

Master's Thesis

# Emulation of TPM on Raspberry Pi

Marcus Sundberg  
Erik Nilsson



Department of Electrical and Information Technology,  
Faculty of Engineering, LTH, Lund University, March 2015.

# Emulation of TPM on Raspberry Pi

Marcus Sundberg Erik Nilsson  
`zbt08msu@student.lu.se` `fys07eni@student.lu.se`

Department of Electrical and Information Technology  
Lund University

Advisor: Martin Hell

March 19, 2015

Printed in Sweden  
E-huset, Lund, 2015

---

# Abstract

---

The **Trusted Platform Module** (TPM) is a dedicated microprocessor designed to secure hardware by integrating cryptographic keys into the non-volatile memory of the module. TPM is specified by the Trusted Computing Group (TCG).

TCG is an initiative started in 2003 by several multinational semiconductor and IT-companies. The initiative is an effort to develop standards for Trusted Computing where hardware is used to provide security support to software. The TPM is typically connected to the LPC bus on the motherboard of a PC and can be used to create and store cryptographic keys, generate random numbers, hash values and encrypt data.

The purpose of this thesis is to develop a TPM learning environment and a laboratory manual for introductory courses in computer security where the students are able to learn about the functionalities of the TPM as a means to secure hardware.

The functions of the TPM will be emulated on the ARM based single board computer Raspberry Pi developed by the Raspberry Pi foundation. The TPM commands will be executed from a PC which will connect to the Raspberry Pi remotely through TCP.

Several exercises related to TPM and its functionalities are provided as an appendix to this report. The exercises are intended for students or others interested in Trusted Computing. This report also provides exercises related to the creation of TPM applications using TSS (Trusted Computing Software Stack).

Keywords: TPM, Trusted Computing, Raspberry Pi

---

## Acknowledgements

---

This master thesis was conducted in the Autumn of 2014 at the faculty of Electrical and Information Technology (EIT) at LTH. We would like to thank Ben Smeets and Martin Hell at EIT for giving us the opportunity of conducting this thesis project.

---

# Contents

---

<b>1</b>	<b>Introduction</b>	<b>2</b>
1.1	Trusted Computing . . . . .	2
1.2	Questions to be evaluated . . . . .	3
1.3	Thesis subgoals and approach . . . . .	3
1.4	Why use TPM emulation? . . . . .	4
<b>2</b>	<b>Basic Cryptography</b>	<b>5</b>
<b>3</b>	<b>Trusted Platform Module</b>	<b>10</b>
3.1	What is a TPM? . . . . .	10
3.2	TPM Architecture . . . . .	12
3.3	TPM and cryptographic keys . . . . .	13
3.4	Key creation . . . . .	17
3.5	Data encryption and decryption . . . . .	17
3.6	Public key signatures . . . . .	18
3.7	Authentication and attestation . . . . .	18
3.8	Communicating with the TPM . . . . .	18
3.9	Developing TPM applications . . . . .	22
<b>4</b>	<b>Method</b>	<b>26</b>
4.1	Chosen hardware . . . . .	26
4.2	Software . . . . .	27
4.3	Evaluating TPM emulators . . . . .	28
4.4	Running the emulator on Raspberry Pi . . . . .	33
4.5	Communicating remotely to Raspberry Pi . . . . .	35
4.6	Preparing to create exercises . . . . .	35
4.7	TSS on Raspberry Pi TPM . . . . .	36
<b>5</b>	<b>Result</b>	<b>41</b>
5.1	Setting up the TPM environment . . . . .	41
5.2	Setting up the IBM software TPM on Raspberry Pi . . . . .	41
5.3	Building for other platforms . . . . .	43
5.4	Setting up libtpm on client . . . . .	44
5.5	Setting up TrouSerS . . . . .	45

5.6	Test TrouSerS with the IBM software TPM . . . . .	46
<b>6</b>	<b>Conclusion</b> _____	<b>47</b>
6.1	Evaluating if thesis goal has been reached . . . . .	47
6.2	Creating a TPM learning exercise . . . . .	48
6.3	Lack of documentation . . . . .	48
6.4	Was an expected solution achieved? . . . . .	49
6.5	Further development . . . . .	49
	<b>Bibliography</b> _____	<b>50</b>
<b>A</b>	<b>Exercises</b> _____	<b>53</b>
<b>B</b>	<b>TrouSerS applications</b> _____	<b>63</b>
<b>C</b>	<b>TrouSerS applications</b> _____	<b>68</b>

---

## List of Figures

---

2.1	The principle of asymmetric cryptography. . . . .	6
2.2	The principle of symmetric cryptography. . . . .	7
2.3	A hash function may be used as a digital fingerprint [1]. . . . .	8
3.1	The TPM installed in a compatible motherboard. . . . .	10
3.2	The architecture of the TPM as described by TCG [2]. . . . .	12
3.3	The Storage Root Key . . . . .	15
3.4	The Platform Migratable Key . . . . .	16
3.5	A Keytree example . . . . .	16
3.6	Architecture of the TCG Software Stack (TSS) [2]. . . . .	19
3.7	Content of the PCR printed in the terminal. . . . .	25
3.8	A list of all the keys loaded in the TPM. . . . .	25
4.1	A Raspberry Pi Revision B running the IBM software TPM with remote access over TCP/IP. . . . .	27
4.2	The architecture of the ETH TPM Emulator. . . . .	29
4.3	The architecture of the IBM Software TPM [3]. . . . .	31
4.4	Result of command <code>tpm_version</code> . . . . .	39
5.1	Software TPM running on Windows 8.1 . . . . .	44



## Glossary

- **AIK** - Attestation Identity Key
- **EK** - Endorsement Key
- **PCR** - Platform Configuration Register
- **SRK** - Storage Root Key
- **TC** - Trusted Computing
- **TCG** - Trusted Computing Group (developed TPM)
- **TDDL** - TPM Device Driver Library
- **TPM** - Trusted Platform Module
- **TrouSerS** - The open-source TCG Software Stack
- **TSS** - TCG Software Stack
  - TPM services provided through the TSS API are:
    - \* Extend data into the TPM's PCRs and log these events
    - \* Random Number Generation
    - \* RSA encryption and decryption using PKCS v1.5 and OAEP padding
    - \* RSA key pair generation
    - \* RSA key storage
    - \* RSA sign/verify
    - \* Seal data to arbitrary PCRs

## 1.1 Trusted Computing

Trusted Computing is a framework for data and network security created by the Trusted Computing Group (TCG). TCG is a non-profit organization with members like IBM, Ericsson, Google, Microsoft, Intel and AMD. TCG was created to develop standards for Trusted Computing by means of hardware solutions, applications and services. According to TCG, the term "Trusted Computing" (TC) refers to applications that leverage hardware-based "roots of trust" at the edge of the network and at the endpoints - sometimes referred to as "hardware anchors in a sea of untrusted software" - for higher assurance.

"Hardware anchors in a sea of untrusted software" generally means that hardware is used to provide security support to software for threats that are difficult to eliminate with software only [4] [5].

The development of the Trusted Platform Module is a step towards Trusted Computing which is a technology developed and promoted by the Trusted Computing Group. The main purpose of the TPM is to generate cryptographic keys in a secure way. Since the TPM is a dedicated hardware device it provides more security in doing this than a software-only solution. The TPM could be used by software to authenticate hardware devices. This makes the TPM a valuable asset when striving for system security.

### 1.1.1 Restrictions

TCG has in 2014 released the TPM 2.0 library specification that provides updates to the previous published TPM main specifications. This thesis restricts the work to TPM version 1.2. The reason for this is that version 2.0 contains several additional features that has not yet been implemented in software emulation. In the near term, it is expected that both TPM 1.2 and TPM 2.0 will be available and that vendors will provide implementations that support both TPM 1.2 and TPM 2.0.

The key changes of TPM 2.0 as compared to the existing TPM 1.2 specification

include:

- Support for additional algorithms
- Agility of algorithms for use by geographies or markets that require specific-use algorithms
- Enhancements to the availability of the TPM to applications
- Enhanced authorization for improved TPM management
- Additional cryptographic services to enhance the security of platform services

When sentences like “TPM supports only asymmetric cryptographic algorithms” is used in this report, what is really meant is that “TPM version 1.2 supports only asymmetric cryptographic algorithms”.

## 1.2 Questions to be evaluated

The following two questions form the heart of this report:

- Could a Raspberry Pi be used as a TPM server?
- Could the user communicate with the TPM remotely in a simple way?

## 1.3 Thesis subgoals and approach

A four-tier approach was developed in order to reach the intended objectives for this thesis:

- Evaluate the best suitable TPM emulator to be used in a learning environment.
- Make the emulator run on the Raspberry Pi single board computer.
- Enable remote communication to the TPM emulator running on the Raspberry Pi, for example through TCP/IP.
- Create exercises to be used by students in order to introduce Trusted Computing.

The thesis follows these four tiers in order to bring answers to the above questions and to create a learning environment used as an introduction to Trusted Computing.

The first step was to evaluate if there already exists any suitable TPM emulators in order not to ‘reinvent the wheel’. If no suitable emulator existed then a basic TPM emulator may have to be implemented to some extent. The main focus of the thesis would then be the development of a TPM emulator. However, if an already existing TPM emulator were to be used, the main focus of the thesis would be to develop the structure of a TPM learning environment.

The first step was evaluated simultaneously with the investigation of basic TPM functionality since knowledge of the TPM are required when evaluating features of existing TPM emulators.

## 1.4 Why use TPM emulation?

In order to learn about the functionality of the Trusted Platform Module (TPM) it is convenient to use software emulation instead of interfacing an actual TPM attached to the hardware. Software emulation of TPM enables some possibilities that are not efficient or possible by using hardware. One example would be to clear the TPM in the case that some application did not work as expected. This is easily done on a TPM emulator but could cause data loss if done on a hardware TPM which may be used by other applications.

Code written for a software TPM may be directly used for a hardware TPM. Programming for a software TPM gives the advantage of being able to debug and reset the TPM without affecting the surrounding system. Developing TPM applications using a software TPM also prevents the developer from getting locked out from the system if anything should go wrong. These are the main reasons for using a software TPM when developing and testing applications.

A TPM that is integrated into a laptop or is connected to a PC motherboard is bound to be used by the specific owner of the system. A TPM is usually shipped with an Endorsement Key (EK) and a certificate created at manufacturing. When using a software TPM, the user can create his or her own EK and certificate to get a better understanding of how the creation of these are performed.

## Basic Cryptography

---

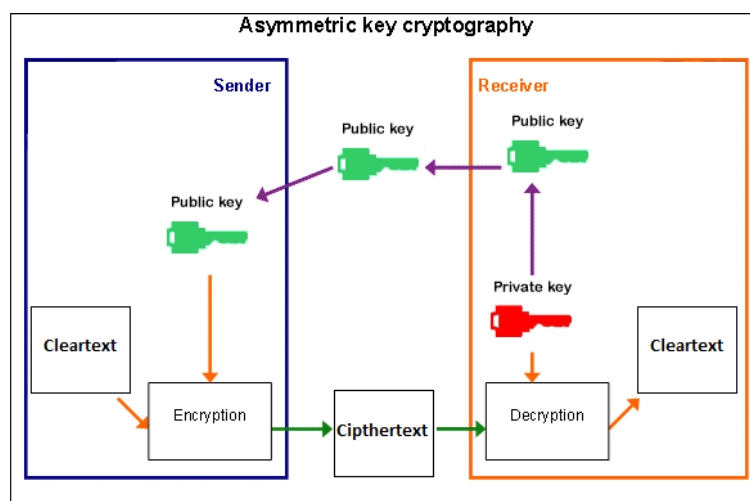
Some basic cryptographic terms are used in this report. This section contains a short explanation of asymmetric and symmetric cryptography, and the differences between them. Certificates, signatures and hash algorithms will also be mentioned. This report will not go into any deeper detail regarding the inner workings of these algorithms.

### 2.0.1 Asymmetric cryptography

In asymmetric cryptography two keys are used, a private key and a public key. The public key is used to encrypt messages and is available for everyone. The private key is used to decrypt messages and must only be known to the recipient of the message. Asymmetric cryptography is like a locked mailbox. Everyone can write letters and put them in the mailbox (encrypt the letters), but only the owner of the mailbox key can open the mailbox and receive the mail inside it (the private key). This principle is illustrated in figure 2.1.

The keys can also be used to sign and verify digital signatures. Here it should be noted that an asymmetric key should not be used for both encryption and signing. When encrypting, the public key is used, whereas when signing, the private key is used.

One example of asymmetric cryptography is RSA. RSA is the cryptosystem used in a TPM. In RSA the user picks two large prime numbers and computes  $N = pq$ . Then an exponent  $e$  is chosen which satisfies  $\gcd(e, (p-1)(q-1)) = 1$ . The  $N$  and  $e$  is the RSA public key. The private key is the numbers  $p$ ,  $q$  and  $d$  where  $d$  is calculated using the formula  $e * d \equiv 1 \pmod{(p-1)(q-1)}$ . The prime numbers  $p$  and  $q$  should be of size 1024-2048 bits. In a TPM RSA keys are used to encrypt, sign and store data as well as other keys. Figure 2.1 illustrates the principle of asymmetric cryptography.



**Figure 2.1:** The principle of asymmetric cryptography.

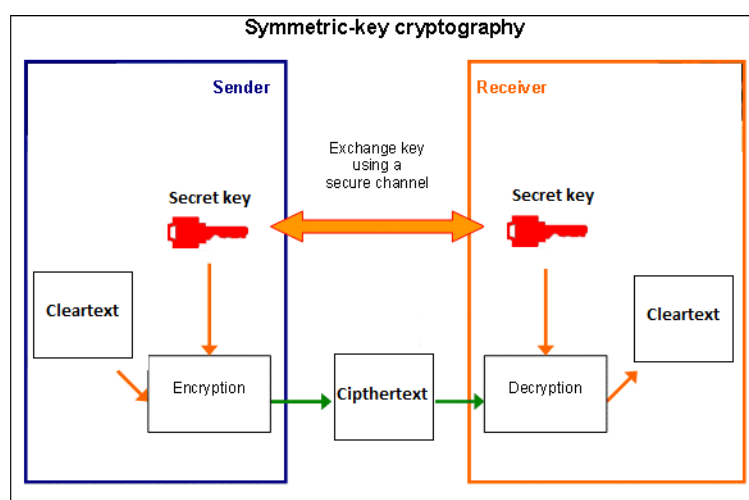
Asymmetric cryptography is typically very slow and should thus not be used to encrypt large amounts of data. Instead, symmetric cryptography is used for bulk encryption [6].

## 2.0.2 Symmetric cryptography

In symmetric cryptography, the key is shared between the parties. That means that both the encrypter and decrypter must know the secret key. Symmetric cryptography is used to encrypt large amounts of data since the symmetric encryption algorithms allow for high performance [3].

TPM version 1.2 does not support native symmetric key creation and storage, but there are ways to store a symmetric key by encrypting it with an RSA key. To take advantage of the speed of symmetric encryption with the strength of asymmetric encryption, a symmetric key can be used to encrypt data and an asymmetric key pair can be used to encrypt the symmetric key. This is called a hybrid encryption [6].

