

TPM Application Note

Trusted Platform Module 2.0



About this document

Scope and purpose

This document explains how the Infineon OPTIGA™ Trusted Platform Module (TPM) 2.0 can be used on a Raspberry Pi® 3. All necessary steps regarding installation of additional software packages and configurations on the Raspberry Pi® will be explained. The Infineon OPTIGA™ TPM 2.0 SLB9670 uses an SPI interface to communicate with the Raspberry Pi®. In this document we refer to the Infineon OPTIGA™ Trusted Platform Module 2.0 SLB 9670 by using simply TPM and to the Raspberry Pi® 3 by using RPi.

An overview of all Infineon OPTIGA™ TPM products can be found on Infineon's website [1].

More information about the TPM in general and how to integrate it into a platform can be found in the corresponding specifications of the Trusted Computing Group (TCG) in reference [14] and [15].

Intended audience

This document is intended for customers who want to increase the security level of their embedded platforms using a TPM 2.0.

Table of Contents

1	Introduction	4
1.1	Motivation.....	4
1.2	Scope.....	5
1.3	Command conventions.....	5
1.4	Acronyms and Abbreviations.....	6
2	Hardware Setup.....	7
2.1	Raspberry Pi® 3.....	8
2.2	Infineon Iridium SLB 9670 TPM 2.0 SPI Board.....	9
3	Software Setup	11
3.1	Developer PC Installation	11
3.2	Raspberry Pi® 3 Installation	11
3.2.1	Installation of Linux.....	12
3.2.2	Download the Linux Kernel	14
3.2.3	Modify and Compile the Linux Kernel	15
3.2.4	Install new Linux Kernel on SD Card	16
3.3	Software Packages and Configuration.....	17
3.3.1	Verifying the Previous Steps.....	17
3.3.2	Additional Packages	18
3.3.3	Install the Intel TPM Software Stack for TPM 2.0.....	18
3.3.4	Install the Intel tpm2.0-tools	19
4	Using the TPM with TSS2	20
4.1	Introduction	20
4.2	Verifying TSS2 functionality.....	20
4.3	Taking TPM Ownership	21
4.4	Using TPM2.0 Tools.....	21
4.4.1	Encrypt and Decrypt a File using RSA 2048	22
4.4.2	Sign and Verify using ECC 256.....	25
5	References.....	27
6	Appendix.....	28
6.1	Dictionary Attack Protection	28

6.2	Cold Reboot of the Raspberry Pi® 3	28
6.3	Troubleshooting	28
6.3.1	Platform Related Problems	28
6.3.2	Driver Related Problems.....	29
6.3.3	TSS2 or TPM 2.0 Usage Related Problems.....	30
6.3.4	Kernel Configuration Related Problems	30
6.3.5	Internet Connection Related Problems	30
6.4	Linux Kernel Support of Infineon SLB 9670 TPM 2.0.....	31
6.5	Schematic and Layout of the Infineon Iridium SLB 9670 TPM 2.0 SPI Board	31
6.6	TPM 2.0 Features	34
6.7	Running commands with root privileges	35
6.7.1	Using sudo	35
6.7.2	Log in as root.....	35
6.8	Interacting with the Raspberry Pi® 3	36
6.8.1	Direct interaction.....	36
6.8.2	Connecting via SSH.....	36
6.9	Exchanging Files	37
6.9.1	Direct Copy on microSD-Card	37
6.9.2	Copying via SCP	37
6.10	Manual Loading of the TPM 2.0 Driver	38

1 Introduction

In case there occur any problems during execution of the steps described in this document, please have a look into the corresponding troubleshooting section in appendix [6.3](#). It might contain helpful information on how to locate the origin of possible problems and how to solve them on your own.

1.1 Motivation

Two of the basic principles of information security when exchanging data are confidentiality and authenticity. While the first one is obvious to almost everyone, the second one often does not get the appropriate amount of attention. Authenticity is just as important as confidentiality, because for example the highest confidentiality of secret data is worth nothing in case the secret data come from or is shared with the wrong communication partner.

A TPM adds additional security features to a platform, which will not only help in gaining confidentiality but also authenticity of exchanged data or a communication partner.

One of the most important features to achieve these goals is that sensitive data such as cryptographic keys or secrets can be stored inside the TPM, where the data is protected by the TPM's hardware from unauthorized access or manipulation. This allows confidentiality of the data on a high level. But in addition to this protection, the data can also be bound to a single TPM and thus to the platform hosting this TPM. This in turn can be used to bring in another authenticity factor, since only the platform or user operating the correct TPM is able to provide the correct authentication secret required to access the protected sensitive data.

Another basic feature of the TPM is that it can function as a starting point for establishing a root of trust, which allows the detection of unauthorized modifications to a platform's hardware or software.

Additionally, the TPM can be used to perform several cryptographic operations on data in hardware, decreasing the vulnerability to several kinds of attacks, such as reading out unencrypted sensitive data from the platform's memory.

More details about TPM features can be found in section [6.6](#).

Introduction

1.2 Scope

This document is intended for users to help them getting familiar how to use a subset of the TPM's functionalities on embedded platforms running Linux. The final goal in this document is to interact with the TPM 2.0 to be able to run the test scripts provided in the GitHub project [tpm2.0-tools](#) [16] and to execute the command line commands and the test scripts available in the same project.

The topics described in this document are:

- The Evaluation Board ([Figure 2](#)) with its expansion header for the Raspberry Pi® 3.
- Usage of a TPM 2.0 on a Raspberry Pi® 3 (running kernel¹ version 4.4).

This document has been generated using exactly kernel version 4.4. Other versions might work applying some additional or different steps but have not been tested.

In this application note we use an RPi Linux distribution based on Debian® to describe the system setup and the application of the TPM 2.0. If not explicitly said otherwise, the application note refers to specific versions of the required Linux packages to prevent compatibility problems. The mentioned versions were publicly available and downloaded at the time of release from the specified links and repositories.

1.3 Command conventions

In this document, the operating system Linux will be used. Since some commands in this document are directly interacting with hardware or accessing protected system files, the commands need to be executed with root privileges (also known as administrative privileges). The convention in this document will be that whenever a command requires root privileges, the actual command is preceded by “#”, whereas commands requiring only normal user privileges are preceded by “\$”:

```
# [command to be executed as root]
```

and

```
$ [command to be executed as user]
```

Information on how to run commands as root can be found in section [6.7](#).

¹ For the remainder of this document the term *kernel* is used equivalently to *Linux kernel*.

Introduction

1.4 Acronyms and Abbreviations

Acronym	Explanation
BIOS	Basic Input / Output System
EK	Endorsement Key
FTD	Flattened Device Tree
LTS	Long Term Support
PCR	Platform Configuration Register
RNG	Random Number Generator
RPi	Raspberry Pi ®
SoC	System on a Chip
SPI	Serial Peripheral Interface
SRK	Storage Root Key
SSH	Secure Shell
SSL	Secure Sockets Layer
TCG	Trusted Computing Group
TCS	TCG Core Services
TCSD	TCS Daemon
TLS	Transport Layer Security
TPM	Trusted Platform Module
TSS	TCG Software Stack
UEFI	Unified Extensible Firmware Interface
VPN	Virtual Private Network

2 Hardware Setup

The required hardware to perform the steps described in this document consists of:

- **Developer PC:**
This platform is used for patching the Kernel, maintaining and interacting with the RPi in a more convenient and faster way compared to doing all actions directly on the RPi. The hardware requirements for the developer PC are:
 - Desktop computer or laptop with x86 architecture and USB 2.0 (or higher)
 - Capable of running Linux, for example Ubuntu® 15.10
 - Connected to the Internet
 - Connected to a network (optional¹)
- **Infineon Iridium SLB 9670 TPM 2.0 SPI Board**
This board contains the Infineon SLB 9670 TPM 2.0 mounted on an easy-to-use hardware board, which can be attached to the Raspberry Pi® 3.
- **RJ45 Ethernet LAN cable:**
 - Internet connection
 - Connected to the same network as the developer PC (optional¹)
- **Raspberry Pi® 3:**
 - Raspberry Pi® 3 Model B
 - Micro SD Card² with at least 8 GB
 - Micro-B USB cable for power supply
- Monitor or projector with HDMI input, HDMI cable, and USB keyboard (optional¹)
- MicroSD-Card Reader

¹ Depends on how you interact with the embedded platform; details will be described in section [6.8](#).

² Some card models may not work. A list of test cards can be found at [2].

2.1 Raspberry Pi® 3

The Raspberry Pi® 3 Model B has a Broadcom® BCM2837 SoC with a 1.2 GHz quad-core ARM™ Cortex-A53 CPU with ARMv7 architecture and 1 GB RAM. An HDMI connector can be used for graphical output. It also has an Ethernet controller for network connectivity and a micro USB port for power supply. A cold reboot of the platform can only be triggered if the Raspberry Pi® 3 is disconnected from power (see section 6.2). Figure 1 shows the Raspberry Pi® 3 with all expansions. For more information visit the official website [3].

