

# TPM

Trusted Platform Module

TPM with Embedded Platform

## Application Note

Infineon SLB 9645 TPM with Embedded Platform  
Revision 4.2, 2016-02-15

**Edition 2016-02-15**

**Published by  
Infineon Technologies AG  
81726 Munich, Germany**

**© 2016 Infineon Technologies AG  
All Rights Reserved.**

#### **LEGAL DISCLAIMER**

THE INFORMATION GIVEN IN THIS APPLICATION NOTE IS GIVEN AS A HINT FOR THE IMPLEMENTATION OF THE INFINEON TECHNOLOGIES COMPONENT ONLY AND SHALL NOT BE REGARDED AS ANY DESCRIPTION OR WARRANTY OF A CERTAIN FUNCTIONALITY, CONDITION OR QUALITY OF THE INFINEON TECHNOLOGIES COMPONENT. THE RECIPIENT OF THIS APPLICATION NOTE MUST VERIFY ANY FUNCTION DESCRIBED 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.

#### **Information**

For further information on technology, delivery terms and conditions and prices, please contact the nearest Infineon Technologies Office ([www.infineon.com](http://www.infineon.com)).

#### **Warnings**

Due to technical requirements, components may contain dangerous substances. For information on the types in question, please contact the nearest Infineon Technologies Office.

Infineon Technologies components may be used in life-support devices or systems only with the express written approval of Infineon Technologies, if a failure of such components can reasonably be expected to cause the failure of that life-support device or system or to affect the safety or effectiveness of that device or system. Life support devices or systems are intended to be implanted in the human body or to support and/or maintain and sustain and/or protect human life. If they fail, it is reasonable to assume that the health of the user or other persons may be endangered.

---

**TPM**

**Revision History: 2016-02-15, Revision 4.2**

Page	Subjects (major changes since last revision)
7, 51, 59, 61	Minor wording improvements
Revision 4.0: All pages	First released version

**Trademarks**

Cortex® and ARM™ of ARM Ltd., Broadcom® of the Broadcom Corporation, Debian® of the Debian Project, OMAP™ and Texas Instruments™ of Texas Instruments Incorporated, Raspberry Pi® of the Raspberry Foundation, SD™ and microSD™ of SD-3C, LLC., Ubuntu® and Canonical® of Canonical Ltd..

Infineon® is a registered trademark of Infineon Technologies Ltd.

Last Trademarks Update 2013-07-04

## Table of Contents

<b>1</b>	<b>Introduction .....</b>	<b>7</b>
1.1	Motivation.....	7
1.2	Scope .....	7
1.3	Command Conventions.....	8
1.4	Acronyms and Abbreviations .....	8
<b>2</b>	<b>Hardware Setup .....</b>	<b>9</b>
2.1	BeagleBoard xM .....	10
2.2	Raspberry Pi® .....	11
2.3	Infineon Iridium SLB 9645 TPM I2C Board .....	12
<b>3</b>	<b>Software Setup .....</b>	<b>15</b>
3.1	Developer PC Installation.....	15
3.2	Embedded Platform Installation .....	16
3.2.1	BeagleBoard xM .....	16
3.2.2	Raspberry Pi® .....	20
3.3	Software Packages and Configuration .....	24
3.3.1	Verifying the Previous Steps .....	24
3.3.2	Correcting the Time Zone and System Time .....	24
3.3.3	Additional Packages for Embedded Platforms.....	25
<b>4</b>	<b>Using the TPM .....</b>	<b>29</b>
4.1	First Time Initialization of the TPM.....	29
4.1.1	Stopping the TCSD.....	29
4.1.2	Initializing the TPM .....	29
4.1.3	Resetting the TPM.....	29
4.1.4	Locking the TPM.....	30
4.2	Using the TPM with TrouSerS .....	31
4.2.1	Introduction .....	31

4.2.2	Starting TrouSerS .....	32
4.2.3	Testing TrouSerS .....	32
4.3	Taking TPM Ownership .....	33
4.4	Establishing a Secure Channel Using a TPM.....	34
4.4.1	Introduction .....	34
4.4.2	OpenSSL .....	35
4.4.3	GnuTLS .....	38
<b>5</b>	<b>References.....</b>	<b>42</b>
<b>6</b>	<b>Appendix .....</b>	<b>43</b>
6.1	Troubleshooting.....	43
6.1.1	Platform Related Problems.....	43
6.1.2	Driver Related Problems .....	43
6.1.3	TPM Startup Related Problems.....	44
6.1.4	TrouSerS or TPM Usage Related Problems .....	44
6.1.5	Clearing the TPM.....	45
6.1.6	Kernel Configuration Related Problems.....	45
6.1.7	Internet Connection Related Problems .....	46
6.2	Linux Kernel Support of Infineon SLB 9645.....	47
6.2.1	Overview .....	47
6.2.2	Patching Kernel Versions 3.7 to 3.9 .....	48
6.3	Schematic and Layout of the Infineon Iridium SLB 9645 TPM I2C Board .....	49
6.4	TPM Features .....	51
6.5	Running Commands with Root Privileges .....	52
6.5.1	Using Sudo .....	52
6.5.2	Logging in as Root .....	52
6.6	Interacting with the Embedded Platforms.....	53
6.6.1	Direct Interaction .....	53

6.6.2	Connecting via RS232 (BeagleBoard xM only) .....	53
6.6.3	Connecting via SSH.....	54
6.7	Manual Loading of TPM Driver .....	56
6.8	TPM States and the TPM Startup Script.....	57
6.9	Cryptographic Principles Used in SSL / TLS .....	59
6.9.1	Cryptography.....	59
6.9.2	Signing .....	59
6.9.3	Certificates .....	60
6.10	The OpenSSL Heartbleed Bug .....	61

## 1 Introduction

This document explains each step from the installation and configuration of a typical embedded platform to work with the Infineon TPM 1.2 (Trusted Platform Module) with I<sup>2</sup>C-Interface, the *Infineon SLB 9645*. An overview on the various TPMs offered by Infineon can be found on Infineon's TPM website [1].

The following typical embedded platforms are being used as host system for the TPM in this document: the BeagleBoard xM and the Raspberry Pi®. The TPM-related examples described in this document are the same for both boards, so one of these boards is sufficient to gain knowledge about the TPM and its applications.

The differences are the underlying hardware architecture, the OS installation and the Linux kernel modification, that is the BeagleBoard xM already uses a *Flattened Device Tree* (or *FTD*) as default to describe its hardware components (while the Raspberry Pi® still uses the "old-fashioned way", where the hardware description is hard-coded in the kernel).

In case you run into any problems during execution of the steps described in this application note, please have a look into the corresponding troubleshooting section in appendix 6.1. 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 is obvious to almost everyone, the second one often does not get the appropriate amount of attendance. Authenticity is just as important as confidentiality, because for example the highest confidentiality of secret data is worth nothing in case the secret data 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 exact 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.

Thus, a TPM can help authenticating a user or client against a server, but also of a server against a client or user. The latter is being performed every time a TLS<sup>1</sup> connection is being established in an internet browser in order to prove to the client that it is connected to the correct server and not a malicious fake server or man-in-the-middle. Unfortunately, in almost every case this platform authentication is being done without a TPM as an additional authentication factor, which in combination with software bugs may lead to critical security vulnerabilities.

Another basic feature of the TPM is that it can function as a starting point for establishing a root of trust, which allows detecting 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 6.4.

### 1.2 Scope

This document is intended for users to help them getting familiar with how to use a subset of the TPM's functionalities on embedded platforms running Linux. The exact goal in this application note is to generate a secure communication channel between a client and a server application (for simplicity both running on the same platform).

The topics described in this document are:

---

<sup>1</sup> TLS = Transport Layer Security, successor of SSL (= Secure Sockets Layer);

## Introduction

- The Infineon Iridium SLB 9645 TPM I2C Board, which has two expansion headers for the BeagleBoard xM (running Linux kernel<sup>1</sup> version 3.14.1+) and Raspberry Pi® (running kernel version 3.14.6+).

*Note: This document has been generated using exactly these kernel versions. Other versions might work, but may need some additional or different steps and have not been tested.*

- Usage of the TPM on the BeagleBoard xM and Raspberry Pi®.
- Create and use an OpenSSL or GnuTLS certificate with their sensitive data protected by the TPM.

This document uses platform-specific Linux distributions based on Debian® to describe the system setup and the application of the TPM. If not explicitly said otherwise, the document refers to specific versions of the required Linux packages to prevent compatibility problems. The mentioned versions were publicly available and downloaded on the 6th of August 2014 from the specified links and repositories.

## 1.3 Command Conventions

In this document, Linux will be used as operating system for all platforms. Since some commands in this Application Note are directly interacting with hardware or accessing protected system files, these commands have to be executed with administrative privileges (also known as root privileges). The convention in this document will be that whenever a command requires root privileges, the actual command is preceded by "#", as for 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 [6.5](#).

## 1.4 Acronyms and Abbreviations

Acronym	Explanation
FTD	Flattened Device Tree
LTS	Long Term Support
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
VPN	Virtual Private Network

---

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



## 2 Hardware Setup

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

- **Developer PC:**  
This platform is used for setting up, maintaining and interacting with the embedded platform in a more convenient and faster way compared to doing all actions directly on the embedded platform. The hardware requirements for the developer PC are
  - Desktop computer or laptop with x86 architecture and USB 2.0
  - Capable of running Linux, for example Ubuntu® 14.04 LTS
  - Connected to the Internet
  - Connected to a network (optional<sup>1</sup>)
- **Infineon Iridium SLB 9645 TPM I2C Board:**  
This board contains the Infineon SLB 9645 mounted on an easy-to-use hardware board, which can be attached to the embedded platform.
- **RJ45 Ethernet LAN cable:**
  - Connected to the Internet
  - Connected to the same network as the developer PC (optional<sup>1</sup>)
- **Embedded platform:**  
This is the actual embedded platform hosting and utilizing the TPM. Required are either
  - BeagleBoard xM Rev C
  - microSD-Card (will be used as mass storage device) with at least 4 GB
  - Mini-B USB cable (will be used for power supply)
  - RS232 Serial-to-USB adapter and USB extension cable for the connection to developer PC (optional<sup>1</sup>)or
  - Raspberry Pi® Model B Rev 2012 (other models might work, but this has not been tested)
  - SD-Card<sup>2</sup> (will be used as mass storage device) with at least 4 GB
  - Micro-B USB cable (will be used for power supply)
- **Monitor or beamer with HDMI and HDMI cable and USB keyboard (optional<sup>1</sup>)**

---

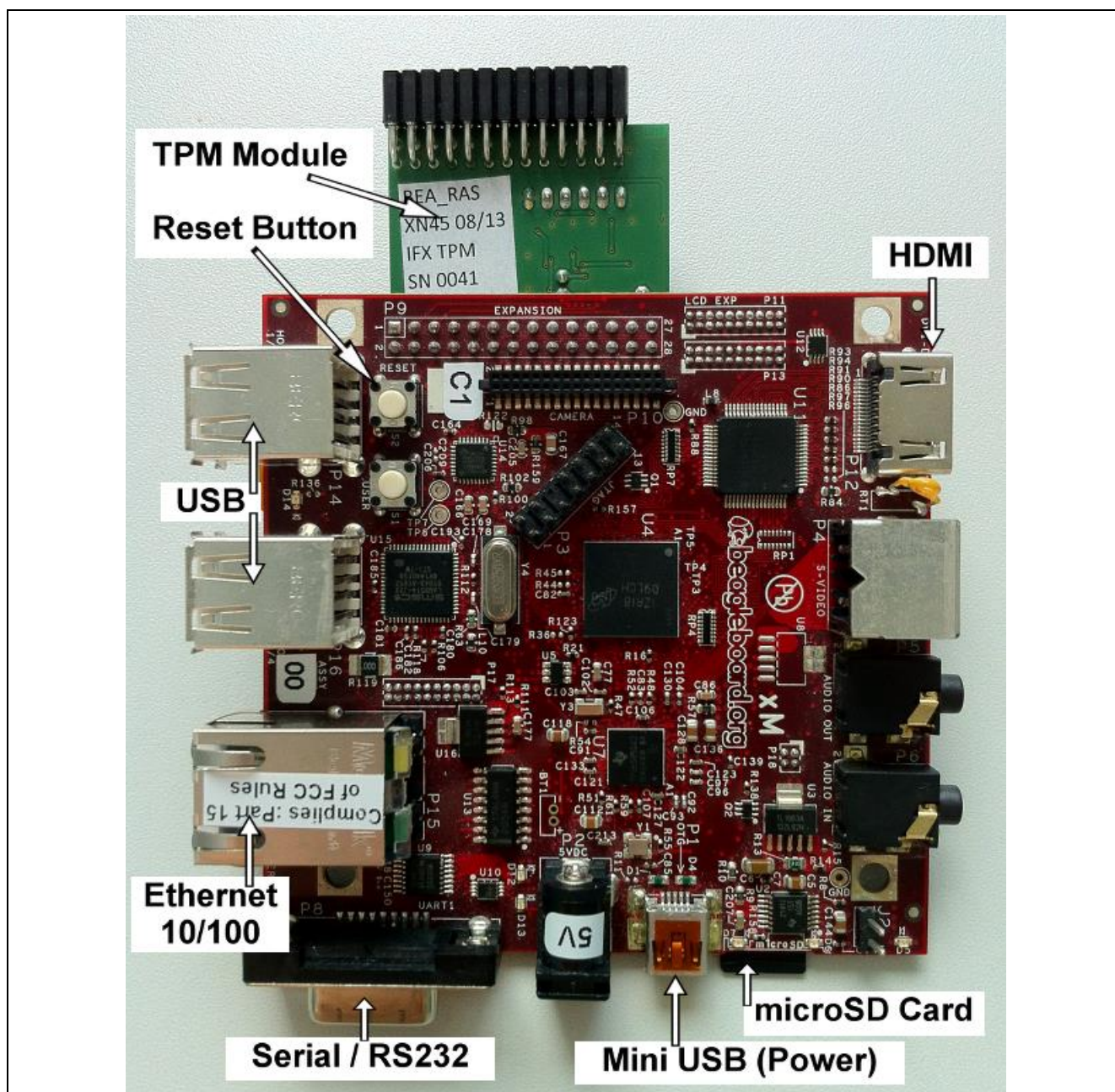
<sup>1</sup> Depends on how you interact with the embedded platform; details will be described in section [6.6](#).

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

## 2.1 BeagleBoard xM

The BeagleBoard xM Rev C has an OMAP™ 3 compatible DM3730 SoC from Texas Instruments™ with a 1000 MHz ARM™ Cortex®-A8 processor and 512 MB RAM. A HDMI or S-Video connector can be used for a graphical output.

**Figure 1** shows the BeagleBoard xM setup with expansions. For more information visit the official website [3].



**Figure 1** BeagleBoard xM with expansions

### Serial / RS232 Port

The developer PC can operate the BeagleBoard xM via a Serial-to-USB adapter and a terminal program.

