Authorization[6] (DARA) and leave provisioning the Storage hierarchy to the end user of the platform.

The following recommendations can be taken:

1. **Disable hierarchies** – can be done using `tpm2_hierarchycontrol` command because all hierarchies are enabled at the start up by default.
2. **Apply policies and authorization** – policies and authorization values for the Endorsement and Storage hierarchies, and the dictionary-attack protection should be set up. Both endorsement policies and authorization values can be set using `tpm2_hierarchyChangeAuth` and `tpm2_setPrimaryPolicy` commands.

The endorsement primary seed (EPS) is produced by the TPM manufacturer and it supposed it is the same during the TPM lifetime.

Both Endorsement Primary Seed (EPS) and Storage Primary Seed (SPS) are generated by the TPM manufacturer. The end user can use the existing SPS or regenerate a new one. Changing SPS leads to invalidating all objects in the Endorsement and Storage hierarchies. The EKs won't be changed and yet are their certificates still valid.

9.4.4 Deprovisioning

Deprovisioning is the procedure for deleting all stored secrets and credentials from the TPM. The deprovisioning can be achieved using Tpm2_Clear command. The authorization associated with this command is TPM_RH_LOCKOUT (Dictionary Attack Reset Authorization DARA) or TPM_RH_PLATFORM (platform authorization). Executing the `Tpm2_Clear` command results the following:

- All TPM-resident objects in either storage or platform hierarchy are flushed.
- The SPS is changed, and thus, any object created under Storage hierarchy can no longer be loaded.
- Any NV index created by the owner is deleted.

---

[6] Triggering the dictionary-attack protection will enforce a large wait before trying-new-password can be allowed.

- Authorization values are reset; the lockout authorization and the policy are cleared.

The dictionary-attack counter, the clock, reset count and restart count all are reset.

9.5 Using TPM for TLS Authentication

TPM has an EK that's unique to each TPM chip, but the EK is not directly accessible from outside the TPM as this is a TPM privacy sensitive issue. Instead, the EK can be used in conjunction with the AIK that will be generated under the Endorsement hierarchy and signed by the EK. The AIK will then be signed by a certain privacy Certificate Authority (CA), which will verify that those keys have been generated by a genuine TPM and will issue an AIK certificate. The private part of the AIK never leaves the TPM, which means that the AIK signatures (and thus AIK certificates) belong to a genuine specific TPM. The type of AIK certificate is an X.509 valid certificate, which can be verified using the standard X.509 verification function [38]. These certificates will make it possible to authenticate devices in authentication protocols such as TLS based on their TPMs. In [39], a method is proposed for full TLS handshake with client using legacy TPM 1.2 certified keys. We propose similar approach based on SKAE extension and using TPM EK, AIK, DevID keys and their certificates. The process is illustrated in **Figure 19** and explained underneath.

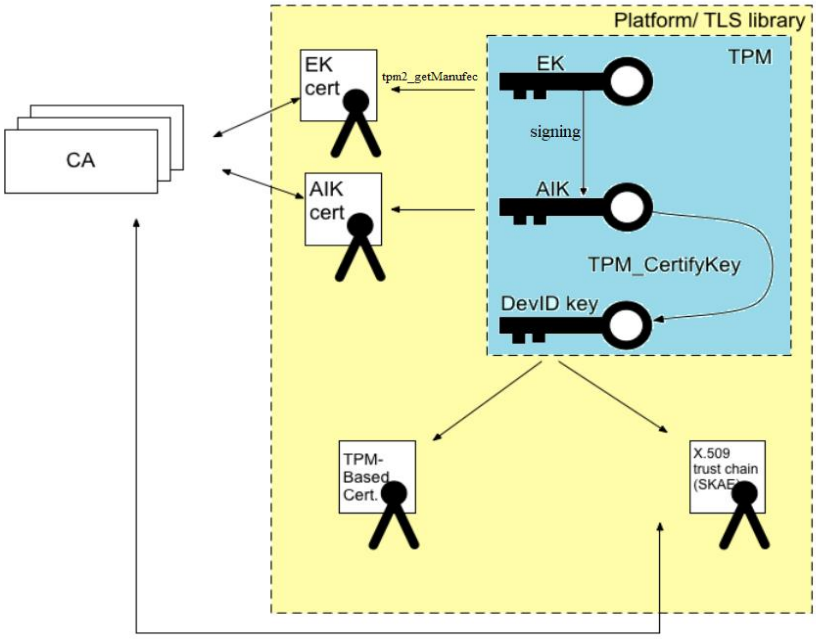

Figure 19 TPM-based certificate obtaining mechanism using Ek and AIK certificates

The process can be recapitulated in the following steps:

1. The TPM will derive EK and AIK keys with an RSA or ECC template using tpm2_getpubek and tpm2_getpubak commands for Endorsement and Attestation keys respectively.

