AN5714

Application note

Integrating the ST33TPHF2xSPI and ST33TPHF2xI2C trusted platform modules with Linux®

Introduction

The objective of this application note is to describe how to use the ST33TPHF2xSPI and ST33TPHF2xI2C trusted platform module (TPM) devices on a Raspberry Pi® board. For that purpose, it explains how to enable TPM support in the Linux® kernel, describes some tools used to communicate with the TPM device, and it presents some practical use cases.

Table 1. **Applicable products**

Type	Product
ST33TPHF2xSPI TPM devices	ST33TPHF2XSPI
	ST33TPHF2ESPI
	ST33TPHF20SPI
ST33TPHF2xI2C TPM devices	ST33TPHF2XI2C
	ST33TPHF2EI2C
	ST33TPHF20I2C

1 Open-source software resources

The table below lists all the public resources needed to take full advantage of this application note.

Table 2. List of open-source software resources

Reference	Open-source software resource
[RPiOS]	The latest version of the Raspberry Pi OS, available from https://www.raspberrypi.org
[WIN32DiskImager]	WIN32 Disk Imager tool available from https://sourceforge.net
[Etcher]	Etcher software available from https://www.balena.io
[RPiImager]	Raspberry Pi Imager available from https://www.raspberrypi.org
[LinuxKernel]	Linux kernel adapted for a Raspberry Pi device available from https://github.com/raspberrypi/linux(1)
[TCG-TPM-I2C-DRV]	STMicroelectronics patch to add support for the ST33TPHF2xI2C I²C products, available from https://github.com/STMicroelectronics/TCG-TPM-I2C-DRV
[TSS]	The TSS available from https://github.com/tpm2-software/tpm2-tss(1)
[RM]	Resource manager (RM) available from https://github.com/tpm2-software/tpm2-abrmd(1)
[TPM2-TSS]	GitHub repository containing projects for specific use cases: https://github.com/tpm2-software(1)
[TPM2-SW]	Github repository containing the TPM software stacks: https://tpm2-software.github.io(1)
[OpenSSL-engine]	*OpenSSL*® engine project accessible from https://github.com/tpm2-software/tpm2-tss-engine(1)
[PKCS#11]	PKCS#11 project available from https://github.com/tpm2-software/tpm2-pkcs11(1)
[TPM2.0 tools]	Set of executable files to run TPM commands: https://github.com/tpm2-software/tpm2-tools(1)

1. *This URL belongs to a third party. It is active at document publication, however, STMicroelectronics shall not be liable for any change, move or inactivation of the URL or the referenced material.*

Table 3. Compliant open-source references

Reference	Version	
[LinuxKernel]	5.4	5.10
[TPM2-TSS]	3.0.3	3.1.0
[TPM2.0 tools]	5.0	5.2

Note: This is a non-exhaustive list of open source package versions compliant with this application note.

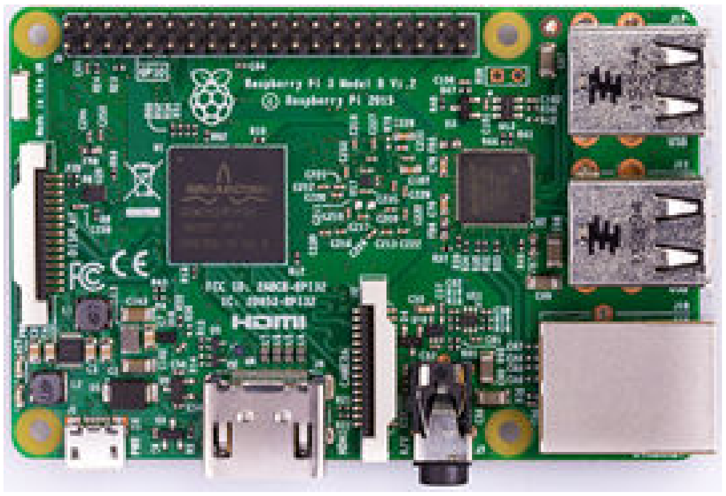
2 Raspberry Pi®

A Raspberry Pi is a miniature computer that can be used to learn programming. The Raspberry Pi is fitted with:

- an HDMI® port to connect an appropriate display
- four USB ports to connect a mouse, a keyboard, and any other device.

In February 2016, the third generation Raspberry Pi (the Raspberry Pi 3 Model B, illustrated below) replaced the Raspberry Pi 2 Model B. In addition to the functionality implemented in the Raspberry Pi 2 (such as the 4 USB ports, 1 Gbyte of RAM and the ethernet port), the Raspberry Pi 3 Model B implements Bluetooth® 4.1. It has the same form factor as the previous Pi 2 (and Pi 1 Model B+), and has complete backward compatibility with Raspberry Pi 1 and 2.

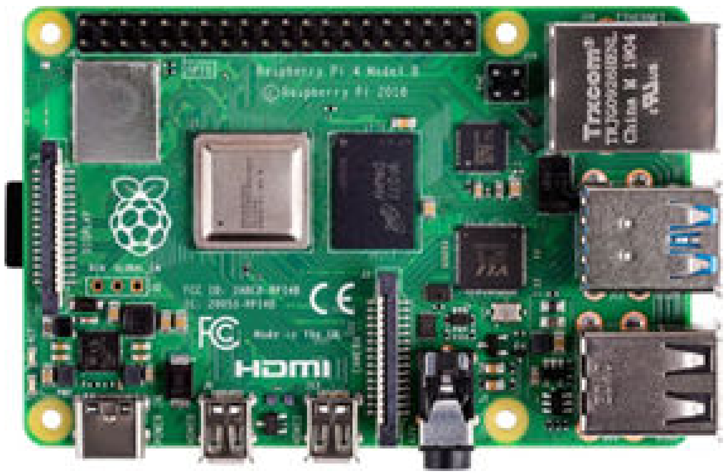
Figure 1. Raspberry Pi 3 model B



The Raspberry Pi 4 Model B (depicted below) was released in 2019. Its main improvements over the Raspberry Pi 3 Model B are:

- Increased RAM capacity support: 2, 4, or 8 Gbytes
- Two USB-3 ports and two USB-2 ports, instead of four USB-2 ports.
- Dual display support, using two micro-HDMI ports.

Figure 2. Raspberry Pi 4 model B



Each Raspberry Pi has a 40-pin connector. The pin description is the same across Raspberry Pi models.

3 STPM4RasPI connector board

The STPM4RasPI board is a daughterboard used to connect a TPM device to a Raspberry Pi.

A TPM is a widely used, standardized solution that acts as the corner stone of personal computer and server security. ST TPM devices are a perfect fit for ecosystems built on Windows® and Linux® operating systems. Certified by the CC and compliant with FIPS 140-2, all ST TPM products meet security and regulatory requirements. The ST product portfolio is qualified for consumer, industrial and automotive applications.

The TPM can be used to store cryptographic keys, platform measurements and other sensitive data. Moreover, it can serve as a hardware random number generator or a cryptographic accelerator. Hereafter, only hardware TPM devices are discussed.

The TPM communicates with the host system via an SPI or an I²C bus. It acts as a slave on the bus, meaning it can only report to the platform, and cannot request an action to be taken.

The host system sends commands to the TPM via byte streams on the SPI or I²C bus. These commands sometimes require authorizations to ensure that only authorized users access sensitive resources.

The STPM4RasPI board includes a soldered ST33TPHF2xSPI or ST33TPHF2xI2C product. For more information, refer to the respective data briefs on the ST website.

The ST33TPHF2xSPI and ST33TPHF2xI2C TPM products have an Arm® core.