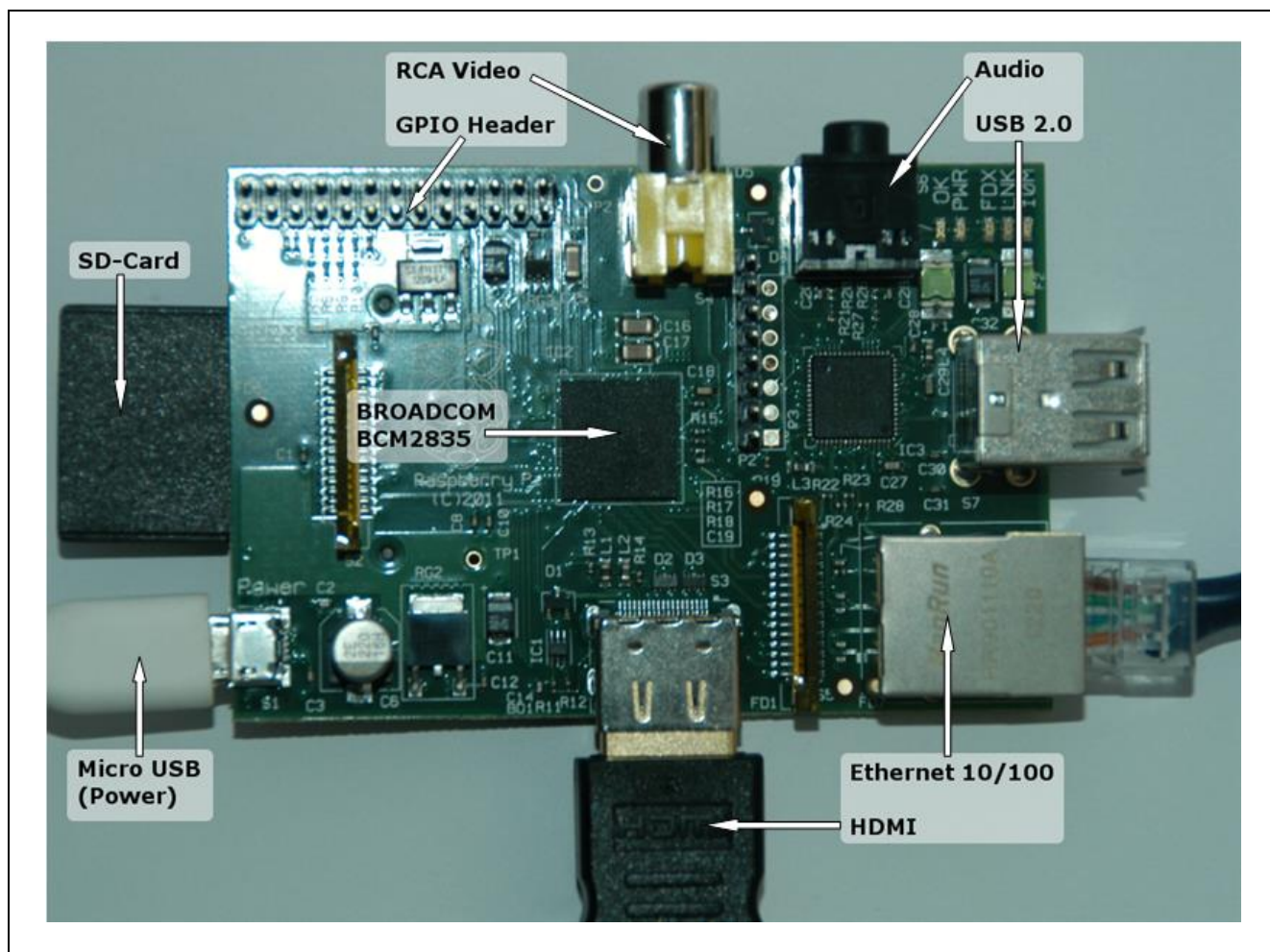
### Reset Button

With the reset button the user can perform a cold reboot of the BeagleBoard xM.

## 2.2 Raspberry Pi®

The Raspberry Pi® version B Rev 2012 has a Broadcom® BCM2835 SoC with a 700 MHz ARM™ 1176JZF-S CPU and 512 MB RAM. A HDMI or RCA connector can be used for graphical output.

**Figure 2** shows the Raspberry Pi® setup with all expansions. For more information visit the official website [4].



**Figure 2** Raspberry Pi® expansions

The Raspberry Pi® has an Ethernet controller for Internet and a micro USB port to supply the Raspberry Pi® with power. Since it has no reset button, a cold reboot can only be triggered if the Raspberry Pi® is disconnected from power.



## 2.3 Infineon Iridium SLB 9645 TPM I2C Board

The Infineon Iridium SLB 9645 TPM I2C Board<sup>1</sup> can connect with BeagleBoard xM or Raspberry Pi® via one of the two expansion headers. For data transfer the board uses the I<sup>2</sup>C bus. The basic layout can be seen in Figure 3.



**Figure 3 Infineon Iridium SLB 9645 TPM I2C Board**

On the top is the Raspberry Pi® Header with 26 pins, on the bottom the BeagleBoard xM Header with 28 pins. The board contains a reset circuit on board, which asserts the reset line of the TPM for the right amount of time after VCC becomes available. The reset on the module is active low and can also be set by connecting the reset and the ground via the reset button. The reset will be needed for a few commands, for example the TPM\_Clear command. This button resets the TPM to a state with all volatile flags initialized to their default values. LED1 shows the status of the 5 volt rail and LED2 the status of the VCC connection.

<sup>1</sup> Order type: Iridium 9645 TPM I2C Linux  
Order number: SP001265088

Figure 4 shows all PIN assignments for the Infineon Iridium SLB 9645 TPM I2C Board:

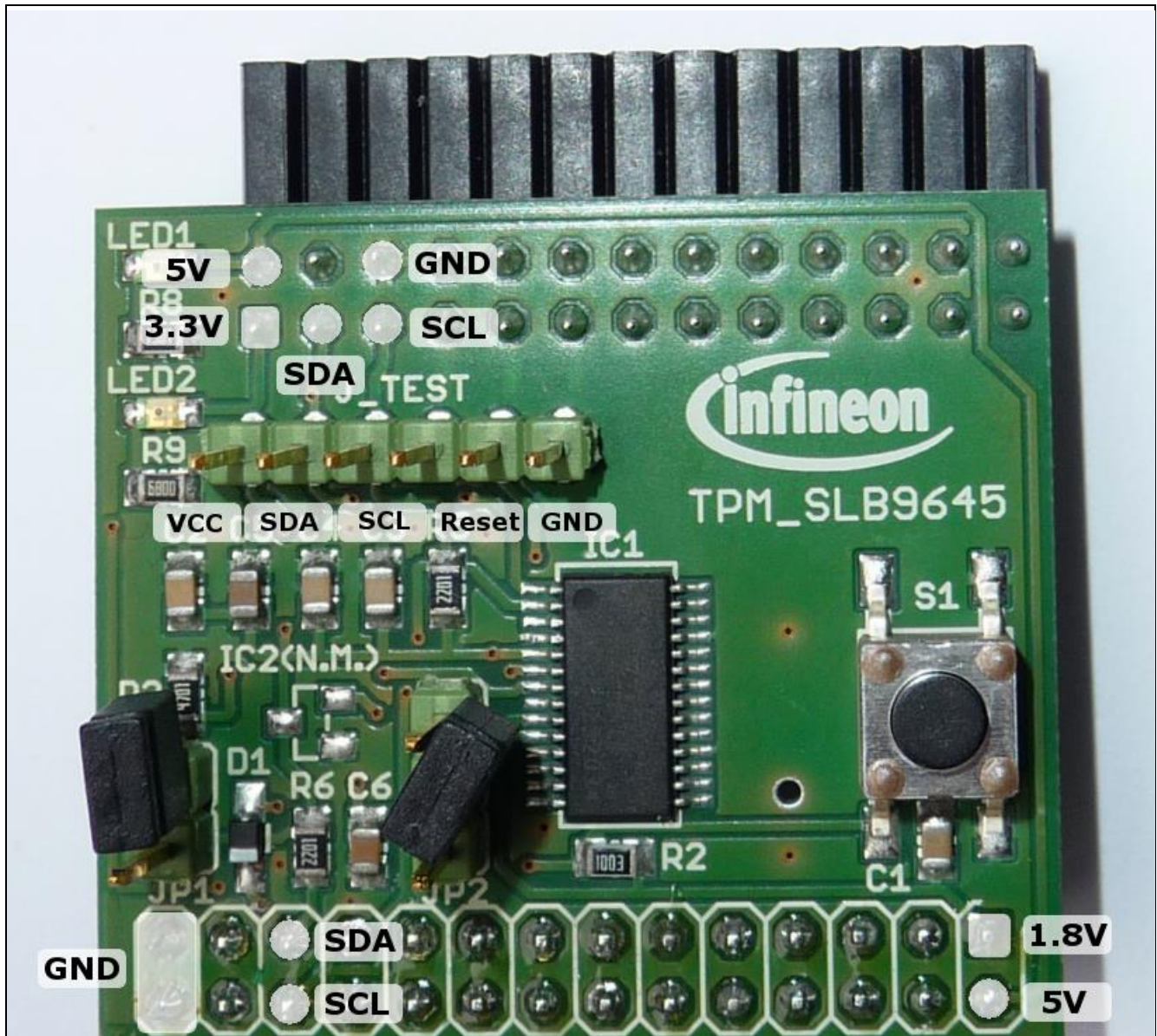


Figure 4 Infineon Iridium SLB 9645 TPM I2C Board PIN assignments

The Raspberry Pi® Header has 3.3 Volt, 5 Volt, SCL, SDA and Ground at the following pins:

**Table 1** Raspberry Pi® Expansion Connector Signals

Pin	1	2	3	4	5	6
Signal	+3.3 V	+5 V	SDA	(not connected)	SCL	GND

The BeagleBoard xM Header has 1.8 Volt, 5 Volt, SCL, SDA, and doubled Ground at the following pins:

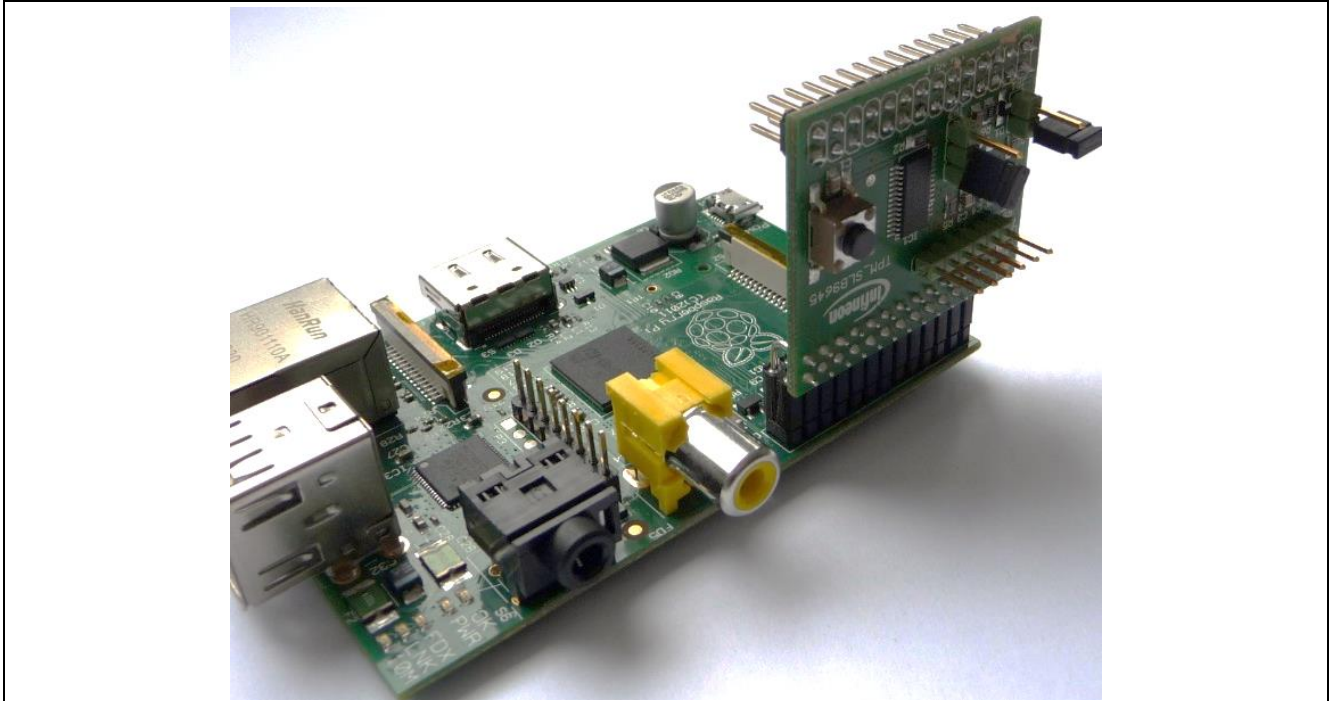
**Table 2** BeagleBoard xM Expansion Connector Signals

Pin	1	2	23	24	27	28
Signal	+1.8 V	+5 V	SDA	SCL	GND	GND

A complete layout of the board can be found in section 6.3.

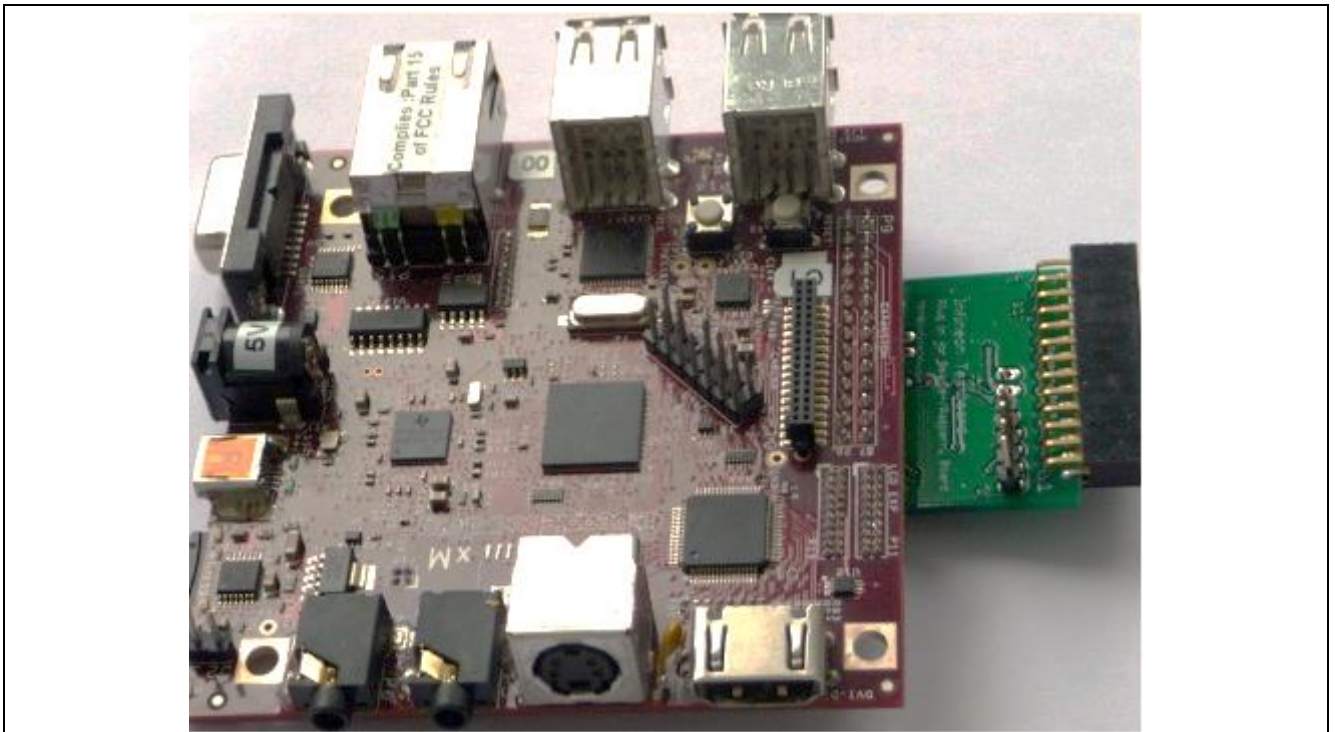


**Figure 5** shows the Infineon Iridium SLB 9645 TPM I2C Board mounted on a Raspberry Pi®:



**Figure 5** TPM on Raspberry Pi®

**Figure 6** the Infineon Iridium SLB 9645 TPM I2C Board mounted on a BeagleBoard xM:



**Figure 6** TPM on BeagleBoard xM

## 3 Software Setup

This section describes the software used in this application note on both, the embedded platforms (BeagleBoard xM and Raspberry Pi®) and the developer PC.

*Note: Some steps in this section have to be done **on the developer PC**, some **on the embedded platform**. There are several ways to interact with the embedded platform, see section 6.6 for more information.*

### 3.1 Developer PC Installation

The developer PC can be either a native or virtual machine and will be used mainly for compiling and setting up the Linux OS for the embedded platforms on a (micro)SD-Card. But it can also be used to control the embedded platforms remotely.

It is recommended to use a Linux Distribution such as Ubuntu®, Debian®, or similar on the developer PC. This document refers to the usage of a native machine running Ubuntu® 14.04 LTS, which can be downloaded at [5].

*Note: All commands and troubleshooting hints in this application note have only been tested on this exact same Ubuntu® version. Other versions can be used on your own risk. To install Ubuntu®, just follow the installation instructions at [5].*

The basic installation of Ubuntu® comes by default with many packages installed depending on the selections made during the installation of Ubuntu®. Beyond that, the following list shows all additional software packages required to be able to perform all steps in this application note. Depending on the used Linux distribution, the list might include some packages which are already installed:

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

Additional packages required for working with BeagleBoard xM:

- minicom (only required for communication with BeagleBoard xM via serial port, see section 6.6.2)
- device-tree-compiler
- lzma
- lzop

Further information on these packages can be found at [6].