As will be shown later, a TPM uses a hybrid encryption when binding data. One or many symmetric keys can be protected (encrypted) by a TPM binding key. The symmetric keys can then be used by the system to perform symmetric encryption. The binding key's responsibility is to keep the symmetric keys safe. Figure 2.2 illustrates the principle of symmetric cryptography.



**Figure 2.2:** The principle of symmetric cryptography.

### 2.0.3 Certificate

A digital certificate, or public key certificate, uses a third party called a certificate authority (CA) which is used to sign the user's public key with the user information. This creates a certificate for the user and his or her public key. Another user A can now receive the public key and check the certificate if the public key truly came from user A. If user B trusts the CA then he or she can now trust that the key truly belongs to user A [3]. In a TPM certificates are used to certify the EK so that the user truly knows that the target is a TPM and not a hostile target that claims to be a TPM in order to receive sensitive information.

### 2.0.4 Hash functions

A hash function is a function which takes arbitrary length bit strings as input and produces a fixed length bit string as output. This can be used as a digital fingerprint to check that data has not been changed. No matter if it is a small text file or a large application, the output of the hash function will still be of the same length. The minimum examples of cryptanalytic attacks the hash must be able to withstand is:

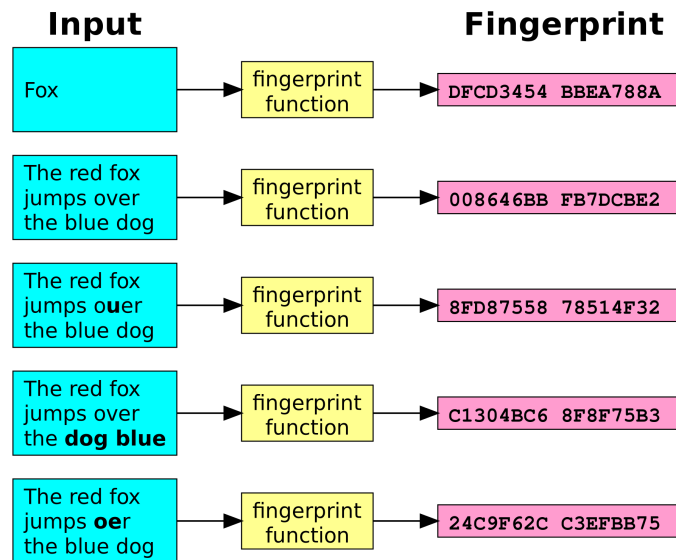
**Pre-image resistance:** Given a hash  $h$  it should be difficult to find any message  $m$  such that  $h = \text{hash}(m)$ .

**Second pre-image resistance:** Given a message  $m_1$  it should be difficult to find another message  $m_2$  such that  $m_1 \neq m_2$  and  $\text{hash}(m_1) = \text{hash}(m_2)$ .

**Collision resistance:** It should be difficult to find two different messages  $m_1$  and  $m_2$  such that  $\text{hash}(m_1) = \text{hash}(m_2)$ .

This leads to the property that if just a single bit of a large application is changed, then the entire hash output value will be different. This is called an avalanche

effect [3]. Figure 2.3 illustrates how a hash function is used as a digital fingerprint. Note how the fingerprint changes completely as the input text varies.



**Figure 2.3:** A hash function may be used as a digital fingerprint [1].

In Trusted Computing, hash functions are used to check the state of e.g. the boot loader or the BIOS. SHA-1 is the hash function used in TPM 1.2 and the output of this function is a 160-bit string.



### 2.0.5 Digital Signature

The private key of an RSA key pair can be used to digitally sign data. This signature can be used to prove that the data has not been changed and that the signer is in possession of the private key. Signing using RSA is performed by first hashing the data and then applying RSA to the hash value using the private key. The public key can then be used to verify the signature. Note that verifying a digital signature does not prove who actually signed the document but rather that the signer knows the private key of the RSA key pair. To answer the question of who really signed the document, the use of digital certificates is needed [6].

---

Trusted Platform Module

---

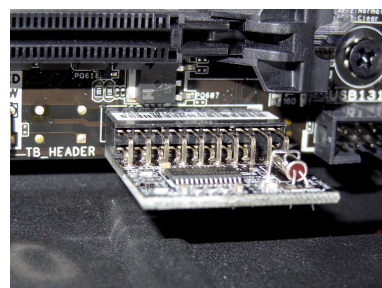
### 3.1 What is a TPM?

A Trusted Platform Module (TPM) is a microchip that is connected to the motherboard of a computer. The intent of a TPM is to create and store cryptographic keys, like RSA keys. Many laptops sold today are shipped with a built in TPM, for example recent versions of HP EliteBook, Dell Latitude, Sony Vaio and Lenovo ThinkPad [7].

Desktop PC motherboards usually does not come shipped with a TPM, but some of the more recent motherboards have support for a TPM that can be bought separately and connected to an onboard TPM connector. The TPM is then enabled and disabled in BIOS. Figure 3.1 displays how the actual TPM looks like when bought separately and when installed.



(a) The TPM chip



(b) Installed TPM

**Figure 3.1:** The TPM installed in a compatible motherboard.

This section provides a list over how the Trusted Platform Module have been used with various hardware and applications:

**Apple switching from PPC to x86:** In 2006, when Apple changed platform from the PowerPC to Intel's x86, they created a software called Rosetta. Rosetta is an emulator that allowed the user to run older software created for PowerPC based Apple computers on more recent Intel-based hardware. To control that Rosetta was only used on Apple's computers and not on any other Intel-based PC, Apple used a TPM. If the TPM was not present on the system, then this proved that the system was not a system created by Apple, and therefore Rosetta would not work [8] [9].

**Intel based Mac for developers:** When Apple switched platform from the PowerPC to Intel's x86, they released development kits for programmers to allow them to start programming software for the new Mac. This Mac was a regular PC that could run the new Intel-based Mac OS X. To prevent that the operating system was installed on regular PCs, the developer Mac used a TPM which needed to be present on the system to be able to install the new Intel-based Mac OS X. If this TPM was not present on the system, then it proved that the PC is not a valid development system and therefore is not permitted to run the Intel-based Mac OS X [8].

**Windows 8 boot process:** New laptops shipped with Windows 8 contains a TPM that is used to protect the user from threats that is activated outside of the operating system. The operating system is unable to detect and remove these threats with traditional antivirus software. One of the main function of the TPM is to measure the components loaded before the operating system, such as BIOS, kernel, boot loader etc. and to report the result to the user when the operating system is loaded. Laptops shipped with Windows 8 have an option called Secure Boot enabled by default in the UEFI BIOS. This means that if the user has not disabled secure boot, then the TPM is activated on every boot to measure the components loaded on the system [10].

**BitLocker:** BitLocker is a drive encryption feature that is created by Microsoft and can be used on newer versions of Windows. Here a TPM can be used to provide keys used for the encryption. BitLocker can also use a TPM to verify the integrity of early boot components and boot configuration data. This helps ensure that BitLocker makes the encrypted drive accessible only if those components have not been tampered with and the encrypted drive is located in the original computer [11].

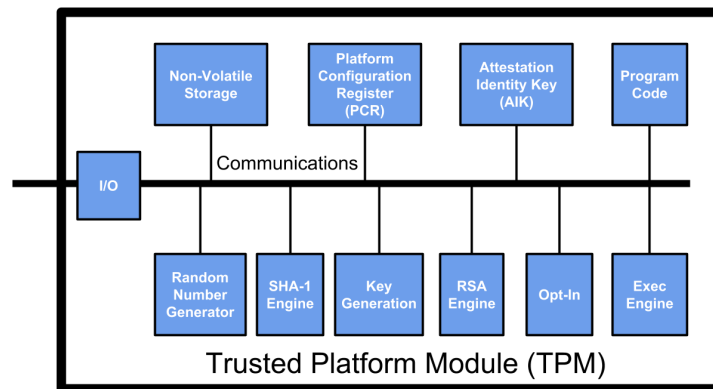
**Chromebooks:** All Chromebooks are shipped with a TPM. The TPM is here used for various reasons. For example to allow the Chromebook to be shared between users but still prevent them from being able to see each other's files. Each users files is encrypted by the system, and the keys are stored inside the TPM [12]. The TPM is also used to allow the users to log into the system while the system is offline. Once the user has logged in for the first time, a hash of the password is saved and encrypted by using the keys that are stored inside the TPM. When accessing the system in offline mode, the

TPM can now be used to authenticate the user in order to access the files. The TPM is also used in the boot process to measure the components loaded before the Chrome OS starts [13].

These five examples illustrates a few ways of how a TPM can be used in various security solutions. Apple has used TPMs to prevent their software from being installed on non-Apple systems. Microsoft and Google uses TPM to detect malicious code that is loaded prior to the operating system and Google also uses TPM in Chromebooks to authenticate the users and protect their files from each other.

## 3.2 TPM Architecture

The following figure 3.2 shows the building blocks of a TPM. The TPM mainly contains hardware support to generate random numbers and to do SHA-1 and RSA calculations.



**Figure 3.2:** The architecture of the TPM as described by TCG [2].

One block to take special notice of is the Platform Configuration Register (PCR).

### 3.2.1 Platform Configuration Register (PCR)

PCRs are registers that can store 20-byte hash digests. TPM version 1.2 contains 24 PCRs. The first 8 is for hardware digests and the following 8 is for software digests. The 20-byte hash digest is created by the SHA-1 hash algorithm, which as mentioned before creates an output string of a fixed size no matter of how large the string input was. The TPM creates this kind of hash digests and saves the values in the 8 first PCRs when booting up the system. If the workstation is powered on and the TPM calculates different hash digests as those that is saved in the PCRs, then this proves that the system has been changed and a threat that no anti-virus software would find has been detected. Checking and saving hash digests of the boot up process in PCRs is called a trusted boot.

Digests of software can also be recorded in the PCRs. This is useful for software attestation i.e. a test of what software has been loaded and as such a test of the state of the machine. By creating a hash digest of for example a banking security software and then trying to connect to an online banking service, then the bank can check the hash digest and see if the software is up to date. If the software is not up to date the bank is able to check the version and notify the user that the software needs to be updated.

Hash digests can only be saved in PCRs by the owner of the TPM through a special TPM command. This command has to be executed on the machine and cannot be executed remotely [14].

### 3.3 TPM and cryptographic keys

So far it has been mentioned that a TPM is a microchip connected to the motherboard that can provide cryptographic functions in hardware, and has been used by multinational companies like Apple, Microsoft and Google. This section will show in more detail what a TPM actually does.

#### 3.3.1 Types of keys

The TPM can create RSA keys to be used for various purposes. The keys created by the TPM can be migratable or non-migratable. Migratable keys may be transferred to another TPM, while non-migratable are locked to the TPM they were created on. When a parent key is migrated to another TPM all its children keys get migrated as well. The different types of keys are:

**Endorsement Key:** The Endorsement Key (EK) is a 2048-bit RSA key pair that is created during manufacturing of the TPM. This key is the only artifact in the TPM that can never be removed or changed. Even if the TPM is reset, the EK remains. The EK is used to identify the TPM as a valid TPM. If a message is encrypted with the public EK, then the only one able to decrypt the message is the TPM that contains the private EK. To make sure that the TPM which the user communicates with is in fact a TPM and not some hostile system claiming to be a TPM, the manufacturer also includes an EK certificate [15].

**Storage Root Key:** When a TPM is activated by the `takeownership` command or reset, it creates a key called Storage Root Key (SRK), and lets the user set a password for the TPM and the SRK. The SRK is a 2048-bit, non-migratable RSA key. The only way to delete the SRK is to reset the TPM. The SRK is the root in the TPM key tree, and all the other keys must be children of the SRK. In a system used by multiple users, it is only the administrator that owns the SRK [16].

**Storage Key:** A storage key is a 2048-bit RSA key that is used to encrypt data and to store other RSA keys. The SRK is one example of a storage key. Another normal storage key to be created is a platform migratable key.

The platform migratable key is a migratable storage key whose parent key is the SRK. This key can be used as a parent to all the different user-migratable keys, so when the platform key is migrated to another TPM, all users migratable keys is migrated as well. This can be useful for backup-purposes. In a system with many users, each root for each user can start with a storage key. For example the user A can create a migratable storage key that is a child of the platform migratable key, and a non-migratable storage key that is the child of the SRK. For user B, the same thing is done. This way all users get their own branches in the key tree structure. When encrypting data using a storage key the data is encrypted with an asymmetric cryptographic algorithm, which means that the encryption will be strong but slow, as mentioned before. It is therefore recommended to only encrypt small chunks of data using a storage key [17].

**Signature Key:** The signature key is an RSA key of size 2048-bits or smaller. This key is only used to sign user data. User data can be signed to validate that a message from a user in fact is from this specific user. The user uses his or her private key of the signature to sign the data, and then the receiver of this data can validate the signature with the public part of the signature key. If the signature is valid then this proves that it was signed with the private key. If the user needs to sign TPM data, then Identity keys is used instead [18].

**Binding Key:** The binding key is an RSA key used to store symmetric keys. As said before, encryption with asymmetric keys is secure but slow, while encryption with symmetric keys is fast but less secure. To combine the speed of symmetric encryption with the security of asymmetric encryption, the data can be encrypted by the system using the symmetric keys stored by the binding keys [19].

**Legacy Key:** The Legacy key is an RSA key that can be used for both encryption and signing. This is however not recommended and therefore it is only recommended to use legacy keys for backward compatibility with older systems. For signing use signature keys and for encryption use storage keys or binding keys [19].

**Identity Key:** An identity key is a non-migratable signing key that is used to sign PCRs when doing machine attestation, and to sign other keys as being non-migratable. The parent key of an identity key must always be the SRK. The identity key is used to sign certificates for the TPM keys [19].

### 3.3.2 Key storage

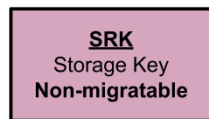
All keys created by a TPM cannot be stored in the TPM. The reason for this is that a TPM is a small, low cost chip with limited amount of storage, which means that there must be a secure way to store the keys created by the TPM outside the chip. This is achieved by using a technique TCG calls secure storage. Secure storage means that data and the keys may be stored securely on a hard drive by encrypting the private key of a key pair with another key pair's public

key, which means that all newly created key pairs must specify another key pair as their parent. This is achieved by creating a tree structure for the keys[20].

### Key hierarchy

The tree structure start with the SRK, the root of the tree. This section illustrates how a typical tree structure may be designed.

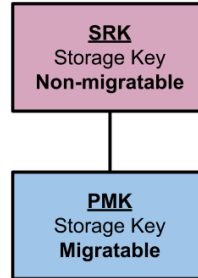
When the user launches the system with the TPM for the first time, the TPM is activated and the SRK is created. The only way to change the SRK is to reset the TPM so that a new SRK can be created. The SRK is stored in the non-volatile memory block in the TPM. Figure 3.3 shows the SRK as the root in the key tree.