Note: Arm is a registered trademark of Arm Limited (or its subsidiaries) in the US and/or elsewhere.

Refer to the STPM4RasPI data brief available from the st.com website for the pin description of the STPM4RasPI board.

4 Raspberry Pi setup

4.1 Raspberry Pi OS

The Raspberry Pi operating system (OS), formerly called Raspbian, is based on a Linux® Debian® OS. Download the latest version of the Raspberry Pi OS from the Raspberry Pi website (see [RPiOS]). For this tutorial, the recommended image is “Raspberry Pi OS with desktop”.

Use a disk imaging tool, such as Etcher or WIN32 Disk Imager, to flash the image onto an SD card. The image is slightly smaller than 4 Gbytes, it is recommended to use an 8 Gbytes or above SD card, to allow for more application memory space.

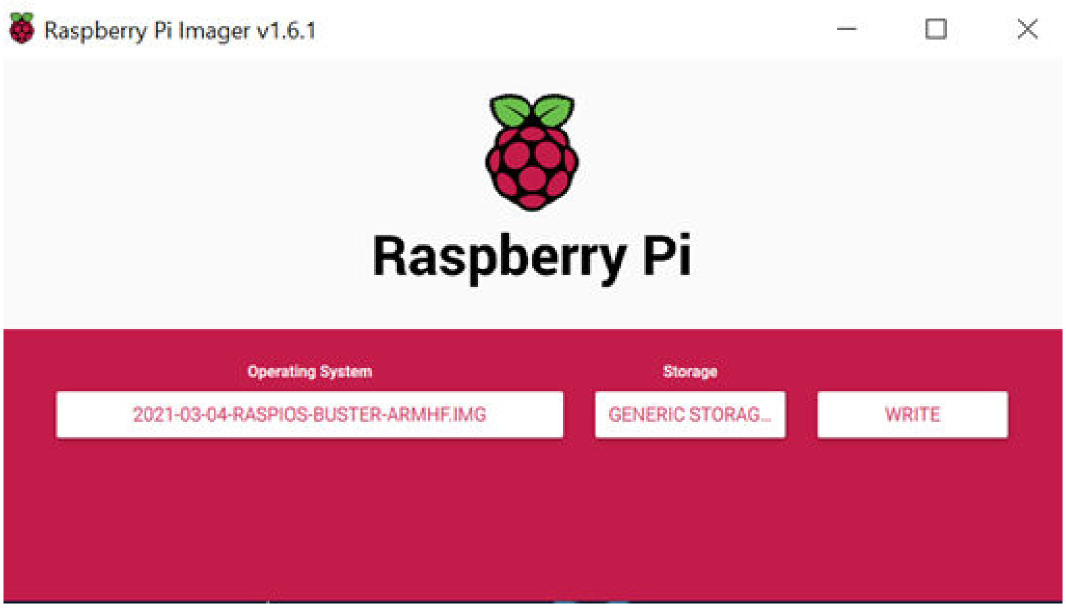
See [WIN32DiskImager] for the link to download the WIN32 Disk Imager tool.

See [Etcher] for the link to download Etcher.

Raspberry Pi also provides a disk imaging tool named the Raspberry Pi Imager, see [RPiImager].

With any of the above tools, simply select the OS image to be copied and the storage location to write to, and press the button to launch the disk imaging process. For instance, with the Raspberry Pi disk imager:

Figure 3. Flashing an SD card with the Raspberry Pi disk imager


Raspberry Pi Imager v1.6.1
Raspberry Pi
Operating System
2021-03-04-RASPIOS-BUSTER-ARMHF.IMG
Storage
GENERIC STORAG...
WRITE

The WIN32 disk imager has the additional functionality, to read from a disk.This means that the disk data is saved as an image file. It is useful to back up the current Raspberry Pi image.

Note: Debian is a registered trademark owned by Software in the Public Interest, Inc.

4.2 Hardware setup

After flashing the SD card, insert it into the SD card slot of the Raspberry Pi device.

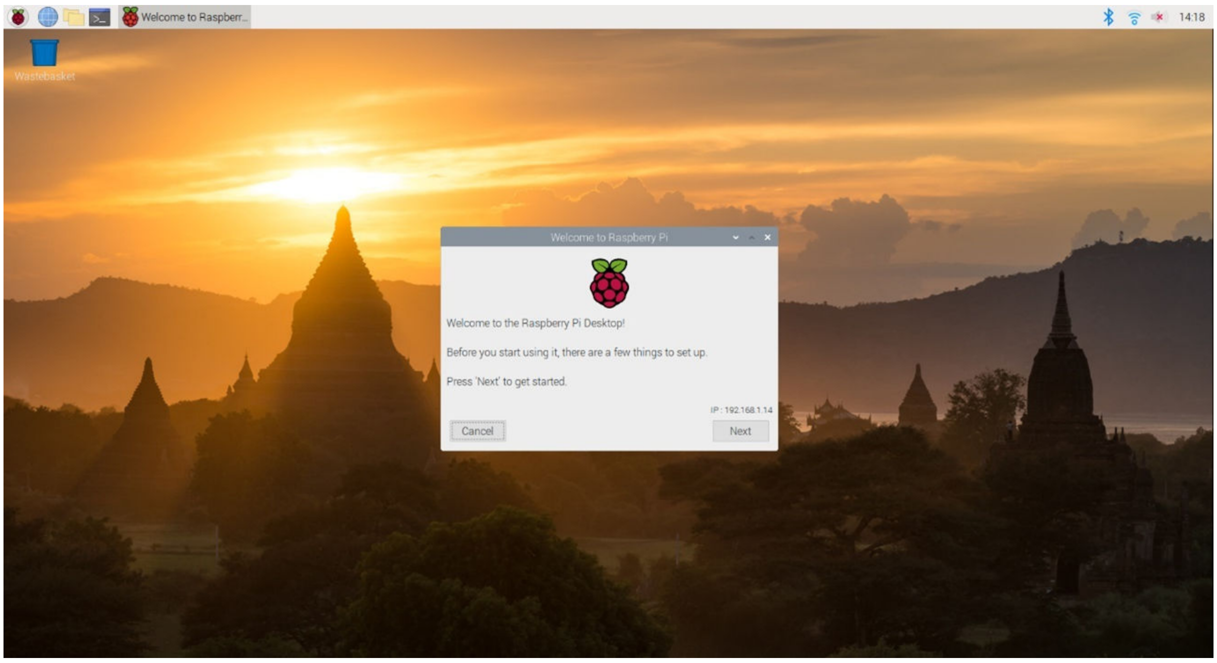
Then connect the computer screen to the Raspberry Pi device via HDMI, and a keyboard and mouse via the USB ports.

Finally, connect the Raspberry Pi power source.

Note: Use the power supply provided with the Raspberry Pi.

After the first boot, the home screen appears as shown below:

Figure 4. Home screen



Welcome to the Raspberry Pi Desktop!

Before you start using it, there are a few things to set up.

Press 'Next' to get started.

IP : 192.168.1.14

Follow the configuration steps to finalize the configuration of your Raspberry Pi device.

5 Enabling TPM support in the Linux kernel

To allow the Raspberry Pi host system to communicate with the TPM device, TPM support must be activated in the Linux kernel. The safest way to do this is to download, compile and install a new Linux kernel.

5.1 SPI/I²C activation

Navigate to [**Menu**]>[**Preferences**]>[**Raspberry Pi Configuration**] and activate I²C and/or SPI in the **Interfaces** item. This corresponds to the communication buses that are used to communicate with the ST33TPHF2xSPI and ST33TPHF2xI2C devices.