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 command 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<sup>1</sup> in the command terminal **on the developer PC**:

```
# apt-get install git openssh-server gddrescue gcc-arm-linux-gnueabi  
libncurses5-dev minicom device-tree-compiler lzma lzop
```

---

<sup>1</sup> **Attention: Remember to always execute commands starting with “#” as root (see section 6.5 on how to do so).**

## 3.2 Embedded Platform Installation

The following sections describe some generic aspects affecting both embedded platforms and the exact software setup of the BeagleBoard xM and Raspberry Pi®.

In order to get the TPM working on one of these platforms, their operating system kernel must have driver support for the TPM, which does not apply to all kernel versions. More information on this topic can be found in section 6.2.

The following sections give detailed instructions how to patch and install a Linux with kernel version 3.14 on the embedded platforms. Other kernel versions might work with more or less effort (see section 6.2). This application note has been specifically written for and tested with kernel version 3.14.

It is recommended to create a subfolder for this application note in your **home** directory on the developer PC. Use a unique name such as **TpmAppNote**. To generate and use such a folder, log on to your system, open a console window (for example by pressing <Ctrl> + <Alt> + <T>) and type the following commands:

```
$ mkdir ~/TpmAppNote
$ cd ~/TpmAppNote
```

This folder is now our working space. Copy the **Src** folder shipped with this Application Note into this folder.

**Attention: All folder references on the developer PC refer to the layout of this workspace, especially the relative location of the Src folder to the current working directory. In case a different folder name and / or structure is being used, the paths in the commands in this document must be adapted accordingly.**

### 3.2.1 BeagleBoard xM

The following sections describe how to install and configure Linux on a BeagleBoard xM. For installation and configuration of Linux on a Raspberry Pi®, skip this section and continue with section 3.2.2. This document describes the usage of the Debian® distribution on the BeagleBoard xM.

#### 3.2.1.1 Installation of Linux

The first step is to download an image of Debian® for the BeagleBoard xM from [7] and unpack the downloaded archive on the developer PC. To do so, switch to your workspace folder:

```
$ cd ~/TpmAppNote
```

Now download and extract the OS image:

```
$ wget https://rcn-ee.net/deb/rootfs/wheezy/debian-7.6-console-armhf-2014-08-13.tar.xz
$ tar -xvaf debian-7.6-console-armhf-2014-08-13.tar.xz
$ cd debian-7.6-console-armhf-2014-08-13
```

After that, connect a microSD-Card via a card reader to the developer PC and execute the following single command, where **sdX** is the device file for the microSD-Card. This has to be replaced by the correct device file for the microSD-Card, for example **sdc**. The device letter for the microSD-Card depends on the system and can be found with the command **fdisk -l** (fdisk requires root privileges). The device is automatically being partitioned, formatted and the basic packages are being installed.

**Attention: 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 microSD-Card, since the command will cause the drive represented by the given device to be completely overwritten!**

```
# ./setup_sdcard.sh --mmc /dev/sdX --dtb omap3-beagle-xm
```

Once completed, connect the network and insert the SD-Card into the BeagleBoard xM, connect a USB keyboard and a monitor (see section 6.6 for different ways how to interact with the BeagleBoard xM), and plug in the power cable. Now the system should start (it might perform several automatic reboots on first start, just wait until they have finished).



You can logon with the shown credentials:

- Username: `debian`
- Password: `temppwd`

You should now update the installed system to get the latest fixes and security patches. To do so, perform the following steps **on the embedded platform** (see section 6.1.7 in case of internet connection problems):

```
# apt-get update
# apt-get upgrade
```

Confirm possibly upcoming questions about new packages to be installed with <Y> and <Enter> and let the upgrade process finish its work.

In the meantime you can continue with the next step, which is to configure and compile a new kernel for the BeagleBoard xM.

### 3.2.1.2 Patching, Configuring and Compiling a Kernel

Some older kernel versions may not support a TPM and in particular not a TPM with an I<sup>2</sup>C interface such as the Infineon SLB 9645. (Details on support for the Infineon TPM with an I<sup>2</sup>C interface can be found in section 6.2.) And even the latest kernel versions may need to be properly patched, configured and recompiled for activating and automatically loading the Infineon SLB 9645 device driver. This section shows how to do so for kernel 3.14 running on the BeagleBoard xM.

*Note: Patch files are using relative paths to modify the original files. Thus, you have to be in the correct folder for them to work, otherwise they will not find the files to modify. To ensure this, please follow each of the described steps exactly.*

To do so, perform the following steps **on the developer PC**, because the compile process takes more than a few hours on the BeagleBoard xM.

First, switch to your workspace folder:

```
$ cd ~/TpmAppNote
```

Now you need the BeagleBoard xM Linux kernel base repository which can be downloaded via git from [8]:

```
$ git clone git://github.com/beagleboard/kernel.git
```

*Note: Behind a proxy you might need to use the slower HTTPS protocol instead of the faster GIT protocol. To do so, replace "git://..." in the URL above with "https://..." and check the documentation of git for the configuration of a proxy.*

Switch to the folder **kernel** generated by the previous command and get the files for kernel version 3.14 with the following two commands:

```
$ cd kernel
$ git checkout origin/3.14 -b 3.14
```

Now execute the **patch.sh** script to download and patch the BeagleBoard kernel.

*Note: Behind a proxy you might need to use the slower HTTPS protocol instead of the faster GIT protocol. To do so, edit the **patch.sh** file and replace "git://..." with "https://..." in every URL.*

```
$ ./patch.sh
```

Switch to the **kernel** subfolder of the current folder:

```
$ cd kernel/
```

The current path should be **~/TpmAppNote/kernel/kernel/** now.

In the next step the original kernel configuration of the kernel installed on the BeagleBoard xM is used as a basis for the configuration of the modified kernel 3.14. For that copy the **/proc/config.gz** file via SSH from the

## Software Setup

BeagleBoard xM to the developer PC. To do so, start the BeagleBoard xM and type the following command **on the developer PC**:

```
$ scp debian@arm:/proc/config.gz ~/TpmAppNote/
```

*Note: In case “arm” cannot be resolved, use the IP address of the BeagleBoard xM instead. You can get it with running `ifconfig` **on the BeagleBoard xM**.*

The current path should still be `~/TpmAppNote/kernel/kernel/`. Then unzip the file **config.gz** and store it with the name **.config** into the current folder:

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

Now copy and apply the patch file named **BeagleBoard\_xM\_Automatic\_TPM\_Driver\_Load.patch**, which is shipped with this document. This patch activates the I<sup>2</sup>C bus and loads the Infineon I<sup>2</sup>C TPM driver automatically on system start.

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

*Note: This patch does not only update the BeagleBoard xM’s flattened device tree, but also the corresponding board file for the “old way” using a hard-coded hardware description, just in case you cannot or do not want to use flattened device trees. Nonetheless, regarding the BeagleBoard xM this application note focuses on using the device tree approach only, thus all other possibilities have not been tested and will not be described or supported in this document. In case this patch is being skipped, the driver has to be loaded manually after every system boot. For information on how to load the driver manually, see section [6.7](#).*

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

```
$ make ARCH=arm CROSS_COMPILE=arm-linux-gnueabi- olddefconfig  
$ make ARCH=arm CROSS_COMPILE=arm-linux-gnueabi- menuconfig
```

*Note: “arm” is the processor architecture of the BeagleBoard xM; “arm-linux-gnueabi-” is the tool chain for cross-compiling the kernel. In case of problems with menuconfig see section [6.1.6](#) for troubleshooting.*

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 1.2 Interface (I2C - Infineon) [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 and generate the **zImage**<sup>1</sup> kernel image file:

```
$ make ARCH=arm CROSS_COMPILE=arm-linux-gnueabi- -jX zImage
```

*Note: **X** 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.*

After that compile the modules:

```
$ make ARCH=arm CROSS_COMPILE=arm-linux-gnueabi- -jX modules
```

---

<sup>1</sup> zImage = A compressed version of the Linux kernel image.

## Software Setup

You then have to install the modules of the new kernel on the microSD-Card; therefore insert the microSD-Card into the card reader of **the developer PC** and type the following command (it is a single command, not two separate commands!):

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

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

*Note: Depending on how the system mounts microSD-Cards this path can be different.*

**Attention: Notice the last line of the output which should be something like “DEPMOD 3.14.1+”. The string “3.14.1+” is the version of the new kernel, which is required later every time this document refers to {kernel\_version}.**

After that compile the device tree files with the following command:

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

Copy the new kernel image and the device tree files from **arch/arm/boot/** to the **boot** folder in the root file system of the microSD-Card:

```
# cp arch/arm/boot/zImage /media/[username]/rootfs/boot/vmlinuz-{kernel_version}  
# mkdir -p /media/[username]/rootfs/boot/dtbs/{kernel_version}  
# find arch/arm/boot/ -iname "*.dtb" -exec cp -v '{}' '  
/media/[username]/rootfs/boot/dtbs/{kernel_version}/ \;
```

Now, open the file **/media/[username]/rootfs/boot/uEnv.txt** with an editor such as nano or vim with the following command:

```
# nano /media/[username]/rootfs/boot/uEnv.txt
```

In this file you have to change the kernel name (marked gray) in the line

```
uname_r=[installed_kernel_version]
```

to {kernel\_version}, so that the line contains for example

```
uname_r=3.14.1+
```

in case “{kernel\_version}” is “3.14.1+”.

*Note: If you are planning to use minicom for interacting with the BeagleBoard xM (see section 6.6), then also insert console tty02,115200n8 into a new line at the beginning of the just opened uEnv.txt.*

Then copy the **tpmstartup.py** script shipped with this document (this script will be described in section 4.1) to the **/media/[username on developer PC]/rootfs/home/[username on BeagleBoard xM]** folder:

```
$ cp ~/TpmAppNote/Src/tpmstartup.py /media/[username]/rootfs/home/debian/
```

Now run **sync** to finish writing to the microSD-Card (there is no screen output, so wait for the command to finish):

```
$ sync
```

**Attention: The sync command is a separate command which commits any write buffer cache content 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. Just wait until a new line appears on the command prompt before you eject the SD-Card.**

Unmount the microSD-Card partitions, insert the microSD-Card into the BeagleBoard xM and press the reset button. Now the new Linux kernel should boot and you can continue with section 3.3.

### 3.2.2 Raspberry Pi®

The following sections describe how to install and configure Linux on a Raspberry Pi®. For installation and configuration of Linux on a BeagleBoard xM, go back to section [3.2.1](#).

#### 3.2.2.1 Installation of Linux

The first step is to download an image of Raspbian from [9] and unpack the downloaded archive **on the developer PC**. To do so, switch to your workspace folder:

```
$ cd ~/TpmAppNote
```

Now download and extract the OS image:

```
$ wget http://downloads.raspberrypi.org/raspbian/images/raspbian-2014-06-22/2014-06-20-wheezy-raspbian.zip
$ unzip 2014-06-20-wheezy-raspbian.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 be found with the command `fdisk -l` (you will be informed by the script if packages are missing). The device is automatically being partitioned, formatted and the basic packages are being installed.

**Attention: 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 drive represented by the given device to be completely overwritten!**

```
# ddrescue 2014-06-20-wheezy-raspbian.img /dev/sdX --force
$ sync
```

**Attention: The `sync` command is a separate command which commits any write buffer cache content 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. Just wait until a new line appears on the command prompt before you eject the SD-Card.**

Once completed, connect the network and insert the SD-Card into the Raspberry Pi®, connect a USB keyboard and a monitor (see section [6.6](#) for different ways how to interact with the Raspberry Pi®), and plug in the power cable. Now the system should start.

After booting the kernel, a setup menu will be shown (in case you are connected via SSH you can manually start the setup menu by executing `# raspi-config` **on the embedded platform**). It is required to execute / configure the following options:

- Update (you can find this option in the *Advanced Options* menu; see section [6.1.7](#) in case of internet connection problems)
- Expand Filesystem
- I2C (you can find this option in the *Advanced Options* menu)

It is also recommended to additionally execute / configure at least the following options accordingly:

- Internationalisation Options
  - Change Locale
  - Change Timezone
  - Change Keyboard Layout
- SSH (you can find this option in the *Advanced Options* menu)

*Note: In case you wish to reconfigure these settings later, you can reopen the setup menu again any time by executing `# raspi-config` from the command line **on the embedded platform**.*

Once completed you can exit the menu and logon with the default Raspbian credentials, which are:

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

**Software Setup**

You should now update the installed system to get the latest fixes and security patches. To do so, perform the following steps **on the embedded platform** (see section 6.1.7 in case of internet connection problems):

```
# apt-get update
# apt-get upgrade
```

Confirm possibly upcoming questions about new packages to be installed with <Y> and <Enter> and let the upgrade process finish its work.

In the meantime you can continue with the next step, which is to compile a new kernel for the Raspberry Pi®.

**3.2.2.2 Patching, Configuring and Compiling a Kernel**

Some older kernel versions may not support a TPM and in particular not a TPM with an I<sup>2</sup>C interface such as the Infineon SLB 9645. (Details on support for the Infineon TPM with an I<sup>2</sup>C interface can be found in section 6.2.) And even the latest kernel versions may need to be properly patched, configured and recompiled for activating and automatically loading the Infineon SLB 9645 device driver. This section shows how to do so for kernel 3.14 running on the Raspberry Pi®.

*Note: Patch files are using relative paths to modify the original files. Thus, you have to be in the correct folder for them to work, otherwise they will not find the files to modify. To ensure this, please follow each of the described steps exactly.*

To do so, perform the following steps **on the developer PC**, because the compile process takes more than a few hours on the Raspberry Pi®.

First of all, you need the Raspberry Pi® Linux kernel base repository which can be downloaded via git from [10]:

```
$ git clone git://github.com/raspberrypi/linux.git
```

*Note: Behind a proxy you might need to use the slower HTTPS protocol instead of the faster GIT protocol. To do so, replace "git://..." in the URL above with "https://..." and check the documentation of git for the configuration of a proxy.*

Switch to the folder **linux** generated by the previous command and get the files for kernel version 3.14 with the following two commands:

```
$ cd linux
$ git checkout rpi-3.14.y
```

In the next step the original kernel configuration of the Raspbian kernel installed on the Raspberry Pi® is used as a basis for the configuration of the modified kernel 3.14. For that copy the **/proc/config.gz** file via SSH from the Raspberry Pi® to the developer PC. To do so, start the Raspberry Pi® and type the following command **on the developer PC**:

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

*Note: In case "raspberrypi" cannot be resolved, use the IP address of the Raspberry Pi® instead. You can get it with running **ifconfig** on the Raspberry Pi®.*

The current path should be **~/TpmAppNote/linux/**. Then unzip the file **config.gz** and store it with the name **.config** into the **linux** folder created when cloning the kernel 3.14 with git as done a few steps before:

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

Then copy and apply the patch file named **Raspberry\_Pi\_Automatic\_TPM\_Driver\_Load.patch**, which is shipped with this document. This patch loads the Infineon I<sup>2</sup>C TPM driver automatically on system start.

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

*Note: In case this patch is being skipped, the driver has to be loaded manually after every system boot. For information on how to load the driver manually, see section 6.7.*

**Software Setup**

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

```
$ make ARCH=arm CROSS_COMPILE=arm-linux-gnueabi- olddefconfig
$ make ARCH=arm CROSS_COMPILE=arm-linux-gnueabi- menuconfig
```

*Note: “arm” is the processor architecture of the Raspberry Pi®; “arm-linux-gnueabi-” is the tool chain for cross-compiling the kernel. In case of problems with menuconfig see section 6.1.6 for troubleshooting.*

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 1.2 Interface (I2C - Infineon) [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 and generate the **zImage**<sup>1</sup> kernel image file:

```
$ make ARCH=arm CROSS_COMPILE=arm-linux-gnueabi- -jX zImage
```

*Note: X 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.*

After that compile the modules:

```
$ make ARCH=arm CROSS_COMPILE=arm-linux-gnueabi- -jX modules
```

You then have to install the modules of the new kernel on the SD-Card; therefore insert the SD-Card into the card reader of **the developer PC** and type the following command (it is a single command, not two separate commands!):

```
# make ARCH=arm CROSS_COMPILE=arm-linux-gnueabi- 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.

*Note: Depending on how the system mounts SD-Cards this path can be different.*

To create a Raspberry Pi® aware kernel, switch to your workspace folder and download the Raspberry Pi® kernel tools via git from [11]:

```
$ cd ~/TpmAppNote
$ git clone git://github.com/raspberrypi/tools.git
```

Copy all files located in the **tools/mkimage** subfolder to the **linux/arch/arm/boot** folder, switch to the folder **linux/arch/arm/boot** and create the kernel image with the following commands:

```
$ cp tools/mkimage/* linux/arch/arm/boot/
$ cd linux/arch/arm/boot/
$ python imagetool-uncompressed.py zImage
```

The python script creates a file named **kernel.img**.