**Figure 3.3:** The Storage Root Key

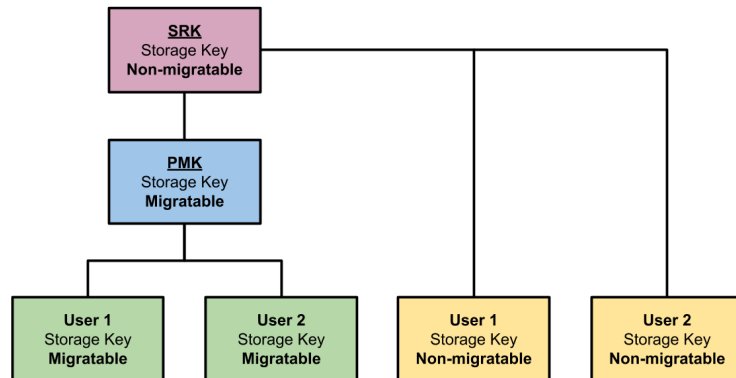
As mentioned earlier the keys can both be migratable and non-migratable. Therefore it may be a good idea to separate the migratable keys from the non-migratable keys in the tree.

Between the non-migratable SRK and the different user's migratable key should be a migratable storage key accessible by the administrator. The reason for this is that the administrator can make a backup of the keys, and if for example the system needs to be updated, then the administrator is able to migrate all the user's migratable keys in one go. This key is usually called the platform migratable key (PMK). The parent key of the PMK can be the SRK, which means that the private key of the PMK key pair is encrypted in a blob by the SRK public key. See figure 3.4. To decrypt a file by using the PMK the private key must first be decrypted by the SRK which requires the usage secret of the SRK



**Figure 3.4:** The Platform Migratable Key

This part of the tree is controlled by the system administrator. The next part is controlled by the different users of the system. Each user gets his own branch of the tree for his migratable and non-migratable key. Figure 3.5 illustrates how the tree might look when it contains two users.



**Figure 3.5:** A Keytree example

Figure 3.5 shows the benefits of storage keys. Since they are used to store other keys and data, they may be used to create a new branch for each user. This makes it very easy to add or remove users to the system. When a parent key is deleted then all its child keys are deleted as well. This means that only two keys need to be removed in order to remove a user from the system. The different users can use these storage keys as parents to their newly created keys. It is not recommended to make the tree deeper than necessary since the user would have to travel from the root down to the required key in order to decrypt the data which could be time consuming if the tree contains many nodes [21].



### 3.4 Key creation

One of the main purposes of a TPM is to create and store cryptographic keys in a secure environment. The reason for this is to extend the security by storing for example private RSA keys in an area where no one except the user is able to reach them, which is in the TPM hardware. If a user creates a RSA key pair by using for example OpenSSL, and then uses the public key to encrypt a message, then the message is kept secret only if the user is able to keep the private key secret. The keys created by a TPM is stored in a tree structure, which means that all new keys must connect with an already existing key in the TPM, a so called parent and child relation. The reason for this was discussed in section 2.5: Key storage.

### 3.5 Data encryption and decryption

When encrypting data, for example messages and large files, the most effective algorithm to use is a symmetric-key algorithm. However, the TPM does not support any symmetric cryptosystems. In order to use symmetric-key algorithms to encrypt data but still take advantage of the TPM's secure storage, a hybrid cryptosystem is used. With a hybrid cryptosystem the speed of symmetric-key encryption remains without the problem of keeping the encryption key secret.

The idea is that the user creates a symmetric key and uses this key to encrypt his data. Then an asymmetric private key is used, in this case a RSA key to encrypt the symmetric key. When using a TPM to encrypt data there are two encryption options:

**Bind data using a Binding key:** Bind data means that the encrypted data is bound to a specific RSA key. Since the data is bound to the key and not to the TPM itself, the data may be transferred to any TPM as long as the key is present. This means that the RSA key can be migratable and can be backed up. When binding data, the first thing that needs to be done is to create a symmetric key by using the random number generator built into the TPM. The next step is to store the symmetric key in a binding key. The encryption can then be done in software, by using a symmetric cryptographic algorithm like AES. This is the only realistic alternative if the user for example is going to back up the hard drive. The TPM has no cryptographic accelerator capabilities, so doing an encryption of this magnitude using a TPM would not be efficient.

**Seal data using a storage key:** Sealing data means that data is encrypted using RSA encryption. Unlike binding data which is bound to a specific RSA key, the data is bound to the TPM and the systems configuration when sealing data. In other words, data sealed by one TPM also has to be unsealed by the same TPM [19].

## 3.6 Public key signatures

A TPM may also sign messages by using a signature key. This can be used to authenticate the user since the signing is performed using the private key of the RSA key pair. A verified signature proves that the verifier has access to the private key, since only someone with knowledge of the private key could sign the message with a key that can be verified with the public key.

## 3.7 Authentication and attestation

The difference between authentication and attestation is [22]:

- Attestation: “*What is the state of machine X?*”
- Authentication: “*Is this machine X?*”

It is always important to be able to authenticate the TPM that the user communicates with to be sure that the TPM is not a hostile system claiming to be a TPM. It can also be important in some situations to know the state of the machine that contains the TPM. For example a banking service or a game server might only allow user access if the software is up to date.

There are a few techniques that can be used to perform a machine authentication and attestation. It can both be done on a high-level using TSS commands (more about this later) and on low-level by using only TPM commands. Since this thesis is about the TPM, the authentication and attestation techniques that will be shown uses only TPM commands.

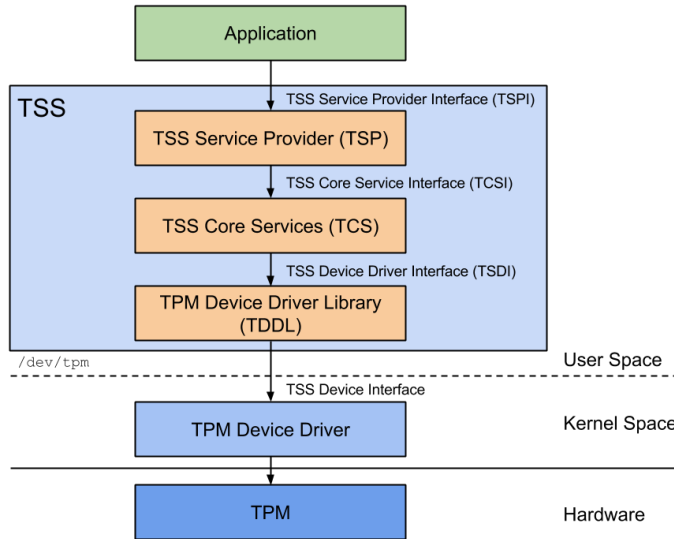
Here the authentication can be done by either signing data using a signature key, or decrypting data using a binding key. The attestation process can be done by either binding a storage key to a PCR value or quoting a PCR value using an identity key.

## 3.8 Communicating with the TPM

There are various ways of communicating with the Trusted Platform Module. When developing more complex applications it is preferred to use a high-level function library as oppose to the more limited low-level native commands of the TPM. The following section describes a common API implementation for developing high-level TPM applications.

### 3.8.1 TCG Software Stack (TSS)

When developing programs that use a TPM, the preferred way to communicate with the TPM is through the TCG Software Stack (TSS). TSS is an API where the programmer may use different layers depending on what kind of application that is going to be developed. The TSS architecture is illustrated in figure 3.6.



**Figure 3.6:** Architecture of the TCG Software Stack (TSS) [2].

When developing for embedded systems where the resources are limited and only basic TPM functionality is needed then the programmer can communicate with the TPM through the lowest layer, the TDDL (TCG device driver library). TDDL provides the user with a low-level API which can be used to run basic TPM functions like creating keys, unseal an encrypted file and calculate SHA-1 values to name a few. Only TPM functions that are implemented in hardware can be executed using TDDL. When communicating with the TPM through the TDDL layer, the programmer needs to use the TDDL API calls to send and receive TPM data [23].

Basic TPM commands are useful for low-level applications, but when developing more complex applications, high-level TPM functionality is typically required. For example when developing low-level applications the user needs to keep track of keys in their hierarchies and sessions. By using a high-level API, the user can let the software handle this instead. It provides the possibility to unload keys from the TPM when the storage is full [24]. This is where the TSS Core Service (TCS) is useful. TCS works almost like a software implementation of a TPM which means that software is used to extend the functionality of a TPM. For example it can perform key context swapping to manage the used storage in the TPM. When communicating with a TPM through the TDDL, the user can only run one command at the time. With the help of TCS, the system can queue the TPM commands that are going to be executed from many applications, and also prioritize the most important TPM commands.

Another benefit of TCS is that the application is not only limited to communicate with the TPM on the same system as itself, but it can also remotely communicate

with a TPM on another system through the TCP/IP protocol. This is possible by letting TCS on one system talk to the TCS on the other system. The TCS layer also converts the API requests into byte streams which can be understood by the TPM [25].

The top layer is called the TCG Service Provider (TSP). TSP is used to provide an object-oriented interface to every application on a system that uses a TPM which makes the code more structured and easier to understand. This is also useful because now the application can rely more on the TSP to perform most of the trusted functions provided by the TPM so the programmer can focus on the rest of the application. TSP also provides dynamic handles that allow for efficient usage of both the application's and TSP's resources. On low-level applications, this must be done by the user.

Many TPM applications may be used on a system at the same time, and there is one TSP for each applications. Since calls to the TPM are executed sequentially, the TCS is used to queue and direct the TPM calls through the TDDL [26] [27].

A company may choose to implement their own TCG Software Stack (TSS) since the specification is publicly available from TCG. TrouSerS is an open source implementation of the TSS that may be used by the public according to the Common Public License (CPL). One of the key objectives of this thesis is to evaluate if the TSS may be used with an emulated TPM. TrouSerS is the TSS of choice for this thesis.

### 3.8.2 TPM commands

This section describes the commands used by the TPM at the lowest level of the TSS to preform all the functions listed in chapter 3 [22].

**Key creation:** The TPM can create five types of keys. These keys are created by using two different TPM commands:

**TPM\_CreateWrapKey:** This command is used to create storage, binding, signing and legacy keys.

**TPM\_MakeIdentity:** This command is used to create Identity keys.

**Loading a key into the TPM:** Since the storage is limited in a TPM, the key is stored on the hard drive and loaded into the TPM when needed. The command to load a key into a TPM is: **TPM\_Loadkey**.

**Evicting a key from the TPM:** When the key is not needed anymore in the TPM, it has to be evicted from the TPM to make room for other keys that needs to be used. The command to evict a key is: **TPM\_EvictKey**.

**Data binding:** One particular thing about data binding is that the encryption is not done by the TPM. Instead the higher layers in the TSS is responsible for the encryption. Therefore the command for data binding is not **TPM\_Bind**, but rather: **TSS\_Bind**.

**Data unbinding:** When unbinding data, the TPM is needed because private keys are used to decrypt the data, and the private keys is only known by the TPM. The command to unbind data is: `TPM_Unbind`.

**Data sealing:** One difference between data binding and data sealing is that with data binding the data is bound to a specific TPM key, while with data sealing the data is sealed to a specific TPM. The data sealing is done by the TPM, so the command to seal data is: `TPM_Seal`.

**Data unseal:** When unsealing data, the user may require the password for the sealing key and for the data, and the PCR values may have to match. Since PCR values and password only can be checked by the TPM, the command to unseal data is: `TPM_Unseal`.

**Signature:** The TPM command to sign data is: `TPM_Sign`.

**Authentication:** Authentications can either be Signing-based or Decryption-based. The key used for Signing-based Authentication is a signature key and the key used for Decryption-based Authentication is a binding key. The TPM commands that is used is: `TPM_Sign` or `TSS_Bind` [28].

**PCR:** The command used to write a new 20 bytes digest to a PCR is: `TPM_Extend`. To calculate SHA-1 hash values and load into the PCRs can also be done by using the commands:

```

TPM_SHA1Start
TPM_SHA1Update
TPM_SHA1CompleteExtend

```

### 3.9 Developing TPM applications

As mentioned before when developing TPM application, TSS will most often be used. TCG has released a specification of the TSS and a package with C header files. It is however up to each vendor to develop their own TSS. One TSS that is free and open source is TrouSerS as mentioned before.

When writing TPM applications on a high level the TSP layer in the TSS architecture is used. This will provide a useful API for easy TPM programming and make the application responsible for key management, memory usage and error handling. A simple TrouSerS application typically looks like this [29]:

---

```
first.c
```

---

```

#include<stdio.h>
#include<string.h>
#include<stdlib.h>
#include<sys/stat.h>
#include<sys/types.h>
#include<tss/platform.h>
#include<tss/tss_defines.h>
#include<tss/tss_typedef.h>
#include<tss/tss_structs.h>
#include<tss/tspi.h>
#include<trousers/trousers.h>
#include<tss/tss_error.h>

#define DEBUG 1
// Macro for debug messages
#define DBG(message, tResult) { if(DEBUG) printf("(Line%d, %s) %s returned \
0x%08x. %s.\n", __LINE__, __func__, message, tResult, \
(char *)Trspi_Error_String(tResult));}

int main(int argc, char **argv)
{
    TSS_HCONTEXT hContext=0;
    TSS_HTPM hTPM = 0;
    TSS_RESULT result;
    TSS_HKEY hSRK = 0;
    TSS_HPOLICY hSRKPolicy=0;
    TSS_UUID SRK_UUID = TSS_UUID_SRK;
    // By default SRK is 20 bytes of 0

```

```

// takeownership -z
BYTE wks[20];
memset(wks,0,20);
// At the beginning
// Create context and get tpm handle
result =Tspi_Context_Create(&hContext);
DBG("Create a context\n", result);
result=Tspi_Context_Connect(hContext, NULL);
DBG("Connect to TPM\n", result);
result=Tspi_Context_GetTpmObject(hContext, &hTPM);
DBG("Get TPM handle\n", result);
// Get SRK handle
// This operation need SRK secret when you takeownership
// if takeownership -z the SRK is wks by default
result=Tspi_Context_LoadKeyByUUID(
                                hContext,
                                TSS_PS_TYPE_SYSTEM,
                                SRK_UUID,
                                &hSRK
                                );
DBG("Get SRK handle\n", result);
result=Tspi_GetPolicyObject(hSRK, TSS_POLICY_USAGE, &hSRKPolicy);
DBG("Get SRK Policy\n", result);
result=Tspi_Policy_SetSecret(hSRKPolicy,TSS_SECRET_MODE_SHA1,20, wks);
DBG("Tspi_Policy_SetSecret\n", result);

// INSERT TPM COMMANDS HERE:

...

// END OF APP
// Free memory
result = Tspi_Context_FreeMemory(hContext, NULL);
DBG("Tspi Context Free Memory\n", result);
result = Tspi_Context_Close(hContext);
DBG("Tspi Context Close\n", result);
return 0;
}

```

### 3.9.1 The basic TSPI commands

These are the basic steps that are needed to connect to the TPM. It requires the TSPI commands:

**Tspi\_Context\_Create:** This command tells the TSP to generate a new context handle for the application. This command must be used since there can be more TPM application running on the same machine, so this context is used by TSP to keep track of each application. It also provides functions for resource management and freeing of memory. The handle to the created

context object is saved in  
`TSS_HCONTEXT_hContext` [30] [31].