5.2 Downloading the Linux kernel

The Linux kernel, adapted for a Raspberry Pi device, is available on the GitHub website (see [LinuxKernel]). The stable Linux kernel version 5.4 is used in this tutorial.

The Linux kernel supports an SPI TPM device driver since version 4.8. To use an STMicroelectronics **SPI** TPM device, the kernel version must therefore be greater than 4.8, or the SPI TPM driver must be ported.

To use the STMicroelectronics **I²C** TPM, the 5.4 Linux kernel must be used, in order to use the STM patch specifically designed to create an adapted I²C TPM driver.

To download the 5.4 Linux kernel, use the following Linux terminal commands:

```
# sudo apt install bc
# cd /home/pi/Downloads
# git clone -b rpi-5.4.y --depth=1 https://github.com/raspberrypi/linux(1)
```

In this case, the selected working directory is `/home/pi/Downloads`, use any convenient directory.

5.3 Enabling TPM support

5.3.1 Drivers

In the downloaded Linux files, the source codes of the TPM drivers are located in the `linux/drivers/char/tpm` directory.

This directory contains the source file of a generic driver for SPI TPM devices, called `tpm_tis_spi.c`, that is used with the ST33TPHF2xSPI products (ST33TPHF2XSPI, ST33TPHF2ESPI and ST33TPHF20SPI).

There is also a driver source code for the STMicroelectronics ST33TPM12SPI and ST33TPM12I2C legacy TPM 1.2 products, available in the `st33zp24` directory.

STMicroelectronics provides a patch to add support for the ST33TPHF2xI2C I²C products (ST33TPHF2XI2C, ST33TPHF2EI2C and ST33TPHF20I2C), and to optimize the SPI driver. See [TCG-TPM-I2C-DRV] for the link to download this patch.

To apply the patch, copy and paste it into the `/home/pi/Downloads/linux/drivers/char` directory. Then run the following commands from a terminal:

```
# cd /home/pi/Downloads/linux/drivers/char
# patch -b -p0 < patchTPMv_5_4_2.patch
```

This generates the `tpm_tis_i2c.c` file, which is the STMicroelectronics I²C TPM driver source code.

5.3.2 Device tree

The hardware present in the Raspberry Pi device uses the Raspberry Pi device tree, which allows the description of onboard features and also external hardware.

As the TPM is an external hardware element, it must be added to the device tree for the system to correctly take it into account.

The `linux/arch/arm/boot/dts` directory contains the device tree description files, which are compiled into DTB (device tree blob) files that are used at each boot of the Raspberry Pi device.

First of all, find the correct *dts* file for the board. For an Raspberry Pi board, the file has the following format: `bcm27xx-{name-of-board}.dts`. Some examples:

- **Raspberry Pi 3B**: `bcm2710-rpi-3-b.dts`
- **Raspberry Pi 3B+**: `bcm2710-rpi-3-b-plus.dts`
- **Raspberry Pi 4B**: `bcm2711-rpi-4-b.dts`

After locating the correct *dts* file for the board, edit it as indicated in the following tables.

Table 4. **Editing the *dts* file for an SPI TPM device**

Replace	By
`spidev0: spidev@0{`	`st33htpm0: st33htpm@0{`
`compatible = "spidev";`	`compatible = "st,st33htpm-spi";`
`reg=<0>; /* CE0 */`	`reg = <0>;`
`#address-cells = <1>;`	`#address-cells = <1>;`
`#size-cells = <0>;`	`#size-cells = <0>;`
`spi-max-frequency = <500000>;`	`spi-max-frequency = <33000000>;`
`};`	`status = "okay";`
	`};`

Table 5. **Editing the *dts* file for an I²C TPM device**

Replace	By
`&i2c1 {`	`&i2c1 {`
`pinctrl-names = "default";`	`pinctrl-names = "default";`
`pinctrl-0 = <&i2c1_pins>;`	`pinctrl-0 = <&i2c1_pins>;`
`clock-frequency = <100000>;`	`clock-frequency = <400000>;`
`};`	`st33htpi: st33htpi@0{`
	`compatible = “st,st33htpm-i2c";`
	`reg = <0x2E>;`
	`status = "okay";`
	`};`
	`};`

5.3.3 Linux kernel configuration

The last file that needs modifying is the Linux kernel configuration file, which contains the list of kernel options that the user can activate or deactivate. In this file, the TPM options must be activated.

The `linux/arch/arm/configs` directory contains the configuration files for the different Raspberry Pi devices. The file to edit depends on the version of the Raspberry Pi board as indicated in the table below.

Table 6. **Configuration file to edit according to Raspberry Pi board**

Raspberry Pi board	Configuration file to edit
Raspberry Pi version 1	`bcmrpi_defconfig`
Raspberry Pi version 2 or 3	`bcm2709_defconfig`
Raspberry Pi version 4	`bcm2711_defconfig`

Modify the configuration file as described in the following tables.

Table 7. **Editing the configuration file for an SPI TPM device**

Replace	By

`CONFIG_TCG_TPM=m`
`CONFIG_TCG_TIS_SPI=m`
	`CONFIG_TCG_TPM=y`
	`CONFIG_TCG_TIS_CORE=y`
	`CONFIG_TCG_TIS_SPI=y`

Table 8. **Editing the configuration file for an I²C TPM device**

Replace	By

`CONFIG_TCG_TPM=m`
`CONFIG_TCG_TIS_SPI=m`
	`CONFIG_TCG_TPM=y`
	`CONFIG_CRC_CCITT=y`
	`CONFIG_TCG_TIS_CORE=y`
	`CONFIG_TCG_TIS_I2C=y`

It is also possible to set the `CONFIG_TCG_TIS_SPI` and `CONFIG_TCG_TIS_I2C` options to `m` instead of `y`. Doing so causes these options to be compiled as loadable modules. This is useful when the user is not sure whether a feature is to be used or not. In this way, the feature can be activated when needed, but is not automatically loaded during platform boot.

5.3.4 Compiling and installing the kernel

After making the appropriate modifications, compile the Linux kernel. The instructions differ slightly according to the Raspberry Pi board version.

First take the configuration into account as shown below.

Table 9. **Configuring the Linux kernel compilation**

Raspberry Pi version 1	Raspberry Pi version 2 or 3	Raspberry Pi version 4

`# cd /home/pi/Downloads/`
`# cd linux`
`# KERNEL=kernel`
`# make bcmrpi_defconfig`
	`# cd /home/pi/Downloads/`
	`# cd linux`
	`# KERNEL=kernel7`
	`# make bcm2709_defconfig`
		`# cd /home/pi/Downloads/`
		`# cd linux`
		`# KERNEL=kernel7l`
		`# make bcm2711_defconfig`

Then compile and install the Linux kernel with the following instructions:

```
# sudo apt install libssl-dev flex bison
# make -j4 zImage modules dtbs
# sudo make modules_install
# sudo cp arch/arm/boot/dts/*.dtb /boot/
# sudo cp arch/arm/boot/dts/overlays/*.dtb* /boot/overlays/
# sudo cp arch/arm/boot/dts/overlays/README /boot/overlays/
# sudo cp arch/arm/boot/zImage /boot/$KERNEL.img
```

Warning: *The process may take some time to complete*

After completion of all the instructions, reboot the Raspberry Pi.

After reboot, the TPM device driver must be present if it was configured as `y`. If it was configured as a module, use the `modprobe` command to load the module as shown below:

- **For an SPI TPM**:

```
# sudo modprobe tpm_tis_spi
```

- **For an I²C TPM**:

```
# sudo modprobe tpm_tis_i2c
```

The TPM driver is present in the `/dev` directory. It is named `/dev/tpm0`. This driver has the same name whether the TPM device uses an SPI or I²C interface, the communication interface is transparent to applications using the driver, and the device driver handles the specificities of the TPM communication interface.