Copy this image to the boot partition of the SD-Card:

```
$ cp kernel.img /media/[username]/boot/
```

**/media/[username]/boot** is the path to the boot partition on the SD-Card.

*Note: Depending on how the system mounts SD-Cards this path can be different.*

---

<sup>1</sup> zImage = A compressed version of the Linux kernel image.

### Software Setup

Then copy the **tpmstartup.py** script shipped with this document (this script will be described in section 4.1) to the **/media/[username]/[rootfs-ID]/home/pi** folder:

```
$ cp ~/TpmAppNote/Src/tpmstartup.py /media/[username]/[rootfs-ID]/home/pi/
```

Now run `sync` 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® and reset it. Now the new Linux kernel should boot and you can continue with section 3.3.

*Note: If the Raspberry Pi® does not boot correctly, try updating the Raspberry Pi® firmware. This may be necessary since older firmware versions have problems with newer kernel images. To do so, follow the introductions on [12].*



### 3.3 Software Packages and Configuration

This section describes further steps to install and configure software on the Raspberry Pi® and Beagle Board xM 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 embedded platform, login and execute following command on the embedded platform:

```
$ uname -a
```

The output should contain 3.14 as kernel version and the actual build date of the modified kernel, such as the following one for the BeagleBoard xM

```
Linux debian 3.14.1+ #1 SMP [build date and time] armv7l GNU/Linux
```

or the following one for the Raspberry Pi®

```
Linux raspberry pi 3.14.6+ #1 PREEMPT [build date and time] armv6l 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 Infineon Iridium SLB 9645 TPM I2C Board to your embedded platform, boot the platform, login and view the contents of the **/dev** folder and verify that the device file **tpm0** exists on the embedded platform with the following commands:

*Note: In case you did not apply the patch for the automatic driver load, you have to perform a manual load of the driver as described in section 6.7.*

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

The output should look like this:

```
/dev/tpm0
```

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

```
$ dmesg | grep tpm
```

The output should look like this:

```
[some tick value] tpm_i2c_infineon 1-0020: 1.2 TPM (device-id 0x1A)
[some tick value] tpm_i2c_infineon 1-0020: Issuing TPM_STARTUP
[some tick value] tpm_i2c_infineon 1-0020: TPM is disabled/deactivated (0x7)
```

**Attention: The content and existence of the last line in the output above depends on the actual TPM state. But in case the first two lines are also not shown, the driver is not loaded. For troubleshooting on this topic, see section 6.1.2.**

#### 3.3.2 Correcting the Time Zone and System Time

The system time zone and clock should be set to the correct current time, because it will be used for the validation of files during package installation, and certificates that are used for authentication and verification in the OpenSSL or GnuTLS implementations. The current time of the system can be checked with the command



**Software Setup**

```
$ date
```

Usually the time is set correctly if the time zone settings are correct and the system has access to the internet in order to retrieve the current time. If the time is not correct, you first have to set the time zone via a wizard started by the following command:

```
# dpkg-reconfigure tzdata
```

After that you can set the system clock to a specific time by executing the command

```
$ date -s "06/25/2013 14:28:00"
```

for setting the date and time manually to 06/25/2013, 2:28 PM.

*Note: An alternative for the manual setting of the system time, the Network Time Protocol (NTP) can be used to automatically synchronize the system time with an NTP server. More information about this method can be found in the Internet.*

### 3.3.3 Additional Packages for Embedded Platforms

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 application note. Depending on the used Linux distribution, the list might include some packages which are already installed:

- trousers
- tpm-tools
- openssl
- libcurl4-openssl-dev
- python
- libtspi-dev
- build-essential

Additional required packages for GnuTLS:

- m4
- libtasn1-3-dev
- libffi-dev
- libtool
- autoconf
- gettext
- lzip

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 the command terminal **on the embedded platform**:

```
# apt-get install trousers tpm-tools openssl libcurl4-openssl-dev python libtspi-dev build-essential m4 libtasn1-3-dev libffi-dev libtool autoconf gettext lzip
```

#### 3.3.3.1 Additional Steps for OpenSSL

In order to be able to use the TPM with OpenSSL, the *OpenSSL TPM Engine* library is also required. It provides the TPM cryptographic engine for OpenSSL. To get it, download the *OpenSSL TPM Engine* sources at [13] and extract them with the following commands **on the embedded platform**:

**Software Setup**

```
$ cd ~
$ wget
http://sourceforge.net/projects/trousers/files/OpenSSL%20TPM%20Engine/0.4.2/openssl_tpm_engine-0.4.2.tar.gz
$ tar -xvaf openssl_tpm_engine-0.4.2.tar.gz
```

Switch to the folder **openssl\_tpm\_engine-0.4.2** generated by the previous command and open the file **create\_tpm\_key.c** with an editor such as nano or vim with the following commands:

```
$ cd openssl_tpm_engine-0.4.2
$ nano create_tpm_key.c
```

Now change the lines 148 and 149 from (differences shown in **gray** marking)

```
148 char      *filename, c, *openssl_key = NULL;
149 int        option_index, auth = 0, popup = 0, wrap = 0;

to

148 char      *filename, *openssl_key = NULL;
149 int        option_index, c, auth = 0, popup = 0, wrap = 0;
```

Then configure the package:

```
$ ./configure --prefix=/usr/
```

After that, open the **Makefile** with an editor such as nano or vim with the following command:

```
$ nano Makefile
```

Now add **-lcrypto** to the line starting with **create\_tpm\_key\_LDADD** (if not already there), so that it looks as follows (addition shown in **gray** marking):

```
303 create_tpm_key_LDADD = -ltspi -lcrypto
```

Finally, compile and install the package with the following commands:

```
$ make
# make install
```

### 3.3.3.2 Additional Steps for GnuTLS

First, switch to the folder where you want to temporarily store the files downloaded in this chapter, for example your home directory:

```
$ cd ~
```

Now download the following packages in the exact given version number with the commands described below:

- **automake 1.14** [14]:  
\$ wget ftp://ftp.gnu.org/gnu/automake/automake-1.14.tar.xz
- **gmp 5.1.3** [15]:  
\$ wget ftp://ftp.gnu.org/gnu/gmp/gmp-5.1.3.tar.xz
- **nettle 2.7.1** [16]:  
\$ wget ftp://ftp.gnu.org/gnu/nettle/nettle-2.7.1.tar.gz
- **p11-kit 0.21.1** [17]:  
\$ git clone git://anongit.freedesktop.org/git/p11-glue/p11-kit.git
- **gnutls 3.2.9** [18]:  
\$ wget ftp://ftp.gnutls.org/gcrypt/gnutls/v3.2/gnutls-3.2.9.tar.lz

**Software Setup**

After that you have to compile and install the downloaded sources with the following commands:

**automake**

```
$ cd ~
$ tar -xvaf automake-1.14.tar.xz
$ cd automake-1.14
$ ./configure --prefix=/usr/
$ make
# make install
```

**gmp**

```
$ cd ~
$ tar -xvaf gmp-5.1.3.tar.xz
$ cd gmp-5.1.3
$ ./configure --prefix=/usr/
$ make
# make install
```

**nettle**

```
$ cd ~
$ tar -xvaf nettle-2.7.1.tar.gz
$ cd nettle-2.7.1
$ ./configure --prefix=/usr/
$ make
# make install
$ export LD_LIBRARY_PATH=/usr/lib:$LD_LIBRARY_PATH
```

**Attention:** In order to make the export permanent, you have to edit the `~/.bashrc` file and add the last line from above to the last line of the file.

**p11-kit**

```
$ cd ~/p11-kit
$ git checkout 0.21.1
$ ./autogen.sh
# make install
$ export LD_LIBRARY_PATH=/usr/local/lib:$LD_LIBRARY_PATH
```

**Attention:** In order to make the export permanent, you have to edit the `~/.bashrc` file and add the last line from above to the last line of the file.

**gnutls**

```
$ cd ~
$ tar -xvaf gnutls-3.2.9.tar.lz
$ cd gnutls-3.2.9
$ ./configure --prefix=/usr/
```

After configuring GnuTLS you get a list with supported tools:

```
version:          3.2.9 shared 57:1:29
Host/Target system: armv7l-unknown-linux-gnueabi
Build system:      armv7l-unknown-linux-gnueabi
Install prefix:    /usr
Compiler:          gcc
CFlags:            -g -O2
Library types:     Shared=yes, Static=yes
```

configure: External hardware support:

```
/dev/crypto:          no
Hardware accel:       none
PKCS#11 support:    yes
TPM support:       yes
```

configure: Optional features:

(note that included applications might not compile properly  
if features are disabled)

```
DTLS-SRTP support:   yes
ALPN support:        yes
OCSP support:        yes
OpenPGP support:     yes
SRP support:         yes
PSK support:         yes
DHE support:         yes
ECDHE support:       yes
Anon auth support:   yes
Heartbeat support:   yes
RSA-EXPORT compat:   yes
Unicode support:     yes
Non-SuiteB curves:   yes
```

configure: Optional applications:

```
crywrap app:         yes
local libopts:       yes
local libtasn1:      yes
```

configure: Optional libraries:

```
Guile wrappers:      no
C++ library:         yes
DANE library:        no
OpenSSL compat:      yes
```

configure: System files:

```
Trust store pkcs11:
Trust store file:    /etc/ssl/certs/ca-certificates.crt
Blacklist file:
CRL file:
DNSSEC root key file: /etc/unbound/root.key
```

*Note: Depending on the platform configuration, a warning that the /etc/unbound/root.key file is missing might be displayed after the output shown above. For the examples in this document this warning can be ignored, but in case verification of DNSSEC responses is required, it is recommended to follow the steps displayed in the warning.*

In the output as shown above check if the **TPM** and **PKCS#11** support are enabled. If not, make sure you have installed all packages as described previously in this section.

If yes, install GnuTLS with the following commands:

```
$ make
# make install
```

## 4 Using the TPM

The following sections describe how to start and use the TPM. Be sure that you have successfully completed all setup steps as described in the previous sections.

*Note: All of the steps in this section have to be done on the embedded platform (unless described otherwise). There are several ways to interact with the embedded platform, see section 6.6 for more information.*

### 4.1 First Time Initialization of the TPM

On first use, the TPM must be enabled and activated. These actions have to be done only once since these state changes will be persistent even after a system reboot.

In order to do so, perform the steps described in the following sections.

#### 4.1.1 Stopping the TCSD

First, stop the *TCS Daemon* (TCSD; it is part of the package *TrouSerS* and will be explained later in section 4.2) with the following command<sup>1</sup>:

```
# pkill -x tcscd
```

#### 4.1.2 Initializing the TPM

Then execute the **tpmstartup.py** python script shipped with this document in Admin-mode in order to start the TPM, assert Physical Presence<sup>2</sup>, and enable and activate the TPM:

```
$ cd ~  
# python tpmstartup.py -a
```

*Note: The python script will not work in case the TCSD is running. More information about the script and TPM states can be found in section 6.8*

#### 4.1.3 Resetting the TPM

Now reset the TPM via the reset button on the Infineon Iridium SLB 9645 TPM I2C Board (see section 2.3 for the reset button location) in order to complete activating of the TPM.

*Note: You can also do a complete system shutdown and reset (also known as “cold reboot”) instead of a simple TPM reset, but this is not necessary. Also be aware, that you need a real reset, that is the whole system must be disconnected from power. A simple system reboot (also known as “warm reboot”) is not sufficient since on some platforms it will NOT reset the TPM but only the OS.*

---

<sup>1</sup> **Attention: Remember to always execute commands starting with “#” as root (see section 6.5 on how to do so).**

<sup>2</sup> *Physical Presence* is a TPM authorization concept allowing to limit commands to be issued only by a user physically present next to the platform. The exact specification how to implement this feature is platform-specific. On a normal PC for example, a TPM-aware BIOS always locks *Physical Presence* before it hands execution control over to the OS boot loader; thus *Physical Presence* authorization can only be performed from within the BIOS.

#### 4.1.4 Locking the TPM

After that, you need to execute the python script again, but this time in User-mode in order to start the TPM and lock Physical Presence:

```
# python tpmstartup.py
```

**Attention:** *This script is important in order to avoid any TPM-state-related problems or security vulnerabilities. So from now on, run the TPM startup script in User-mode after every TPM reset or system reboot (unless of course you plan to execute commands requiring physical presence; in that case you have to execute the script in Admin-mode). In case of a system reboot, do so only for those reboots that are causing a TPM reset (cold reboots), otherwise the script might fail! In case of “only” a manual TPM reset (that is an explicit TPM reset without a system reboot) you also have to perform all the steps in the sections [4.1.1](#), this section and section [4.2.2](#) again! Please be also aware, that locking Physical Presence is persistent until the next TPM reset. Asserting Physical Presence (and thus running the script in Admin-mode) is only possible in case Physical Presence is not locked. For troubleshooting on these topics, also see sections [6.1.3](#) and [6.1.4](#).*

## 4.2 Using the TPM with TrouSerS

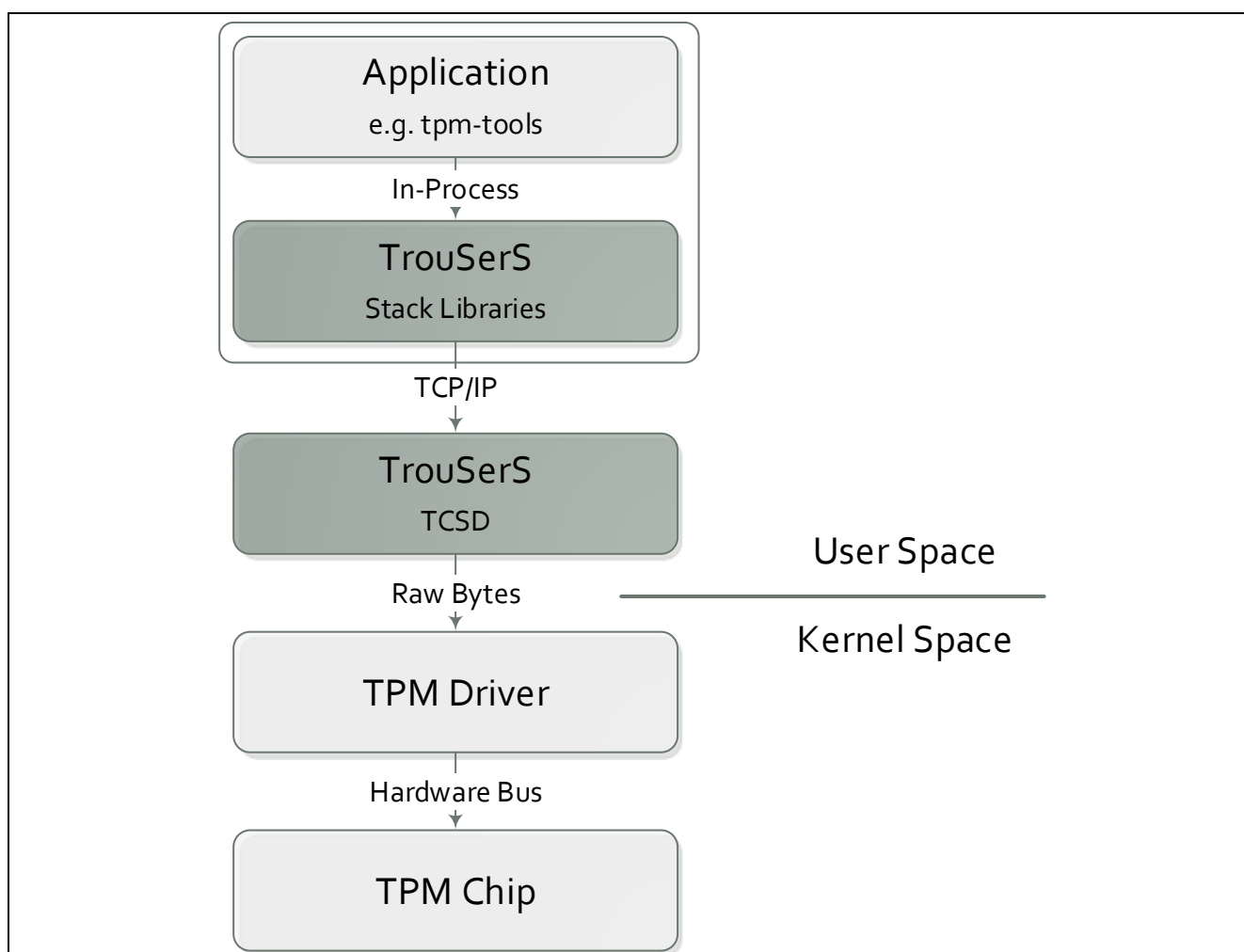
This section describes how to use the TPM on the embedded platform with *TrouSerS*.

### 4.2.1 Introduction

*TrouSerS* is a Trusted Computing Software Stack (TSS) [19] and basically consists of the *TCSD* (TCS Daemon) and the actual *TrouSerS* stack libraries as shown in **Figure 7**. *TCSD* 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. *TrouSerS* 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 a 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.

*tpm-tools* is a set of executables, which can be used to issue TPM commands from the user shell. *tpm-tools* calls the corresponding *TrouSerS* function(s), which sends *TCSD*-understandable commands to the *TCSD*. *TCSD* converts these commands to raw bytes and sends them to the TPM driver.



**Figure 7** TPM-related software

## Using the TPM

With a running *TrouSerS* daemon and *tpm-tools* various TPM actions can be performed. Here are some examples of commands which can be executed always independently of the state of the TPM:

```
$ /usr/sbin/tpm_version  
$ /usr/sbin/tpm_selftest
```

*Note: If a TrouSerS command fails, please see troubleshooting section 6.1.4. The provisioning of a TPM with the command tpm\_takeownership will be shown in section 4.3 of this application note. For general information about TrouSerS and its commands see the TrouSerS website [20].*

## 4.2.2 Starting TrouSerS

In order for *TrouSerS* to work, the TCSD must be started. Normally, this is being done on system boot, but in case you stopped it in section 4.1 you need to restart it manually.

In order to get an output on the console if TCSD could be started or not, you should start it in the foreground and then put it in the background:

```
# tcscd -f
```

Press <Ctrl+Z> to stop it, then type and execute the following command to restart it in the background:

```
# bg1
```

*Note: When TCSD is running, the python script (see section 4.1) will not work. Thus, in case you need to run the startup script again you first have to stop the TCSD by executing `kill -x tcscd`.*

**Attention: In case of a manual TPM reset (that is an explicit TPM reset without a system reboot) you have to perform all the steps in the sections 4.1.1, 0 and this section again! For troubleshooting, also see section 6.1.4.**

## 4.2.3 Testing TrouSerS

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

```
$ /usr/sbin/tpm_version
```

*Note: Depending on how your user environment variables are set, it might be sufficient to just call `tpm_version`.*

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

```
TPM 1.2 Version Info:  
Chip Version:      1.2.133.32  
Spec Level:       2  
Errata Revision:   3  
TPM Vendor ID:     IFX  
Vendor Specific data: 85200050 0074706d 3438ffff ff  
TPM Version:       01010000  
Manufacturer Info: 49465800
```

*Note: If the command fails, please see troubleshooting section 6.1.4. For general information about TrouSerS and its commands see the TrouSerS website [20].*

---

<sup>1</sup> In case you are logged in as normal user and you are using `sudo` in order to run commands as root, then you must not use `sudo` for the `bg` command, since it will not work. Instead, simply run `$ bg` directly afterwards as normal user.



### 4.3 Taking TPM Ownership

Before the TPM can be fully used, a TPM Owner has to be set. Setting an owner password is optional, but in a real world scenario it is highly recommended since it protects most of the administrative TPM commands from unauthorized access.

For every TPM command in this section and the next sections, an enabled and activated TPM is required (see section 4.1 for more information).

Now you can take the TPM ownership with the following command sequence:

```
$ /usr/sbin/tpm_takeownership
```

*Note: Depending on how your user environment variables are set, it might be sufficient to just call `tpm_takeownership`.*

Now you should be asked for two passwords:

Enter owner password: *[Enter here a password, for example owner123]*

Confirm password: *[Repeat owner password]*

Enter SRK password: *[Enter here a password, for example srk456]*

Confirm password: *[Repeat SRK password]*

**Attention: The SRK<sup>1</sup> password must not be empty! It is needed in the following sections.**

*Note: Besides the security aspect to set important passwords, it is still recommended to always set both passwords since some applications using TPM functions are not able to work correctly if these passwords are empty.*

In case the operation fails see section 6.1.4 for troubleshooting. If it is successful, the output might be empty or the following one:

```
TCSO TCS Unloading a public key of size 0!
```

After this operation the TPM is initialized together with the *TrouSerS* stack. *TrouSerS* internally stores specific initialization values of the TPM and manages the keys used in the following sections.

You can test if taking ownership has worked by performing the following commands, where you will be asked for the owner password for the commands to complete:

```
$ /usr/sbin/tpm_setenable -s
```

Enter owner password: *[Enter your owner password, in our example it is owner123]*

This command uses owner authorization to obtain the current TPM enable/disable state. The output should be the following one:

```
Disabled status: false
```

*Note: In case of an error make sure that you have entered the password correctly and that the TPM is enabled.*

---

<sup>1</sup> SRK = Storage Root Key. This key is random and the parent key for all stored keys and will be generated newly on each TPM\_TakeOwnership.

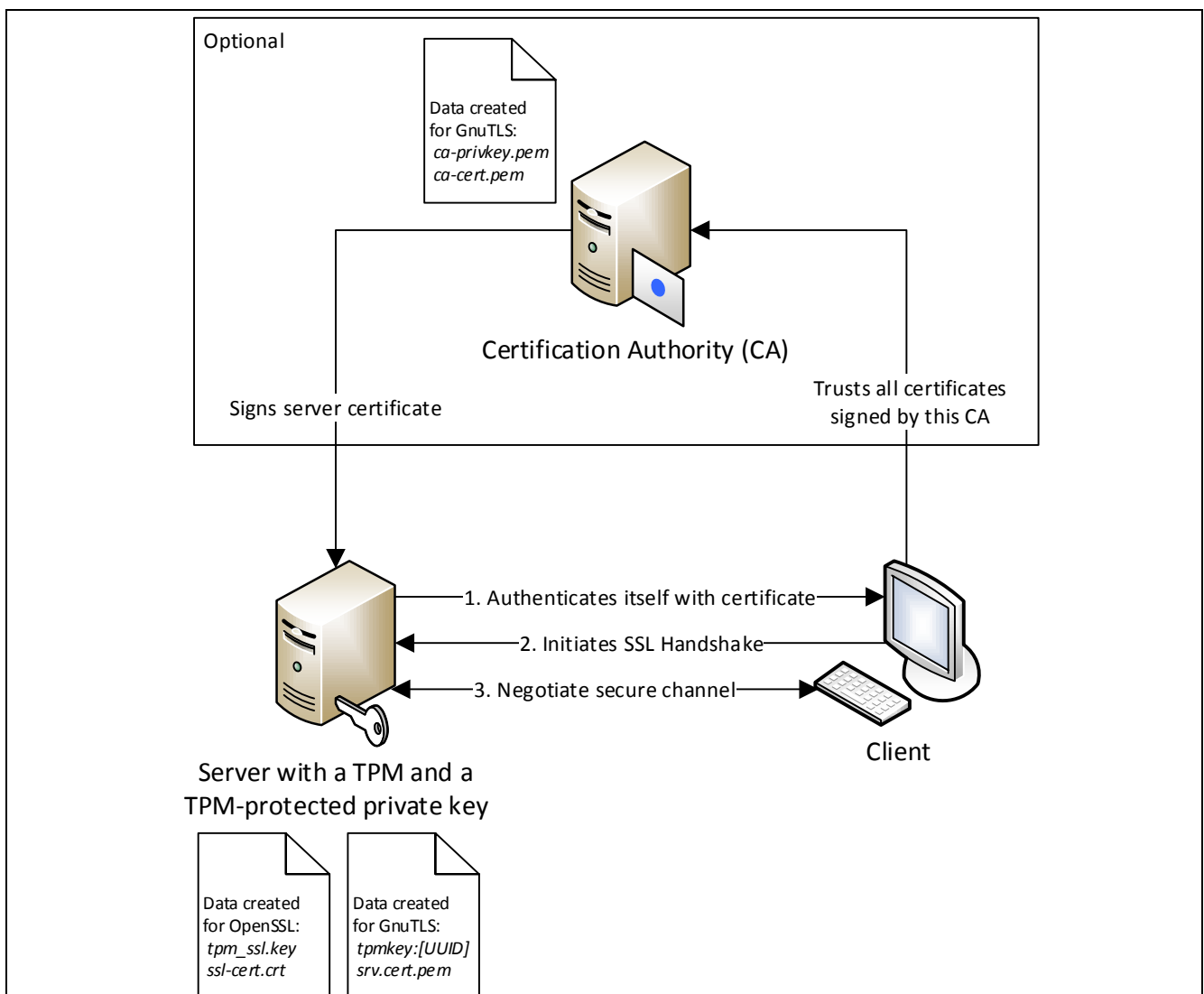
## 4.4 Establishing a Secure Channel Using a TPM

In this document, a *Secure Channel* refers to an SSL/TLS-secured communication between two communication partners. This chapter briefly explains secure channels and will demonstrate the use of OpenSSL (section 4.4.2) or GnuTLS (section 0) for creating a TPM-protected TLS certificate. The certificate will then be used for authentication and to establish a TLS channel between a server and a client.

### 4.4.1 Introduction

SSL/TLS is used in almost every secured channel between a server and a client in the internet. It basically provides three authentication use cases: one where the server authenticates itself against the client, another one where the client authenticates itself against the server, and one where both authenticate themselves against each other.

The most common use case is the first one. A simplified view of this *server authentication* use case can be seen in **Figure 8** (including the conceptual<sup>1</sup> location of the sensitive data created in the examples in the following sections):



**Figure 8** Simplified overview of Server Authentication with SSL/TLS

<sup>1</sup> The actual location of all data created in the examples is the embedded platform, since the examples do not use different machines.

### Using the TPM

Here, the server owns a certificate which is being used to prove the server's authenticity against a client. Additionally, the server certificate is normally signed by a *Certification Authority* (CA), so a client can validate a server's identity without knowing the server's certificate in advance.

If now the server's private key is not protected well-enough and could be obtained by an attacker<sup>1</sup>, the attacker would be able to authenticate himself against the client as the valid server, since the rest of the server's certificate is public by design. Thus, a server's private key should enjoy the highest protection possible.

This is where the TPM comes in: It is possible to use a TPM-protected certificate, where the private key is being created and protected by the TPM via encryption. This key can only be decrypted (and thus used) inside the TPM, so it is always protected: either via encryption outside the TPM or via hardware-protection inside the TPM.

*Note: In case a non-migratable key is being used, this automatically helps in verifying the server authenticity since only this server with the very same physical TPM can have the corresponding private key.*

## 4.4.2 OpenSSL

In this section an *OpenSSL* certificate utilizing a TPM will be created and used.

### 4.4.2.1 Creating a TPM-protected Server Certificate

First you have to create a TPM-protected signing key, which will be called **tpm\_ssl.key**. This key will be generated by the TPM, encrypted and then the encrypted key will be exported to the given file on the hard disk. You will need the SRK password in the next steps.

*Note: You will also be asked to provide a key usage password. This is optional, but in case you provide one, you will need it later, too.*

```
$ create_tpm_key -s 2048 -a tpm_ssl.key  
SRK Password: [Enter your SRK password, in our example it is srk456]
```

*Note: You can get a description of all command line parameters by executing `create_tpm_key --help`.*

Now create a certificate with the **tpm\_ssl.key** file. The certificate will be saved in **ssl-cert.crt**. The OpenSSL certificate uses the TPM as cryptographic engine and for generating the **tpm\_ssl.key**, which means that the TPM key generated in the previous step is being used as private key for the certificate. The generated certificate is self-signed, based on cryptography standard X.509<sup>2</sup> and will be valid for 365 days (in case you have provided a key usage password in the previous step, you will need it in this step, but here it is called "TPM Key Password"):

```
$ openssl req -keyform engine -engine /usr/lib/openssl/engines/libtpm.so -key  
tpm_ssl.key -new -x509 -days 365 -out ssl-cert.crt
```

*Note: In case the command does not run, the path to **libtpm** may be wrong. To fix this, search for the **libtpm.so** and replace the wrong path in the command above.*

The output should be:

```
engine "tpm" set.  
SRK authorization: [Enter your SRK password, in our example it is srk456]
```

In the upcoming questions you can choose to either enter some data or use the defaults by just pressing <Enter>.

---

<sup>1</sup> The infamous *OpenSSL Heartbleed Bug* found in 2014 was such a bug, where in the worst case an attacker was able to obtain the server's private key from the server's memory. More information can be found in section [6.10](#).

<sup>2</sup> X.509 is (simply spoken) a standard for digital certificates. X.509 certificates contain several optional and mandatory certificate properties, for example the issuer, validity period or the certificate's purposes. For more information about certificates see section [6.9.3](#).

#### 4.4.2.2 Using the Certificate

Now you can test the new certificate. Therefore start an OpenSSL server on the BeagleBoard xM / Raspberry Pi® with the previously generated certificate, the TPM as engine and the key in the **tpm\_ssl.key** file as private key:

```
$ openssl s_server -cert ssl-cert.crt -accept 4433 -keyform engine -engine  
/usr/lib/openssl/engines/libtpm.so -key tpm_ssl.key
```

*Note: In case the command does not run, the path to libtpm may be wrong. To fix this, search for the **libtpm.so** and replace the wrong path in the command above.*

The server output looks like this:

```
engine "tpm" set.  
SRK authorization: [Enter your SRK password, in our example it is srk456]  
Using default temp DH parameters  
Using default temp ECDH parameters  
ACCEPT
```

Now start the client (for simplicity reasons running on the same platform here):

*Note: You should use a second terminal session<sup>1</sup> on the BeagleBoard xM / Raspberry Pi® for running the client. You can also try to run the client on the developer PC. Of course you then have to replace **localhost** in the command below with the host name or IP address of the embedded platform.*

```
$ openssl s_client -connect localhost:4433
```

The client output with a successful validation (sample data):

```
CONNECTED(00000003)  
depth=0 C = AU, ST = Some-State, O = Internet Widgits Pty Ltd  
verify error:num=18:self signed certificate  
verify return:1  
depth=0 C = AU, ST = Some-State, O = Internet Widgits Pty Ltd  
verify error:num=9:certificate is not yet valid  
notBefore=Apr  2 10:05:55 2014 GMT  
verify return:1  
depth=0 C = AU, ST = Some-State, O = Internet Widgits Pty Ltd  
notBefore=Apr  2 10:05:55 2014 GMT  
verify return:1  
---  
Certificate chain  
 0 s:/C=AU/ST=Some-State/O=Internet Widgits Pty Ltd  
  i:/C=AU/ST=Some-State/O=Internet Widgits Pty Ltd  
---  
Server certificate  
-----BEGIN CERTIFICATE-----  
MIIDXTCCAkWgAwIBAgIJAITiOARKezMXMA0GCSqGSIb3DQEBBQUAMEUxCzAJBgNV  
...  
    [Base64-encoded certificate data]  
...  
WA==  
-----END CERTIFICATE-----  
subject=/C=AU/ST=Some-State/O=Internet Widgits Pty Ltd  
issuer=/C=AU/ST=Some-State/O=Internet Widgits Pty Ltd  
---
```

---

<sup>1</sup> It is enough to put OpenSSL server in the background, but in order to test the connection you have to open two separate terminal connections.

## Using the TPM

```

No client certificate CA names sent
---
SSL handshake has read 1546 bytes and written 375 bytes
---
New, TLSv1/SSLv3, Cipher is ECDHE-RSA-AES256-SHA
Server public key is 2048 bit
Secure Renegotiation IS supported
Compression: NONE
Expansion: NONE
SSL-Session:
    Protocol    : TLSv1.1
    Cipher      : ECDHE-RSA-AES256-SHA
    Session-ID: F2261F609C5FB5180AEEF47B54775CF7ED5CD00FAF69342B12913B005F3DF489
    Session-ID-ctx:
    Master-Key:
35D956D93CC2C41E8E3488DFFE9B8237EEB6670EA0510D56FA00D58CF3DAAEAE25BA49E7EEB80FCDFE
3D9BAAF392F389
    Key-Arg     : None
    PSK identity: None
    PSK identity hint: None
    SRP username: None
    TLS session ticket lifetime hint: 300 (seconds)
    TLS session ticket:
0000 - 33 83 e4 e0 ec db d0 38-c7 3c a1 5a 6c 3e d1 9e    3.....8.<.Zl>..
0010 - 27 46 7e 96 a1 3b 70 d1-8e de ae b8 a3 ba b6 e0    'F~...;p.....
0020 - 9d e0 c8 6d c2 63 fc c2-c9 96 fb 58 da ec 98 33    ...m.c.....X...3
0030 - 23 7a 2d 49 bb ee ea dc-ca ea dd f4 fb 6c 25 df    #z-I.....l%.
0040 - 49 7f a9 fd c6 6d 15 8a-82 51 29 87 9f 55 8a 7c    I....m...Q)..U.|
0050 - 85 7e e1 69 a7 0f a0 83-d3 37 b8 1a b2 06 76 0a    .~.i.....7....v.
0060 - e8 b6 80 95 24 e2 5e 5a-fe 82 df 4a 03 07 96 47    ....$.^Z...J...G
0070 - b9 b5 b8 0b f7 ed 92 2e-d7 ac 42 1d 62 c0 ba 89    .....B.b...
0080 - c1 bc 5e 9b 96 e4 1f e0-c9 95 55 83 b3 f0 93 5f    ..^.....U...._
0090 - 83 fe 19 ce c8 04 5a 92-83 e7 69 4e 01 e2 0b c1    .....Z...iN....