**Tspi\_Context\_Connect:** This command connects the TSP context to a TCS provider. The local TCS is represented by `Null` as seen in the code. If the TPM is not activated on the system then this command will return `TSS_E_NO_CONNECTION`. If the TPM is not activated then it is recommended to end the program after this command and activate the TPM before executing again [32] [33].

**Tspi\_Context\_GetTpmObject:** After a TCS connection is made in the command `Tspi_Context_Connect`, then this command is used to retrieve a handle to the TPM object [33].

**Tspi\_Context\_LoadKeyByUUID:** This command creates an object of a key and loads it into the TPM. All information about the key is handled by the TCS. In the initial step the SRK is loaded into the TPM with this command.

**Tspi\_GetPolicyObject:** This command is used to locate the current authorization policy associated with the context [34].

**Tspi\_Policy\_SetSecret:** This command is used to create a policy object of the authorization data (owner password), and returns a handle for this object [35].

When the TPM is initialised the user may test its functionality by running:

**Tspi\_TPM\_GetRandom:** Return a random number of a specified size.

Freeing allocated memory is done with the command: `Tspi_Context_FreeMemory`

Last command to be used is the command that closes the context: `Tspi_Context_Close`

The TPM application can easily be compiled with GCC. The command is [29]:

```
gcc trousersApp -o trousersApp.c -ltspi -Wall
```

In Appendix B are two examples where the usefulness of TSS is demonstrated. The first program will print a list of the PCR content in the terminal. A capture of this is displayed in figure 3.7.



**Figure 3.7:** Content of the PCR printed in the terminal.

```
swichblade@swichblade-1015PN: ~/Downloads/programming_with_trousers-master/StartUp$
swichblade@swichblade-1015PN: ~/Downloads/programming_with_trousers-master/StartUp$ ./keyRead2
returned 0x00000000. Success.
(Line92, main) Tspi Context Close
returned 0x00000000. Success.
swichblade@swichblade-1015PN: ~/Downloads/programming_with_trousers-master/StartUp$ ./keyRead2
returned 0x00000000. Success.
(Line43, main) Create a context
returned 0x00000000. Success.
(Line45, main) Connect to TPM
returned 0x00000000. Success.
(Line47, main) Get TPM handle
returned 0x00000000. Success.
(Line52, main) get SRK handle
returned 0x00000000. Success.
(Line54, main) Get SRK Policy
returned 0x00000000. Success.
(Line56, main) Tspi_Policy_SetSecret
returned 0x00000000. Success.
(Line64, main) Tspi Context_GetRegisteredKeysByUUID
returned 0x00000000. Success.
Registered key 0:
Version : 1.1.0.0
UUID : 00000000 0000 0000 00 00 000000000001
parent UUID : 00000000 0000 0000 00 00 000000000000
auth : YES
vendor data : (0 bytes)

(Line76, main) Tspi Context Free Memory
returned 0x00000000. Success.
(Line78, main) Tspi Context Close
returned 0x00000000. Success.
swichblade@swichblade-1015PN: ~/Downloads/programming_with_trousers-master/StartUp$
```

**Figure 3.8:** A list of all the keys loaded in the TPM.

A list of all TSPI commands can be found at the TSS specification [36].

## 4.1 Chosen hardware

### 4.1.1 Raspberry Pi

Raspberry Pi is a single board computer developed by the Raspberry Pi foundation. It features an ARM CPU running at 700 MHz and 512 MB of RAM. Combined with its low power usage it makes for an ideal platform to run software intended to emulate the Trusted Platform Module. Raspbian Wheezy is the OS of choice in this thesis. Based on the Debian Linux distribution, it seemed like the ideal platform for installing and using a software based TPM Emulator on.

The Raspberry Pi was developed to be used in schools to teach students about computer science and over 2.5 million units have been sold [37].

Combining a well-known device like Raspberry Pi with a relative unknown technique like the TPM may contribute to the interest in Trusted Computing and perhaps even inspire students to create new TPM applications by using the result from this thesis. Raspberry Pi Revision B shown in figure 4.1 was used in this thesis.



**Figure 4.1:** A Raspberry Pi Revision B running the IBM software TPM with remote access over TCP/IP.

## 4.2 Software

A variety of software were used in this thesis in order to test and evaluate TPM functionality. Some of the most frequently used software are listed here.

**Ubuntu 14.04** was the Linux distribution that was used to compile source code and to test and develop the thesis. However, any version of Ubuntu could have been used.

**Raspbian Wheezy** is an operating system developed for Raspberry Pi based on the Debian "Wheezy" Linux distribution. Raspbian is a popular choice among the existing operating systems for Raspberry Pi and it uses the same package manager as the popular Ubuntu Linux distribution which is also based on Debian. These were the key factors that made Raspbian the operating system of choice for the TPM on Raspberry Pi solution.

**WinSCP 5.5.6** (Windows Secure CoPy) is a SFTP, SCP and FTP client for Windows that was used to transfer files to and from the Raspberry Pi. This application made it easy to transfer files from a PC to the Raspberry Pi through SSH.

**PuTTY 0.63** is the serial console that were used on Windows to connect to the Raspberry Pi through SSH.

**VMware Player 7.0.0** has been used in the thesis to virtualize Ubuntu on a Windows machine. This proved very handy for testing the various TPM emulator

features.

### 4.3 Evaluating TPM emulators

At the start of this thesis there existed two TPM emulators that could be suitable for use as a learning environment. These two software TPMs were identified as:

- “TPM Emulator” (referred in this report as the *ETH TPM Emulator*)
- “Software TPM” by IBM (referred in this report as the *IBM software TPM*)

The first emulator was created by Mario Strasser at the Department of Computer Science, Swiss Federal Institute of Technology Zurich (ETH). This emulator had been released as an open source solution under the license GNU GPL (General Public License). This emulator was fairly well documented and it seemed to have been popular among users interested in a software TPM solution as it was linked to from the TCG website [38].

The second emulator was created by IBM and is targeted toward application development, education, and virtualization. This emulator is provided “as is” with an open source code.

Appendix C lists the license terms for the IBM software TPM. This emulator was somewhat less documented than the first emulator but still proved to be relatively easy to setup and use [3].

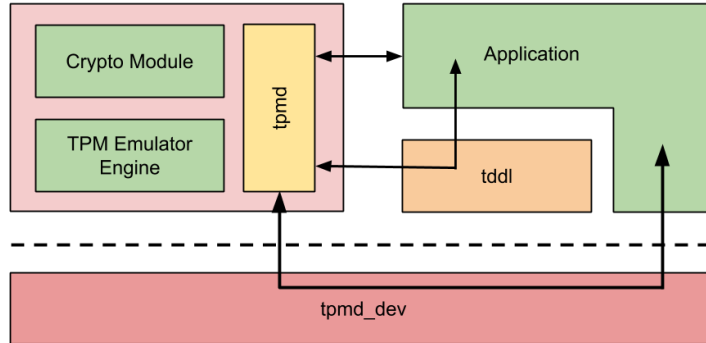
#### 4.3.1 ETH TPM Emulator

The initial choice fell on the ETH TPM Emulator. This emulator provides full support for TPM 1.2 and is portable due to its open source implementation. This solution actually also contains the MTM - Mobile Trusted Module for use in embedded devices. MTM was however not considered in this thesis [39] [40].

During this thesis, the ETH TPM emulator were in revision 0.7.3 and was last updated in 2011. The figure 4.2 shows how the ETH TPM emulator engine communicates with its surroundings. The compiled ETH TPM Emulator application runs a daemon in user space that is called `tpmd`. In order to directly communicate from an application to the ETH TPM Emulator daemon one needs to load the kernel module `tpmd_dev` into the Linux kernel.

The first experiments with the ETH TPM emulator were conducted on Ubuntu with full system access. The source code for the ETH TPM Emulator was downloaded and compiled without any major problems. It took a while in order to figure out how the emulator actually worked and how it was supposed to respond when interfaced with.

The first experience of running actual TPM commands was through the TPM



**Figure 4.2:** The architecture of the ETH TPM Emulator.

Device Driver Library (TDDL). A test application that utilized TDDL was included in the ETH TPM Emulator source code tree. This application allowed the user to open a connection to the simulated TPM device and send low-level TPM commands.

The ETH TPM Emulator provided the test application `test_tddl` for testing of low-level commands to the ETH TPM Emulator engine. The TDDL interface is the standard interface that applications use for communicating directly with the TPM. This interface is part of the TSS library and is in fact used by the TSS stack to talk to the TPM. Although this interface is available for use by applications, its direct use is typically best avoided [40].

Test of the ETH TPM emulator by running the `TPM_GetCapability` command:

---

```
// TPM_GetCapability command:
BYTE getcapability[] =
{
    0, 0xC1,          // TPM_TAG_RQU_COMMAND
    0, 0, 0, 18,      // blob length, bytes
    0, 0, 0, 101,     // TPM_ORD_GetCapability
    0, 0, 0, 6,       // TCP_CAP_VERSION
    0, 0, 0, 0        // no sub capability
};
```

---

If the ETH TPM emulator daemon is running, the test application could be used to test various TDDL commands. The result of the `TPM_GetCapability` command is displayed below. The ETH TPM emulator responds with a result string of as many bytes as the command.

---

```
pi@raspberrypi ~/tpm/tpm_emulator-0.7.3/build/tddl $ sudo ./test_tddl
```

```
Transmit: 00 c1 00 00 00 12 00 00 00 65 00 00 00 06 00 00 00 00
Result:    00 c4 00 00 00 12 00 00 00 00 00 00 00 04 01 01 00 00
```

```
Success, Bye!
```

```
pi@raspberrypi ~/tpm/tpm_emulator-0.7.3/build/tddl $
```

---

The output of a `test_tddl` application written to test TDDL [41]. An install script was written in order to automate the download and compilation process. This made it easy to build and install the emulator on a Debian system.

A simple install script for the ETH TPM Emulator:

---

```
installTPMEmulator.sh
```

---

```
#!/bin/bash

# required stuff
#apt-get install cmake libgmp3-dev

# versions
export TPM_V=tpm_emulator-0.7.3

# install path
export TPM_INSTALL_PATH=/...

cd $TPM_INSTALL_PATH
mkdir $TPM_V
cd $TPM_V

# download and extract
wget http://sourceforge.net/projects/tpm-emulator.berlios/files/$TPM_V.tar.gz
tar -zxvf $TPM_V.tar.gz

# build
cd $TPM_V
mkdir build
cd build
cmake ../
make
make install
```

```
# clean up
cd ../../
find . -name "
```

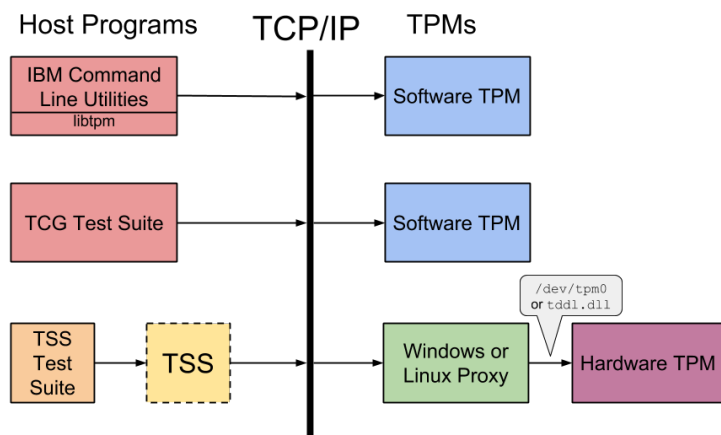
### 4.3.2 Software TPM by IBM

Another interesting solution for emulating the Trusted Platform Module is the IBM software TPM. The following description is derived from the website of the IBM software TPM [3].

The IBM software TPM is targeted toward application development, education, and virtualisation. The intent is that an application can be developed using the software TPM and then run on a hardware TPM.

Some advantages of this approach:

- In contrast to a hardware TPM, the emulator runs on many platforms and is generally faster.
- Application errors are easily reversed by simply removing the TPM state and starting over.
- The debugging abilities as well as the supporting TPM demonstration utilities helps to understand how a TPM works.



**Figure 4.3:** The architecture of the IBM Software TPM [3].

### Software TPM modules

The IBM software TPM package contains four modules for use with TPM development and testing as displayed in the IBM software TPM architecture figure 4.3. These modules are [3]:

- Software TPM
  - Current to TPM 1.2 revision 116 with updates to 117
- libtpm
  - libtpm supports the Utilities. It compiles to a shared object or DLL. It provides a low-level API to TPM command ordinals.
- Utilities
  - Utilities is a number of command line programs. Each typically maps directly to TPM command ordinals, but some support authorization session setup or context saving.
- TPM Proxy
  - The TPM Proxy acts on one side as a TCP/IP socket server and on the other side as an interface to the TPM device driver. It passes commands from the socket to the device driver and passes responses from the device driver to the socket.

This description seemed to fit the purpose of this thesis well. The IBM software TPM uses TCP/IP sockets as a communication interface. In practice this would mean that this emulator was ready to establish a connection between two systems, one running the actual emulator and the other running the communicating application [42].

The build and install process for the IBM software TPM is very straightforward, just download, extract and compile using the appropriate `make` file.

The following is a simple install script for the IBM software TPM.

---

`installIBMEulator.sh`

---

```
#!/bin/bash

# required stuff
#apt-get install libtool automake libssl-dev openssl

# versions
export TPM_V=tpm4720

# install path
export TPM_INSTALL_PATH=...
```



```
cd $TPM_INSTALL_PATH
mkdir $TPM_V
cd $TPM_V

# download and extract
wget -c http://downloads.sourceforge.net/project/ibmswtpm/$TPM_V.tar.gz
tar -zxvf $TPM_V.tar.gz

# build
cd tpm
make --file=makefile-ts
cd ../libtpm
./autogen
./configure
make

# clean up
cd ..
find . -name "
# set env vars
export TPM_PATH=$TPM_INSTALL_PATH/$TPM_V/tpm
export TPM_PORT=6543
```

---

Since the ETH TPM Emulator seemed to be the most well documented emulator and it acted like an actual TPM due to the kernel module the decision was made that this emulator would be the primary choice. Information were found on how to use the emulator together with the open source TSS TrouSerS which made the emulator even more interesting.

The ETH TPM Emulator had to be used with root access and was somewhat difficult to debug. The ETH TPM Emulator could however be reconstructed from a daemon application to a more basic foreground application. This would make the debugging of the source code easier since the application would simply stop executing at the exact point of the occurred error. No further research were made on this since focus was changed to the IBM software TPM.

## 4.4 Running the emulator on Raspberry Pi

The following section describes efforts and problems that were encountered when trying to make the TPM emulators run on Raspberry Pi.

### 4.4.1 ETH TPM Emulator

In order to install all components of the ETH TPM Emulator on any system, root access is needed. This caused some concern for using the ETH TPM Emulator

in an educational environment. On the Raspberry Pi this caused no major issues since root access is given just by typing `sudo` (without any password) on the default Raspbian Wheezy installation.

The ETH TPM Emulator was successfully run on the Raspberry Pi. Low-level TPM commands were tested through the provided TDDL test application.