There is a second TPM driver named `/dev/tpmrm0` that integrates the in-kernel resource manager. Use the following command to check whether the driver is loaded:

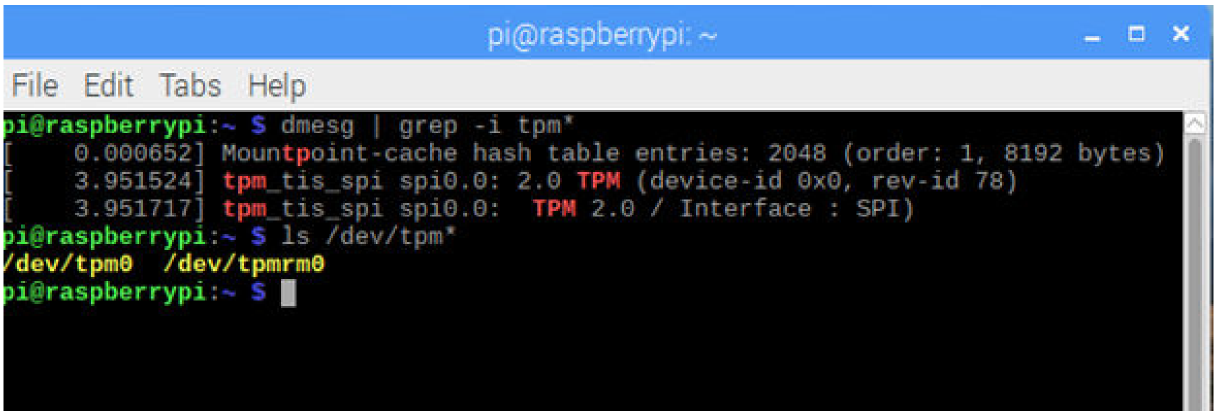
```
# ls /dev/tpm*
```

It is also possible to check the boot logs for TPM activity with the following command:

```
# dmesg | grep -i tpm
```

The figure below shows an image of the expected output.

Figure 5. Expected output once the TPM driver is loaded



```
pi@raspberrypi:~ $ dmesg | grep -i tpm*
[    0.000652] Mountpoint-cache hash table entries: 2048 (order: 1, 8192 bytes)
[    3.951524] tpm_tis_spi spi0.0: 2.0 TPM (device-id 0x0, rev-id 78)
[    3.951717] tpm_tis_spi spi0.0:  TPM 2.0 / Interface : SPI)
pi@raspberrypi:~ $ ls /dev/tpm*
/dev/tpm0  /dev/tpmrm0
pi@raspberrypi:~ $
```

If both an SPI TPM and an I²C TPM are active on the same system, then the `/dev/tpm1` and `/dev/tpmrm1` drivers are also present.

6 Low-level communication

The `/dev/tpm0` driver ensures the lowest level of communication with the TPM device. Scripts can be used to write byte sequences to the `/dev/tpm0` driver, which then correctly formats and sends the data to the TPM via the SPI or I²C bus.

Note: *Other methods exist other than scripts. In this section, only the scripting method is discussed, other methods are presented in the following sections.*

6.1 Linux terminal I²C commands

A TPM communication is a series of bytes written and read from the TPM registers. These registers can be written and read on an I^{2}C TPM using the `i2cset` and `i2cget` commands. The `i2cset` command is for writing to the I²C bus whereas the `i2cget` command is for reading from the bus.

The TPM device address on the I²C bus is 0x2E.

The `i2cset` command is particularly useful in the case where two I²C TPM devices are connected to the Raspberry Pi. In that case, modify the `.dts` file to integrate the following lines:

```
&i2c0 { pinctrl-names = "default";
        pinctrl-0 = <&i2c1_pins>;
        clock-frequency = <400000>;
        st33htpi: st33htpi@0{
                    compatible = "tcg,tpm-tis-i2c";
                    reg = <0x2F>;
      };

&i2c1 { pinctrl-names = "default";
        pinctrl-0 = <&i2c1_pins>;
        clock-frequency = <400000>;
        st33htpi: st33htpi@0{
                    compatible = "tcg,tpm-tis-i2c";
                    reg = <0x2E>;
      };
```

On first boot, connect only one TPM device, and change its address to 0x2F, so as to have different addresses for the two TPM devices.

Use the following command to change the TPM device address on the bus:

```
i2cset -y -f 1 0x2E 0x38 0x802F w
```

Where:

- 1 is the I²C bus number
- 0x2E is the TPM device current address
- 0x38 is the register in which the TPM device new address needs to be written
- 0x802F is the value that is to be written to the register in order to change the TPM device address.

Then switch off the Raspberry Pi board and connect the second TPM. When connected, the second TPM device is automatically associated with address 0x2E.

Turn the board back on. On reboot, two TPM devices are present: `/dev/tpm0` and `/dev/tpm1`.

6.2 Python scripts

Python scripts can be used to send simple commands to the TPM device. For instance, hereafter is a simple script for sending a `TPM2_GetRandom` command and requesting 16 random bytes.

```
import binascii
with open('/dev/tpm0','r+b',buffering=0) as tpm :
    tpm.write(binascii.unhexlify(b'80010000000c0000017b0010'))
    print(tpm.read())
```

After writing the desired code to a file (for instance named `TPM2_GetRandom.py`), execute it with the following command:

```
# sudo python3 TPM2_GetRandom.py
```

6.3 C language scripts

In the same way as Python scripts, C language scripts can be used to send commands to the TPM device. The previous example for sending a TPM2_GetRandom command and requesting 16 random bytes is also achieved with the following code:

```
#include <stdio.h>
int main()
{
     FILE *tpm;
     char str[] = "\x80\x01\x00\x00\x00\x0c\x00\x00\x01\x7b\x00\x10";
     char buffer[100];
     int i, n;
     tpm = fopen("/dev/tpm0", "rb+");
     if (tpm == NULL){
         printf("ERROR: Could not open driver file.\n");
         return -1;
     }
     n=fwrite(str, 1, sizeof(str), tpm);
     if (n != sizeof(str)){
         printf("ERROR: Could not write bytes to TPM.\n");
         fclose(tpm);
         return -1;
     }
     n = fread(buffer, 1, sizeof(buffer), tpm);
     printf("TPM Response: ");
     for (i=0; i<n; i++){
         printf("%x ", buffer[i]);
 }
     printf("\n");
     fclose(tpm);
     return 0;
}
```

Save these instructions to a `TPM2_GetRandom.c` file. C language scripts are different from Python scripts because they have to be compiled before they are executed by the terminal. To compile the script, use the following command:

```
# gcc TPM2_GetRandom.c –o TPM2_GetRandom
```