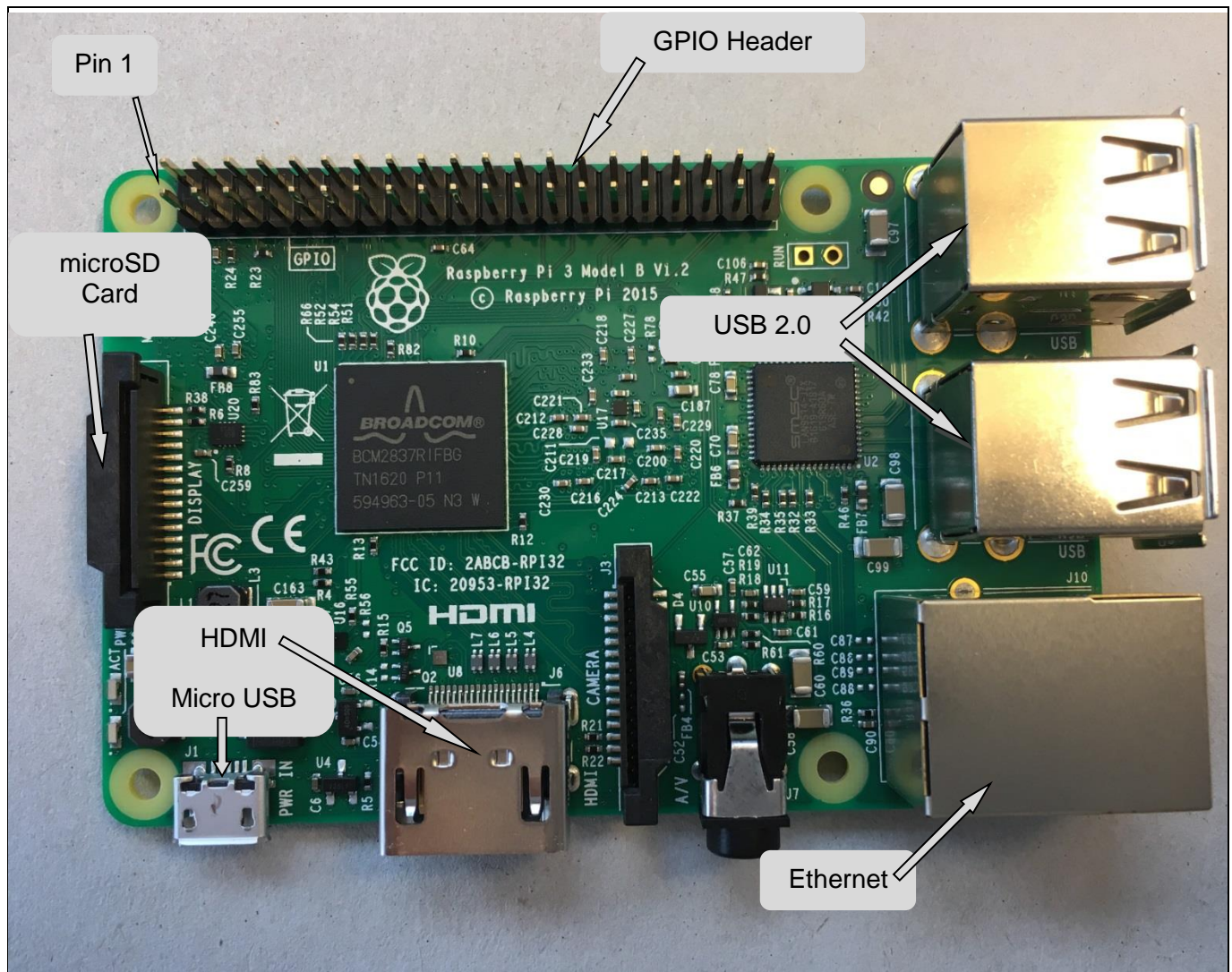


Figure 1 Raspberry Pi® 3 Expansions

2.2 Infineon Iridium SLB 9670 TPM 2.0 SPI Board

The Infineon Iridium SLB 9670 TPM 2.0 SPI Board¹ can be connected to a Raspberry Pi® 3 via its extension header. For data transfer the board uses the SPI bus. The board can be seen in Figure 2.

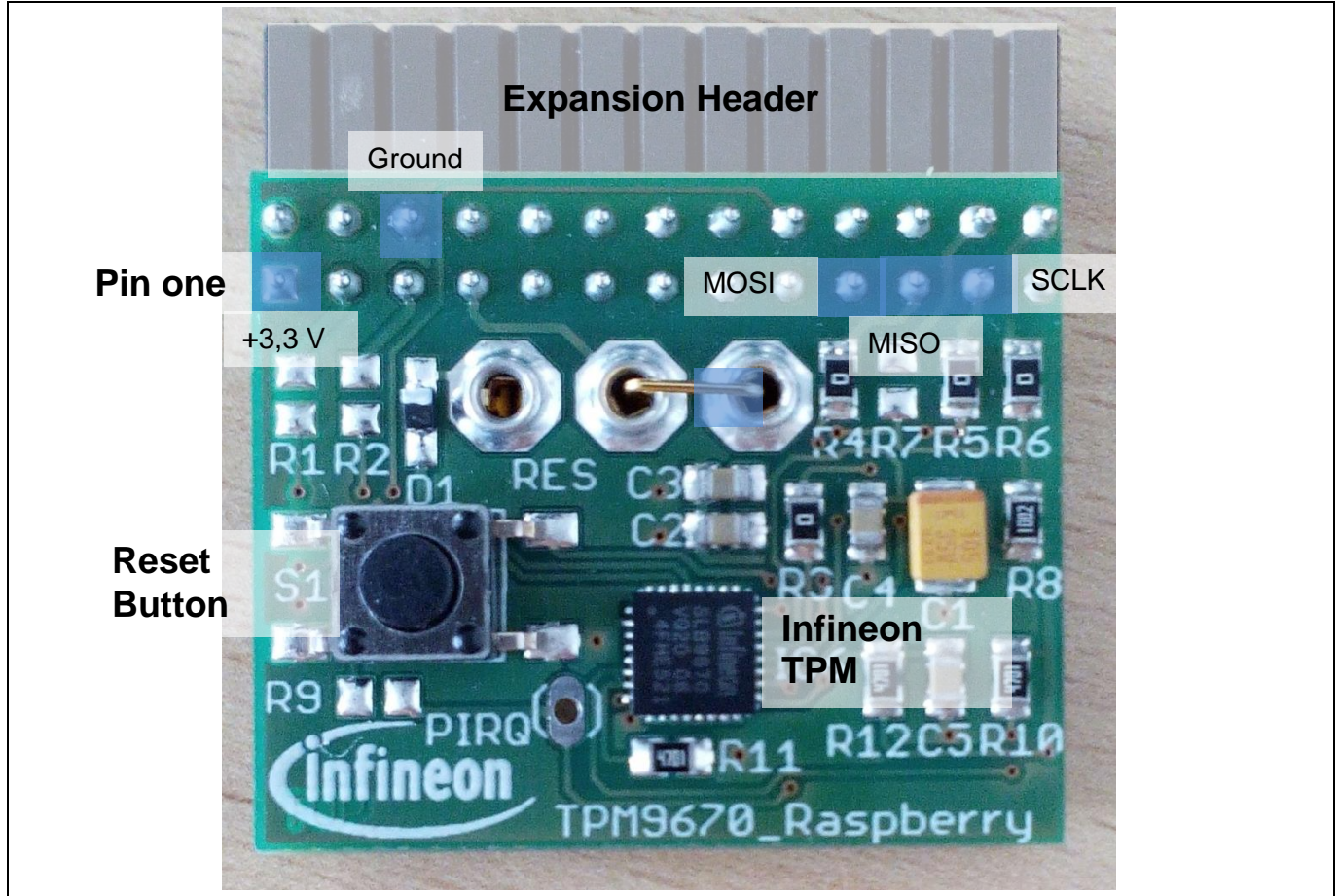


Figure 2 Infineon Iridium SLB 9670 TPM 2.0 SPI Board

On the top is the Raspberry Pi® Header with 26 pins. The board contains a reset circuit on board, which pulls the reset line of the TPM to GND for the right amount of time after VCC becomes available. The reset of the module is active low and can also be set by connecting reset and ground via the reset button.

The Raspberry Pi® 3 Header has VDD, GND, MOSI, MISO and SCLK at the following pins:

Pin	1	6	19	21	23
Signal	VDD, 3.3 V	GND	MOSI	MISO	SCLK

A complete layout of the board can be found in section 6.5. Figure 3 shows how the Evaluation Board should be mounted on the RPi.

¹ Order type: IRIDIUM9670 TPM2.0
Order number: SP001596592



Figure 3 Infineon Iridium SLB 9670 TPM 2.0 SPI Board on Raspberry Pi® 3

3 Software Setup

This section describes all necessary steps needed to use the SLB 9670 TPM 2.0 with a Raspberry Pi® 3. Some steps in this section have to be done on a developer PC and some on the Raspberry Pi® 3.

3.1 Developer PC Installation

The developer PC can be either a native Linux PC or a virtual machine running Linux on a non-Linux machine (e.g. Windows). It will be used mainly for compiling and setting up the Linux Operating System for the Raspberry Pi® 3 on a microSD-Card. It can also be used to control the Raspberry Pi® 3 remotely via an SSH connection. It is recommended to use a Linux Distribution like Ubuntu®, Debian®, or similar on the developer PC. This document refers to the usage of a native machine running Ubuntu® 15.04, which can be downloaded at [4].

The basic installation of Ubuntu® comes by default with many packages installed. Beyond that, the following list shows all additional software packages required to be able to perform all steps in this application note.

- gddrescue
- git
- libncurses5-dev
- gcc-arm-linux-gnueabi

Further information on these packages can be found at [5].

The easiest way to make sure that all packages and their dependencies are installed is to call the installation routine for all of them from a terminal via the command “apt-get install [package name]” (requires an active Internet connection). Packages already installed will not be reinstalled, but updated if necessary. First the package information should be updated:

```
# apt-get update
```

Now install all packages with the following command in a terminal on the developer PC:

```
# apt-get install gcc-arm-linux-gnueabi gddrescue git libncurses5-dev
```

3.2 Raspberry Pi® 3 Installation

The following sections describe the software setup of the Raspberry Pi® 3. In order to get the TPM 2.0 working on the Raspberry Pi® 3, the kernel must have driver support for the TPM. More information on this topic can be found in section 6.4. The following sections give detailed instructions on how to patch and install a Linux with kernel version 4.4 on the Raspberry Pi® 3. Other kernel versions might work with more or less effort. The basic idea of the required steps for setting up the RPi to interact with a TPM 2.0 is shown in Table 1. The detailed descriptions can be found in the referenced chapters in the table. Since some steps need to be executed on a PC and some have to be executed on the Raspberry Pi® itself, one can be confused very easily on which platform a certain step shown in the subsequent sections has to be executed. Therefore, the two rightmost columns of the enumerated list depicted in Table 1 show on which platform a command has to be executed.

Software Setup

Table 1 Summarized steps needed to prepare the PC and the Raspberry Pi® to get started with a TPM 2.0.