However there were some issues when trying to establish communication between applications and the TPM Emulator engine. The main reason for these problems were the fact that efforts to build and initialise the `tpmd_dev` kernel module on Raspbian proved to be fruitless. The error was traced to conflicting versions of the Linux headers used to compile the module. Despite efforts to build the seemingly correct Linux headers for Raspbian the `tpmd_dev` module could not be probed on Raspbian. This problem made it difficult to test the communication between an application and the TPM emulator engine.

---

```
pi@raspberrypi
~/tpm/tpm_emulator-0.7.3/build/tpmd_dev/Linux $ sudo modprobe tpmd_dev
ERROR: could not insert 'tpmd_dev': Exec format error
pi@raspberrypi
~/tpm/tpm_emulator-0.7.3/build/tpmd_dev/linux $ dmesg |tail -n 1
[171963.273854] tpmd_dev: disagrees about version of symbol module_layout
```

---

When the ETH TPM Emulator did not work on Raspbian, further attempts were made to try it on another Raspberry Pi Linux distribution. Efforts were made to build Gentoo for Raspberry Pi, however it proved to be somewhat time consuming. When Gentoo was up and running the ETH TPM Emulator could be installed by using the Gentoo package manager Portage. The ETH TPM Emulator was marked as masked which means that the package had been blocked for installations and had to be unmasked in order to be installed. After this was done, the ETH TPM Emulator, TPM-tools, OpenSSL and TrouSerS were installed.

However the ETH TPM Emulator did not work at all on Gentoo. The kernel module `tpmd_dev` could be loaded using `modprobe`, but no TPM command could be executed. Not even the provided test applications worked. The error messages given by the ETH TPM Emulator did not give any clues on how to solve the problem. When changing to the IBM software TPM there was no difficulty to compile and run the emulator.

#### 4.4.2 Software TPM by IBM

The Software TPM by IBM was built and installed on Raspbian without any problems. Since the Software TPM by IBM had built-in support for network

communication and did not need system root access it was considered to be the primary choice of TPM emulator for this thesis.

## 4.5 Communicating remotely to Raspberry Pi

These options were investigated in order to communicate with the TPM environment running on the Raspberry Pi:

- Remotely (no physical connection)
- Using a network cable between Raspberry Pi and workstation to communicate through LAN.
- Using a USB cable between Raspberry Pi and workstation to enable serial communication.
- Websockets
- Other solutions like SSH or CGI.

None of these options were considered efficient when using the ETH TPM Emulator that did not provide any built-in network communication. However, by using the Software TPM by IBM, these problems were eliminated. The Software TPM by IBM provided native TCP/IP network communication which made it easy to establish a connection between the emulator running on the Raspberry Pi and the host computer. All focus was hereby shifted to the Software TPM by IBM.

## 4.6 Preparing to create exercises

A goal of this thesis is to provide an easy and effective way of giving students a clear overview of the Trusted Platform Module and its applications without having to dive too deep into technicalities. This means giving the students the opportunity to write their own TPM applications by using a high-level API. The IBM software TPM came bundled with a set of command line utilities that could be used to demonstrate various TPM commands.

These utilities demonstrated how to do all basic TPM tasks like key creation, key loading, key evicting, PCR writing and reading, data binding and sealing to name a few. The developer of the IBM software TPM, Kenneth Goldman also confirmed through mail correspondence that these utilities may be used for educational purposes when no C programming was desired.

The demo command line utilities were used when creating the lab manual in order to allow users of different TPM and programming knowledge to use them. For a deeper understanding of how a TPM works, these exercises can be used as an overview of a TPM before diving deeper into its functionality.

## 4.7 TSS on Raspberry Pi TPM

When the IBM software TPM was functional on the Raspberry Pi, focus was shifted to testing the TCG Software Stack (TSS). TrouSerS (The open-source TCG Software Stack) is an open source implementation of TSS. This solution seemed to be most suitable for this thesis since it is fairly well documented. The next step was to investigate if the IBM software TPM supported TrouSerS.

By looking in the installation instructions it was evident that the IBM software TPM and TrouSerS can be used together since the installation package contained descriptions on how to setup TrouSerS. The IBM software TPM website also provided a description on how to setup TrouSerS. One problem was that the install file instructions and the website provided different settings. While the install file instructions described the setup steps as:

---

Examples - varies with TrouSerS install path

```
> export LD_LIBRARY_PATH=/root/trousers-0.3.1/src/tspi/.libs
```

Add to /usr/local/etc/tcsd.conf:

```
remote_ops = seal,unseal,registerkey,unregisterkey,loadkey,createkey,sign,
random,getcapability,unbind,quote,readpubek,getregisteredkeybypublicinfo,
getpubkey,selftest
```

Must be owner/group/mode tss/tss/0600

```
> export TESTSUITE_OWNER_SECRET="ownerAuth"
> export TESTSUITE_SRK_SECRET="srkAuth"
> /root/trousers-0.3.1/src/tcsd/tcsd -f > tss.log 2>&1
> ./tpmbios
> ./createek
> ./takeown -pwdo ownerAuth -pws srkAuth

./tsstests.sh -v 1.2 &> logfile
```

---

The IBM software TPM website described the steps like this [3]:

---

Begin provisioning the TPM using the SW TPM utilities as per the INSTALL instructions.

Additional steps are required if the TPM is disabled or deactivated.

```
> tpmbios                                (each time the TPM is started)
> createek                              (only required once)
> ./nv_definespace -in ffffffff -sz 0    (only required once)
```

DO NOT take ownership using the SW TPM utilities.

Running tcspd:

Set the environment variable TCSD\_TCP\_DEVICE\_PORT to match the TPM's TPM\_PORT socket number. As root:

```
# /usr/sbin/tcpsd -e -f
```

---

Different approaches had to be tested since the instructions did not match. The instructions provided in the installation package did not work. The problem is due to the fact that the IBM software TPM runs on another machine where TrouSerS can't find it. The error message states **TCSD ERROR: Could not find a device to open!**

The other approach found on the website was then tested, but to no avail. The IBM software TPM needed to be cleared since the instructions states that no ownership should be taken by using the software TPM tools. According to this instruction the environment variable TCSD\_TCP\_DEVICE\_PORT should be set to match the environment variable TPM\_PORT. Therefore this was set to port number 6543 which is the same as TPM\_PORT.

Next step was to run the command `tcspd -e -f` with full system access. When executing `tcspd` this time, the option `-e` was used in order to establish a connection to the IBM software TPM through TCP.

TrouSerS needs to know the location of the IBM software TPM in order to be able to execute. These settings are enabled in `/usr/local/etc/tcpsd.conf`:

---

```
remote_ops=seal,unseal,registerkey,unregisterkey,loadkey,createkey,
sign,random,getcapabiliy,unbind,quote,readpubek,
getregisteredkeybypublicinfo,getpubkey,selftest
```

---

The environment variable TCSD\_TCP\_DEVICE\_PORT has to be set since this tells TrouSerS at which port the TPM will listen to for commands. The option `-e` also has to be set when executing `tcspd` since this tells TrouSerS that the connection to the TPM is going to be made over TCP.

Since it could not be found how the TPM destination is specified, the next step was to try to get TrouSerS to communicate with the TPM when it runs locally on the PC. When the software TPM is started and communication is established, the next step is to connect TrouSerS to the TPM. Since the port 6543 is used for communication over TCP by the TPM, it is this port number the variable TCSD\_TCP\_DEVICE\_PORT is set to. The communication still did not work which raised the question about additional missing settings.

Here several different approaches were tested. From recompiling the TPM with a different makefile, to trying different versions of TrouSerS and changing configurations in `tcsd.conf`, but to no avail. After some additional searching, this text was found in TrouSerS README [43]:

---

README.txt

---

If you're attempting to make the TCS Core Services daemon communicate with a software TPM through TCP, you must call it using the `-e` option.

```
# /usr/local/sbin/tcsd -e
```

The default values for hostname, port and UN socket device path are "localhost", "6545" and "/var/run/tpm/tpmd\_socket:0". It will search for the IN socket device, then for an UN socket one, and then for the real TPM in this order. The default values match with the current open source project required values, if for instance case you need to set values of your choice, the environment variables for them are `TCSD_TCP_DEVICE_HOSTNAME`, `TCSD_TCP_DEVICE_PORT` if using an IN socket and `TCSD_UN_SOCKET_DEVICE_PATH` if running an UN socket.

---

This reveals two useful settings. The variable `TCSD_TCP_DEVICE_PORT` is set to 6545 by default and the environment variable called `TCSD_TCP_DEVICE_HOSTNAME` specifies the TPM location. This is set to `localhost` by default which refers to the current machine TrouSerS is used on.

Here it was suspected that TrouSerS for some reason does not check the new value of `TCSD_TCP_DEVICE_PORT` and instead sets it to its default value which is 6545. To try this theory the software TPM was initialised with environment variables `TPM_PORT` and `TPM_SERVER_PORT` set to 6545. This time when executing TrouSerS with the command `sudo tcsd -e -f` it worked to start the `tcsd` server.

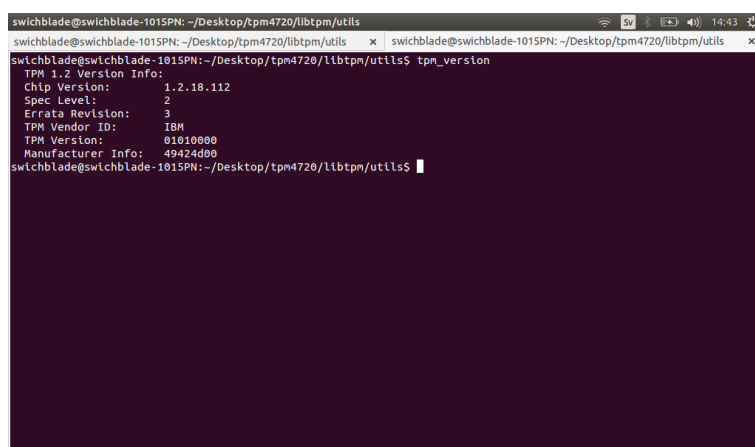
To see if TPM commands could be sent to the TrouSerS the commands that is included with `tpm_tools` were tried. The command `tpm_version` outputs:

This indicates that it received information from the TPM. Now when it was possible to use TrouSerS with the software TPM locally, the next task would be to connect to a TPM from another machine other than the one where TrouSerS is installed.

Now the problem remained that TrouSerS does not check the environment variables

`TCSD_TCP_DEVICE_PORT` and `TCSD_TCP_DEVICE_HOSTNAME`. This could be tested by declare some random values like:

```
export TCSD_TCP_DEVICE_PORT=777
```



```

swichblade@swichblade-1015PN: ~/Desktop/tpm4720/libtpm/utlis
swichblade@swichblade-1015PN: ~/Desktop/tpm4720/libtpm/utlis  x  swichblade@swichblade-1015PN: ~/Desktop/tpm4720/libtpm/utlis  x
swichblade@swichblade-1015PN:~/Desktop/tpm4720/libtpm/utlis$ tpm_version
TPM 1.2 Version Info:
Chip Version:      1.2.18.112
Spec Level:       2
Errata Revision:   3
TPM Vendor ID:    IBM
TPM Version:      01010000
Manufacturer Info: 49424d00
swichblade@swichblade-1015PN:~/Desktop/tpm4720/libtpm/utlis$

```

**Figure 4.4:** Result of command `tpm_version`

```
export TCSD_TCP_DEVICE_HOSTNAME=192.168.1.77
```

(Note that this IP-address is different from the PC and the Raspberry Pi.)

When executing `tcspd` with these values, TrouSerS still did connect to the TPM locally through port 6545. The next step was to examine why TrouSerS did not read these values and to figure out if there is any other way to specify the destination address. When searching the source code in `tddl.c` the following code was found:

---

```

tddl.c

```

---

```

if (getenv("TCSD_USE_TCP_DEVICE")) {
    if ((tcp_device_hostname =
        getenv("TCSD_TCP_DEVICE_HOSTNAME")) == NULL)
        tcp_device_hostname = "localhost";
    if ((un_socket_device_path =
        getenv("TCSD_UN_SOCKET_DEVICE_PATH")) == NULL)
        un_socket_device_path = "/var/run/tpm/tpmd_socket:0";
    if ((tcp_device_port_string =
        getenv("TCSD_TCP_DEVICE_PORT")) != NULL)
        tcp_device_port = atoi(tcp_device_port_string);
    else
        tcp_device_port = 6545;
}

```

---

This means that there is a fourth environment variable called `TCSD_USE_TCP_DEVICE` that can be set. However setting this variable did not solve the problem.

After many attempts, the problem with setting the environment variables persisted. The last attempt to make TrouSerS communicate with the software TPM

was to simply hard code the port and destination into the source code. Therefore, the variable

`tcp_device_hostname = "localhost"` was changed to `tcp_device_hostname = "192.168.1.160"` which was the IP addressed allocated to the Raspberry Pi and the port number remained the default. When recompiling the source code with this modification the communication between the software TPM and TrouSerS worked. This is not the optimal solution, but it works and was therefore used.

This shows that it is possible to have TrouSerS communicating with the Software TPM by IBM through TCP, allowing a learning environment with e.g., a Raspberry Pi being used as a TPM emulator. Also developers can take advantage of this when developing TPM applications using TSS.



## 5.1 Setting up the TPM environment

The IBM software TPM emulator was chosen for this thesis because it is easy to use, requires no root access and has built in support for TCP/IP communication. Since this thesis is supposed to be used in an educational environment it is imperative that the students are able to start, stop and reset the TPM without the risk of deleting important data.

## 5.2 Setting up the IBM software TPM on Raspberry Pi

Since the Raspberry Pi runs on a different hardware architecture than a PC (ARM for Raspberry Pi and x86 for PC), there are two choices when compiling the emulator. The emulator can be compiled natively or it can be cross-compiled. When compiling natively the source code is compiled on the target platform. This is a straightforward way to compile source code since the toolchain is installed on the target platform, however the compile time will most likely increase on a CPU-weak platform like the Raspberry Pi.

The other alternative is cross-compiling. Cross-compiling describes the practice of building binaries for a target hardware platform on another platform. In order to do this, a tool-chain is required on the compiling platform. Such a toolchain exists for the Raspberry Pi and the following code illustrates how to build by using cross-compilation:

```
make ARCH=arm CROSS_COMPILE=arm-linux-gnueabi- [39]
```

Since the software TPM is a small application the source code was compiled natively on the Raspberry Pi during this thesis.

The IBM software TPM contains three folders: `libtpm`, `tpm`, and `tpm_proxy`. The folder `tpm` contains the TPM emulator, the `libtpm` contains the TPM commands and `tpm_proxy` contains proxies which can be used to connect a TCP/IP based TPM interface to a hardware TPM device driver. This will not be needed in this project. On Raspberry Pi, the content of the folder `tpm` is needed. Since

the Raspberry Pi Linux distribution is running from an SD card the easiest way to copy the `tpm` folder to the card is by inserting the SD card in a PC with a card reader. A suitable place to copy the folder is for example to `/home/pi`

A software that can do cryptographic operations needs to be installed on Raspberry Pi before the TPM emulator can be compiled. The supported software is either OpenSSL or FreeBL. In this thesis OpenSSL was used, which can be installed on Raspberry Pi with the command:

```
sudo apt-get install openssl libssl1.0.0
```

Five makefiles with various options are provided:

1. `makefile-tpm` (standard TPM)
2. `makefile-en-ac` (TPM activated by default)
3. `makefile-ts` (TPM with `TPM_Init` disabled for TCG test suite)
4. `makefile-freebl` (TPM using the FreeBL crypto library)
5. `makefile-freebl-ts` (TPM using the FreeBL crypto library with `TPM_Init` disabled for TCG test suite)

In this thesis, the second option is used. Now the source code can be compiled by running `make`. This creates the binary `tpm_server` which starts the TPM emulator.