2. Both EK and AIK should be made persistent and use the EK to sign the AIK.
3. Obtain an EK certificate from TPM manufacturer's endorsement certificate hosting server using tpm2_getManufect command.
4. Obtain an AIK certificate from a Privacy CA.
5. Generate a new non-migratable asymmetric key (i.e. DevID), make it persistent and certify the (DevID) key with the AIK using tpm2_certify command.
6. Generate a self-signed certificate X.509 around the certified key DevID key including an extension known as Subject Key Attestation Evidence (SKAE). The SKAE extension is an X.509 extension that has been defined to carry the certified structure returned by tpm2_certify command.
7. Send EK certificate, AIK certificate with a request to a CA to sign the X.509 certificate.
8. The result is a standard X.509 certificate that can be used together with DevID key for TLS handshake.

The future work can be to find a way to implement this approach, besides to find answers to the following questions:

A. Find trusted method to retrieve certificates from TPM manufacturer's endorsement certificate hosting server.
B. Find secure way to send CA certificate requests and receive certificates.
C. Analyze the applicability and reliability of this approach to create and request all the certificates necessary to TLS authentication.
D. Analyze the latency of key generation and other cryptographic operation compared to the latency of other software libraries like OpenSSL or cryptographic protocols like SSH.

Bibliography

[1] Eriksson M, Pourzandi M, Smeets B. *Trusted Computing for Infrastructure*. Ericsson Review. 2014; 8: (pp. 1-8).

[2] Chen L, Franklin J, Regenscheid A. *Guidelines on hardware-rooted security in mobile devices* (Draft). *NIST Special Publication*. 2012 Oct;800(164): (pp. 10-l).

[3] Schrecker S, Soroush H, Molina J. *Industrial Internet of Things Volume G4: Security Framework.* Needham, Massachusetts: *Industrial Internet Consortium*; 2016. 173 p.

[4] Schmitz J, Loew J, Elwell J, Ponomarev D, Abu-Ghazaleh N. TPM-SIM: *a framework for performance evaluation of trusted platform modules.* InDesign Automation Conference (DAC), 2011 48th ACM/EDAC/IEEE 2011 Jun 5 (pp. 236-241). IEEE.

[5] P. Dwyer, *Cybersecurity and sustainability, don't WannaCry?* 2017. [cited 30 10 2018]. Available at: http://thepurposebusiness.com/cybersecurity-and-sustainability-dont-wannacry/

[6] Kirsch D. *The Value of Bringing Analytics to the Edge*. Needham Heights, Massachusetts: *Hurtwitz & Associates*; 2015. 10 p.

[7] Trusted Computing Group. *TCG Guidance for Securing Network Equipment Using TCG Technology*. 2018 Jun 17. [cited 2018 07 16] Available at: https://trustedcomputinggroup.org/wpcontent/uploads/TCG_Guidance_for_Securing_NetEq_1_0r26b_Public-Review.pdf.

[8] Trusted Computing Group. *ARCHITECT'S GUIDE FOR SECURING NETWORK EQUIPMENT*. 2018 Jun 17. [cited 2018 07 16] Available at: https://trustedcomputinggroup.org/wpcontent/uploads/ArchitectsGuide_Securing_Network_Equipment.pdf.

[9] Trusted Computing Group. *Trust Computing*. 2018. [cited 2018 07 16]. Available at: https://trustedcomputinggroup.org/trusted-computing/.

[10] Technical University Darmstad / Sadeghi AR. *Introduction to Trusted Computing*. 2017 [cited 2018 may 27]. Available at: https://www.trust.informatik.tu-darmstadt.de/fileadmin/user_upload/Group_TRUST/LectureSlides/STC-WS2016/02_Introduction_to_Trusted_Computing.pdf.

[11] Arthur W, Challener D, Goldman K. *A practical guide to TPM2.0*. Berkeley: Apress; 2015.