Then, to execute the script:

```
# sudo ./TPM2_GetRandom
```

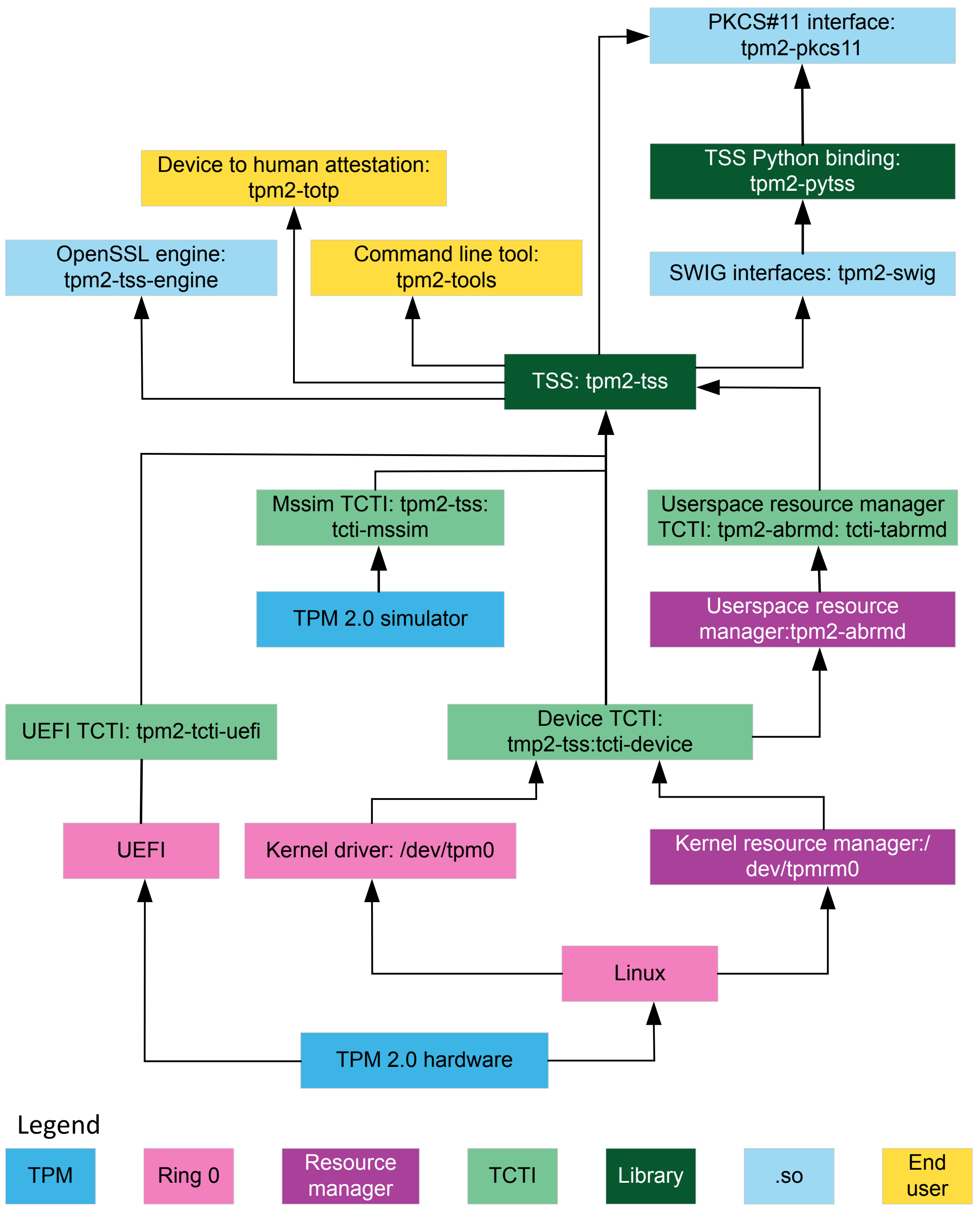
7 TPM software stack (TSS)

The Python and C language scripts are very useful for rapidly sending short commands. But some commands require complex authorizations and can be hundreds of bytes long. This makes defining the byte sequences manually very difficult.

The TPM software stack exists for such purposes. It is an API that allows commands to be sent to the TPM device via C language programs.

A high-level view of the TPM software stack that is used in this application note is presented in the graph illustrated below and sourced from [TPM2-SW]. Additional tools relying on the TSS, such as the TPM2.0 tools, are also covered in the next sections of this application note.

Figure 6. TPM software stack illustration


PKCS#11 interface: tpm2-pkcs11
TSS Python binding: tpm2-pytss
Device to human attestation: tpm2-totp
OpenSSL engine: tpm2-tss-engine
Command line tool: tpm2-tools
SWIG interfaces: tpm2-swig
TSS: tpm2-tss
Mssim TCTI: tpm2-tss: tcti-mssim
Userspace resource manager TCTI: tpm2-abrmd: tcti-tabrmd
TPM 2.0 simulator
Userspace resource manager:tpm2-abrmd
UEFI TCTI: tpm2-tcti-uefi
Device TCTI: tmp2-tss:tcti-device
UEFI
Kernel driver: /dev/tpm0
Kernel resource manager:/ dev/tpmrm0
Linux
TPM 2.0 hardware
Legend
TPM
Ring 0
Resource manager
TCTI
Library
.so
End user

7.1 Installing the TSS

The TSS is available from the GitHub web site (see [TSS]), with the following instructions:

```
# sudo apt install autoconf libtool autoconf-archive libjson-c-dev libcurl4-gnutls-dev
       doxygen
# cd /home/pi/Downloads
# git clone -b 3.1.0 https://github.com/tpm2-software/tpm2-tss(1)
```

There are three ways of using the TSS: using the in-kernel built-in resource manager, with the TAB/RM (TPM access broker and resource manager), or without a resource manager. The in-kernel RM is included in the Linux kernels 4.12 and greater, but the separate RM project is maintained for compatibility purposes

From a performance point of view, it is more interesting to bypass the resource manager. The RM adds latency to the end application by swapping resources in and out of TPM memory.

If any STMicroelectronics vendor commands are used, the RM must not be used, as the current implementation blocks all vendor commands.

If a resource manager is not used, volatile resources will need to be swapped in and out of TPM memory manually.

7.1.1 Adding SHA3 support to the TSS

To add SHA3 support to the TSS, modify the following files as described hereafter:

- `include/tss2/tss2_tpm2_types.h`
- `src/tss2-esys/esys_crypto.c`
- `src/tss2-esys/esys_crypto_ossl.c`
- `src/tss2-esys/fapi_crypto.c`
- `src/tss2-fapi/tpm_json_deserialize.c`
- `src/tss2-fapi/tpm_json_serialize.c`
- `src/tss2-mu/tpmu-types.c`

Table 10. include/tss2/tss2_tpm2_types.h file

After	Add

`#define TPM2_SHA512_DIGEST_SIZE    64`	`#define TPM2_SHA3_256_DIGEST_SIZE    32`
	`#define TPM2_SHA3_384_DIGEST_SIZE    48`

`BYTE sha512[TPM2_SHA512_DIGEST_SIZE];`	`BYTE sha3_256[TPM2_SHA3_256_DIGEST_SIZE];`
	`BYTE sha3_384[TPM2_SHA3_384_DIGEST_SIZE];`

Table 11. src/tss2-esys/esys_crypto.c file

After	Add

`case TPM2_ALG_SHA512:`
`    *size = TPM2_SHA512_DIGEST_SIZE;`
`    break;`
	`case TPM2_ALG_SHA3_256:`
	`    *size = TPM2_SHA3_256_DIGEST_SIZE;`
	`    break;`
	`case TPM2_ALG_SHA3_384:`
	`    *size = TPM2_SHA3_384_DIGEST_SIZE;`
	`    break;`

Table 12. src/tss2-esys/esys_crypto_ossl.c file

After	Add

`case TPM2_ALG_SHA512:`
`    return EVP_sha512();`
`    break;`
	`case TPM2_ALG_SHA3_256:`
	`    return EVP_sha3_256();`
	`    break;`
	`case TPM2_ALG_SHA3_384:`
	`    return EVP_sha3_384();`
	`    break;`