The last thing to do before the TPM emulator can be executed is to set two environment variables. Environment variables allows the user to specify strings that will be available to all applications. The strings can be locations of files or IP-addresses etc [44].

The environment variables that the TPM emulator will look for is called `TPM_PATH` and `TPM_PORT`. The `TPM_PATH` contains the location to the folder where the TPM non-volatile data will be saved, which means data that is saved even when the device is powered off. The `TPM_PORT` defines the port number where the TPM server will listen for commands when used over TCP. These two environment variables are set with the command:

```
export TPM_PATH=/home/pi/tpm_trousers/NV_Storage
export TPM_PORT=6545
```

This means that the location of the NV-data is `/home/pi/tpm_trousers/NV_Storage`, and the port 6545 will be used to send TPM commands. To set these environment variables on every Raspberry Pi reboot, a script file called `tpm.sh` was created and placed in the location `/etc/profile.d/`

These environment variables with these values will be set on each startup. Now the `tpm_server` can be executed by writing `./tpm_server` and no root access is required.

### 5.3 Building for other platforms

Both TPM emulators were developed to be quite portable. This was a handy feature since the user probably will use different platforms. To test the portability of the Software TPM by IBM once it was selected as the emulator of choice for this thesis it was built for Windows as well. In order to compile the source code for Windows, MinGW (Minimalist GNU for Windows) was used. The Win32 port of OpenSSL v1.0.1i was also needed. The correct path to the compiler and OpenSSL were set in the makefiles, after that the Software TPM and the demonstration utilities compiled without any problems. The IBM software TPM on Windows was tested by running the included test programs from another system running in VirtualBox. It worked without any problems. Figure 5.1 shows the running TPM emulator on Windows 8.1.

A simple Windows batch file for easy initialization of software TPM is shown below.

---

```
@echo off
title TPM SERVER

setx TPM_PATH "C:\PATH_TO_EXECUTABLE"
setx TPM_PORT "6543"

rm 00.permall
tpm_server.exe
```

---

```

TPM_SERVER
TPM_SHA1InitCmd:
TPM_SHA1_valist: Digesting 765 bytes
TPM_SHA1Update: length 765
TPM_SHA1FinalCmd:
TPM_SHA1_valist: Digest 79 9c da 6f
TPM_SHA1Delete:
TPM_PermanentAll_Store: Appending integrity digest
TPM_PermanentAll_NUStore: Require 785 bytes
TPM_NVRAM_StoreData: Io name permall
TPM_NVRAM_GetFilenameForName: For name permall
TPM_NVRAM_GetFilenameForName: File name C:\Users\Lifabook\Desktop\00.permal
TPM_NVRAM_StoreData: Opening file C:\Users\Lifabook\Desktop\00.permal
TPM_NVRAM_StoreData: Writing 785 bytes of data
TPM_NVRAM_StoreData: Closing file C:\Users\Lifabook\Desktop\00.permal
TPM_NVRAM_StoreData: Closed file C:\Users\Lifabook\Desktop\00.permal
TPM_Maininit: Creating global TPM instance 0
TPM_Maininit: Run limited self tests on TPM 0
TPM_LimitedSelfTestTPM:
TPM_SizedBuffer_Delete:
TPM_LimitedSelfTestTPM: Set testState to 1
TPM_Global_Delete:
mainLoop:
TPM_IO_Connect: Waiting for connection on port 6543...

```

Figure 5.1: Software TPM running on Windows 8.1

## 5.4 Setting up libtpm on client

With the TPM emulator running on Raspberry Pi the next step is to be able to communicate with it remotely through TCP. The functions needed for this task are located in the `libtpm` folder. The various build scripts are presented here.

`comp-sockets.sh` TCP/IP socket interface, standard TPM

`comp-chardev.sh` character device interface, standard TPM

`comp-serialp.sh` character device interface, TPM on serial port

`comp-unixio.sh` Unix domain socket interface, standard TPM

`comp-sockets.sh` is chosen by default and is the one that will be needed for this thesis. To build these tools the following commands is used:

```

./autogen
./configure
make

```

This fills the folder `libtpm/Utils` with binaries that represents the TPM commands. Some examples are:

```

createkey
loadkey
evictkey
listkeys
createek
listpubek

```

The use of these binaries are self-explanatory. Some preparations are necessary in order to be able to send these commands to the TPM emulator.

The environment variables called `TPM_SERVER_NAME` and `TPM_SERVER_PORT` needs to be set:

```
export TPM_SERVER_NAME=192.168.1.160
export TPM_SERVER_PORT=6545
```

`TPM_SERVER_NAME` is used to specify the location of the TPM emulator i.e the IP address of Raspberry Pi.

`TPM_SERVER_PORT` specifies the port which the command will be sent through. This port number must be the same as `TPM_PORT`.

After these environment variables are set the communication with the TPM emulator can be established. Every time the TPM emulator is started on Raspberry Pi, the program `tpmbios` must be executed on the PC with the command `./tpmbios`, which is located in `libtpm/utls`. This simulates the startup process for the TPM and makes the TPM emulator operational.

Note that no EK has been created. This is the first thing that needs to be done when using the TPM emulator for the first time. This is done with the command: `./createek`

## 5.5 Setting up TrouSerS

TrouSerS may be downloaded either directly from SourceForge or by a package manager on a Linux system. For Debian based distributions use:

```
sudo apt-get install trousers
```

If TrouSerS is downloaded from SourceForge then it has to be compiled and installed. This is done with the following commands:

```
sh bootstrap.sh
./configure --enable-debug
make
sudo make install
```

When setting up TrouSerS, the first step is to add the line:

```
remote_ops = seal,unseal,registerkey,unregisterkey,loadkey,
createkey,sign,random,getcapability,unbind,quote,readpubek,
getregisteredkeybypublicinfo,getpubkey,selftest
```

```
in /usr/local/etc/tcsd.conf
```

Next step is to initialise the environment variables. Since TrouSerS will communi-

cate with the TPM over TCP, the environment variables `TCSD_USE_TCP_DEVICE`, `TCSD_TCP_DEVICE_PORT` and `TCSD_TCP_DEVICE_HOSTNAME` will be used.

```
export TCSD_USE_TCP_DEVICE=true
export TCSD_TCP_DEVICE_PORT=6545 (default port)
export TCSD_TCP_DEVICE_HOSTNAME=192.168.1.160 (default IP for Raspberry Pi)
```

Now `tcsd` can be executed using the command `sudo /usr/local/sbin/tcsd -e -f`. If `tcsd` is started then a message that it waits for connections will be shown.

To test if the communication with the TPM works one may try the command `tpm_version` from the package `tpm-tools`. The next step is to run the command `tpm_takeown` to take ownership over the TPM. This command is also included in `tpm-tools` so it can be used by just writing `tpm_takeown` in the terminal. Here the user has to set both owner password and SRK password. Now the user controls the TPM and is able to create and execute TPM applications.

## 5.6 Test TrouSerS with the IBM software TPM

The user should now have a Raspberry Pi with the software TPM running and the following environment variables initialised.

```
TPM_PATH=/home/pi/tpm_trousers/NV_Storage
TPM_PORT=6545
```

The server is started with the command `./tpm_server`.

The user should have the IBM TPM utils and TrouSerS installed.  
The user should have set the environment variables

```
TPM_SERVER_NAME=192.168.1.160
TPM_SERVER_PORT=6545
```

and the IP address to the Raspberry Pi and the port number used to send information should be set in the TrouSerS source code. The IBM TPM utils command that always needs to be executed is `./tpmbios`. If no EK has been created it should be created with the command `./createek`

TrouSerS should be executed with the command `sudo tcsd -e -f`.

The user can now take ownership of the TPM with the command `tpm_takeown -z -y`. This will set a default, well known password to the owner and to the SRK.

## 6.1 Evaluating if thesis goal has been reached

The questions to be answered by this thesis project could be summarized as:

- Can a Raspberry Pi be used as a TPM server?
- Can the user communicate with the TPM environment remotely?

The first choice of TPM emulator used in this thesis was the ETH TPM Emulator. This emulator was successfully run on the Raspberry Pi and the low-level TPM commands were tested through the provided TDDL test app.

With this emulator there were some problems to establish communication between applications and the TPM Emulator engine. The main reason for these problems originated from the fact that the `tmpd_dev` kernel module could not be compiled on Raspbian due to conflicting versions of the Linux headers used to compile the module. Despite efforts to build the seemingly correct Linux headers for Raspbian the `tmpd_dev` module could not be probed.

The emulator chosen for this thesis was the IBM software TPM. Even if this emulator was designed in a different way than the ETH TPM Emulator found on the TCG website it turned out to have all features required for this project. First of all it was designed to run as a server/client, where the server is the TPM emulator and the client is the user. Secondly it could be used without root access which is more suitable for an educational environment. Building the IBM software TPM for Raspbian was unproblematic. The Raspberry Pi could hence be used to run a TPM server.

The second task was to investigate if the user could communicate with the TPM remotely. Communicating with the TPM directly is however not recommended. The correct way for a user to communicate with a TPM is through the TCG Software Stack (TSS). In this thesis the open source TSS implementation called TrouSerS were used. These are the TPM communication subtask questions:

- Is the user able to communicate with the TPM environment remotely by using the TPM emulator?

- Is the user able to communicate with the TPM environment remotely by using TrouSerS?

The IBM software TPM contained a folder containing a set of demo command line utilities which according to the documentation could be built to communicate with the TPM emulator over TCP. This could be done by just declaring two environment variables containing the IP address of the TPM server and the port number of which the TPM would send and listen for commands.

The second subtask was the actual challenge. TrouSerS had to be communicating with the TPM emulator remotely in order to be able to develop high-level TPM applications. TrouSerS could communicate with a TPM over TCP, but this did not confirm that it would work if the TPM emulator was located on a different machine. The lack of documentation and that no one else seemed to have tried this experiment before made this a trial and error challenge. There were instructions on how to make the IBM software TPM and TrouSerS work together but these instructions were conflicting. Since TrouSerS is an open source project, the source code could be investigated in order to see how TrouSerS establishes the connection. The IBM software TPM could eventually be configured to work with TrouSerS remotely. This makes us feel like the purpose of this thesis has been reached.

## 6.2 Creating a TPM learning exercise

The purpose of this thesis was not only to create a TPM learning environment but also to create some exercises that can be used to learn about the workings and applications of the Trusted Platform Module. In order to do this, one must first learn how the TPM works and how it communicates with its surroundings. Native TPM commands are very low-level and should best be avoided.

The purpose of the exercises is to illustrate what a TPM is capable of doing and to show what each TPM command is used for without having to actually program the TPM. The IBM software TPM comes bundled with a set of command line utilities that demonstrates various TPM commands and functionality.

This is useful for someone who is learning how to use the TPM. Before they start writing TPM commands they may try out a demonstration of the command to see what the command does and what parameters are needed. For example, when creating a key one has to specify a parent key to the new key. Without this parent key the new key cannot be created. This is demonstrated in the command line demonstration utilities.

## 6.3 Lack of documentation

Very little publically documented work involving TPM emulation has been done before this thesis. When trying to get the ETH TPM Emulator to work there were sometimes errors that no one seemed to have experienced before this thesis



project. Not many concrete examples were found on how to get TrouSerS to work with a TPM emulator, the same goes for programming the actual TPM. *Opensecuritytraining.info* proved to be a good website for learning about the TPM [45].

## 6.4 Was an expected solution achieved?

The goal of this thesis was to investigate if a TPM could be emulated on a dedicated hardware platform which would then act as the actual TPM that the user could communicate with. The result of this thesis is having a Raspberry Pi running a TPM emulator from IBM that can be connected to from any workstation over the Internet. This also allows for more than one user to work with the same TPM emulator from different locations.

The TPM emulator works with an open source TSS implementation called TrouSerS that is used to interface the TPM in order to write high-level applications.

The results of this thesis project seems to be consistent with the original intent of running a software TPM on a dedicated platform and to develop a laboratory manual in order to introduce Trusted Computing.

## 6.5 Further development

One of the most important reasons why the Raspberry Pi was chosen to act as a TPM server is the potential for further development. Some suggestions are stated here:

- Developing a web-based user interface. Since Raspberry Pi is a fully functional Linux-supported computer, a free webserver, for example Apache or nginx can be installed to allow the monitoring of the TPM to be more visual. It can show the content of the PCRs, show history over executed TPM commands etc.
- Have various TPM events communicating with the world via the GPIO of the Raspberry Pi, for example outputting commands via LCD display.
- Investigate if there is any easy way to separate the TCS part from the TSP part in the TSS. This would show a more correct way of how the TSS is used since one TCS is used for all application, and one TSP for each application is used. The TCS could then also be installed on the Raspberry Pi and the TSP could be installed on each workstation.

---

## Bibliography

---

- [1] Shows a typical fingerprint function at work. the fingerprints seen here (in hexadecimal format) are actually the first eight bytes (64 bits) of the sha-1 cryptographic hash functions of those text examples. <http://en.wikipedia.org/wiki/File:Fingerprint.svg>.
- [2] Trusted platform module (tpm) components. [http://www.trustedcomputinggroup.org/files/resource\\_files/AC652DE1-1D09-3519-ADA026A0C05CFAC2/TCG\\_1\\_4\\_Architecture\\_Overview.pdf](http://www.trustedcomputinggroup.org/files/resource_files/AC652DE1-1D09-3519-ADA026A0C05CFAC2/TCG_1_4_Architecture_Overview.pdf).
- [3] IBM. Software tpm introduction. <http://ibmswtpm.sourceforge.net/>.
- [4] TCG. Trusted platform module. [https://www.trustedcomputinggroup.org/?e=category.developerDetail&urlpath=trusted\\_platform\\_module&resource\\_type\\_id=1](https://www.trustedcomputinggroup.org/?e=category.developerDetail&urlpath=trusted_platform_module&resource_type_id=1).
- [5] TCG. What is trusted computing? [http://www.opensecuritytraining.info/IntroToTrustedComputing\\_files/Day1-3-what-is-trusted-computing.pdf](http://www.opensecuritytraining.info/IntroToTrustedComputing_files/Day1-3-what-is-trusted-computing.pdf).
- [6] Smart, Nigel. Cryptography: An introduction. <http://www.cs.umd.edu/~waa/414-F11/IntroToCrypto.pdf>.
- [7] us.hardware. Tpm laptops. <http://us.hardware.info/products/2054/trusted-platform-module-tpm-laptopstables#allproducts>.
- [8] OSx86. Osx86 faq. <http://wiki.osx86project.org/wiki/index.php/FAQ>.
- [9] Black Hat. The trusted computing revolution. <http://www.blackhat.com/presentations/bh-usa-06/BH-US-06-Potter-trusted.pdf>.
- [10] Windows. Securing the windows 8 boot process. <http://technet.microsoft.com/en-us/windows/dn168167.aspx>.
- [11] Windows Server. Bitlocker drive encryption overview. <http://technet.microsoft.com/en-us/library/cc732774.aspx>.
- [12] Google Chrome Blog. Chromebook security: browsing more securely. <http://chrome.blogspot.se/2011/07/chromebook-security-browsing-more.html>.