[12] Malipatlolla S, Feller T, Huss SA. *An Adaptive System Architecture for Mitigating Asymmetric Cryptography Weaknesses on TPMs*. In *NASA/ESA Conference on Adaptive Hardware and Systems*; 2012; (pp. 221-226).

[13] Tomlinson A. *Introduction to the TPM*. In Mayes K, Markantonakis K. *Smart Cards, Tokens, Security and Applications*. Boston: Springer; 2008; (pp. 155-172).

[14] Seaman M, editor. *802.1AR-2009 – IEEE Standard for Local and Metropolitan Area Networks - Secure Device Identity*. *IEEE Std 802-2014*. New York: IEEE Computer Society; 2014; (pp. 1-74)

[15] Trusted Computing Group. *TPM 2.0 Library Specification.* 2016 Sep 09. [cited 2018 07 16]. Available at: https://trustedcomputinggroup.org/resource/tpm-library-specification/.

[16] Trusted Computing Group. *Trusted Platform Module Library. Part 1: Architecture*. 2016 Sep 09. [cited 2018 07 16]. Available at: https://trustedcomputinggroup.org/wp-content/uploads/TPM-Rev-2.0-Part-1-Architecture-01.38.pdf.

[17] Trusted Computing Group. *TCG Glossary.* 2012 Des 13. [cited 2018 07 20]. Available at: https://www.trustedcomputinggroup.org/wp-content/uploads/TCG_Glossary_Board Approved_12.13.2012.pdf.

[18] Sanders L. *Secure Boot of Zynq-7000 All Programmable SoC. XAPP1175*, 2015 April 3; (pp. 5-63).

[19] Jacob N, Heyszl J, Zankl A, Rolfes C, Sigl G. *How to Break Secure Boot on FPGA SoCs through Malicious Hardware. Fraunhofer Research Institution AISEC.* 2017; (pp. 2-16).

[20] ELTT2. *Infineon Embedded Linux TPM Toolbox 2 for TPM 2.0.* 2018 [cited 2018 7 11]. Available at: https://github.com/Infineon/eltt2.

[21] Trusted Computing Group. *TSS 2.0 Enhanced System API (ESAPI) Specification.* 2017 [cited 2018 6 20]. Available at: https://trustedcomputinggroup.org/wp-content/uploads/TSS_TSS-2.0-Enhanced-System-API_V0.9_R03_Public-Review-1.pdf.

[22] Trusted Computing Group. *TCG Software Stack Feature API, Family "2.0" Level 00 Revision 12*. 2014. [cited 2018 7 23]. Available at: https://trustedcomputinggroup.org/wp-content/uploads/TSS-Feature-API-version-.12_Review.pdf.

[23] Infineon. *Getting Started with OPTIGA™ TPM 2.0 SLB 9670 and Raspberry Pi® 3*; 2017. [cited 2018 7 24]. Available at: https://www.infineon.com/dgdl/InfineonTPM20_Embedded_SLB_9670_AppNote-AN-v01_00 EN.pdf?fileId=5546d46265257de8016537f329595e5c.

[24] Package tpm2-tools. *Package tpm2-tools and TPM 2.0 command templates*. 2016. [cited 2018 7 23]. Available at: https://www.mankier.com/package/tpm2-tools.

[25] Github. *Linux TPM2 & TSS2 Software*. 2018. [cited 2018 7 27]. Available at: https://github.com/tpm2-software .

[26] Military Embedded Systems/Edwards S. *Establishing a root of trust: Trusted computing and Intel-based systems.* 2017 [cited 2018 05 24]. Available at: http://milembedded.com/articles/establishing-root-trust-trusted-computing-intel-based-systems/

[27] Shpantzer G. *Implementing Hardware Roots of Trust: The Trusted Platform Module Comes of Age. 2013.* SANS Institute; (pp. 2-14).

[28] Sabt M, Achemlal M, Bouabdallah A. *Trusted Execution Environment: What It is, and What It is Not. Trustcom/BigDataSE/ISPA*, 2015; 1: (pp. 57-64).

[29] Zhao S, Zhang Q, Hu G, Qin Y, Feng D. *Providing root of trust for ARM Trust Zone using on-chip SRAM. InProceedings of the 4th International Workshop on Trustworthy Embedded Devices*. 2014; (p. 25-36).

[30] Proudler G, Chen L, Dalton C. *Trusted Computing Platform TPM2.0 in Context*. New York Dordrecht London: Springer; 2014.