    Start Time: 1396433038
    Timeout    : 300 (sec)
    Verify return code: 9 (certificate is not yet valid)
---

```

The certificate validation is done automatically on the client and will fail for example in case the certificate has been expired or has not the correct format or is issued to another host.

You can test the connection by entering "Test Message" in the OpenSSL command prompt on the server (or client) console and pressing <Enter>. Now on the OpenSSL client (or server) console the entered text should appear. You can stop the server and the client by entering Q and pressing <Enter> on the server or client console with the following results: If entered on the server console, it will stop both the client and the server. But if entered on the client console it will only stop the client.

*Note: Using Q on the server console only works in case a client is connected. In case of no client connection you can stop the server by pressing <Ctrl+C>.*

### 4.4.3 GnuTLS

In this section a *GnuTLS* certificate utilizing a TPM will be created and used.

#### 4.4.3.1 Creating a TPM-protected Server Certificate

First, you have to create a TPM-protected key:

```
$ export LD_LIBRARY_PATH=/usr/lib/:$LD_LIBRARY_PATH
$ tpmtool --generate-rsa --bits=2048 --register --user
```

The terminal output looks like this (sample data):

```
tpmkey:uuid=fcf82685-83fb-423b-89c1-20da0d4189a9;storage=user
```

The TPM has generated a user key and the UUID is a link to it. With the command `tpmtool --list` you can see all UUIDs to the TPM-keys.

```
$ tpmtool --list
```

Available keys (example):

```
0: tpmkey:uuid=fcf82685-83fb-423b-89c1-20da0d4189a9;storage=user
1: tpmkey:uuid=00000000-0000-0000-0000-000000000001;storage=system
```

With the newly generated UUID you can create a public key which will be stored in **pubkey.pem**.

```
$ tpmtool --pubkey "tpmkey:uuid=fcf82685-83fb-423b-89c1-20da0d4189a9;storage=user"
--outfile=pubkey.pem
```

To create a valid GnuTLS server certificate, the server certificate must be issued by a Certification Authority (CA). This is important for certificate validation on the client and will be explained later.

An exemplary CA key **ca-privkey.pem** and certificate **ca-cert.pem** can be created by the following two commands:

```
$ certtool --generate-privkey --outfile ca-privkey.pem
$ certtool --generate-self-signed --load-privkey ca-privkey.pem --outfile ca-
cert.pem
```

In the upcoming questions you can choose to either enter some data or use the defaults by just pressing <Enter>. But you **must** enter an expiration time and answer the questions “Does the certificate belong to an authority?” and “Will the certificate be used to sign other certificates?” with ‘y’.

The hostname at “Enter a dnsName of the subject of the certificate” and “Enter a URI of the subject of the certificate” (press <Enter> twice after entering a name) can be chosen freely since it is only a fake CA certificate.

After that, you especially have to answer the last question with ‘y’ in order to get a valid certificate. The important part of the questions and their answers can look like this (input marked **gray**):

```
...
The certificate will expire in (days): 365
```

Extensions.

```
Does the certificate belong to an authority? (y/N): y
Path length constraint (decimal, -1 for no constraint):
Is this a TLS web client certificate? (y/N):
Will the certificate be used for IPsec IKE operations? (y/N):
Is this a TLS web server certificate? (y/N):
Enter a dnsName of the subject of the certificate: [fake CA hostname]
Enter a dnsName of the subject of the certificate:
```

## Using the TPM

```
Enter a URI of the subject of the certificate: [fake CA hostname]
Enter a URI of the subject of the certificate:
Enter the IP address of the subject of the certificate:
Enter the e-mail of the subject of the certificate:
Will the certificate be used to sign other certificates? (y/N): y
...
Is the above information ok? (y/N): y
```

Signing certificate...

Now you have all files to generate the GnuTLS server certificate. This command uses the private key on the TPM and the created public key stored under `pubkey.pem` and generates the file **srv-cert.pem**:

```
$ certtool --generate-certificate --outfile srv-cert.pem --load-privkey
"tpmkey:uuid=fcf82685-83fb-423b-89c1-20da0d4189a9;storage=user" --load-pubkey
pubkey.pem --load-ca-certificate ca-cert.pem --load-ca-privkey ca-privkey.pem
```

First, you have to enter a PIN for the 'SRK' token, which in fact is the SRK password.

In the upcoming questions you can choose to either enter some data or use the defaults by just pressing <Enter>. But you **must** enter an expiration time and answer the question "Is this a TLS web server certificate?" with 'y'.

The hostname at "Enter a `dnsName` of the subject of the certificate" and "Enter a URI of the subject of the certificate" (press <Enter> twice after entering a name) **must** be the host name of your embedded platform.

*Note: You can get the hostname by executing the command `hostname`.*

After that, you especially have to answer the last question with 'y' in order to get a valid certificate. The important part of the questions and their answers can look like this (input marked gray):

```
...
The certificate will expire in (days): 365
```

Extensions.

```
Does the certificate belong to an authority? (y/N):
Is this a TLS web client certificate? (y/N):
Will the certificate be used for IPsec IKE operations? (y/N):
Is this a TLS web server certificate? (y/N): y
Enter a dnsName of the subject of the certificate: [hostname]
Enter a dnsName of the subject of the certificate:
Enter a URI of the subject of the certificate: [hostname]
Enter a URI of the subject of the certificate:
Enter the IP address of the subject of the certificate:
...
Is the above information ok? (y/N): y
```

Signing certificate...

#### 4.4.3.2 Using the Certificate

Now you can test the new certificate. Therefore start a GnuTLS server on the BeagleBoard xM / Raspberry Pi® with the previously generated certificates and the TPM key with the given UUID as private key:

```
$ gnutls-serv --x509keyfile "tpmkey:uuid=fcf82685-83fb-423b-89c1-20da0d4189a9;storage=user" --x509certfile srv-cert.pem -p 4433
```

Now start the client (for simplicity reasons running on the same platform here):

*Note: You must use a separate terminal session<sup>1</sup> on the BeagleBoard xM / Raspberry Pi® to start the client. You can also try to run the client on the developer PC. Of course you then have to install it and replace localhost in the command below with the host name or IP address of the embedded platform.*

```
$ gnutls-cli --x509cafile ca-cert.pem -p 4433 [hostname]
```

The certificate validation is done automatically on the client, but in addition to the previous example with OpenSSL, the built-in certificate validation will also fail in case the CA certificate is not valid or not trusted by the client.

The client output with a successful validation (indicator marked in **gray**) looks like this:

```
Processed 1 CA certificate(s).
Resolving 'debian'...
Connecting to '127.0.1.1:443'...
- Peer's certificate is trusted
- The hostname in the certificate matches 'debian'.
- Successfully sent 0 certificate(s) to server.
- Session ID:
BE:AD:C9:F1:9B:28:B9:F3:4F:94:69:36:37:83:90:D3:F4:48:DD:E1:F3:75:3A:1B:73:2A:98:B
0:2D:8F:D5:4D
- Server has requested a certificate.
- Certificate type: X.509
- Got a certificate list of 1 certificates.
- Certificate[0] info:
  - subject `O=debian', issuer `', RSA key 2048 bits, signed using RSA-SHA256,
  activated `2000-01-01 00:30:18 UTC', expires `2000-12-31 00:30:20 UTC', SHA-1
  fingerprint `85ec56ec4be04f6b24c912179bab4df323d'
    Public Key Id:
      ed8ef4a8872cdabb7b50aa9b3ed4651268f49b78d
    Public key's random art:
      +---[ RSA 2048]-----+
      |
      |      o +
      |      . O +
      |      + E..
      |      .S .
      |      .. .
      |      oo.. o
      |      o++oo* .
      |      +B+=o+
      +-----+
- Ephemeral EC Diffie-Hellman parameters
- Using curve: SECP192R1
- Curve size: 192 bits
- Version: TLS1.2
- Key Exchange: ECDHE-RSA
```

<sup>1</sup> It is not enough to put GnuTLS server in the background, since it requires the SRK password on each start. Thus, you have to open two separate terminal connections.



**Using the TPM**

- Cipher: AES-128-CBC
- MAC: SHA1
- Compression: NULL
- Handshake was completed
- Simple Client Mode:

You can exit the applications by pressing <Ctrl+C>.

## 5 References

*Note: 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#Working\\_.2F\\_Non-working\\_SD\\_cards](http://elinux.org/RPi_SD_cards#Working_.2F_Non-working_SD_cards)
- [3] <http://beagleboard.org/>
- [4] <http://www.raspberrypi.org/>
- [5] <http://www.ubuntu.com/download/desktop>
- [6] <http://packages.ubuntu.com/>
- [7] <https://rcn-ee.net/deb/rootfs/wheezy/debian-7.6-console-armhf-2014-08-13.tar.xz>
- [8] <git://github.com/beagleboard/kernel.git>
- [9] <http://downloads.raspberrypi.org/raspbian/images/raspbian-2014-06-22/2014-06-20-wheezy-raspbian.zip>
- [10] <git://github.com/raspberrypi/linux.git>
- [11] <git://github.com/raspberrypi/tools.git>
- [12] [http://elinux.org/RPi\\_Kernel\\_Compilation#Get\\_the\\_firmware](http://elinux.org/RPi_Kernel_Compilation#Get_the_firmware)
- [13] [http://sourceforge.net/projects/trousers/files/OpenSSL%20TPM%20Engine/0.4.2/openssl\\_tpm\\_engine-0.4.2.tar.gz](http://sourceforge.net/projects/trousers/files/OpenSSL%20TPM%20Engine/0.4.2/openssl_tpm_engine-0.4.2.tar.gz)
- [14] <ftp://ftp.gnu.org/gnu/automake/automake-1.14.tar.xz>
- [15] <ftp://ftp.gnu.org/gnu/gmp/gmp-5.1.3.tar.xz>
- [16] <ftp://ftp.gnu.org/gnu/nettle/nettle-2.7.1.tar.gz>
- [17] <git://anongit.freedesktop.org/git/p11-glue/p11-kit.git>
- [18] <ftp://ftp.gnutls.org/gcrypt/gnutls/v3.2/gnutls-3.2.9.tar.lz>
- [19] [http://www.trustedcomputinggroup.org/resources/tcg\\_software\\_stack\\_tss\\_specification](http://www.trustedcomputinggroup.org/resources/tcg_software_stack_tss_specification)
- [20] <http://trousers.sourceforge.net/man.html>
- [21] <http://www.raspberrypi.org/faqs>
- [22] <http://beagleboard.org/Support/FAQ>
- [23] <https://www.modmypi.com/blog/tutorial-how-to-give-your-raspberry-pi-a-static-ip-address>
- [24] [http://www.trustedcomputinggroup.org/resources/tpm\\_main\\_specification](http://www.trustedcomputinggroup.org/resources/tpm_main_specification)
- [25] [http://www.trustedcomputinggroup.org/resources/pc\\_client\\_work\\_group\\_specific\\_implementation\\_specification\\_for\\_conventional\\_bios\\_specification\\_version\\_12](http://www.trustedcomputinggroup.org/resources/pc_client_work_group_specific_implementation_specification_for_conventional_bios_specification_version_12)

## **6 Appendix**

### **6.1 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:

- Run `dmesg` from the command terminal. By typing for example `dmesg | grep -i tpm` the output can be filtered for messages containing a given string (here the string is “tpm”).
- Take a look into the standard logs in `/var/log/messages` and `/var/log/syslog`
- Check if your kernel has TPM support by running `zcat /proc/config.gz | grep -i tpm`
- Look for the TPM device with `ls /dev/tpm*`  
If nothing is found, then the driver has not been loaded.

#### **6.1.1 Platform Related Problems**

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

Please make sure that:

- Everything is properly connected.
- The power supply provides sufficient power (for example some USB-ports do not provide enough power for the Raspberry Pi® to start).
- The (micro)SD-Card is properly partitioned and formatted.
- The OS is properly installed.
- The `uEnv.txt` file has been modified properly (see the steps after copying the kernel image and device tree files in section [3.2.1.2](#))

For additional information, please consult the general FAQ and / or troubleshooting pages of the Raspberry Pi® [21] and the BeagleBoard xM [22].

#### **6.1.2 Driver Related Problems**

Examples: Loading the driver fails or TPM is not found.

Please make sure that:

- The TPM is properly connected.
- You have the required (root) permissions, that is all driver-related actions and some TPM actions (for example taking ownership) require root permissions. For information on how to obtain root privileges, see section [6.5](#).
- You are using the correct kernel version (should be 3.14) by running `uname -a` or `cat /proc/version`. This may happen, since after a software upgrade by using `apt-get upgrade` it may be possible that a new kernel has been installed. In that case you have to install the self-compiled kernel again. If you still have the wrong version, make sure to successfully complete all steps in section [3.2](#).
- The driver is loaded. You can try to load it manually as described in section [6.7](#) in order to find out if it is just the automatic driver load or if the driver does not run at all. In case it cannot be loaded or automatic load is configured but does not work, make sure to successfully repeat all steps in section [3.2](#).

You can also try to:

- Reboot and run `i2cdetect` as described in section [6.7](#). If everything works as described, the chip is properly connected.
- Look if the I<sup>2</sup>C device is registered at the bus by executing `ls -lah /sys/bus/i2c/devices/i2c-2/2-0020/` (the numbers after “.../devices/i2c-“ may vary and depend on the bus the TPM is attached to).

### 6.1.3 TPM Startup Related Problems

Examples: The TPM startup script returns "Error opening device" or "Device or resource busy" or a command in the script fails.

Please make sure that:

- You have executed the script with root permissions. For information on how to obtain root privileges, see section 6.5.
- The TPM driver is loaded.
- The TCSD is **not** running (you can stop it by calling `# pkill -x tcscd` with root privileges). Otherwise the script will fail.
- Physical Presence is not locked when you want to run the TPM startup script in Admin-mode. In case you are not sure or Physical Presence is locked, you can press the reset button on the TPM board (see section 2.3) and then re-run the TPM startup script with parameter -a.
- You performed a TPM reset every time when this application note or the screen output tells you to do so, that is a simple platform reboot is not enough; instead, either use the reset button on the TPM board (see section 2.3) or do a real shutdown, disconnect the TPM from the power supply, then re-connect it and then restart the platform.
- The TPM is properly connected.

### 6.1.4 TrouSerS or TPM Usage Related Problems

Example: Starting *TrouSerS* or executing a TPM command fails unexpectedly.

At first, try a real shutdown of the system by performing the following steps:

1. `# shutdown -hP now`
2. Disconnect power from embedded platform for a few seconds
3. Reboot embedded platform and logon
4. Perform steps in section 4.1.1, section 0 (or section 4.1.2 in case you need Physical Presence asserted, for example in case of clearing the TPM) and section 4.2.2.
5. Continue with previously failed command

Now retry the command that has failed before. If the error is still there, please make sure that:

- The TPM is started, physical presence is asserted (only if necessary, see section 6.8), the TPM is enabled and activated (all done by startup script; see section 4.1 on how to do this).
- The TCSD has been started properly. In case of errors you should also try to restart it. To do so, you have to kill it with `# pkill -x tcscd` and start it again (see section 4.2.2 on how to do this).
- The TPM is in the required state, for example for a take ownership the TPM must not have an owner.
- The TPM driver is loaded (see section 6.7 on how to do this), that is check that `/dev/tpm0` exists. In case of driver loading problems (for example shown by "Error opening device"), reboot your system and try to load the driver again.
- You performed a real TPM reset when this application note or the screen output tells you to do so, that is a simple platform reboot is not enough; instead, do a real shutdown, disconnect the TPM from the power supply, then re-connect it and then restart the platform. In case you only need a TPM reset and no TPM reboot (for example required for (de)activating the TPM), you can use the reset button on the TPM board (see section 2.3).
- The TPM is not in lifetime lock state (this only happens in case the TPM is explicitly set to be this state by issuing the corresponding TPM command, but this command is not part of this application note).
- The TPM is properly connected.
- You have the required (root) permissions, that is all driver-related actions and some TPM actions (for example taking ownership) require root permissions. For information on how to obtain root privileges, see section 6.5.
- You have entered a password correctly, if one is required. In case you forgot your password, you have to clear the TPM in order to reset it to its factory defaults. Clearing the TPM is described in section 6.1.5.

## Appendix

- Your TPM is not in dictionary attack<sup>1</sup> lockout. You can ensure this by executing the command `/usr/sbin/tpm_resetdalock`. This command resets the dictionary attack counters and requires the owner password which may have been set during taking ownership.

In case you still have problems, try clearing the TPM as described in section 6.1.5.

### 6.1.5 Clearing the TPM

**Attention: Clearing the TPM will result in the fact that all keys and data stored on or protected with the TPM is not available anymore. This also applies to certificates using a TPM-generated key as private key!**

In case your TPM has an owner and you know the owner password, you can clear the TPM with the following command:

```
$ /usr/sbin/tpm_clear
```

Now enter the owner password.

In case you do not know the owner password or your TPM does not have an owner, you have to perform the following steps:

1. Stop the TCSD:  
# `pskill -x tcscd`
2. Reset the TPM via the reset button on the TPM board (see section 2.3).
3. Run the **tpmstartup.py** script in Admin Mode:  
# `python tpmstartup.py -a`
4. Start the TCSD as described in section 4.2.2.
5. Clear the TPM using physical presence authorization:  
\$ `/usr/sbin/tpm_clear -f`

After clearing the TPM you have to do the following steps:

1. **DO NOT** reboot the system or reset the TPM
2. Stop TCSD
3. Run the startup script
4. Reset the TPM (you can do so either via the reset button on the TPM board or by shutting down the system and disconnecting it from the power supply; in the latter you have to load the driver after boot)
5. Run the startup script
6. Start the TCSD

and then continue with the steps in the corresponding action sequence starting from (and including) taking ownership of the TPM (see section 4.3).

### 6.1.6 Kernel Configuration Related Problems

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-gnueabi- menuconfig`. In that case, try to delete the current configuration and reconfigure the kernel with the following commands:

```
# make mrproper
# zcat ~/TpmAppNote/config.gz > .config
$ make ARCH=arm CROSS_COMPILE=arm-linux-gnueabi- olddefconfig
$ make ARCH=arm CROSS_COMPILE=arm-linux-gnueabi- menuconfig
```

Now continue with the kernel compilation steps as described in the corresponding subsection of section 3.2.

---

<sup>1</sup> See section 6.4 for more information of the TPM's Dictionary Attack mechanism.

### 6.1.7 Internet Connection Related Problems

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

1. Since network configuration is being done when the kernel loads, 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` with root privileges 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.

## 6.2 Linux Kernel Support of Infineon SLB 9645

**Attention:** This application note applies to the respective Linux distributions with kernel version 3.14 for both embedded platforms. There is no warranty that the steps in this application note are the same or will even work for other kernel versions.

### 6.2.1 Overview

The basic circumstance in Linux affecting the driver support of specific hardware devices is the kernel version. Since the Infineon SLB 9645 is available since 2013 and thus the open source driver for it was not available before, especially older or customized kernel versions might not have built-in support for it. Thus, for some kernel versions the hardware drivers have to be added by patching the kernel, compiling it and using that kernel on the embedded platform.

On top of that, older kernel versions must receive more patches to support the SLB 9645 than newer ones, which increases the effort and complexity of the patch process. The following table shows the most recent kernel versions, their hardware support state and the effort necessary to patch them for working with the SLB 9645.

*Note: Please be aware, that these are not the standard Linux kernels, but versions specifically adapted for the BeagleBoard xM and the Raspberry Pi®. They can be found on [8] and [10]. The TPM support though is the same as in standard Linux kernels.*

Kernel Version	Embedded Platform(s)	Patch Files Available	Patch Effort
< 3.4	BeagleBoard xM	Yes	High (some driver files must be replaced)
	Raspberry Pi®		
3.4 - 3.6	BeagleBoard xM	yes	Medium (more patches necessary)
	Raspberry Pi®		
3.7	BeagleBoard xM	yes	low
	Raspberry Pi®	Kernel version not available for Raspberry Pi®	
3.8, 3.9	BeagleBoard xM	yes	low
	Raspberry Pi®		
3.10 or higher, including 3.14	BeagleBoard xM	yes	low, only required for automatic driver loading
	Raspberry Pi®		

Kernel 3.6 and older versions need more patching, therefore they are out of scope for this application note.

Kernel 3.7 is not available for the Raspberry Pi® at all.

Beginning with kernel 3.10, the kernel already has built-in support for the SLB 9645, so except for an automated driver load no patching is necessary and the steps in this application note should be almost the same (that is at some places, the command output might look slightly different).

### 6.2.2 Patching Kernel Versions 3.7 to 3.9

This section describes how to patch older kernel versions in order to enable SLB 9645 support. Although it should work in most cases without problems, no support can be given for kernel versions other than kernel 3.14. The patches must be applied before configuring the kernel.

First, download the patch files. This can be done with the following commands:

```
$ wget https://patchwork.kernel.org/patch/2196511/raw/ -O RESEND-char-tpm-Convert-struct-i2c_msg-initialization-to-C99-format.patch
$ wget https://patchwork.kernel.org/patch/2213161/raw/ -O v2-tpm-Add-support-for-new-Infineon-I2C-TPM-SLB-9645-TT-1.2-I2C.patch
```

*Note: The `-O` parameter in the commands above has a capitalized “O”, which has a different meaning than the `-o` parameter with a non-capitalized “o”.*

These files are modifying some of the files in the Linux kernel sources; for that, they are using relative paths. Thus, you have to be in the correct folder for them to work, otherwise they will not find the files to modify.

*Note: The command `patch --dry-run < [patch name]` does not change any files; it just prints if the patch can be installed correctly or not.*

Apply the patches with:

```
$ patch -p1 < RESEND-char-tpm-Convert-struct-i2c_msg-initialization-to-C99-format.patch
$ patch -p1 < v2-tpm-Add-support-for-new-Infineon-I2C-TPM-SLB-9645-TT-1.2-I2C.patch
```

In case of patching errors, revert the changes by repeating the same command (enter “yes” when asked “Assume -R?” and apply them manually via opening the patch file and copying the failed patch lines with a text editor into the corresponding source file line.

After that, continue with configuring and compiling the kernel as described in the corresponding subsection in section 3.2.



### 6.3 Schematic and Layout of the Infineon Iridium SLB 9645 TPM I2C Board

Figure 9 shows the schematic of the Infineon Iridium SLB 9645 TPM I2C Board:

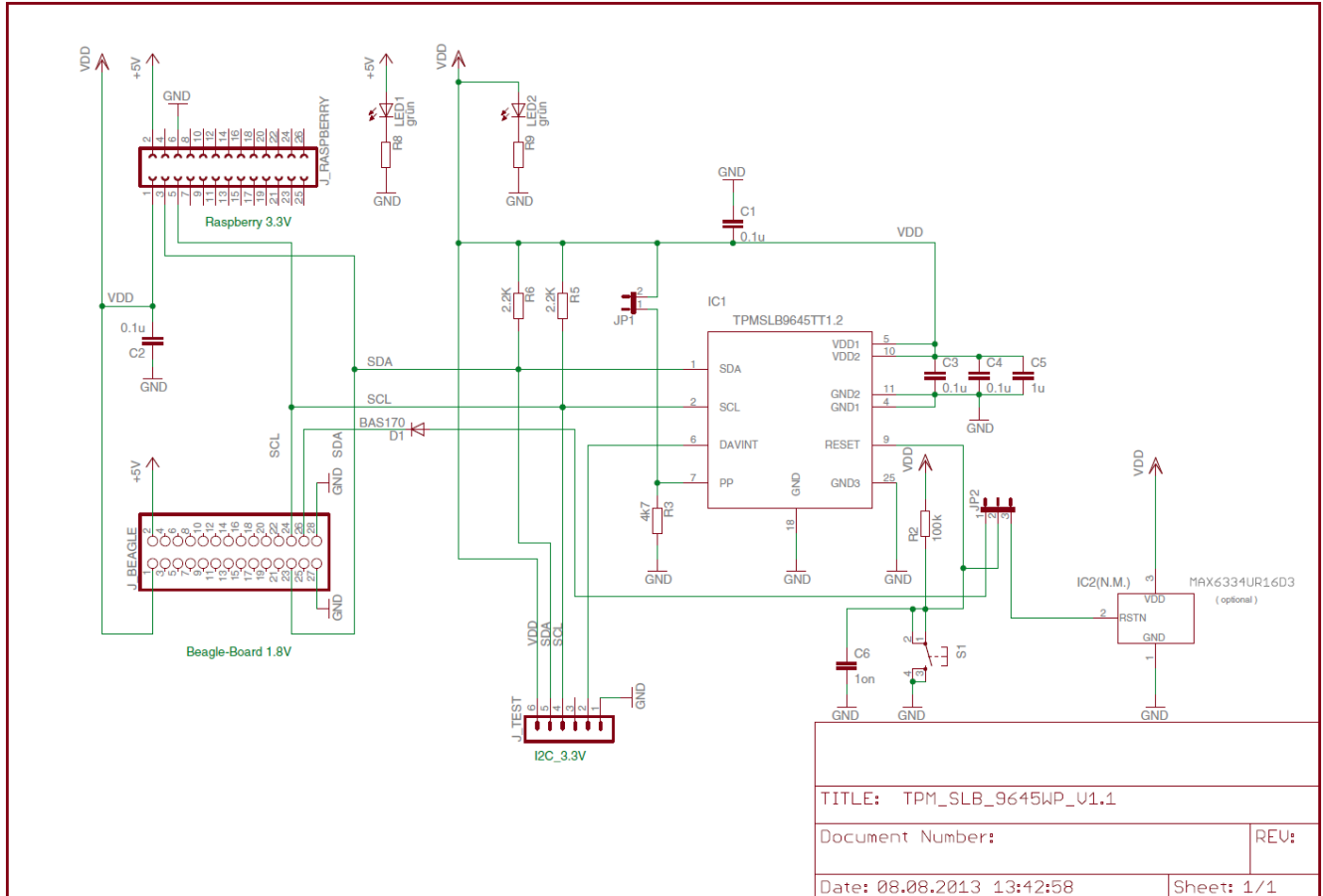
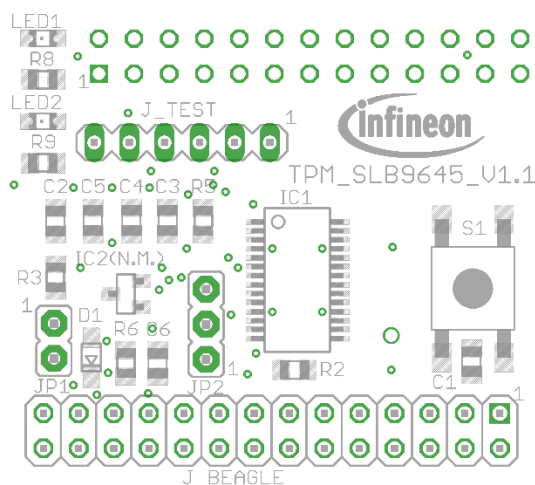


Figure 9 Schematic of the Infineon Iridium SLB 9645 TPM I2C Board

Figure 10 shows the placement of the Infineon Iridium SLB 9645 TPM I2C Board:

Raspberry PI socket connector (female), angled on solder side



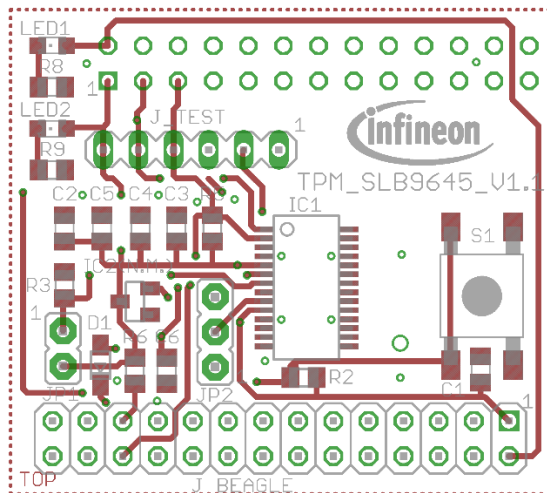
Beagle XM connector on solder side

Figure 10 Placement of the Infineon Iridium SLB 9645 TPM I2C Board

Appendix

**Figure 11** shows the layout of the component side of the Infineon Iridium SLB 9645 TPM I2C Board:

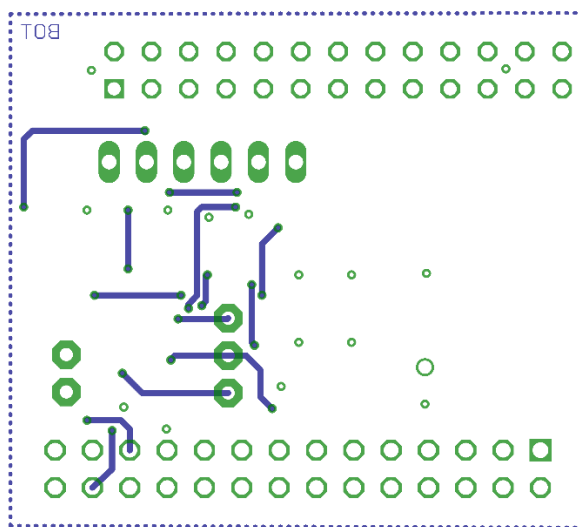
Raspberry PI socket connector (female), angled on solder side



Beagle XM connector on solder side

**Figure 11** Layout of the component side of the Infineon Iridium SLB 9645 TPM I2C Board

**Figure 12** shows the layout of the solder side of the Infineon Iridium SLB 9645 TPM I2C Board:



**Figure 12** Layout of the solder side of the Infineon Iridium SLB 9645 TPM I2C Board

## **6.4 TPM Features**

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

- Secure storage for 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 is being externally generated, imported and securely stored in the TPM's NV (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 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 access 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 itself.
- 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 RNG (random number generator) is protected against manipulation, thus these numbers are more reliable than numbers generated with a software RNG.
- Dictionary attack prevention mechanisms
  - Implemented in hardware; delays brute force attacks on the TPM by disabling the chip after a certain amount of failed authentication attempts and 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.

## **6.5 Running Commands with Root Privileges**

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

### **6.5.1 Using Sudo**

The package *sudo* is installed by default in Ubuntu® and Raspbian OS, but is not available for the Debian® distribution for the BeagleBoard xM.

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
```

In case a command consists of multiple sub commands, it is necessary to enclose the whole command in a *sudo* statement, required for example when loading the TPM driver manually (otherwise only the *echo tpm\_i2c\_infineon* command would be executed with root privileges, but not the redirection to the device file):

```
$ sudo sh -c "echo tpm_i2c_infineon 0x20 > /sys/bus/i2c/devices/i2c-2/new_device"
```

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 command terminal 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.

### **6.5.2 Logging 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 on as root user. Obviously this is not as comfortable as using *sudo* but it helps in case *sudo* is not available.

## 6.6 Interacting with the Embedded Platforms

For user interaction with the BeagleBoard xM / Raspberry Pi®, either a monitor and keyboard can be attached directly, or a remote connection via RS232 (only on BeagleBoard xM) or SSH can be used. This section describes all possibilities.

### 6.6.1 Direct Interaction

In order to directly interact with the BeagleBoard xM / Raspberry Pi®, just attach a USB keyboard to one of the USB ports of the BeagleBoard xM / Raspberry Pi® and a monitor, for example via HDMI.

This enables the user to view the complete boot process of the embedded platform.

### 6.6.2 Connecting via RS232 (BeagleBoard xM only)

A remote, but comparable way to a direct interaction (where the boot process can be observed) is connecting to the BeagleBoard xM via the RS232 port and a terminal program such as minicom.

To do so, connect the RS232 Serial-to-USB adapter shown in [Figure 13](#) with a USB extension cable to your developer PC. No driver installation is required.



**Figure 13 RS232 Serial-to-USB adapter**

*Note: If your developer PC has an RS232 port you can directly connect the BeagleBoard xM to it without an adapter via an RS232 cable instead, but this is out of scope for this application note.*

Now configure and use **minicom** as described in the following steps.