Table 13. src/tss2-esys/fapi_crypto.c file

After	Add
`case TPM2_ALG_SHA512:` `return EVP_sha512();`	`case TPM2_ALG_SHA3_256:` `return EVP_sha3_256();` `case TPM2_ALG_SHA3_384:` `return EVP_sha3_384();`
`case TPM2_ALG_SHA512:` `return TPM2_SHA512_DIGEST_SIZE;` `break;`	`case TPM2_ALG_SHA3_256:` `return TPM2_SHA3_256_DIGEST_SIZE;` `break;` `case TPM2_ALG_SHA3_384:` `return TPM2_SHA3_384_DIGEST_SIZE;` `break;`
`case TPM2_ALG_SHA512:` `return EVP_sha512();` `break;`	`case TPM2_ALG_SHA3_256:` `return EVP_sha3_256();` `break;` `case TPM2_ALG_SHA3_384:` `return EVP_sha3_384();` `break;`

Table 14. src/tss2-fapi/tpm_json_deserialize.c file

After	Add
`{ TPM2_ALG_CAMELLIA,   "CAMELLIA" },`	`{ TPM2_ALG_SHA3_256, "SHA3_256" },` `{ TPM2_ALG_SHA3_384, "SHA3_384" },`
`SUBTYPE_FILTER(TPMI_ALG_HASH,` `TPM2_ALG_ID, TPM2_ALG_SHA1,` `TPM2_ALG_SHA256, TPM2_ALG_SHA384,`	`TPM2_ALG_SHA3_256, TPM2_ALG_SHA3_384,`
`case TPM2_ALG_SHA512:` `hash_size = TPM2_SHA512_DIGEST_SIZE;` `buffer = &out->sha512[0];` `break;`	`case TPM2_ALG_SHA3_256:` `hash_size = TPM2_SHA3_256_DIGEST_SIZE;` `buffer = &out->sha3_256[0];` `break;` `case TPM2_ALG_SHA3_384:` `hash_size = TPM2_SHA3_384_DIGEST_SIZE;` `buffer = &out->sha3_384[0];` `break;`

Table 15. src/tss2-fapi/tpm_json_serialize.c file

After	Add
`{ TPM2_ALG_CAMELLIA, "CAMELLIA" },`	`{ TPM2_ALG_SHA3_256,   "SHA3_256" },` `{ TPM2_ALG_SHA3_384, "SHA3_384" },`
`CHECK_IN_LIST(TPMI_ALG_HASH, in,` `TPM2_ALG_SHA1, TPM2_ALG_SHA256,` `TPM2_ALG_SHA384, TPM2_ALG_SHA512,`	`TPM2_ALG_SHA3_256, TPM2_ALG_SHA3_384,`
`case TPM2_ALG_SHA512:` `size = TPM2_SHA512_DIGEST_SIZE;` `buffer = &in->sha512[0];` `break;`	`case TPM2_ALG_SHA3_256:` `size = TPM2_SHA3_256_DIGEST_SIZE;` `buffer = &in->sha3_256[0];` `break;` `case TPM2_ALG_SHA3_384:` `size = TPM2_SHA3_384_DIGEST_SIZE;` `buffer = &in->sha3_384[0];` `break;`

Table 16. src/tss2-mu/tpmu-types.c file

After	Add

static TSS2_RC marshal_hash_sha384(BYTE const *src, uint8_t buffer[],
size_t buffer_size, size_t *offset)
{
    return marshal_tab(src, buffer, buffer_size, offset, TPM2_SHA384_DIGEST_SIZE);
}
	static TSS2_RC marshal_hash_sha3_256(BYTE const *src,
	uint8_t buffer[], size_t buffer_size, size_t *offset)
	{
	    return marshal_tab(src, buffer, buffer_size, offset, TPM2_SHA3_256_DIGEST_SIZE);
	}
	
	static TSS2_RC marshal_hash_sha3_384(BYTE const *src, uint8_t buffer[], size_t buffer_size, size_t *offset)
	{
	    return marshal_tab(src, buffer, buffer_size, offset, TPM2_SHA3_384_DIGEST_SIZE);
	}

static TSS2_RC unmarshal_hash_sha384(uint8_t const buffer[],
size_t buffer_size, size_t *offset, BYTE *dest)
{
    return unmarshal_tab(buffer, buffer_size, offset, dest, TPM2_SHA384_DIGEST_SIZE);
}
	static TSS2_RC unmarshal_hash_sha3_256(uint8_t const buffer[],
	size_t buffer_size, size_t *offset, BYTE *dest)
	{
	    return unmarshal_tab(buffer, buffer_size, offset, dest, TPM2_SHA3_256_DIGEST_SIZE);
	}
	
	static TSS2_RC unmarshal_hash_sha3_384(uint8_t const buffer[], size_t buffer_size, size_t *offset, BYTE *dest)
	{
	    return unmarshal_tab(buffer, buffer_size, offset, dest, TPM2_SHA3_384_DIGEST_SIZE);
	}

TPMU_MARSHAL2(TPMU_HA, TPM2_ALG_SHA1, ADDR, sha1[0], marshal_hash_sha, TPM2_ALG_SHA256, ADDR, sha256[0], marshal_hash_sha256, TPM2_ALG_SHA384, ADDR, sha384[0], marshal_hash_sha384, TPM2_ALG_SHA512, ADDR, sha512[0], marshal_hash_sha512,	TPM2_ALG_SHA3_256, ADDR, sha3_256[0], marshal_hash_sha3_256, TPM2_ALG_SHA3_384, ADDR, sha3_384[0], marshal_hash_sha3_384,
TPMU_UNMARSHAL2(TPMU_HA, TPM2_ALG_SHA1, sha1[0], unmarshal_hash_sha, TPM2_ALG_SHA256, sha256[0], unmarshal_hash_sha256, TPM2_ALG_SHA384, sha384[0], unmarshal_hash_sha384, TPM2_ALG_SHA512, sha512[0], unmarshal_hash_sha512,	TPM2_ALG_SHA3_256, sha3_256[0], unmarshal_hash_sha3_256, TPM2_ALG_SHA3_384, sha3_384[0], unmarshal_hash_sha3_384,

7.1.2 Patching Fapi_Provision

The `Fapi_Provision` function must be patched to disable certificate verification, as the format of the certificate provisioned in the TPM device is not correctly recognized.

To do so, modify the `src/tss2-fapi/tpm_json_serialize.c` file as described below.

Table 17. src/tss2-fapi/tpm_json_serialize.c file

Replace	By

`statecase(context->state, PROVISION_INIT_GET_CAP2);`
`    if (context->config.ek_cert_less == TPM2_YES) {`
`        /* Skip certificate validation. */`
`        context->state = PROVISION_EK_WRITE_PREPARE;`
`        return TSS2_FAPI_RC_TRY_AGAIN;`
` }`
	`statecase(context->state, PROVISION_INIT_GET_CAP2);`
	`// if (context->config.ek_cert_less == TPM2_YES) {`
	`        /* Skip certificate   validation. */`
	`        context->state = PROVISION_EK_WRITE_PREPARE;`
	`        return TSS2_FAPI_RC_TRY_AGAIN;`
	`// }`

7.1.3 Compiling without any RM

For this option, the TPM driver needs to be writable without root privileges. To change the configuration, edit the `/etc/rc.local` file by using the following command:

```
# sudo nano /etc/rc.local
```

Add the following lines before “exit 0”:

```
./bin/chmod 766 /dev/tpm0
```

Reboot the Raspberry Pi device, then compile and install the TSS:

```
# cd /home/pi/Downloads/tpm2-tss
# ./bootstrap
# ./configure --with-device=/dev/tpm0
# make
# sudo make install
```