Step	Section	PC	Raspberry Pi®
1	3.2.1	Download RPi image “Raspbian Jessie” and write image to SD Card	
2	3.2.1		Insert SD-Card to RPi and boot the image
3	3.2.1		Configure basic settings to enable e.g. SPI interface
4	3.2.2	Download Linux Kernel from GitHub	
5	3.2.2	Apply patch to Linux Kernel	
6	3.2.3		Create configuration file of RPi and transfer it to the PC
7	3.2.3	Edit configuration file to enable TPM support	
8	3.2.3	Compile the Kernel	
9	3.2.4	Replace kernel of image on SD Card with modified kernel	
10	3.2.4		Insert SD-Card with modified kernel to RPi and boot the image

3.2.1 Installation of Linux

It is recommended to create a new folder which will work as the working space for all following instructions in the home directory of the developer PC, e.g. `TpmAppNote`. Now, copy the `src` folder shipped with this document into the `TpmAppNote` folder. The first step is to download an image of Raspbian from [6] and unpack the downloaded archive on the developer PC. Therefore, open a terminal (`Ctrl-Alt-T`) and switch to your workspace folder:

```
$ cd ~/TpmAppNote
```

Now download the archive (via command line or web browser) and extract the OS image:

```
$ wget https://downloads.raspberrypi.org/raspbian/images/raspbian-2017-01-10/2017-01-11-raspbian-jessie.zip
```

```
$ unzip 2017-01-11-raspbian-jessie.zip
```

After that, connect an SD-Card via a card reader to the developer PC and execute the following two commands, where `sdX` is the device file for the SD-Card. This has to be replaced by the correct device file for the SD-Card, for example `sdb`. The device letter for the SD-Card depends on the system and can

Software Setup

be found with the command `$ lsblk` and identified over its size. The device is automatically being partitioned, formatted and the basic packages are being installed.

In order to avoid data loss, do not issue the following command unless you are absolutely sure that you have replaced `sdX` with the correct device file for the SD-Card, since the command will cause the specified device to be completely overwritten!

```
# ddrescue 2015-11-21-raspbian-jessie.img /dev/sdX --force
$ sync
```

The `sync` command is a separate command which commits any write buffer cache contents to disk, thus it also finishes writing to the SD-Card. Depending on the amount of data in the buffer, this can take several minutes and it is normal that there will be no screen output. Also note that the SD-Card is not automatically mounted after `ddrescue` and `sync` have been executed. You either have to mount it manually or eject and insert it again in case you want to verify the content of the two partitions.

Once completed, connect the network and insert the SD-Card into the Raspberry Pi® 3, connect a USB keyboard and a monitor (see section 6.8 for different ways how to interact with the Raspberry Pi® 3) and plug in the power cable. Now the system should start, otherwise see section 6.3 for troubleshooting.

The default Raspbian credentials required to logon are:

- Username: `pi`
- Password: `raspberrypi`

On first boot, a desktop will be shown. Open a terminal on the Raspberry Pi® 3 (or connect to the Raspberry Pi® 3 via SSH) and execute the following command:

```
# raspi-config
```

It is required to configure at least the following options:

- Expand Filesystem
- Advanced Options: SPI (enable the interface and let it load by default when asked to)
- Internationalization Options: Change Time zone

Optionally, you can also configure the following options:

- Boot Options (e.g. choose “Console Auto login” to speed up the boot process)
- Internationalization Options: Change Keyboard Layout (not applicable when connected via SSH)

Once the above steps have been completed you can finish the wizard and reboot the RPi.

3.2.2 Download the Linux Kernel

This section shows how to download the Linux Kernel from a repository. The following list contains explanations for the terms used in the following sections.

- Patching the kernel means modifying the kernel source code and / or the kernel's FTD¹.
 - Modifying the FTD is only required in case the TPM driver should be loaded automatically during system boot. This applies to all kernel versions.
 - Modifying the kernel source code is only required in case your kernel version does not already have built-in support for your TPM with the chosen communication interface (I²C or SPI).
- Configuring the kernel means activating the TPM driver in the kernel configuration so that the driver is being compiled. The TPM driver can be configured to be built either as separate kernel module or to be built directly into the kernel image.
This is always necessary unless you are using a kernel with the corresponding TPM driver already activated.
- Compiling the kernel means building the kernel image, its modules and / or the FTD with the chosen configuration for the embedded platform. This is required in case you did patch and / or configure your kernel.

Download the Raspberry Pi® 3 Linux kernel from the GitHub repository [7]. Execute the following commands in a terminal on the developer PC:

```
$ cd ~/TpmAppNote
$ git clone --depth=1 git://github.com/raspberrypi/linux.git -b rpi-4.4.y
```

After that, switch to the `linux` folder inside the downloaded repository:

```
$ cd linux
```

¹ FTD = Flattened Device Tree; a mechanism on embedded platforms to configure and populate attached hardware to the kernel.

3.2.3 Modify and Compile the Linux Kernel

Now, the current path should be `~/TpmAppNote/linux/`. Copy and apply the patch file named `Rpi_3_SPI_TPM_Driver_And_Automatic_Driver_Load.patch`, which is shipped with this application note. This patch changes portions of the previously downloaded Kernel. The Kernel now loads the Infineon SPI TPM driver automatically on system start.

```
$ cp ~/TpmAppNote/Src/RPi_3_SPI_TPM_Driver_And_Automatic_Driver_Load.patch .
$ patch -p1 < RPi_3_SPI_TPM_Driver_And_Automatic_Driver_Load.patch
```

In the next step the original kernel configuration of the Raspbian kernel installed on the Raspberry Pi® 3 is used as a basis for the configuration of the modified kernel 4.4. Therefore, the configuration file must be copied via SSH from the running Raspberry Pi® 3 to the developer PC. Attach the Raspberry Pi® 3 to your network and (re)start the Raspberry Pi®. It is also possible to do this with any USB storage device. Execute the following command and copy the `config.gz` file to the developer PC:

```
# modprobe configs
```

Now execute the following commands on the developer PC:

```
$ scp pi@raspberrypi:/proc/config.gz ~/TpmAppNote/
```

In case “`raspberrypi`” cannot be resolved, use the IP address of the Raspberry Pi® 3 instead. You can get it with running `$ ifconfig` on the Raspberry Pi® 3. Then unzip the file `config.gz` and store it with the name `.config` into the `linux` folder:

```
$ zcat ~/TpmAppNote/config.gz > .config
```

Before compiling anything you need to define the name under which the kernel will be known on the SD-Card. Double-check to take ‘kernel seven’ and not ‘kernel seventeen’ as the `KERNEL` value.

```
$ KERNEL=kernel7
```

After that, the TPM driver needs to be enabled as kernel module. Execute the following commands to initialize and bring up the kernel configuration:

```
$ make ARCH=arm CROSS_COMPILE=arm-linux-gnueabihf- olddefconfig
```

```
$ make ARCH=arm CROSS_COMPILE=arm-linux-gnueabihf- menuconfig
```

In the upcoming menu, navigate with the up and down arrow keys as described to the corresponding menu entries listed below and press the key(s) also listed below (marked in **gray**):

```
Device Drivers [Enter] →
```

```
Character devices [Enter] →
```

```
TPM Hardware Support [M] & [Enter] →
```

```
TPM Interface Specification 1.3 Interface TPM 2.0 FIFO Interface - native SPI [M]
```

Then exit each sub menu by selecting `Exit` in the lower menu bar with the right arrow key and pressing `[Enter]`. Do so for all sub menus and save the configuration when asked to do so.

Now use the following command to compile the kernel, modules and FTD blobs and generate the `zImage`¹ kernel image file:

```
$ make ARCH=arm CROSS_COMPILE=arm-linux-gnueabihf- -jX zImage modules dtbs
```

¹ `zImage` = A compressed version of the Linux kernel image.

Software Setup

The variable `x` in the previous command is the number of parallel threads created and used for compiling. You will most likely get the fastest result when using the number of logical processor cores + 1 of your development PC here.

3.2.4 Install new Linux Kernel on SD Card

You then have to install the modules of the new kernel on the SD-Card. Therefore, shutdown the RPi, insert the SD-Card into the card reader of the developer PC and execute the following two commands on the developer PC (make sure that the placeholders in brackets (“[...]”) are replaced with the correct values). Depending on how the system mounts SD-Cards these paths can be different.

```
# make ARCH=arm CROSS_COMPILE=arm-linux-gnueabihf- modules_install  
INSTALL_MOD_PATH=/media/[username]/[rootfs-ID]/
```

`/media/[username]/[rootfs-ID]/` is the path to the root file system on the SD-Card.

`/media/[username]/boot` is the mount point of the boot partition on the SD-Card.

Rename the current Kernel image with to create a backup for the original Kernel.

```
# mv /media/[username]/boot/$KERNEL.img /media/[username]/boot/$KERNEL-  
backup.img
```

Copy the new Kernel image and additional configuration files to the boot partition on the SD-Card.

```
# scripts/mkknlimg arch/arm/boot/zImage /media/[username]/boot/$KERNEL.img  
# cp arch/arm/boot/dts/*.dtb /media/[username]/boot/  
# cp arch/arm/boot/dts/overlays/*.dtb* /media/[username]/boot/overlays/
```

Now run the following command like before to finish writing to the SD-Card (there is no screen output, so wait for the command to finish):

```
$ sync
```

Unmount the SD-Card partitions, insert the SD-Card into the Raspberry Pi® 3 and reset it. Now the new Linux kernel should boot and you can continue with section [3.3](#). If the Raspberry Pi® 3 is not booting, see section [6.3](#) for Troubleshooting.

3.3 Software Packages and Configuration

This section describes further steps to install and configure software on the Raspberry Pi® 3 to enable working with the TPM.

3.3.1 Verifying the Previous Steps

The first thing you should do now is to validate that the previous steps (modifying and replacing the kernel) have been successful.

3.3.1.1 Verifying the Linux Kernel Replacement

In order to verify if the new kernel has been installed correctly, boot the Raspberry Pi® 3, log in and execute the following command on the Raspberry Pi® 3:

```
$ uname -a
```

The output should contain 4.4 as kernel version and the build date of the modified kernel, such as the following one

```
Linux raspberrypi 4.4.48-v7+ #1 SMP [build date and time] armv7l GNU/Linux
```

3.3.1.2 Verifying the Linux Kernel Modification

In order to verify if the new kernel contains and loads the TPM driver, attach the corresponding Infineon Iridium Board to your Raspberry Pi® 3, boot the platform, log in, view the contents of the `/dev` folder and verify that the device file `tpm0` exists on the Raspberry Pi® 3 with the following commands:

```
$ ls /dev/tpm*
```

The output should look like this:

```
/dev/tpm0
```

Additionally, you can view the kernel messages and look for TPM-related messages¹:

```
$ dmesg | grep tpm
```

For the Infineon SLB 9670 the output should look like this:

```
[some tick value] tpm_spi_tis spi0.1: 2.0 TPM (device-id 0xB892, rev-id 16)
[some tick value] tpm_spi_tis spi0.1: A TPM error (256) occurred continue
selftest
[some tick value] tpm_spi_tis spi0.1: Firmware has not started TPM
```

Even though an error message is shown, the TPM 2.0 should work.