- [13] Katherine Fang, Deborah Hanus, Yuzhi Zheng. Security of google chrome-book. <http://dhanus.mit.edu/docs/ChromeOSSecurity.pdf>.
- [14] Steven Kinney. Trusted platform module basics using tpm in embedded systems page 53.
- [15] David Challener,Kent Yoder,Ryan Catherman,David Safford,Leendert Van Doorn . Ibm.press.a.practical.guide.to.trusted.computing.jan.2008 page 29.
- [16] David Challener,Kent Yoder,Ryan Catherman,David Safford,Leendert Van Doorn . Ibm.press.a.practical.guide.to.trusted.computing.jan.2008 page 40.
- [17] David Challener,Kent Yoder,Ryan Catherman,David Safford,Leendert Van Doorn . Ibm.press.a.practical.guide.to.trusted.computing.jan.2008 page 31.
- [18] Ariel Segall. Tpm keys creating, certifying, and using them. [http://opensecuritytraining.info/IntroToTrustedComputing\\_files/Day1-7-tpm-keys.pdf](http://opensecuritytraining.info/IntroToTrustedComputing_files/Day1-7-tpm-keys.pdf).
- [19] David Challener,Kent Yoder,Ryan Catherman,David Safford,Leendert Van Doorn . Ibm.press.a.practical.guide.to.trusted.computing.jan.2008 page 36.
- [20] David Challener,Kent Yoder,Ryan Catherman,David Safford,Leendert Van Doorn . Ibm.press.a.practical.guide.to.trusted.computing.jan.2008 page 19.
- [21] Steven Kinney. Trusted platform module basics using tpm in embedded systems page 42.
- [22] Ariel Segall . Using the tpm:machine authentication and attestation. [http://opensecuritytraining.info/IntroToTrustedComputing\\_files/Day1-7-tpm-keys.pdf](http://opensecuritytraining.info/IntroToTrustedComputing_files/Day1-7-tpm-keys.pdf).
- [23] David Challener,Kent Yoder,Ryan Catherman,David Safford,Leendert Van Doorn . Ibm.press.a.practical.guide.to.trusted.computing.jan.2008 page 64.
- [24] Ariel Segall . Programming for the tpm and other practical topics. [http://opensecuritytraining.info/IntroToTrustedComputing\\_files/Day2-4-programming-tpm.pdf](http://opensecuritytraining.info/IntroToTrustedComputing_files/Day2-4-programming-tpm.pdf).
- [25] David Challener,Kent Yoder,Ryan Catherman,David Safford,Leendert Van Doorn . Ibm.press.a.practical.guide.to.trusted.computing.jan.2008 page 141.
- [26] TCG. Tcg software stack (tss) specification. [https://www.trustedcomputinggroup.org/files/resource\\_files/647B51B6-1D09-3519-AD0E37E883F62329/TSS\\_Version\\_\\_1.1.pdf](https://www.trustedcomputinggroup.org/files/resource_files/647B51B6-1D09-3519-AD0E37E883F62329/TSS_Version__1.1.pdf).
- [27] David Challener,Kent Yoder,Ryan Catherman,David Safford,Leendert Van Doorn . Ibm.press.a.practical.guide.to.trusted.computing.jan.2008 page 141.
- [28] Decryption-based authentication. [http://opensecuritytraining.info/IntroToTrustedComputing\\_files/Day2-1-auth-and-att.pdf](http://opensecuritytraining.info/IntroToTrustedComputing_files/Day2-1-auth-and-att.pdf).
- [29] David Challener. An introduction to programming the tpm. <https://www.cylab.cmu.edu/tiw/slides/challener-TPM.pdf>.
- [30] TCG. Tss\_1\_2\_errata\_a-final.pdf page 182.

- [31] David Challener, Kent Yoder, Ryan Catherman, David Safford, Leendert Van Doorn . Ibm.press.a.practical.guide.to.trusted.computing.jan.2008 page 79.
- [32] TCG. Tss\_1\_2\_errata\_a-final.pdf page 191.
- [33] David Challener, Kent Yoder, Ryan Catherman, David Safford, Leendert Van Doorn . Ibm.press.a.practical.guide.to.trusted.computing.jan.2008 page 81.
- [34] David Challener, Kent Yoder, Ryan Catherman, David Safford, Leendert Van Doorn . Ibm.press.a.practical.guide.to.trusted.computing.jan.2008 page 304.
- [35] TCG. Tss\_1\_2\_errata\_a-final.pdf page 232.
- [36] Tss specification. [http://www.trustedcomputinggroup.org/files/resource\\_files/6479CD77-1D09-3519-AD89EAD1BC8C97F0/TSS\\_1\\_2\\_Errata\\_A-final.pdf](http://www.trustedcomputinggroup.org/files/resource_files/6479CD77-1D09-3519-AD89EAD1BC8C97F0/TSS_1_2_Errata_A-final.pdf).
- [37] Number of sold raspberry pi devices. <http://www.raspberrypi.org/a-birthday-present-from-broadcom/>.
- [38] Mario Strasser. Software-based tpm emulator for linux. <http://archiv.infsec.ethz.ch/education/projects/archive/tpmemulatortalk.pdf>.
- [39] Mario Strasser and Heiko Stamer.
- [40] Jan-Erik Ekberg and Markku Kylänpää.
- [41] Tpm emulator at sourceforge. <http://tpm-emulator.sourceforge.net/installation.html>.
- [42] Ibm software tpm at sourceforge. <http://sourceforge.net/projects/ibmswtpm/>.
- [43] Trousers readme. <http://sourceforge.net/p/rousers/rousers/ci/master/tree/README>.
- [44] Environmentvariables. <https://wiki.debian.org/EnvironmentVariables>.
- [45] Introduction to trusted computing. <http://www.opensecuritytraining.info/IntroToTrustedComputing>.
- [46] Mingw at sourceforge. <http://sourceforge.net/projects/mingw/files/latest/download?source=files>.
- [47] Win32openssl. <http://code.x2go.org/releases/binary-win32/3rd-party/Win32OpenSSL/>.
- [48] David Challener, Kent Yoder, Ryan Catherman, David Safford, Leendert Van Doorn . Ibm.press.a.practical.guide.to.trusted.computing.jan.2008 page 63.

---

## Appendix A

# Exercises

---

These exercises can be used to try out different TPM functions before an actual TPM application is developed. We believe that this is the best way to get an idea of what a TPM is used for instead of diving into technicalities right from the start.

A set of command lines utilities designed to demonstrate various TPM functionality comes bundled with the IBM TPM emulator. These utilities can be used to demonstrate the following functions:

- Key creation
- Key migration
- File encryption
- Authentication
- Attestation

In order to develop an actual TPM application one would normally use the TCG Software Stack (TSS). TrouSerS is an open source implementation of the TSS and is used in this thesis. Development using TrouSerS will be described in some of the following exercises.

### Exercise 1: Download software

In order to download, build and install the Software TPM by IBM, we have written some build scripts to automate the process. The build scripts are available in their entirety under “Software TPM by IBM” in the report.

#### Linux

When writing these exercises, the current version of the Software TPM was `tpm4720`. Two parameters should be set by the user:

```
# versions
export TPM_V=tpm4720

# install path
```

```
export TPM_INSTALL_PATH=/...
```

In order to build the Software TPM, some packages must be installed on the users system:

- libtool
- automake
- libssl-dev
- OpenSSL

On a Debian-based system, run `apt-get install libtool automake libssl-dev openssl` in a terminal shell.

Then simply run `./build.sh` in a terminal shell. The build script downloads and compiles the Software TPM at the user's location of choice.

## Windows

In order to build the Software TPM by IBM on a Windows system, a port of the GCC (GNU Compiler Collection) is needed. A port of GCC is e.g., included in the MinGW (Minimalist GNU for Windows) [46] development environment.

A port of OpenSSL (the open-source implementation of the SSL and TLS protocols) to Windows is also needed. This should be downloaded and installed on the user's system.

Win32OpenSSL [47].

- set user environment variables:
  - `TPM_PATH`
  - `TPM_PORT`
- modify `makefile.mak`
  - set correct path to MinGW
  - set correct path to OpenSSL
  - remove `OBJFILES += applink.o`
- `make -file=makefile.mak`

Run the makefile in the MinGW shell in order to compile the Software TPM. TrouSerS has unfortunately very little support for Windows at the moment of writing this, therefore these exercises will only cover the Linux build of TrouSerS and the software TPM.

## Exercise 2: TPM Provisioning (getting TPM ready for use)

### Introduction:

TPM provisioning means to get the TPM ready for use. This is achieved in a few steps. The first step is to activate the TPM. A TPM that has been newly installed on the system is normally deactivated. The TPM is activated in the BIOS. The next step is to ensure that the TPM contains an EK keypair and a certificate for the EK. It should be the manufacturer that creates the EK and certificate at manufacturing but unfortunately this isn't always the case. If the EK is not created at manufacturing it has to be created and certified as the second step in the TPM provisioning. The third step is to take ownership of the TPM. It is in this step where the SRK is created. Two passwords has to be set in this step. One for the owner of the TPM and one for the SRK. The owner password is needed when for example TPM settings is changed and the SRK password is needed for when the SRK is used.

### Quiz:

1. Which TPM command is used to create the Endorsement Key (EK)?
2. If the user is forced to create and certify his or her own EK, the public key of the EK has to be saved and sent to a certification authority. Which command is used to save the public key of the EK?
3. Which TPM command is used to take ownership of a TPM?

**Hint:** Use the document "TPM Main Part 3 Commands" to read about all TPM commands.

### Exercise 2.1: Establish a connection to the TPM emulator over TCP/IP.

1. An actual hardware TPM is enabled and disabled in the BIOS. Before a connection to the IBM software TPM is made the BIOS process has to be simulated. This is done by using the binary `tpmbios` in the `util` directory. This does a `TPM_Startup`, which is the TPM command used to either deactivate the TPM, start up the TPM with a reset of the PCR registers, or start up the TPM with a restore of PCR values from their saved state [48].

Note: `./tpmbios` has to be reentered every time the TPM emulator is started.

2. In the second step the EK will be created. When using the emulator the EK has to be created by the user. This is done by using the utility `./createek`.

Keep an eye on the TPM emulator terminal when the command `./createek` is executed. The EK public key will be printed to the terminal. Take a screen capture of it.

3. Next step is to take ownership of the TPM. This will create the SRK and

set a password to the TPM and the SRK. The utility used for this is `./takeown -pudo ooo -puds sss`, where `ooo` and `sss` is the passwords to the TPM owner and the SRK. These passwords can be set to other values by the users. Memorize the passwords as they will be needed later. After these steps the TPM is setup and ready for usage.

### Exercise 3: Key hierarchy

#### Introduction:

The first key that gets loaded into the chip after the machine is booted is usually a platform migratable key. This key which is usually owned by the system administrator has the well-known secret for its authorization, but requires the system owner's authorization to migrate. If it is migrated, all other migratable keys (which in this design will be children or grandchildren of this key) will also be migrated.

After the platform migratable key is loaded, a user can load his base migratable storage key and his base non-migratable storage key because the TPM now knows the private key of the platform migratable key necessary to decrypt the user migratable key.

#### Quiz:

1. Motivate why it is recommended that the key tree does not get too deep.
2. The identity key is one type of signature key. Describe some differences between an identity and a signature key.
3. Which keys can be used for file encryption?
4. There is one type of key that exists, but it is not recommended to use it. Which key is that, and why does it exist?

#### Exercise 3.1: Create a Key hierarchy using the TPM emulator:

Create a key hierarchy that contains:

- a platform migratable storage key,
- a migratable storage key,
- a non migratable storage key,
- a migratable signing key,
- a non migratable signing key,
- a migratable binding key,
- a migratable symmetric key,
- a non migratable identity key.

Make a picture of your key hierarchy and motivate your solution.



## Exercise 4: Key Migration

### Introduction:

The TPM keys can either be migratable or non-migratable. By using key migration the user can transfer the migratable keys to another system. This is useful for backups and when the TPM is transferred to another system. If a parent key is migrated all the children of this key is migrated as well. When doing a key migration, a blob is created which is then transferred to the target.

### Quiz:

1. Is it possible for a migratable key to be the parent of a non-migratable key?
2. Which command is the first to be executed when performing a key migration?
3. Give a short description of the command `TPM_ConvertMigrationBlob`.
4. Which TPM command loads the migrated keys into the TPM?
5. Is it the TPM or the TSS that handles the transfer of the migration blob?

**Hint:** Use the document "TPM Main Part 3 Commands" to read about all TPM commands.

### Exercise 4.1: Key migration in the TPM emulator

In this exercise we will show how migratable keys can be migrated and then reloaded in the TPM emulator using the demo utilities. First create a migration key blob that can be saved and then reloaded to this TPM or another TPM. Migrate the keys using the utility `./migrate`. Reload the keys to this, or another TPM using the utility `./loadmigrationblob`.

```

Hints: ./migrate -hp <SRK handle in hex> -pwdp <SRK password>
        -pwo <TPM owner password>
        -pwm MIGRATION_PASSWORD
        -ik STORAGEKEY_FILENAME.key
        -im TPM2_STORAGEKEY_FILENAME.key
        -pwdk TPM2_STORAGEKEY_PASSWORD
        -ok migrationblob.bin

        ./loadmigrationblob -hp TPM2_STORAGEKEY_HANDLE
        -if migrationblob.bin -pwdp TPM2_STORAGEKEY_PASSWORD

```

## Exercise 5: Extending values to PCRs

### Introduction:

The PCRs (Platform Configuration Registers) is used to save SHA-1 hash digests of applications. This is used for both trusted boots and attestations. By calculating and saving a digest of an application, it can be shown if the application has been tampered with by an intruder. If one single bit is changed in the application, the hash digest will be completely different.

### Quiz:

1. How many PCRs does the TPM version 1.2 contain?
2. Describe one TPM command that can be used to extend a SHA-1 digest to a PCR.
3. Describe which TPM command that can be used to read a PCR value.
4. The SHA-1 calculations can either be done by the TPM or the TSS before the digest is saved to a PCR. Describe one situation when it has to be done by the TPM.

### Exercise 5.1: SHA-1 calculation and PCR extending using the TPM emulator

This exercise will show how a TPM can calculate a SHA-1 digest of an application and then extend this to a PCR. This is done by using the commands:

```

TPM_SHA1Start
TPM_SHA1Update
TPM_SHA1CompleteExtend

```

The first command starts the SHA-1 calculation, and if the file is too large then it passes the remaining calculation to the second command. The last command finishes the calculation and extends the result into a PCR.

The demo function that can be used to illustrate these commands is  
`./sha -if <filename> -ix <PCR index>.`

Calculate the hash value of for example the binary file `tpmbios` and extend it to the PCR 11. Keep an eye on the TPM emulator terminal. Read the PCR value using the utility `./pcrread` when the calculation is done.