7.1.4 Compiling with the kernel RM

For this option, the TPM driver with the in-kernel RM needs to be writable without root privileges. To change the configuration, edit the `/etc/rc.local` file by using the following command:

```
# sudo nano /etc/rc.local
```

Add the following lines before “exit 0”:

```
./bin/chmod 766 /dev/tpmrm0
```

Reboot the Raspberry Pi device, then compile and install the TSS:

```
# cd /home/pi/Downloads/tpm2-tss
# ./bootstrap
# ./configure --with-device=/dev/tpmrm0
# make
# sudo make install
```

7.1.5 Compiling with the separate TAB/RM

Execute the following code:

```
# cd /home/pi/Downloads/tpm2-tss
# ./bootstrap
# ./configure
# make
# sudo make install
```

Download the resource manager (RM) and compile it:

```
# sudo apt install libglib2.0-dev
# cd /home/pi/Downloads
# git clone -b 2.3.3 https://github.com/tpm2-software/tpm2-abrmd(1)
# cd tpm2-abrmd
# ./bootstrap
# ./configure --with-dbuspolicydir=/etc/dbus-1/system.d
# make
# sudo make install
# sudo ldconfig
```

Before using the TSS or any other associated tools, write the following command in a terminal to start the resource manager:

```
# sudo tpm2-abrmd --allow-root --dbus-name=com.intel.tss2.Tabrmd
```

Warning: *This command is mandatory only with the separate TAB/RM. It must NOT be sent if the in-kernel RM is used.*

The TSS compilation generates libraries that can be used in C language programs. These libraries are located in the `/usr/local/lib` folder, and are used by the tools presented in the next sections.

8 TPM2.0 tools

The TSS is useful for sending complex commands to the TPM device, however, using it still requires a certain level of knowledge of the TPM device. The [TPM2.0 tools] are designed to allow very simple and intuitive use of the TPM device with little TPM knowledge.

The [TPM2.0 tools] are a set of executable files. Each executable file corresponds to a TPM command. Sending the command to the TPM device is simply a case of executing the file, possibly with some command line parameters.

To download the necessary packages and the [TPM2.0 tools], execute the following:

```
# sudo apt install pandoc uuid-dev
# cd /home/pi/Downloads
# git clone -b 5.0 https://github.com/tpm2-software/tpm2-tools(1)
# cd tpm2-tools
```

8.1 Adding SHA3 support to the TPM2 tools

To add SHA3 support to the TPM2 tools, modify the following files before compilation as described hereafter:

- `lib/tpm2_alg_util.c`
- `lib/tpm2_eventlog.c`
- `lib/tpm2_eventlog.h`
- `lib/tpm2_eventlog_yaml.c`
- `lib/tpm2_openssl.c`
- `scripts/utils/man_to_bashcompletion.sh`
- `tools/misc/tpm2_checkquote.c`
- `tools/tpm2_pcrevent.c`

Table 18. lib/tpm2_alg_util.c file

After	Add

`case TPM2_ALG_SHA512:`
`    return TPM2_SHA512_DIGEST_SIZE;`
	`case TPM2_ALG_SHA3_256:`
	`    return TPM2_SHA3_256_DIGEST_SIZE;`
	`case TPM2_ALG_SHA3_384:`
	`    return TPM2_SHA3_384_DIGEST_SIZE;`

Table 19. lib/tpm2_eventlog.c file

After	Add

`} else if (alg == TPM2_ALG_SHA512) {`
`    pcr = ctx->sha512_pcrs[pcr_index];`
`    ctx->sha512_used |= (1 << pcr_index);`
	`} else if (alg == TPM2_ALG_SHA3_256) {`
	`    pcr = ctx->sha3_256_pcrs[pcr_index];`
	`    ctx->sha3_256_used |= (1 << pcr_index);`
	`} else if (alg == TPM2_ALG_SHA3_384) {`
	`    pcr = ctx->sha3_384_pcrs[pcr_index];`
	`    ctx->sha3_384_used |= (1 << pcr_index);`

Table 20. lib/tpm2_eventlog.h file

After	Add

`uint32_t sha512_used;`	`uint32_t sha3_256_used;`
	`uint32_t sha3_384_used;`

`uint8_t sha512_pcrs[TPM2_MAX_PCRS]`
`[TPM2_SHA512_DIGEST_SIZE];`
	`uint8_t sha3_256_pcrs[TPM2_MAX_PCRS]`
	`[TPM2_SHA3_256_DIGEST_SIZE];`
	`uint8_t sha3_384_pcrs[TPM2_MAX_PCRS]`
	`[TPM2_SHA3_384_DIGEST_SIZE];`

Table 21. lib/tpm2_eventlog_yaml.c file

After	Add

`if (ctx->sha512_used   != 0) {`
`    tpm2_tool_output(" sha512:\n");`
`    for(unsigned i = 0 ; i < TPM2_MAX_PCRS ; i++) {`
`        if ((ctx->sha512_used & (1 << i)) == 0)`
`            continue;`
`        bytes_to_str(ctx->sha512_pcrs[i], sizeof(ctx->sha512_pcrs[i]), hexstr, sizeof(hexstr));`
`        tpm2_tool_output(" %-2d : 0x%s\n", i, hexstr);`
`    }`
`}`
	`if (ctx->sha3_256_used != 0) {`
	`    tpm2_tool_output(" sha3_256:\n");`
	`    for(unsigned i = 0 ; i < TPM2_MAX_PCRS ; i++) {`
	`        if ((ctx->sha3_256_used & (1 << i)) == 0)`
	`            continue;`
	`        bytes_to_str(ctx->sha3_256_pcrs[i], sizeof(ctx->sha3_256_pcrs[i]), hexstr, sizeof(hexstr));`
	`        tpm2_tool_output(" %-2d : 0x%s\n", i, hexstr);`
	`    }`
	`}`
	
	`if (ctx->sha3_384_used != 0) {`
	`    tpm2_tool_output(" sha3_384:\n");`
	`    for(unsigned i = 0 ; i < TPM2_MAX_PCRS ; i++) {`
	`        if ((ctx->sha3_384_used & (1 << i)) == 0)`
	`            continue;`
	`        bytes_to_str(ctx->sha3_384_pcrs[i], sizeof(ctx->sha3_384_pcrs[i]), hexstr, sizeof(hexstr));`
	`        tpm2_tool_output(" %-2d : 0x%s\n", i, hexstr);`
	`    }`
	`}`

Table 22. lib/tpm2_openssl.c file

After	Add

`case TPM2_ALG_SHA512:`
`    return NID_sha512;`
	`case TPM2_ALG_SHA3_256:`
	`    return NID_sha3_256;`
	`case TPM2_ALG_SHA3_384:`
	`    return NID_sha3_384;`

`case TPM2_ALG_SHA512:`
`    return EVP_sha512();`
	`case TPM2_ALG_SHA3_256:`
	`    return EVP_sha3_256();`
	`case TPM2_ALG_SHA3_384:`
	`    return EVP_sha3_384();`

`case TPM2_ALG_SHA512:`
`    return SHA512;`
	`case TPM2_ALG_SHA3_256:`
	`    return SHA256; // workaround`
	`case TPM2_ALG_SHA3_384:`
	`    return SHA384; // workaround`