¹ The command `grep` filters text for a given string; here the filter string is `tpm`.

3.3.2 Additional Packages

Before the TPM can be used with the applications described in this document, some packages must be installed. The following list shows all required additional software packages for this document. Depending on the used Linux distribution, the list might include some packages which are already installed:

- build-essential
- ~~libtsi-dev~~
- autoconf
- autoconf-archive
- automake
- libtool

The easiest way to make sure that all packages and their dependencies are installed is to call the installation routine for all of them again via `apt-get install [package name]` (requires an active Internet connection). Packages already installed will not be reinstalled, but updated if necessary. First the package information should be updated:

```
# apt-get update
```

Now install all packages with the following command in a terminal on the Raspberry Pi® 3:

```
# apt-get install build-essential libtspi-dev autoconf autoconf-archive  
automake libtool
```

If your microSD-card is less than 8 GB of size, you should also remove some packages not required for this document in order to have enough free disk space for the remaining steps in this document:

```
# apt-get autoremove libreoffice scratch wolfram-engine  
# apt-get autoclean
```

3.3.3 Install the Intel TPM Software Stack for TPM 2.0

To compile and install the Intel TSS for TPM 2.0 from [17] conduct the following steps.

Clone the repository from the GitHub project Intel TPM2.0 TSS. In the next subsection, the TPM2.0 tools will be installed. Since the names are quite similar and can be mixed up easily, the following naming convention is used. The software stack in this section is written in upper-case characters (TPM2.0-TSS) whereas the project containing the TPM2.0 tools (section 3.3.4) will be written in lower-case characters (tpm2.0-tools).

```
$ git clone https://github.com/01org/TPM2.0-TSS.git
```

If problems in the subsequent steps occur, use version 92f7adc. This version was verified and working while the application note was created..

```
$ git checkout 92f7adc
```

In the project, there is an `INSTALL` file ([18]) which shows how the TSS2 can be installed. If you follow the installation procedure listed in the file, the following commands will be executed.

```
$ ./bootstrap  
$ ./configure  
$ make  
# make install
```


Software Setup

```
# ldconfig
```

Now, the TSS 2 is installed and can be used.

3.3.4 Install the Intel tpm2.0-tools

The Intel Open Source Technology Center (01org) provides some TPM 2.0 tools [16] which allow interaction with the TPM 2.0 on a command line level. The tools are explained in detail in the manual available in the project ([19]).

```
$ git clone https://github.com/01org/tpm2.0-tools.git
```

If problems in the subsequent steps occur, use version ee62e78. This version was verified and working while the application note was created.

```
$ git checkout ee62e78
```

To configure and install the tpm2.0-tools, follow the build process.

```
$ ./bootstrap
```

```
$ ./configure
```

```
$ make
```

```
# make install
```

To use the tpm2.0-tools, the resource manager from the previous project (TPM2.0-TSS) needs to be running. This can be done starting the resource manager either manually every time the tpm2.0-tools need to be executed or automatically during boot time.

To start it manually, execute the following command:

```
$ cd TPM2.0-Tools/resourcemgr/
```

```
# ./resourcemgr
```

To start it automatically during the boot process, add the line from code sequence 001: to the file /etc/rc.local.

```
001: nohup sudo resourcemgr &
```

After a reboot, the resource manager runs in the background and manages all commands sent to the TPM 2.0 from the upper software layers. To verify a successful installation, run one of the easy to handle tpm2.0-tools commands. Listing all PCR values available in the TPM2.0 does neither require any authorization nor any additional command line parameters. It can be executed simply by executing the command in a terminal.

```
$ tpm2_listpcrs
```

4 Using the TPM with TSS2

The following sections describe how to start and use the TPM 2.0 on the Raspberry Pi® 3 with the TSS2 GitHub projects ([16], [17]) from Intel. Be sure that you have successfully completed all setup steps as described in the previous sections. For general information about the TPM, its operational states and requirements see the TPM specification, which can be found at [14]. All of the steps in this section have to be done on the Raspberry Pi® 3 (unless described otherwise). There are several ways to interact with the Raspberry Pi® 3, see section 6.8 for more information.

4.1 Introduction

TSS 2.0 is a TPM Software Stack (TSS) for the TPM 2.0 and basically consists of the resource manager and the actual software stack libraries. The resource manager runs as a system daemon providing a TCP/IP interface. It is a middle layer between the user application and the TPM driver in the kernel space. The resource manager which is part of the Intel TSS2.0 GitHub project ([17]) provides several functionalities, for example:

- Simplifies the execution of TPM commands by generating and processing TPM data structures by utilizing cryptographic functions.
- Marshals and unmarshals TPM commands to / from raw byte streams.
- Supports the establishment and management of secure communication from the host processor to the TPM chip with transport sessions.
- Provides resource management for the TPM, for example regarding sessions or the generation and storage of cryptographic keys.
- Multiplexes access to the TPM, so that different applications and / or separate users can independently use the TPM functions.

The second project from 01org (Intel Open Source Technology Center) is tpm2.0-tools. It is a set of executables, which can be used to issue TPM 2.0 commands from the user shell. tpm2.0-tools calls the corresponding TSS2 function, which sends commands to the resource manager. It converts these commands to raw bytes and sends them to the TPM driver in the kernel.

4.2 Verifying TSS2 functionality

In case you performed all steps in this document correctly so far, TSS2 should work now. You can easily test this by executing the following command:

```
$ tpm2_dump_capability -c properties-fixed
```

The output should look like this (the values might differ depending on your TPM):

```
TPM_PT_FAMILY_INDICATOR:
  as UINT32:      0x08322e3000
  as string:      "2.0"
TPM_PT_LEVEL:     0
TPM_PT_REVISION:  1.00
TPM_PT_DAY_OF_YEAR: 0x000000d1
TPM_PT_YEAR:      0x000007df
TPM_PT_MANUFACTURER: 0x49465800
TPM_PT_VENDOR_STRING_1:
  as UINT32:      0x534c4239
  as string:      "SLB9"
TPM_PT_VENDOR_STRING_2:
  as UINT32:      0x36373000
  as string:      "670"
```

...

If the command fails, please see troubleshooting section 6.3.3.

4.3 Taking TPM Ownership

Taking ownership of a TPM means basically to create several authorization values and a storage primary key.

1. Creating the authorization values:
 - a. The endorsement hierarchy authorization: endorsementAuth
 - b. The endorsement hierarchy policy: endorsementPolicy
 - c. The storage hierarchy authorization: ownerAuth
 - d. The storage hierarchy policy: ownerPolicy
 - e. The dictionary attack authorization: lockoutAuth
 - f. The dictionary attack policy: lockoutPolicy
2. Creating a storage primary key
3. Optional: Making the storage primary key resident in the TPM non-volatile memory

When the TPM is in an un-owned state (after a TPM2_Clear or when the TPM is in the default state after shipment) all authorization values will be set to an EmptyAuth and the policies will be set to an Empty Buffer.

Note: An EmptyAuth can be easily satisfied using the NULL password while an Empty Policy can never be satisfied.

Changing the authorization values (endorsementAuth, ownerAuth, lockoutAuth) will be performed with TPM2_HierarchyChangeAuth and changing the policies (endorsementPolicy, ownerPolicy, lockoutPolicy) will be done with TPM2_SetPrimaryPolicy.

The following flow shows a secure way to take ownership:

- Check capabilities to see if ownership is enabled
TPM2_GetCapability with TPM_PT_PERMANENT and TPM_PT_STARTUP_CLEAR checking for ownerAuthSet == 0 and shEnable == 1
- TPM2_CreatePrimary() to create an endorsement key (EK) for which a certificate exists
- Check the EK certificate
- Start an authorization session using the public portion of the endorsement key pair (EKpub) to protect the secret
- TPM2_HierarchyChangeAuth() using parameter encryption to protect the new auth values (endorsementAuth, ownerAuth, lockoutAuth)
- TPM2_SetPrimaryPolicy using the previously set auth values to set the corresponding policies (endorsementPolicy, ownerPolicy and lockoutPolicy)
- TPM2_CreatePrimary() to create a storage primary key
- TPM2_EvictControl to make the storage primary key persistent

Configure the dictionary attack parameters with TPM2_DictionaryAttackParameters.

4.4 Using TPM2.0 Tools

A complete overview of all tpm2.0-tools can be seen in [19]. It shows explanations for every command and its parameters. In Table 2, there is a summary of all available commands currently available in the GitHub project. Furthermore it shows two columns with the two basic use cases encrypt/decrypt using RSA and sign/verify using an elliptic curve cryptographic algorithm. To get an idea what commands are necessary to execute both use cases, the necessary commands are marked with an “x” in the appropriate rows.

Table 2 TPM2.0 tools command list and two use cases

Using the TPM with TSS2

Category	Commands	Encrypt/Decrypt	Sign/Verify
NV tools	tpm2_nvdefine tpm2_nvrelease tpm2_nvread tpm2_nvreadlock tpm2_nvwrite tpm2_nvlist		
Attestation tools	tpm2_takeownership tpm2_getpubek tpm2_getmanufec tpm2_getpubak tpm2_akparse tpm2_makecredential tpm2_activatecredential tpm2_listpcrs tpm2_quote tpm2_listpersistent	x	x
Key management tools	tpm2_createprimary tpm2_create tpm2_evictcontrol tpm2_load tpm2_loadexternal	x x x x	x x x (x)
Encryption tools	tpm2_encryptdecrypt tpm2_rsaencrypt tpm2_rsadecrypt tpm2_unseal	 x x 	
Signing tools	tpm2_sign tpm2_verifysignature tpm2_certify		x x
Utilities	tpm2_getrandom tpm2_hash tpm2_hmac tpm2_readpublic		

4.4.1 Encrypt and Decrypt a File using RSA 2048

Using asymmetric cryptography to securely exchange data between two peers is the basis for a lot of applications. The public key can be distributed to anyone who wants to send encrypted data to the owner of the private key. Usually, only small portions of data are encrypted by an asymmetric cryptographic algorithm because it takes significant amount of CPU power to do the mathematical calculations. These small data can be a symmetric key which is used in a later step for bulk encryption or configuration data for IoT applications. This section shows how a file can be encrypted and decrypted using the TPM 2.0 which is plugged to a RPi. In a real world scenario, the encryption may be done on a remote device which wants to send confidential data to the RPi. The encryption can be done on non-secure hardware as well because there is no data involved in the encryption process (plain-text, public key and cipher-text) which can be used to get any information about the private key. Therefore, using the TPM as an encryption service is possible, but not necessary.