[31] LENOVO/Cui N. *A Technical Introduction to the Use of Trusted Platform Module 2.0 with Linux*. Lenovo Press; 2017 Sep 8. [cited 2018 03 26]. Available at: https://lenovopress.com/lp0599-technical-introduction-tpm-20-with-linux.

[32] GPL/Dinh TTA, Ryan MD. *Protected storage and Root of Trust for Storage*. 2006 [cited 2018 5 31]. Available at: https://www.cs.bham.ac.uk/~mdr/teaching/modules/security/lectures/TrustedComputingTCG.html.

[33] Oxford University/Martin A. *The ten-page introduction to Trusted Computing*. 2008 November. [cited 2018 5 31]. Available at: https://ora.ox.ac.uk/objects/uuid:a4a7ae67-7b2a-4516-801d-9379d613bab4.

[34] Ebrary. *TPM Provisioning*. 2018. [cited 2018 08 13]. Available at: https://ebrary.net/24846/computer_science/provisioning.

[35] Trusted Computing Group. *TCG TSS 2.0 Overview and Common Structures Specification*; 2018 [cited 2018 08 20]. Available at: https://trustedcomputinggroup.org/resource/tss-overview-common-structures-specification/.

[36] Birnstill P, Haas C, Hassler D, Beyerer J, *Introducing Remote Attestation and Hardware-based Cryptography to OPC UA*, IEEE (ETFA). 2017, (pp. 1-8).

[37] Yuan S. Zhao B, Yu Z, Zhang H. *A Security-Improved Scheme for Virtual TPM Based on KVM*, Wuhan Univ. J. Nat. Sci. (2015) 20: p. 505. https://doi.org/10.1007/s11859-015-1126-5.

[38] Trusted Computing Group. *TCG Infrastructure WG TPM Keys for Platform Identity for TPM 1.2, Specification.* August 2015 [cited 2018 08 20]. Available at: https://trustedcomputinggroup.org/wpcontent/uploads/TPM_Keys_for_Platform_Identity_v1_0_r3_Final.pdf.

[39] University of Fribourg/Latze C. et al, *Transport Layer Security (TLS) Extensions for the Trusted Platform Module (TPM)*. 2011 [cited 2018 08 25]. Available at: https://tools.ietf.org/html/draft-latze-tls-tpm-extns-02#ref-SKAE.

[40] Trusted Computing Group. *Trusted Platform Module Library Part 3: Commands, Family "2.0" Level 00 Revision 01.38*. September 29, 2016. [cited 2018 07 23]. Available at: https://www.trustedcomputinggroup.org/wp-content/uploads/TPM-Rev-2.0-Part-3-Commands-01.38.pdf .

[41] Jain R. *The art of computer systems performance analysis, techniques for experimental design, measurement, simulation and modeling*, Wiley 1992.

[42] Microsoft, *Understanding and Evaluating Virtual Smart Cards*, [online], Accessed on November 2018.

[43] Lin K. et. al, *Using TPM to Improve Boot Security at BIOS Layer*, 2012 IEEE International Conference on Consumer Electronics (ICCE).

[44] Zhao S. et. al, *Providing Root of Trust for ARM TrustZone using On-Chip SRAM*, Proceeding TrustED '14 Proceedings of the 4th International Workshop on Trustworthy Embedded Devices, Pages 25-36, Scottsdale, Arizona, USA — November 03 - 03, 2014, ACM

[45] Ekberg J. et. al, *Trusted Execution Environments on Mobile Devices*, November 2013 with 247 Reads DOI: 10.1145/2508859.2516758, Proceedings of the 2013 ACM SIGSAC conference on Computer & communications security.

[46] Yang X. et. al, Trust-E : *A Trusted Embedded Operating System Based on the ARM TrustZone*, 2014 IEEE 11th Intl Conf on Ubiquitous Intelligence and Computing and 2014 IEEE 11th Intl Conf on Autonomic and Trusted Computing and 2014 IEEE 14th Intl Conf on Scalable Computing and Communications and Its Associated Workshops.

[47] Aaraj N. et. al, *Energy and Execution Time Analysis of a Software-based Trusted Platform Module*, Published in: 2007 Design, Automation & Test in Europe Conference & Exhibition.