First, you need to configure it by running the following command in the command terminal on the developer PC:

```
# minicom -s
```

After that you will see the minicom menu, which looks like this:

```
+-----[configuration]-----+
| Filenames and paths         |
| File transfer protocols     |
| Serial port setup          |
| Modem and dialing           |
| Screen and keyboard         |
| Save setup as dfl            |
| Save setup as ..            |
| Exit                        |
| Exit from Minicom           |
+-----+-----+-----+-----+
```

To configure minicom to communicate with the BeagleBoard xM via the serial port, use the up and down arrows to select **Serial port setup** and press <Enter>.

Now you will see the next menu:

```
+-----+
| A -   Serial Device       : /dev/ttyUSB0      |
| B - Lockfile Location    : /var/lock          |
| C -   Callin Program      :                   |
| D -   Callout Program     :                   |
| E -   Bps/Par/Bits        : 115200 8N1        |
| F - Hardware Flow Control : No                |
| G - Software Flow Control : No                |
|                                     |
|   Change which setting?              |
+-----+
```

In this menu, do the following steps:

1. Press <A> on your keyboard to setup a serial device name such as /dev/ttyUSB0
  - ttyUSB0 is just an example; we use a second terminal **on the developer PC** to figure out which **tty** device the BeagleBoard xM uses: `dmesg | grep tty`
2. Press <E> to setup Baud Rate, Parity- and Stop Bits to the following settings:
  - 115200 8N1
3. Press <F> to set Hardware Flow Control = No
4. Press <Esc> to exit

Back in the main menu select **Save setup as dfl** (dfl = default) and press <Enter>, then select **Exit** and press <Enter>.

You are now in the minicom console, which you can also start by typing the following command from the command terminal:

```
# minicom
```

The terminal program should connect automatically to the BeagleBoard xM when you plug the serial-to-USB adapter in and power it up.

In order to disconnect and exit minicom, press <Ctrl+A> followed by <X>.

### 6.6.3 Connecting via SSH

To connect via Ethernet to the Raspberry Pi® or the BeagleBoard xM, you can also use SSH, a remote terminal. Additionally, with SSH you can open multiple simultaneous terminal sessions with the same or different users. This may come handy in sections 4.4.2.2 and 0 or when performing single tasks as root without *sudo* (see section 6.5).

SSH should be active by default. In case you have disabled SSH, you can reinstall it by running the following command from a terminal either directly with an attached keyboard and monitor (see section 6.6.1) or via minicom, if possible (see section 6.6.2):

```
# 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 embedded platform), 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**).

## Appendix

To use SSH, attach both the developer PC and the embedded platform 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 [23] for example on how to do so).

**Attention: For SSH to work, the IP addresses of both the developer PC and the embedded platform 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 embedded platform will also publish its host name in the LAN, which you can get by executing the command `hostname` on the embedded platform. 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® for example the default user is “pi”, the default hostname is “raspberrypi” and the default password for “pi” is “raspberrypi”. Thus, the SSH command would be the following:

```
$ ssh pi@raspberrypi
```

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 embedded platform. 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 terminal on the developer PC:

```
$ exit
```

## 6.7 Manual Loading of TPM Driver

Before you can load the driver you might have to check to which I<sup>2</sup>C bus the TPM is connected to. This must be done only once, since the bus is fixed and will not change neither after a reset of the TPM nor a reset of the whole embedded platform.

To do so, use the command `i2cdetect -r 1`, `i2cdetect -r 2`, and (if necessary) `i2cdetect -r 3` (the actual bus number depends on your installation). You might need to install the corresponding package first via the following command:

```
# apt-get install i2c-tools
```

If the chip is properly connected (that is there is no electrical problem), the terminal output looks like this (in case of errors, see section 6.1 for troubleshooting):

```
# i2cdetect -r 2
WARNING! This program can confuse your I2C bus, cause data loss and worse!
I will probe file /dev/i2c-2.
I will probe address range 0x03-0x77.
Continue? [Y/n]
    0  1  2  3  4  5  6  7  8  9  a  b  c  d  e  f
00:          -- -- -- -- -- -- -- -- -- -- -- -- -- --
10: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
20: 20 -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
30: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
40: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
50: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
60: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
70: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
```

In this example, the TPM is connected to I<sup>2</sup>C bus number 2 at address 0x20. Now you can properly load the TPM driver with this information:

```
# echo tpm_i2c_infineon 0x20 > /sys/bus/i2c/devices/i2c-2/new_device
```

If you are working on the BeagleBoard xM and after that no terminal input line appears, just press <Enter> again (it is a minicom issue).

After loading the Infineon TPM driver you can view the kernel log with the following command:

```
$ dmesg
```

The last few lines should look like this:

```
[some tick value] tpm_i2c_infineon 1-0020: 1.2 TPM (device-id 0x1A)
[some tick value] tpm_i2c_infineon 1-0020: Issuing TPM_STARTUP
[some tick value] tpm_i2c_infineon 1-0020: TPM is disabled/deactivated (0x7)
```

**Attention: The content and existence of the last line in the output above depends on the actual TPM state. But in case the first two lines are also not shown, the driver is not loaded. In case you did not configure the kernel to load the driver automatically, you have to load the TPM driver manually with the command shown above after every system restart! For troubleshooting on these topics, see section 6.1.2.**

Now the TPM driver is loaded and the TPM has been started. In order to get access to the full functionality, the TPM has to be additionally initialized, which is described in the following section.



## **6.8 TPM States and the TPM Startup Script**

This chapter briefly explains and refers to some of the TPM's possible states. Since explaining all the different TPM states in full detail would exceed this Application Note's scope, please refer to the TPM specification for further information about TPM states. The TPM specification can be found at [24].

After powering on, the TPM by design always is in a state, which provides no TPM functionalities except the ability to finish the TPM initialization by explicitly issuing a TPM\_Startup command.

Also, in order to get the TPM into an operational state, a TPM\_SelfTestFull or TPM\_ContinueSelfTest command should be issued in order to run the TPM's internal functionality checks.

*Note: In case you perform a complete system reset of your embedded platform, both steps will be done by the TPM driver on driver load, but in case you only do a TPM reset without a system reboot, you have to do these steps manually.*

The TPM is now operational, but it can still be in a deactivated and / or disabled state (for example the factory default is disabled and deactivated, so after clearing<sup>1</sup> the TPM its state will always be disabled and deactivated).

Additionally, for some critical commands (for example for (de)activating the TPM), the TPM must have *Physical Presence* asserted (at least as long as ownership<sup>2</sup> of the TPM has not been taken).

*Note: (De)activating (and thus also clearing) of a TPM additionally requires a TPM reset for the action to be completed.*

In brief, several actions with specific preconditions have to be done before a user can really use a factory-new TPM. In a PC, these tasks would be manageable or even automatically done by the PC's BIOS (of course only if the BIOS has TPM support; more information can be found at [25]), but on embedded platforms without a BIOS the user has to take care of these tasks by himself.

In order to simplify all of these actions, all the steps described above can be done by the script `tpmstartup.py` shipped with this document. It can be executed in two different modes called "*Admin-mode*" and "*User-mode*" here.

The motivation for these two modes is to show the reader how to bring the TPM either in a

- more manageable but less secure<sup>3</sup> state (equivalent to the state that a TPM has during TPM management actions in a TPM-aware PC BIOS, where simply spoken all TPM commands requiring Physical Presence to be asserted are executable), or a
- less manageable but more secure state (equivalent to the normal operation mode of a TPM in a PC, for example during or after startup of the OS; Physical Presence is locked and cannot be asserted in this mode in order to prevent a normal user to execute critical TPM commands that for security reasons only an administrator knowing the TPM owner authorization secret should be allowed to do).

Thus, it is logical that actions requiring Physical Presence asserted can and will only be performed in Admin-mode.

---

<sup>1</sup> Clearing the TPM means to issue a TPM\_Clear command, which resets the TPM to its factory defaults. This also results in the fact that data that was protected by the TPM is not accessible anymore since after clearing a TPM the data itself or the encryption key that has been used to encrypt the data has been deleted.

<sup>2</sup> There are two authorization techniques for performing critical TPM actions: Owner Authorization and Physical Presence Authorization. But as long as ownership has not been taken, only Physical Presence Authorization can be used to perform tasks such as (de)activating the TPM.

<sup>3</sup> Even with Physical Presence asserted, it is not possible for any attacker to read out secret data like user keys or protected data unless he knows the proper authorization secret. But an attacker can misuse Physical Presence to clear a TPM without the need for providing any further authorization secret resulting in an unrecoverable loss of the protected data.

The following list shows which commands performed with the two modes:

Steps in Script	Admin-mode	User-mode
Issue a TPM startup <sup>1</sup>	x	x
Run a TPM self test <sup>1</sup>	x	x
Set a flag that allows to assert Physical Presence via Software	x	
Assert Physical Presence	x	
Lock Physical Presence		x
Enable the TPM <sup>2</sup>	x	
Activate the TPM <sup>2</sup>	x	
Set a flag that allows to take ownership of the TPM	x	

In other words, the Admin-mode initializes the TPM, asserts Physical Presence and enables and activates the TPM. This mode is required for example for the first use of the TPM or after resetting the TPM to factory settings with a TPM\_Clear command. The Admin-mode can be executed with the following command:

```
# python tpmstartup.py -a
```

If you execute the script without “-a”, the script will be executed in User-mode. This mode is for the general use of the TPM and just initializes the TPM and then locks Physical Presence. To assert Physical Presence you have to reset the TPM with the reset button and execute the script in Admin-mode.

**Attention: In order to avoid any TPM-state-related problems during execution of the steps described in this application note, run the TPM startup script after every TPM or embedded platform reset! Also make sure to use the correct mode as described in the corresponding chapters. For troubleshooting on these topics, see sections 6.1.3 and 6.1.4.**

<sup>1</sup> Although both these steps are being performed by the driver, both these steps are being done every time the script runs. This keeps the script simple and also avoids usage errors with TPMs accidentally left uninitialized (for example in case the TPM has been reset without a system reboot).

<sup>2</sup> Although the TPM might already be enabled and activated, both these steps are being done every time the script runs in Admin-mode. This keeps the script simple and also avoids usage errors with TPMs accidentally left disabled or deactivated (for example after a reset to factory settings).

## 6.9 Cryptographic Principles Used in SSL / TLS

SSL / TLS uses several cryptographic key principles to establish a secure channel where a sender is able to send data without disclosure or undetected manipulation to an authenticated recipient without the drawbacks of the single cryptographic principles. In short, it uses

1. *Symmetric Cryptography* for encryption of the actual transfer data in order to achieve a higher performance.
2. *Asymmetric Cryptography* for exchanging the data encryption key in order to solve the key-exchange problem of symmetric cryptography.
3. *Certificates* (optionally with *Certificate Chains*) to check the authenticity of one or both communication partners.

The following section describes the single principles in more detail.

### 6.9.1 Cryptography

The basic principle behind modern cryptography is that the used cryptographic algorithm is standardized and well-known, while the security only relies on the assumption that the used cryptographic key (or to be more specific: the key required for decryption) is only known to the recipient(s) and has not been disclosed to anyone else.

#### 6.9.1.1 Symmetric Cryptography

In *Symmetric* cryptography the same key is used for encryption and decryption of data. It is usually fast and still offers appropriate security (as for all types of cryptography both attributes of course strongly dependent on the used cryptographic algorithm). The problem with symmetric cryptography is how to exchange the used key. Obviously the plain key cannot be sent along with the encrypted data, because in that case every potential listener or attacker is able to intercept and decrypt the encrypted data. Thus, the encryption key must be encrypted as well. But using symmetric cryptography for that would again lead to the same problem.

#### 6.9.1.2 Asymmetric Cryptography

In *Asymmetric* cryptography the encryption is being done with a public key while decryption can only be done with a different and secret private key matching the public key used for encryption. The idea behind this is that the security is not compromised if everyone can encrypt data as long as only the correct and valid receiver of data can decrypt it.

In order for this to work, the recipient first has to publish the public key matching to his own private key. Now, any sender can encrypt data with this public key and send it to the recipient while being sure that only the sender can decrypt it, since only the recipient has the matching private key.

Obviously, the sensitive part here is the private key which should be protected at all costs.

The advantage of asymmetric cryptography is that there is no key exchange problem. The public key can be distributed while the private key can remain secret.

Unfortunately, asymmetric cryptography is slow compared to symmetric cryptography rendering it impractical for large amounts of data.

### 6.9.2 Signing

While encryption and decryption prevents disclosure of data, it does not make sure that the data is from the correct sender. *Signing* uses asymmetric cryptography to address this topic.

Here, a sender encrypts the data to be sent with his private key. Now, the recipient can validate that the data comes from the right sender by decrypting the data with the public key of the sender.

This can also be combined with normal data encryption, where the data is being signed before encryption. But even though symmetric cryptography could be used for encryption, there is still an asymmetric cryptographic operation done, the signing itself. Thus, signing is impractical for ensuring the origin for large amounts of data.

### **6.9.3 Certificates**

#### **6.9.3.1 General Information**

A Certificate is used to authenticate<sup>1</sup> the certificate holder against a requester. It contains public information about the holder of a certificate, including a public key matching the holder's private key. With that information anyone can send a challenge (for example a random number encrypted with the public key) to a supposed certificate holder. Now, by decrypting the random number and sending it back to the challenger, she / he proves that she / he really is the certificate holder since no one else has the matching private key.

Obviously, the private key is the sensitive information here, since its secrecy prevents an attacker from being able to impersonate her- / himself as the valid communication partner. But for this to be effective, also the certificate must be associated to the correct communication partner.

Thus, the sender must either know the correct communication partner's certificate in advance or be able to validate and trust unknown certificates somehow.

#### **6.9.3.2 Certificate Chain and Certification Authority**

The problem with knowing the certificate or public key of a communication partner in advance is the dynamic nature and vast number of potential communication partners. It is simply not possible that everyone can know the certificate / public key of everyone else at any time. And obviously trusting every certificate implicitly is also not an option.

One solution for this is to add a constraint that any unknown certificate must be issued by a commonly accepted institution called *Certification Authority* (or *CA*). Such an authority is responsible for making sure that the certificate holder really is the one she / he claims to be.

After that, the CA issues the certificate by adding information about the CA to it and then digitally signs it with the CA's own private key. This is called a *Certificate Chain*. Now, any certificate with a valid chain can be trusted implicitly, even though it was unknown to the sender before.

---

<sup>1</sup> Authentication is only one of the possible purposes of a certificate. Certificates can also be used for signing and data exchange (encryption / decryption). Furthermore, certificates that comply with the X.509 standard may contain information on the intended certificate purposes restricting the actions possible with these certificates to a specific feature set.

## 6.10 The OpenSSL Heartbleed Bug

Since there is much information on the *OpenSSL Heartbleed Bug* itself in the internet, this application note will concentrate on explaining the improvements and remaining vulnerabilities of servers running affected OpenSSL versions with a TPM.

The Heartbleed Bug allowed an attacker to read out parts of the server's memory. The worst case scenario here is that these parts contain the server's private key. This key is most notably being used to authenticate a server's identity against a client, and it is also part of the SSL handshake and thus the key exchange protocol.

Now, if an attacker can obtain this private key, he is for example able to run a man-in-the-middle attack on secured connections and – depending on the session settings, that is not to use *Forward Secrecy* – also to retroactively decrypt the encrypted communication of recorded SSL sessions.

Obviously, using the OpenSSL TPM integration does not fix the Heartbleed Bug, since it is a bug in the OpenSSL software. Thus, an attacker can still read out the memory of servers running vulnerable OpenSSL versions. But at least the memory can never contain the server's private key since it is protected by the TPM, so the worst case scenario cannot occur.

Of course, the bug has been fixed in OpenSSL quickly after it has been detected, but nobody knows how many server keys have been compromised until then. And since software is and will always be a complex matter, nobody can guarantee that exploits with similar effects will not happen again in the future or will be detected right away if they happen.

Thus, the general threat remains, so it is always a good choice to use and rely more on certified and well-tested hardware protection modules for sensitive data such as the TPM.

*Note: To check if your installed OpenSSL-Version is vulnerable to the Heartbleed Bug, execute the following command:*

```
$ apt-cache policy openssl
```

*The output should contain a line such as the following:*

```
Installed: 1.0.1e-2+rvt+deb7u5
```

*Important here are the version and the update level: in case the version is higher than 1.0.1e, or it is 1.0.1e and has at least update 5 (as shown here by the string `deb7u5`) your OpenSSL version is not vulnerable to the Heartbleed Bug.*

[www.infineon.com](http://www.infineon.com)