Using the TPM with TSS2

The commands which are needed to execute the encryption and decryption use case are:

```
001: tpm2_createprimary
002: tpm2_create
003: tpm2_loadexternal
004: tpm2_rsaencrypt
005: tpm2_load
006: tpm2_rsadecrypt
```

These commands can be executed in a terminal on the RPi. To successfully run these commands, command line parameters have to be provided. An example for a correct execution of the commands is available in the test folder of the tpm2.0-tools GitHub project (e.g. `test_tpm2_rsadecrypt.sh` [20]). It contains many different shell scripts which execute the commands in the right order with the appropriate parameters. The following section shows the commands with parameters from the previous unordered list and explanations for each command to get an idea why each parameter is required. In the test script, all parameters are provided as variables. Using the scripts for automatic testing purposes, this is a good solution to reuse the parameters which are needed in different commands. In the context of this application note, all variables are replaced by their values to better see what is processed in every command. The original bash script variables have some unique names which were created to run all tests without having naming collisions. That's why the original context name (`context.p_B1`) is replaced by a simplified and more self-explaining name in 001: (`primary_ctx`).

```
tpm2_createprimary -A e -g 0x000B -G 0x0001 -C context.p_B1
001: tpm2_createprimary -A e -g 0x000B -G 0x0001 -C primary_ctx
                        IN |   IN   |   IN   |   OUT
```

The command `tpm2_createprimary` creates and loads a Primary Object under one of the Primary Seeds. To specify which seed should be used, the parameter `-A` has to be either `o`: Owner, `p`: Platform, `e`: Endorsement, `n`: Null. In this example, the Primary Object is created in the Endorsement Hierarchy. The command line option `-g` defines the hash algorithm used for computing the name of the object (`0x000B`: SHA256), `-G` defines the algorithm associated with this object (`0x0001`: RSA) and `-C` defines the file to save the object context (`primary_ctx`). The line below 001: shows which parameters are used as an input (IN) and which ones are used as output (OUT) from the command line function. The output is then used in the subsequent command as an input. In code line 001: the hierarchy, hash algorithm and the object algorithm are provided as input and the object reference (`-c primary_ctx`) is returned by the `tpm2_createprimary` function. This reference is then used in 002: as an input besides `-g` and `-G`.

```
002: tpm2_create -g 0x000B -G 0x0001 -c primary_ctx -o key_pub -O key_priv
                        IN   |   IN   |   IN   |   OUT   |   OUT
```

The `tpm2_create` command creates an encryption and decryption key and returns both the public and private portion of the key to the application (`key_pub`, `key_priv`). Both parts are stored as files in the folder where the command was executed. The private part is encrypted with the public part of the parent key and can therefore only be decrypted and used inside the TPM.

```
003: tpm2_loadexternal -H n -u key_pub -C rsaencrypt_key_ctx
                        IN |   IN   |   OUT
```

Due to resource limitations of the TPM, the keys are stored outside the chip, the public part unencrypted, the private part encrypted. This allows creating and storing an unlimited amount of keys with a single TPM 2.0 without any security drawback. The only disadvantage is that every key which shall be used for a cryptographic operation must be loaded into the TPM in advance of the actual operation. This mechanism can be simplified and accelerated using persistent storage inside the TPM for frequently used keys. Since performance is not a requirement for this sample use case, the keys are not stored permanently inside the TPM and must be loaded beforehand. The encryption key (`key_pub`) is loaded inside the TPM using the command shown in 003: It needs to have a parameter defining the hierarchy

Using the TPM with TSS2

where the encryption key should be loaded, the public key itself and returns the reference which must be used in the actual encryption process shown in 004:

```
004:  tpm2_rsaencrypt -c rsaencrypt_key_ctx -I input_data -o cipher_data
           IN                |           IN           |           OUT
```

To encrypt data with the TPM (004:), a filename for the plain-text input data (`input_data`) and a filename for the encrypted data (`cipher_data`) must be provided.

```
005:  tpm2_load -c primary_ctx -u key_pub -r key_priv -n name
           IN                |           IN           |           IN           |           OUT
           -C rsadecrypt_key_ctx
           OUT
```

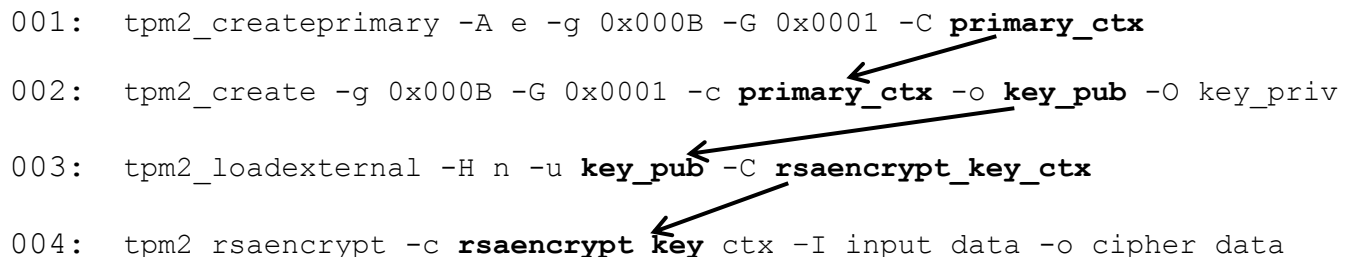
The command `tpm2_load` is used to load objects into the TPM when both the public and private part are needed for a certain command to be present inside the TPM. Regarding a decryption operation involving a private key, this requirement is fulfilled. The decryption process needs a key context (`rsadecrypt_key_ctx`) for referencing the private key inside the TPM. This handle is created by the `tpm2_load` command as an output shown in the second line of 005:

```
006:  tpm2_rsadecrypt -c rsadecrypt_key_ctx -I cipher_data -o output_data
           IN                |           IN           |           OUT
```

Additionally, the decryption command needs the previously created file with cipher-text data (`cipher_data`) and a file descriptor for the output data (`output_data`) as command line parameters. After executing this command, the content of `output_data` is the same as the `input_data`.

As a summary, Code Listing 1 and Code Listing 2 show all required commands with their appropriate parameters to encrypt or decrypt data respectively. The arrows depict what output(s) of a command is (are) used as input(s) in subsequent ones.

```
001:  tpm2_createprimary -A e -g 0x000B -G 0x0001 -C primary_ctx
002:  tpm2_create -g 0x000B -G 0x0001 -c primary_ctx -o key_pub -O key_priv
003:  tpm2_loadexternal -H n -u key_pub -C rsaencrypt_key_ctx
004:  tpm2_rsaencrypt -c rsaencrypt_key_ctx -I input_data -o cipher_data
```



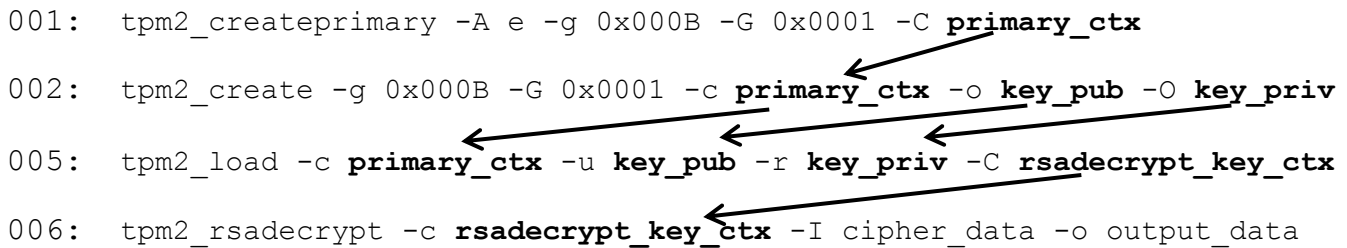
Code Listing 1 Encrypting data with RSA 2048 on the TPM 2.0

In Code Listing 2, lines 003: and 004: are missing because they are not needed for executing the commands in line 005: and 006:.. Merely the first two lines are shown to see what parameters which were created there are used in the decryption process afterwards. Furthermore, drawing the arrows in one listing is very confusing to distinguish the different parameters and their usage in the appropriate commands.

```

001: tpm2_createprimary -A e -g 0x000B -G 0x0001 -C primary_ctx
002: tpm2_create -g 0x000B -G 0x0001 -c primary_ctx -o key_pub -O key_priv
005: tpm2_load -c primary_ctx -u key_pub -r key_priv -C rsadecrypt_key_ctx
006: tpm2_rsadecrypt -c rsadecrypt_key_ctx -I cipher_data -o output_data

```



Code Listing 2 Decrypting data with RSA 2048 on the TPM 2.0

4.4.2 Sign and Verify using ECC 256

Creating a signature with a private key can be done for any data. The only precondition is that it is possible to create a hash value of the data which is e.g. a file or a message. If that's possible, the output of the hash function will be used as the input for the signing operation. After that, the signature can be used to check both the authenticity and integrity of data which was sent by the owner of the private key and received at a remote location. Therefore, the receiver needs to have the public key to successfully verify the signature. In terms of the TPM the following commands have to be executed to sign a file and verify the signature. The use case is available in the test script folder of the tpm2.0-tools project (test_tpm2_verifysignature.sh, [21]).

```

001: tpm2_createprimary
002: tpm2_create
003: tpm2_load
004: tpm2_sign
005: (tpm2_loadexternal)
006: tpm2_verifysignature

```

Listing 001: and 002: are basically the same as for the previous section. Both use cases need to have an asymmetric key pair available which can be used for encryption/decryption or signature/verification respectively.

```

001: tpm2_createprimary -A e -g 0x000B -G 0x0001 -C primary_ctx
                        IN  |  IN   |  IN   |  OUT