Appendix

1. TSS Stack setup

The following are instructions for downloading, configuring, compiling and building tpm2-tss, tpm2-abrmd and tpm2-tools projects in Linux, and putting all together in one project folder. All instructions below are taken from [25] and are according to Developer Community Group for implementing APIs and infrastructure from the TCG TSS2 specifications.

First the following dependencies are required:

- GNU Autoconf
- GNU Automake
- GNU Libtool
- pkg-config
- C compiler
- C Library Development Libraries and Header Files (for pthreads headers)
- SAPI - TPM2.0 TSS SAPI library and header files
- OpenSSL libcrypto library and header files
- Curl library and header files

These packages can be installed using the following commands:

```
$ sudo dnf -y update && sudo dnf -y install automake libtool \
autoconf autoconf-archive libstdc++-devel gcc pkg-config \
uriparser-devel libgcrypt-devel dbus-devel glib2-devel \
compat-openssl10-devel libcurl-devel PyYAML
```

Secondly, the following commands should be executed in order:

Tpm2-tss install and build

```
$ git clone git://github.com/tpm2-software/tpm2-tss.git
$ cd tpm2-tss
$ ./bootstrap
$ ./configure --prefix=/usr
$ make -j5
$ sudo make install
```

Tpm2-abrmd install and build

```
$ git clone git://github.com/tpm2-software/tpm2-abrmd.git
```

```
$ cd tpm2-abrmd
$ ./bootstrap
$ ./configure --with-dbuspolicydir=/etc/dbus-1/system.d
--with-udevrulesdir=/usr/lib/udev/rules.d
--with-systemdsystemunitdir=/usr/lib/systemd/system
--libdir=/usr/lib64 --prefix=/usr
$ make -j5
$ sudo make install
```

Tpm2-tools install and build

```
$ git clone git://github.com/tpm2-software/tpm2-tools.git
$ cd tpm2-tools
$ ./bootstrap
$ ./configure --prefix=/usr
$ make -j5
$ sudo make install
```

2. TPM simulator setup

TPM simulator install and build

```
$ mkdir ibmtpm && cd ibmtpm
$ wget https://downloads.sourceforge.net/project/ibmswtpm2/ibmtpm1119.tar.gz
$ tar -zxvf ibmtpm1119.tar.gz
$ cd src
$ make -j5
```

3. TPM Encryption/Decryption with protection

TPM 2.0 commands for encryption/decryption with a parent key password “abc123” and child key password “def123”:

*/*Encryption with password*/*

```
tpm2_createprimary -H e -K abc123 -g sha256 -G rsa -C primary.ctx
```

```
tpm2_create -c primary.ctx -P abc123 -K def123 -g sha256 -G rsa -u key.pub -r
key.priv
```

```
tpm2_loadexternal -H n -u key.pub -C rsaenc_key.ctx
```

*/*Decryption with password*/*

```
tpm2_load -c primary.ctx -P abc123 -u key.pub -r key.priv -C rsadec_key.ctx
```

```
tpm2_rsadecrypt -c rsadec_key.ctx -P def123 -I cipher.enc -o output.txt
```

4. Make an object persistent/transient

*/*Store the previously created primary object inside TPM and make it persistent under the handle 0x81010001*/*

```
tpm2_evictcontrol -A o -c primary.ctx -S 0x81010001
```

*/*display persistent objects with their handles and attributes*/*

```
tpm2_listpersistent
```

*/*Remove persistent object at the handle 0x81010001*/*

```
tpm2_evictcontrol -A o -H 0x81010001 -S 0x81010001
```

5. TPM Sign/Verify commands

TPM 2.0 Signing commands with ECC-key and using SHA-256 hashing algorithm:

*/*create a signing ECC key from a persistent key with handle 0x81010001 and load it t the TPM */*

```
tpm2_create -g sha256 -G ecc -H 0x81010001 -u key.pub -r key.priv
```

```
tpm2_load -H 0x81010001 -u key.pub -r key.priv -C eccSignKey.ctx
```

*/*Make the signing key persistent and sign a 12-byte size text file. Verify and generate a ticket contains the validation structure that proves it's done by the TPM*/*

```
tpm2_evictcontrol -A o -c eccSignKey.ctx -S 0x81010004
```

```
tpm2_sign -k 0x81010004 -g sha256 -m d.txt -s d_signature
```

```
tpm2_verifysignature -k 0x81010004 -g sha256 -m d.txt -s d_signature -t ticket
```