## Exercise 6: File encryption

There are four commands that handle the encryption and decryption of data in the TPM. Three commands are supported by the TPM: `TPM_UnBind`, `TPM_Seal`, and `TPM_UnSeal`. The fourth command is supported by the TSS, `TSS_Bind`.

### Quiz:

1. Why is `TSS_Bind` a TSS command, and not a TPM command?
2. Give some differences between Data binding and Data sealing.
3. Can a key used for data sealing be migrated to another TPM?

### Data binding

#### Exercise 6.1: Data binding using the TPM emulator

Create a binding key using the command `./createkey`. Then encrypt a file with some text using the `.pem` file created by `./createkey`. The command used for this is `./bindfile`. (Note that the key does not have to be loaded into the TPM). Then try to decrypt the file using the command `./unbindfile`. Note that the command `./loadkey` has to be executed before decryption is possible.

Why doesn't the key have to be loaded inside the TPM when encrypting, but it has to be when decrypting?

### Data sealing

#### Exercise 6.2: Data Sealing using the TPM emulator

Create a storage key and load it into the TPM using the commands `./createkey` and `./loadkey`. Seal a textfile using the storage key with the command `./sealfile`. Unseal the file using the command `./unseal`.

Test if you can do a sealing with a legacy key, a binding key or a signing key. If not, why?

## Exercise 7: Machine Authentication

### Introduction:

Machine authentication is used to bring answer to the question "Is this machine X?" It is always important to be able to authenticate the TPM that the user communicates with to be sure that the TPM in fact is a TPM and not a hostile system claiming to be a TPM.

### Signature based:

The goal of this exercise is to show how a TPM can authenticate itself by digitally signing a file using a signature key, and then letting the target verify the signature using the public part of the signature key pair. If the signature is valid then this proves that the TPM knows the private key of the signature key pair and this proves that the TPM is who he claims to be.

#### Quiz:

1. Which TPM command is used to sign a file?
2. Does a TPM has to be present in order to verify the signed file? Why/why not?

**Exercise:** Sign a file with some text in it by loading a signature key into the TPM and use this key to sign the file using the utility `./signfile`. Let another TPM verify the signature by using the utility `./verifyfile`. Could the `verifyfile` command have been done by another TPM?

### Decryption based:

The goal of this exercise is to show how a TPM can authenticate itself by decrypting a file. If a TPM encrypts a file using its public storage key, then the only user able to decrypt the file is the one with the private key of the key pair. If a user can decrypt the file, then this user possesses the private key and is therefore the correct user.

#### Quiz:

1. Is the TPM used to encrypt the file, store the binding key, or both?
2. Which TPM command is used to decrypt the file?
3. Can the decryption based authentication be done by using data sealing instead of binding?

**Exercise 7.1.:** Encrypt a file by creating a binding key and load it into the TPM and then encrypt a text file using the command `./bindfile` Then decrypt it using the command `./unbindfile`.

### Exercise 8: Attestation

Attestation is a mechanism by which one wants to obtain a proof that the right software was loaded (by recording its hash in a PCR). The goal of attestation is to prove to a remote party that your application software are intact and trustworthy. This is useful by for example banks who now could ask the user to show that he has the correct versions of the banking software. Just like with authentication, attestation can either be signature based or decryption based.

**Quiz:**

1. Machine authentication answers the question “Is this machine X?”. What question does attestation bring answer to?

Signature based:

**Exercise 8.1:**

Create an AIK (Attestation Identity Key) using the command `./identity`. Use it to quote a PCR value, like the PCR with the hash digest of `tpmbios`. The verification of the quote will be done automatically.

**Hints:**

```
./identity -pwdo <owner password>
```

```
-la <a label>
```

```
-pwds <SRK password>
```

```
-ok <key filename>
```

```
./quote -hk <key handle in hex> -bm <pcr hash digest> -pwdk <key password>
```

Decryption-based:

With a decryption based attestation we bind a PCR value to a storage key. The key can only be used for decryption if the PCR value that was bound to the key is unchanged or has obtained the same value after a restart of the system. So if a program has been changed, then the PCR value will be different and then the key cannot be used for decryption.

**Exercise 8.2:**

Create a text file and extend the hash digest to a PCR. Create a storage key and bind it to the PCR value using the command `./createkey` (`-ix` is used to specify the PCR index). Load the key into the TPM. Seal the text file using the storage key with the command `./sealfile`. Unseal the file using the command `./unsealfile` (Should be successful).

Change the text in the text file and extend the PCR with the new hash digest of the text file. Try decrypt the file again (should not work, PCR value bound to the storage key has changed). Clear ownership of the TPM using the command `./forceclear`.

### Exercise 9: Write your own TPM application using TrouSerS

On Debian:

```
sudo apt-get install trousers
sudo apt-get install tpm_tools
> export TCSD_TCP_DEVICE_PORT=<your RPi port>
> export TCSD_TCP_DEVICE_HOSTNAME=<your RPi IP>
```

Take command of the TPM using `tpm_tools` by writing `tpm_takeown -z -y` in the terminal. Use the examples found in the section "Developing TPM applications" and write your own TPM application. Here are some examples for TPM applications:

Generate random numbers using `Tspi_TPM_GetRandom`

Extend values in PCRs using `Tspi_TPM_PcrExtend`

---

TrouSerS applications

---

## List PCRs

---

ListPCRs.c

---

```
#include<stdio.h>
#include<string.h>
#include<stdlib.h>
#include<unistd.h>
#include<sys/stat.h>
#include<sys/types.h>
#include<tss/platform.h>
#include<tss/tss_defines.h>
#include<tss/tss_typedef.h>
#include<tss/tss_structs.h>
#include<tss/tspi.h>
#include<trousers/trousers.h>
#include<tss/tss_error.h>

#define DEBUG 1
#define DBG(message, tResult) { if(DEBUG) printf("(Line%d, %s) \
%s returned 0x%08x. %s.\n", __LINE__, __func__, message, \
tResult, (char *)Trspi_Error_String(tResult));}

int main(int argc, char **argv) {

    BYTE *rgbPcrValue, *rgbNumPcrs;
    UINT32 ulPcrValueLength;
    UINT32 exitCode, subCapSize, numPcrs, subCap, i, j;

    TSS_HCONTEXT hContext=0;
    TSS_HTPM hTPM = 0;
    TSS_RESULT result;
    TSS_HKEY hSRK = 0;
    TSS_HPOLICY hSRKPolicy=0;
    TSS_UUID SRK_UUID = TSS_UUID_SRK;
    //By default SRK is 20bytes 0
    //takeownership -z
    BYTE wks[20];
```

```

memset(wks,0,20);

//At the beginning
//Create context and get tpm handle
result =Tspi_Context_Create(&hContext);
DBG("Create a context\n", result);
result=Tspi_Context_Connect(hContext, NULL);
DBG("Connect to TPM\n", result);
result=Tspi_Context_GetTpmObject(hContext, &hTPM);
DBG("Get TPM handle\n", result);
//Get SRK handle
//This operation need SRK secret when you takeownership
//if takeownership -z the SRK is wks by default
result=Tspi_Context_LoadKeyByUUID(
    hContext,
    TSS_PS_TYPE_SYSTEM,
    SRK_UUID,
    &hSRK);
DBG("Get SRK handle\n", result);
result=Tspi_GetPolicyObject(hSRK,
    TSS_POLICY_USAGE, &hSRKPolicy);
DBG("Get SRK Policy\n", result);
result=Tspi_Policy_SetSecret(hSRKPolicy,
    TSS_SECRET_MODE_SHA1, 20, wks);
DBG("Tspi_Policy_SetSecret\n", result);

subCap = TSS_TPMCAP_PROP_PCR;
//Retrieve number of PCR's from the TPM
result = Tspi_TPM_GetCapability(hTPM,
    TSS_TPMCAP_PROPERTY,
    sizeof(UINT32),
    (BYTE *)&subCap,
    &ulPcrValueLength,
    &rgbNumPcrs);
if (result == TSS_SUCCESS) {
    if (ulPcrValueLength != sizeof(UINT32)) {
        printf("GetCapability(TSS_TPMCAP_PROP_PCR)
            returns value != sizeof(UINT32)!\n");
        Tspi_Context_FreeMemory(hContext, NULL);
        Tspi_Context_Close(hContext);
        exit(result);
    }
    //Algorithm found at trousers/testsuite
    numPcrs = *(UINT32 *)&rgbNumPcrs;
    printf("\nPCR List\n");
    for (i = 0; i < numPcrs; i++) {
        result = Tspi_TPM_PcrRead(hTPM,i,
            &ulPcrValueLength,
            &rgbPcrValue);
        printf("PCR%02u: ", i);
        for (j = 0; j < ulPcrValueLength; j++) {

```



```

        printf("%02x", rgbPcrValue[j] & 0xff);
    }
    printf("\n");
}
printf("\n");
}

//Free memory
result = Tspi_Context_FreeMemory(hContext, NULL);
DBG("Tspi Context Free Memory\n", result);
result = Tspi_Context_Close(hContext);
DBG("Tspi Context Close\n", result);
return 0;
}

```

## List Keys

ListKeys.c

```

#include<stdio.h>
#include<string.h>
#include<stdlib.h>
#include<sys/stat.h>
#include<sys/types.h>
#include<tss/platform.h>
#include<tss/tss_defines.h>
#include<tss/tss_typedef.h>
#include<tss/tss_structs.h>
#include<tss/tspi.h>
#include<trousers/trousers.h>
#include<tss/tss_error.h>

#define DEBUG 1
#define DBG(message, tResult) { if(DEBUG) printf("(Line%d, %s) \
%s returned 0x%08x. %s.\n", __LINE__ , __func__ , message, tResult, \
(char *)Trspi_Error_String(tResult));}

int main( int argc, char **argv ) {

    UINT32 i,pulKeyHierarchySize;
    TSS_HKEY hKey;
    TSS_KM_KEYINFO* ppKeyHierarchy;

    TSS_HCONTEXT hContext=0;
    TSS_HTPM hTPM = 0;
    TSS_RESULT result;
    TSS_HKEY hSRK = 0;
    TSS_HPOLICY hSRKPolicy=0;
    TSS_UUID SRK_UUID = TSS_UUID_SRK;

```

```

//By default SRK is 20bytes 0
//takeownership -z
BYTE wks[20];
memset(wks,0,20);

//At the beginning
//Create context and get tpm handle
result =Tspi_Context_Create(&hContext);
DBG("Create a context\n", result);
result=Tspi_Context_Connect(hContext, NULL);
DBG("Connect to TPM\n", result);
result=Tspi_Context_GetTpmObject(hContext, &hTPM);
DBG("Get TPM handle\n", result);
//Get SRK handle
//This operation need SRK secret when you takeownership
//if takeownership -z the SRK is wks by default
result=Tspi_Context_LoadKeyByUUID(hContext, TSS_PS_TYPE_SYSTEM, SRK_UUID,
&hSRK);
DBG("Get SRK handle\n", result);
result=Tspi_GetPolicyObject(hSRK, TSS_POLICY_USAGE, &hSRKPolicy);
DBG("Get SRK Policy\n", result);
result=Tspi_Policy_SetSecret(hSRKPolicy,TSS_SECRET_MODE_SHA1,20, wks);
DBG("Tspi_Policy_SetSecret\n", result);

//Get Registered Keys By UUID
//Out:pulKeyHierarchySize - size of the key list
//ppKeyHierarchy- the keys in the keylist
result = Tspi_Context_GetRegisteredKeysByUUID(
                                hContext,
                                TSS_PS_TYPE_SYSTEM, NULL,
                                &pulKeyHierarchySize,
                                &ppKeyHierarchy
                                );

if (result == TSS_SUCCESS) {
    DBG("Tspi_Context_GetRegisteredKeysByUUID\n", result);
    //Print the key info for each loaded key in the tpm.
    for (i = 0; i < pulKeyHierarchySize; i++) {
        printf("Registered key %u:\n", i);
        print_KM_KEYINFO(&ppKeyHierarchy[i]);
    }
}

Tspi_Context_CloseObject(hContext, hKey);
result = Tspi_Context_FreeMemory(hContext, NULL);
DBG("Tspi Context Free Memory\n", result);
result = Tspi_Context_Close(hContext);
DBG("Tspi Context Close\n", result);
return 0;
}

```

```
//method to print the key info from print_KM_KEYINFO
//This method was taken from trousers/testsuite.
void print_KM_KEYINFO(TSS_KM_KEYINFO *k) {
    printf("Version: %hhu.%hhu.%hhu.%hhu\n",
        k->versionInfo.bMajor,
        k->versionInfo.bMinor,
        k->versionInfo.bRevMajor,
        k->versionInfo.bRevMinor
    );
    printf("UUID: %08x %04hx %04hx %02hhx %02hhx \
        %02hhx%02hhx%02hhx%02hhx%02hhx%02hhx\n",
        k->keyUUID.ulTimeLow,
        k->keyUUID.usTimeMid,
        k->keyUUID.usTimeHigh,
        k->keyUUID.bClockSeqHigh,
        k->keyUUID.bClockSeqLow,
        k->keyUUID.rgbNode[0] & 0xff,
        k->keyUUID.rgbNode[1] & 0xff,
        k->keyUUID.rgbNode[2] & 0xff,
        k->keyUUID.rgbNode[3] & 0xff,
        k->keyUUID.rgbNode[4] & 0xff,
        k->keyUUID.rgbNode[5] & 0xff
    );
    printf("parent UUID : %08x %04hx %04hx %02hhx \
        %02hhx %02hhx%02hhx%02hhx%02hhx%02hhx%02hhx\n",
        k->parentKeyUUID.ulTimeLow,
        k->parentKeyUUID.usTimeMid,
        k->parentKeyUUID.usTimeHigh,
        k->parentKeyUUID.bClockSeqHigh,
        k->parentKeyUUID.bClockSeqLow,
        k->parentKeyUUID.rgbNode[0] & 0xff,
        k->parentKeyUUID.rgbNode[1] & 0xff,
        k->parentKeyUUID.rgbNode[2] & 0xff,
        k->parentKeyUUID.rgbNode[3] & 0xff,
        k->parentKeyUUID.rgbNode[4] & 0xff,
        k->parentKeyUUID.rgbNode[5] & 0xff
    );
    printf("auth: %s\n", k->bAuthDataUsage ? "YES" : "NO");
    if (k->ulVendorDataLength)
        printf("vendor data : \"%s\" (%u bytes)\n",
            k->rgbVendorData,
            k->ulVendorDataLength
        );
    else
        printf("vendor data: (0 bytes)\n");

    printf("\n");
}
```

---

---

## TrouSerS applications

---

### IBM License file

---

LICENSE.txt

---

\$Id: LICENSE 4702 2013-01-03 21:26:29Z kgoldman \$  
(c) Copyright IBM Corporation 2006, 2010.

All rights reserved.

Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.

Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.

Neither the names of the IBM Corporation nor the names of its contributors may be used to endorse or promote products derived from this software without specific prior written permission.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT HOLDER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.



**LUND**  
UNIVERSITY

Series of Master's theses  
Department of Electrical and Information Technology  
LU/LTH-EIT 2015-434

<http://www.eit.lth.se>