```

The `tpm2_createprimary` command is not needed in case the primary key was already loaded. Assuming that one executes the use case of the previous section and can reuse the primary context, the `tpm2_createprimary` needs not to be executed before the subsequent commands (e.g. `tpm2_create`). For completeness, the parameters used in the “create primary” command define the hierarchy (-A), the hash algorithm for object name (-g) and the algorithm for the primary key (-G). The command returns the primary context which is then used in subsequent commands (-C). The values provided in the parameter list specify that the primary key is loaded in the endorsement hierarchy (e), uses the SHA256 hash algorithm (0x000B) and takes RSA as the object algorithm (0x0001).

```

002: tpm2_create -g 0x000B -G 0x0023 -c primary_ctx -o key_pub -O key_priv
                        IN   |  IN   |  IN   |  OUT  |  OUT

```

The second command is almost the same one as in the previous section. The only difference to the `tpm2_create` command from the encryption/decryption use case is that instead of choosing RSA (0x0001) as an asymmetric cryptographic algorithm, ECC is used (0x0023). The key parameters (-o `key_pub` -O `key_priv`) are fairly self-explaining, the others have already been explained in the previous section or can be looked up in the manual file of the project (e.g. `tpm2_create`, lines 507-540, [19]). In the TPM specification (“Part 3: Commands, Revision 1.16”, Section 12.1.1, [23]), the `tpm2_create` command is summarized like this:

Using the TPM with TSS2

“This command [tpm2_create] is used to create an object that can be loaded into a TPM using TPM2_Load(). If the command completes successfully, the TPM will create the new object and return the object's creation data (*creationData*), its public area (*outPublic*), and its encrypted sensitive area (*outPrivate*). Preservation of the returned data is the responsibility of the caller. The object will need to be loaded (TPM2_Load()) before it may be used.”

This describes the same behavior as we have seen in the command. We get the public key (*outPublic* = *key_pub*), the encrypted private key (*outPrivate* = *key_priv*) and the creation data (*creationData* = *primary_ctx*).

```
003:  tpm2_load -c primary_ctx -u key_pub -r key_priv -n name
           IN          |      IN      |      IN      |      OUT      |
           -C eccsigning_key_ctx
           OUT
```

Since the signing operation is done using the private key of the asymmetric key pair, the command `tpm2_load` has to be executed. That's because only this command loads the private key into the TPM and provides a key context which can be then used to create a signature. In the TPM 2.0 specification (“Part 3: Commands, Revision 1.16”, Section 12.2.1, [23]), this is explained like this:

“This command [tpm2_load] is used to load objects into the TPM. This command is used when both a TPM2B_PUBLIC and TPM2B_PRIVATE are to be loaded. If only a TPM2B_PUBLIC is to be loaded, the TPM2_LoadExternal command is used.”

Thus, for the signing operation, the command `tpm2_load` instead of `tpm2_loadexternal` must be used. It needs to have the primary key context (`-c primary_ctx`) to decrypt the private key portion and both keys itself (`-u key_pub, -r key_priv`). The output parameter specifying the signing key context (`-C eccsigning_key_ctx`) is then used in the actual signing operation.

```
004:  tpm2_sign -c eccsigning_key_ctx -g 0x000B -m input_data -s signature
           IN          |      IN      |      IN      |      OUT      |
```

The `tpm2_sign` command makes use of the signing key (`-C eccsigning_key_ctx`) and creates a signature which is then saved in the `signature` file. Before signing the file, a digest from the `input_data` is created using SHA256 (`-g 0x000B`).

```
005:  tpm2_verifysignature -c eccsigning_key_ctx -g 0x000B -m input_data
                           IN          |      IN      |      IN      |
                           -s signature -t ticket_data
                           IN          |      OUT      |
```

Now, the command `tpm2_verifysignature` can be used to verify the signature which was created just before. There is no need to reload the public key into the TPM because it was already loaded with the load command in 003: as it is explained in the excerpt from the TPM specification. In case there is a reboot of the system or the signature is verified on a different system, then the public key has to be loaded into the TPM before the signature can be verified. This can be done using the command `tpm2_loadexternal`.

References

5 References

All links starting with “git://...” in the list below are links intended to be used with the GIT source code management system. If you like to inspect the GIT link target in a browser, replace “git://...” with “https://...” in the link’s address.

- [1] <http://www.infineon.com/tpm>
- [2] http://elinux.org/RPi_SD_cards
- [3] <http://www.raspberrypi.org/>
- [4] <http://www.ubuntu.com/download/desktop>
- [5] <http://packages.ubuntu.com/>
- [6] <https://downloads.raspberrypi.org/raspbian/images/raspbian-2017-01-10/2017-01-11-raspbian-jessie.zip>
- [7] <git://github.com/raspberrypi/linux.git>
- [8] <ftp://ftp.gnu.org/gnu/automake/automake-1.15.tar.xz>
- [9] <ftp://ftp.gnu.org/gnu/gmp/gmp-6.1.0.tar.xz>
- [10] <ftp://ftp.gnu.org/gnu/nettle/nettle-3.1.1.tar.gz>
- [11] <git://anongit.freedesktop.org/git/p11-glue/p11-kit.git>
- [12] <http://www.raspberrypi.org/faq>
- [13] <https://www.modmypi.com/blog/tutorial-how-to-give-your-raspberry-pi-a-static-ip-address>
- [14] http://www.trustedcomputinggroup.org/resources/tpm_main_specification
- [15] http://www.trustedcomputinggroup.org/resources/pc_client_work_group_specific_implementation_specification_for_conventional_bios
- [16] <https://github.com/01org/tpm2.0-tools>
- [17] <https://github.com/01org/TPM2.0-TSS>
- [18] <https://github.com/01org/TPM2.0-TSS/blob/master/INSTALL>
- [19] <https://github.com/01org/tpm2.0-tools/blob/master/manual>
- [20] https://github.com/01org/tpm2.0-tools/blob/master/test/system/test_tpm2_rsadecrypt.sh
- [21] https://github.com/01org/tpm2.0-tools/blob/master/test/system/test_tpm2_verifysignature.sh
- [22] https://github.com/01org/tpm2.0-tools/blob/master/src/tpm2_create.c
- [23] <https://trustedcomputinggroup.org/wp-content/uploads/TPM-Rev-2.0-Part-3-Commands-01.16-code.pdf>

6 Appendix

6.1 Dictionary Attack Protection

Besides other security mechanisms the TPM has a protection against guessing or exhaustive searches of authorization values stored within the TPM. To provide a suitable protection the parameters for the dictionary attack protection need to be carefully chosen and programmed into the TPM with the command `TPM2_DictionaryAttackParameters`.

The following parameters must be configured:

- `newMaxTries`: 32 or less
- `newRecoveryTime`: 7200 seconds or more
- `lockoutRecovery`: 86400 seconds or more

6.2 Cold Reboot of the Raspberry Pi® 3

A complete system reset (also known as *cold reboot*) of a Raspberry Pi® 3 must be done by performing the following steps:

1. Execute `# shutdown -hP now` as a root on the Raspberry Pi® 3 (for information on how to obtain root privileges, see section 6.7)
2. Disconnect the power supply and the HDMI cable from Raspberry Pi® 3 and wait for at least 5 seconds
3. Reconnect the power supply and the HDMI cable to the Raspberry Pi® 3

6.3 Troubleshooting

This section describes some useful hints in case you run into problems.

In general, to find out what causes an error, there are several possibilities that can be performed from a terminal on the Raspberry Pi® 3:

- Checking kernel messages:


```
$ dmesg | grep tpm
```

```
$ grep tpm /var/log/messages
```

```
$ grep tpm /var/log/syslog
```
- Checking whether the TPM device driver has been loaded:


```
$ ls /dev/tpm*
```

6.3.1 Platform Related Problems

Examples: Platform freezes during boot or does not boot at all.

Please make sure that:

- The used platform is a Raspberry Pi® 3 (a kernel for the Raspberry Pi® 3 will not boot on a Raspberry Pi® 1/2).
- Everything is properly connected.
- The power supply provides sufficient power (for example some USB ports do not provide enough power for the Raspberry Pi® 3 to start).
- The HDMI cable and the power cable have been unplugged before attempting to boot (otherwise the reset might not be complete due to remaining electrical capacity or standby power supply).
- The microSD-Card is supported (the Raspberry Pi® 3 is incompatible or has issues with some cards).

Appendix

- The microSD-Card is properly plugged in, partitioned and formatted.
- The OS is properly installed.
- The image has been completely written to the SD card using `sync` (sometimes the OS is not booting even after using the `sync` command; in this case simply rewrite the image to the SD card).

For additional information, please consult the general FAQ and / or troubleshooting pages of the Raspberry Pi® 3 [12].

6.3.2 Driver Related Problems

Examples: Loading the driver fails, TPM device file `/dev/tpm0` does not exist or TPM is not found. Please make sure that:

- The TPM is properly connected.
- Check if your kernel has TPM support by running

```
# zcat /proc/config.gz | grep -i tpm.
```

The output should look like this:

```
CONFIG_HW_RANDOM_TPM=m
CONFIG_TCG_TPM=m
```

You might need to run `# modprobe configs` first to create the `/proc/config.gz` archive. Using the wrong kernel may happen after a software upgrade. In that case you have to install the self-compiled kernel again.

- Check that you are using the correct kernel (it should be your self-built kernel 4.4) by running

```
$ uname -a or
```

```
$ cat /proc/version.
```

- The driver is loaded. You can try to load it manually as described in section [6.10](#) in order to find out if the automatic driver load mechanism is not working or if the driver does not run at all.

6.3.3 TSS2 or TPM 2.0 Usage Related Problems

Starting TSS2 or executing a TPM command fails unexpectedly.

Now retry the command that has failed before. If the error is still there, please make sure that the TPM is not in a dictionary attack lockout state. This happens when a wrong authorization value was entered too many times or when the lockout authorization was entered wrongly.

6.3.4 Kernel Configuration Related Problems

In case you cannot find the file `/proc/config.gz`, execute the following command on the Raspberry Pi® 3:

```
# modprobe configs
```

In some rare cases it may occur that you cannot activate the Infineon TPM driver inside the configuration menu started with `$ make ARCH=arm CROSS_COMPILE=arm-linux-gnueabihf- menuconfig`. In that case, try to delete the current configuration and reconfigure the kernel with its defaults with the following commands on the developer PC:

```
$ cd ~/TpmAppNote/linux
```

```
# make mrproper
```

```
$ make ARCH=arm CROSS_COMPILE=arm-linux-gnueabihf- bcm2709_defconfig
```

Now enable the TPM driver with the following command:

```
$ make ARCH=arm CROSS_COMPILE=arm-linux-gnueabihf- menuconfig
```

6.3.5 Internet Connection Related Problems

In case of connection problems with the internet, perform the following steps:

1. Since network configuration is being done on kernel startup, make sure that your device was already connected to your network on boot. If you are not sure, connect your device to the network, perform a platform reboot by executing `# shutdown -r now` and try the internet connectivity again.
2. Make sure that your device is properly connected to your network (all cables and interfaces are connected correctly) and that internet access is generally possible from this network.
3. In case you have to use a proxy server for connecting to the internet, make sure it has been properly configured. Configuring a proxy server is out of scope of this document.

6.4 Linux Kernel Support of Infineon SLB 9670 TPM 2.0

This document applies to the respective Linux distributions with kernel version 4.4 for the Raspberry Pi® 3. Other versions might work but may need some additional or different steps and have not been tested.

The basic circumstance in Linux affecting the driver support of specific hardware devices is the kernel version. Especially older and / or customized kernel versions do not have built-in support for the Infineon SLB 9670 and TPM2.0. For these kernel versions the hardware drivers have to be added by patching the kernel, enabling the driver in the kernel, compiling the kernel and using this modified kernel instead on the Raspberry Pi® 3.

6.5 Schematic and Layout of the Infineon Iridium SLB 9670 TPM 2.0 SPI Board

This section contains the schematics and layouts of the used Infineon Iridium TPM boards.

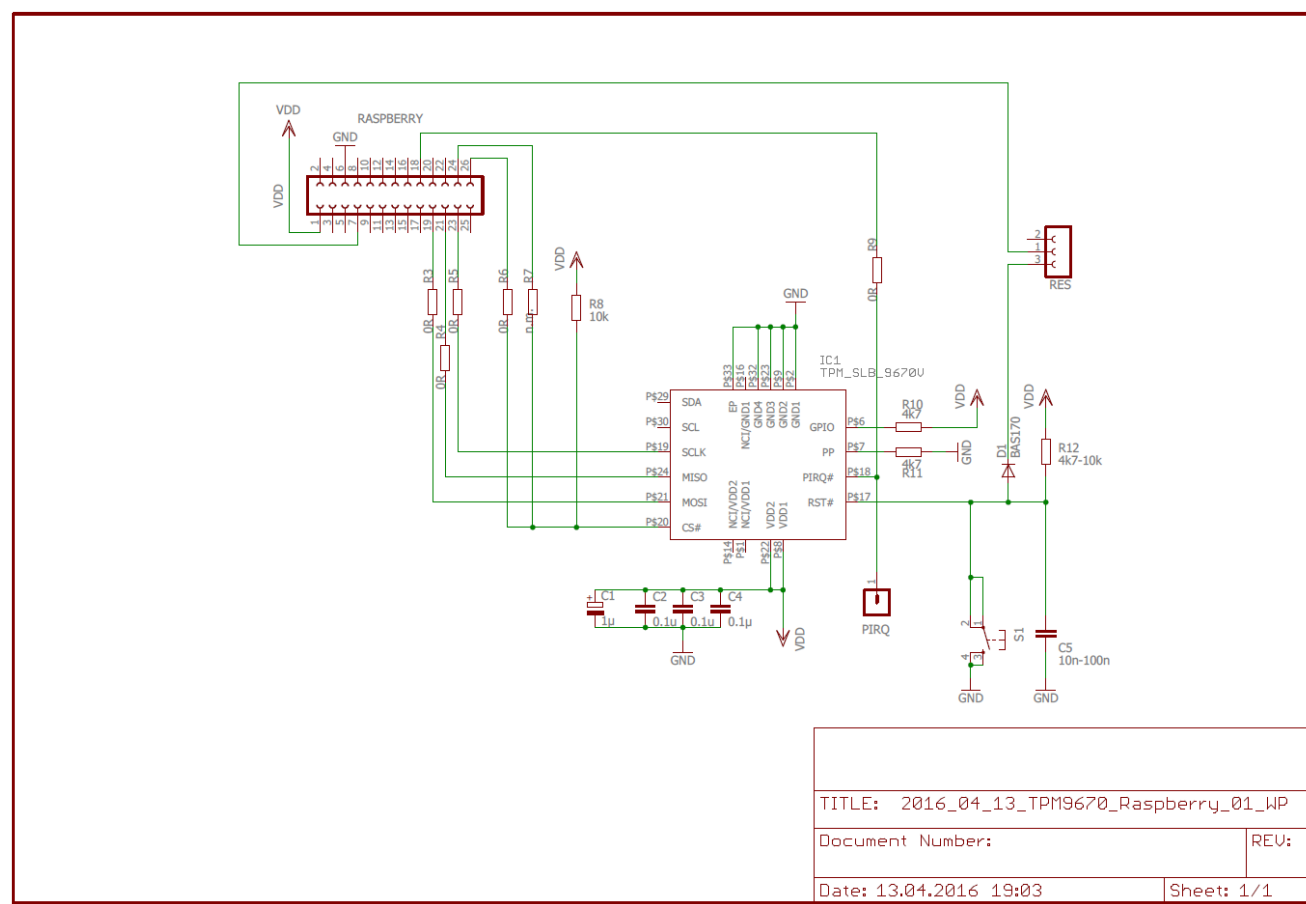


Figure 4 Schematic of the Infineon Iridium TPM 2.0 SPI Board

Appendix

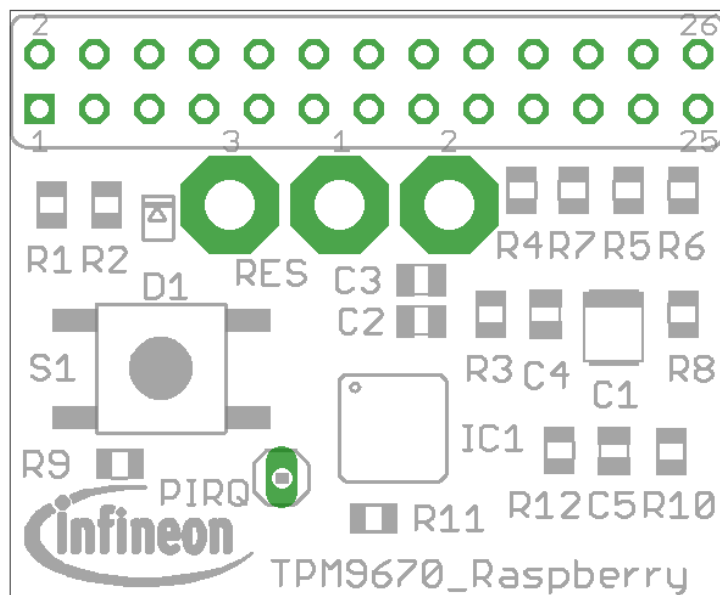


Figure 5 Placement of the Infineon Iridium SLB 9670 TPM 2.0 SPI Board

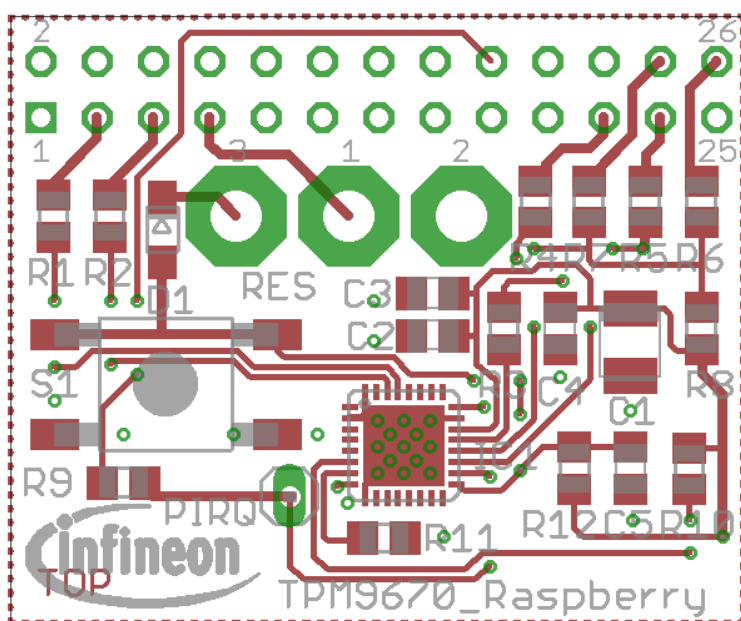


Figure 6 Layout of the component side of the Infineon Iridium SLB 9670 TPM 2.0 SPI Board

Appendix

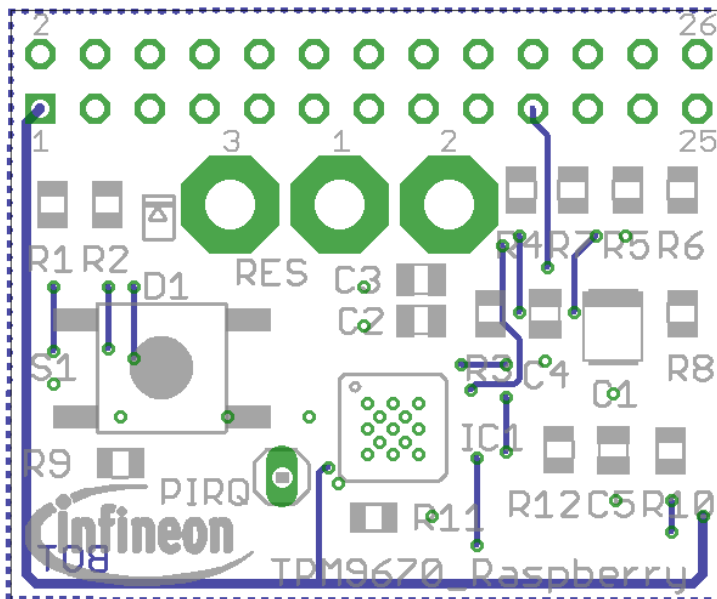


Figure 7 Layout of the solder side of the Infineon Iridium SLB 9670 TPM 2.0 SPI Board

Appendix

6.6 TPM 2.0 Features

The TPM provides the following features and extension possibilities to a platform:

- Secure storage of data
 - This storage is protected against unauthorized access through hardware and software attacks.
- Genuine trust anchor
 - During the process of personalization (that is one of the last manufacturing steps) a unique Endorsement Key and an Endorsement Certificate issued by the TPM manufacturer are being externally generated, imported and securely stored in the TPM's non-volatile memory.
 - The personalization happens in a highly secured and certified manufacturing environment, so by validating the Endorsement Certificate, one can verify that the TPM is genuine and thus trustworthy.
- Key hierarchy
 - A TPM key is always generated by derivation from a parent key (except for the storage root key (SRK)) and can in turn have multiple child keys.
 - Except for the SRK, every parent key also has a parent key.
 - Eventually, all keys are derived from the SRK. Thus, the SRK is the root for all keys.
 - By deleting a parent key, all of its child keys become invalid.
 - A TPM can have multiple keys for multiple users, applications and purposes.
 - Access to critical keys is secured by a secret, for example a user password in case of a user key, so the keys of one user can be protected from being accessed by another user.
- Ability to securely generate, store and use keys
 - The generated keys are protected by encryption with a parent key.
 - Before usage, this key must be loaded into the TPM, where it will be decrypted for usage.
 - Thus, data protected by this key is to be decrypted inside the TPM.
- Support for non-migratable keys
 - Such keys cannot be migrated to other TPMs; they are bound to the chip and the platform.
 - The parent key of a non-migratable key must also be non-migratable, thus obviously the SRK is always non-migratable.
 - A non-migratable key can have both migratable and non-migratable child keys.
 - The migration ability is set during key generation or import and cannot be changed afterwards.
- Hardware random number generator
 - Can generate non-predictable, cryptographically strong random numbers
 - This hardware random number generator (RNG) is protected against manipulation, thus these numbers are more reliable than numbers generated with a software RNG.
- Dictionary attack prevention mechanisms
 - Implemented in hardware and delays brute force attacks on the TPM by disabling the chip after a certain amount of failed authentication attempts. It is exponentially increasing the lockout time after every further failing attempt. This makes brute force attacks with a larger number of tries extremely time consuming and thereby nearly practically impossible which effectively hinders a successful attack.

6.7 Running commands with root privileges

This chapter describes how to run commands with root privileges from a terminal.

6.7.1 Using `sudo`

The package `sudo` is installed by default in Ubuntu® and Raspbian OS®.

If installed, it is possible to obtain root privileges by simply adding `sudo` before the actual command, for example when issuing the `shutdown` command:

```
$ sudo shutdown -hP now
```

Another possibility is to put your whole terminal session into a “root mode” by executing the following command:

```
$ sudo -s
```

This command elevates the privileges of the current terminal session to root privileges, thus all commands started in this terminal will now be executed as root. The root mode can be left by issuing an `exit` command.

6.7.2 Log in as root

Another possibility (though it might not be possible on all distributions) is of course to log off from the current user session and log in as root user. Obviously this is not as comfortable as using `sudo` but it helps in case `sudo` is not available.

6.8 Interacting with the Raspberry Pi® 3

For user interaction with the Raspberry Pi® 3, either a monitor and keyboard can be attached directly, or a remote connection via SSH can be used. This section describes both possibilities.

6.8.1 Direct interaction

In order to directly interact with the Raspberry Pi® 3, attach a USB keyboard to one of the USB ports of the Raspberry Pi® 3 and a monitor via HDMI. This enables the user to view the complete boot process of the Raspberry Pi® 3.

6.8.2 Connecting via SSH

To connect via Ethernet to the Raspberry Pi® 3, you can use the remote terminal SSH. Additionally, with SSH you can open multiple simultaneous terminal sessions with the same or different users.

SSH should be active by default. In case you have disabled SSH, you can reinstall it by running the following command from a terminal directly with an attached keyboard and monitor (see section 6.8.1):

```
# apt-get install ssh
```

In order to avoid warnings regarding missing system locales (that is if your PC uses locales that are not installed on the Raspberry Pi® 3), it is also recommended to disable the corresponding SSH option. To do so, uncomment the corresponding SSH option in the file `/etc/ssh/ssh_config` on the developer PC:

```
# nano /etc/ssh/ssh_config
```

Now change the line with

```
SendEnv LANG LC_*
```

to

```
# SendEnv LANG LC_*
```

(differences are marked gray).

To use SSH attach both the developer PC and the Raspberry Pi® 3 to the same network. In case you have a DHCP server running in your LAN, the boards will automatically obtain an IP address on boot, otherwise you have to manually configure an IP address for both of them (see [13] for example on how to do so).

For SSH to work, the IP addresses of both the developer PC and the Raspberry Pi® 3 must be in the same subnet, that is for IPv4 the first three blocks must be the same. For example, 192.168.100.1 and 192.168.100.2 are in the same subnet while 192.168.101.3 is in a different subnet. Normally, the DHCP server in your network (DHCP server can be a dedicated server or run on your router) takes care of this, but in case you configure IP addresses manually, you have to keep that in mind.

Depending on your network configuration, the Raspberry Pi® 3 will also publish its host name in the LAN, which you can get by executing the command `hostname` on the Raspberry Pi® 3. In this case, you can use SSH by executing the following command from a terminal on the developer PC:

```
$ ssh [username]@[hostname]
```

On a Raspberry Pi® 3 the default user is “pi”, the default hostname is “raspberrypi” and the default password for “pi” is “raspberry”. Thus, the SSH command would be the following:

```
$ ssh pi@raspberrypi
```

Appendix

If you cannot open a connection or your network configuration does not support to connect via the host name, you can also use the IP address, which you can get by executing `ifconfig` on the Raspberry Pi® 3. Now you can use SSH by executing the following command from a terminal on the developer PC:

```
$ ssh [user name]@[IP Address]
```

You can disconnect SSH by executing the following command in the same terminal on the developer PC:

```
$ exit
```

6.9 Exchanging Files

This section describes how to exchange files between the Raspberry Pi® 3 and the developer PC.

6.9.1 Direct Copy on microSD-Card

You can insert the microSD-Card into a card reader attached to the developer PC. Ubuntu® will then automatically mount the partitions on the card and you can copy files to it.

Mounting the Raspberry Pi® 3's micro-SD-card works reliably only on Linux-based operating systems, since other operating systems may only be able to mount the ext4 file system of the partitions on the microSD-card and / or may only be able to mount more than one partition on a microSD-card.

6.9.2 Copying via SCP

SCP uses SSH to copy files between two machines in a network. It requires a working SSH client installation on the local machine and a running SSH server on the remote machine.

The syntax is as follows:

```
$ scp [source] [target]
```

[source] and [target] can either be a normal file path (only in case of files on the local machine) or must have the following form (for files on the remote machine):

```
[remote username]@[remote hostname or IP address]:[remote file path]
```

Appendix

6.10 Manual Loading of the TPM 2.0 Driver

To manually load the TPM driver, execute the following command on the embedded platform:

```
# modprobe tpm_spi_tis
```

Note: To make this command work, the driver still must have been patched into the kernel.

To check whether the driver has been loaded successfully, execute the command

```
$ lsmod
```

The screen output should contain the following:

Module	Size	Used by
...		
tpm_spi_tis	2500	0
tpm	26497	1 tpm_spi_tis
...		

Alternatively, you can call

```
$ dmesg | grep tpm
```

The output should look like this:

```
[some tick value] tpm_spi_tis spi0.1: 2.0 TPM (device-id 0xB892, rev-id 16)
[some tick value] tpm_spi_tis spi0.1: A TPM error (256) occurred continue
selftest
[some tick value] tpm_spi_tis spi0.1: Firmware has not started TPM
```

In case you did not configure the kernel to load the driver automatically, you have to load the TPM driver manually after every system restart.

Revision History

Major changes since the last revision

Page or Reference	Description of change
Revision 1.0	Initial version

Trademarks of Infineon Technologies AG

μHVIC™, μIPM™, μPFC™, AU-ConvertIR™, AURIX™, C166™, CanPAK™, CIPOS™, CIPURSE™, CoolDP™, CoolGaN™, COOLiR™, CoolMOS™, CoolSET™, CoolSiC™, DAVE™, DI-POL™, DirectFET™, DrBlade™, EasyPIM™, EconoBRIDGE™, EconoDUAL™, EconoPACK™, EconoPIM™, EiceDRIVER™, eupec™, FCOS™, GaNpowIR™, HEXFET™, HITFET™, HybridPACK™, iMOTION™, IRAM™, ISOFACE™, IsoPACK™, LEDriviR™, LITIX™, MIPAQ™, ModSTACK™, my-d™, NovalithIC™, OPTIGA™, OptiMOS™, ORIGA™, PowIRaudio™, PowIRstage™, PrimePACK™, PrimeSTACK™, PROFET™, PRO-SiL™, RASIC™, REAL3™, SmartLEWIS™, SOLID FLASH™, SPOC™, StrongIRFET™, SuplIRBuck™, TEMPFET™, TRENCHSTOP™, TriCore™, UHVIC™, XHP™, XMC™

Trademarks updated November 2015

Other Trademarks

All referenced product or service names and trademarks are the property of their respective owners.

Edition 2017-03-16

Published by
Infineon Technologies AG
81726 Munich, Germany

©2017 Infineon Technologies AG.
All Rights Reserved.

Do you have a question about this document?

Email: erratum@infineon.com

Chipcard & Security
document reference

IMPORTANT NOTICE

The information contained in this application note is given as a hint for the implementation of the product only and shall in no event be regarded as a description or warranty of a certain functionality, condition or quality of the product. Before implementation of the product, the recipient of this application note must verify any function and other technical information given herein in the real application. Infineon Technologies hereby disclaims any and all warranties and liabilities of any kind (including without limitation warranties of non-infringement of intellectual property rights of any third party) with respect to any and all information given in this application note.

The data contained in this document is exclusively intended for technically trained staff. It is the responsibility of customer's technical departments to evaluate the suitability of the product for the intended application and the completeness of the product information given in this document with respect to such application.

For further information on the product, technology, delivery terms and conditions and prices please contact your nearest Infineon Technologies office (www.infineon.com).

WARNINGS

Due to technical requirements products may contain dangerous substances. For information on the types in question please contact your nearest Infineon Technologies office.

Except as otherwise explicitly approved by Infineon Technologies in a written document signed by authorized representatives of Infineon Technologies, Infineon Technologies' products may not be used in any applications where a failure of the product or any consequences of the use thereof can reasonably be expected to result in personal injury.