Table 23. scripts/utils/man_to_bashcompletion.sh file

After	Add
`local hash_methods=(sha1 sha256 sha384 sha512`	`sha3_256 sha3_384`

Table 24. tools/misc/tpm2_checkquote.c file

After	Add

`} else if (sel->hash == TPM2_ALG_SHA512 && pcr->size == TPM2_SHA512_DIGEST_SIZE) {`
`    pcr_e = eventlog_ctx.sha512_pcrs[pcr_id];`
	`} else if (sel->hash == TPM2_ALG_SHA3_256 && pcr->size == TPM2_SHA3_256_DIGEST_SIZE) {`
	`    pcr_e = eventlog_ctx.sha3_256_pcrs[pcr_id];`
	`} else if (sel->hash == TPM2_ALG_SHA3_384 && pcr->size == TPM2_SHA3_384_DIGEST_SIZE) {`
	`    pcr_e = eventlog_ctx.sha3_384_pcrs[pcr_id];`

Table 25. tools/tpm2_pcrevent.c file

After	Add

`case TPM2_ALG_SHA512:`
`    bytes = d->digest.sha512;`
`    size = sizeof(d->digest.sha512);`
`    break;`
	`case TPM2_ALG_SHA3_256:`
	`    bytes = d->digest.sha3_256;`
	`    size = sizeof(d->digest.sha3_256);`
	`    break;`
	`case TPM2_ALG_SHA3_384:`
	`    bytes = d->digest.sha3_384;`
	`    size = sizeof(d->digest.sha3_384);`
	`    break;`

8.2 Changing the default PCR allocation

If the default banks of the TPM device being used are SHA256 and SHA284 PCR, modify the `tools/misc/tpm2_pcrevent.c` file as described in the table below.

Table 26. tools/misc/tpm2_pcrevent.c file modification

Replace	By

`.pcrSelections = { {`
`    .hash = TPM2_ALG_SHA1,`
`    .sizeofSelect = 3,`
	`.pcrSelections = { {`
	`    .hash = TPM2_ALG_SHA256,`
	`    .sizeofSelect = 3,`

`}, {`
`    .hash = TPM2_ALG_SHA256,`
`    .sizeofSelect = 3,`
	`}, {`
	`    .hash = TPM2_ALG_SHA384,`
	`    .sizeofSelect = 3,`

8.3 Installing the TPM2.0 tools

To compile and install the tools:

```
# ./bootstrap
# ./configure
# make
# sudo make install
# sudo ldconfig
```

Once the [TPM2.0 tools] have been installed, they can be used immediately. To display the list of the tools with the Linux autocompletion function, type `tpm2` in a Linux terminal, then press the **TAB** key twice.

Note: If the TSS was compiled with the separate resource manager (RM), then the RM has to be running when the tools are executed. Start the RM with the following command:

```
# sudo tpm2-abrmd --allow-root --dbus-name=com.intel.tss2.Tabrmd
```

There are almost 100 tools. A simple example to test if the setup has been correctly done is to send a `TPM2_GetRandom` command, used to generate random bytes:

```
# tpm2 getrandom --hex 20
```

The command returns 20 random bytes to the Linux terminal.

8.4 Example use cases

This section presents a few practical use cases for the [TPM2.0 tools]. The commands in the following sections must all be run from the same directory, for instance /home/pi or /home/pi/Documents. Any input or output files for these examples must also be stored in the directory. Use the `cd.` command in a terminal to navigate to the chosen directory.

8.4.1 RSA encryption and decryption

The goal of this use case is to create an RSA key pair that is stored to the TPM device. The RSA key pair is then used to encrypt and decrypt a data file. The keys use the owner hierarchy.

First of all, create a primary key, to serve as a parent to the RSA key. With the primary key created, use the print command to show information about the created key, as shown below.

```
# tpm2 createprimary -G rsa -C o -c primarykey
# tpm2 print -t TPMS_CONTEXT primarykey
```

Then, create the child RSA key pair. Its parent is *primarykey*. The child key is automatically loaded into the TPM device and the resulting context is stored in *loadedkey*. The public and private keys are exported: the public part (*key.pub*) is in clear, but the private part (*key.priv*) is encrypted with the parent key so that it remains protected.

```
# tpm2 create -G rsa -C primarykey -u key.pub -r key.priv -c loadedkey
# tpm2 print -t TPMS_CONTEXT loadedkey
# tpm2 print -t TPM2B_PUBLIC key.pub
```

Create a data file named *mydata.txt* in the current directory such as /home/pi for example. This file can contain any text. In a real-world use case, this could be a file containing user passwords. For instance:

```
(mydata.txt)
Hello world!
```

Then, encrypt this file with the RSA key. The encrypted data is stored in a file named `encrypted_data.txt`.

```
# tpm2 rsaencrypt -c loadedkey -o encrypted_data.txt mydata.txt
```

Reading `encrypted_data.txt` produces a series of cryptic symbols. This is the encrypted message.

Use the RSA key to decrypt the encrypted data:

```
# tpm2 rsadecrypt -c loadedkey -o decrypted_data.txt encrypted_data.txt
```

This creates the `decrypted_data.txt` file that contains the original data.

At the end of the use case, flush the keys from the TPM memory.

```
# tpm2 flushcontext -t
```

By default, the keys are deleted at each reboot. The `TPM2_EvictControl` command allows a key to be made persistent:

```
# tpm2 evictcontrol -C o -c loadedkey
```

And conversely, it allows a persistent object to be made no longer persistent. The persistent handle of the object must be passed to the evictcontrol utility, as illustrated by {persistent-object-handle} here.

```
# tpm2 evictcontrol -C o -c {persistent-object-handle}
```

8.4.2 NV indexes

An NV index can be used to persistently store data in TPM memory. It can be configured to restrict read and write access to authorized entities. In this use case, the goal is to create a symmetric AES key, and store it to an NV index protected by a password.

First, create the AES key using *OpenSSL®*:

```
# openssl enc -nosalt -aes-256-cbc -pbkdf2 -k mykeypass -P > aes_key.txt
```

Read the created `aes_key.txt` file. It contains two values: the AES key further referred to as `{aes_key}` and the IV, hereafter referred to as `{iv}`.

Then, encrypt a file with the symmetric key:

```
# openssl enc -nosalt -aes-256-cbc -in mydata.txt -out aes_encrypted_data.txt -K {aes_key}
-iv {iv}
```

The resulting `aes_encrypted_data.txt` file contains the encrypted bytes.

However, the AES key is not stored in a secure place yet. Prior to storing it to a TPM NV index, first define the NV index attributes:

- NV index handle is 0x1500015.
- NV index attributes bit field is 0x40004.

This means that the NV index can only be written and read if the correct `auth` value is provided (in this example, the `myNVpassword` password).

```
# tpm2 nvdefine -C o -p myNVpassword -a 0x40004 0x1500015
```

After defining the NV index attributes, write the AES key data to the NV index:

```
# tpm2 nvwrite -i aes_key.txt -P myNVpassword 0x1500015
```

The key is now stored in the NV index. The `aes_key.txt` file is no longer needed and can be deleted. To release the AES key context, provide the correct authorization for the NV index. If a wrong password is provided, the key is not released:

```
# tpm2 nvread -o aes_unlocked -P wrongpassword 0x1500015
```

However, with the correct password, the key context is released:

```
# tpm2 nvread -o aes_unlocked -P mypassword 0x1500015
```

Now use the AES key to decrypt the encrypted data:

```
# openssl enc -nosalt -aes-256-cbc -d -in aes_encrypted_data.txt -out aes_decrypted_data.txt
-K {aes_key} -iv {iv}
```

The `aes_decrypted_data.txt` file is the same as the original `mydata.txt` file.

To clean up after this use case, delete the NV index:

```
# tpm2 nvundefine -C o 0x1500015
```

This was a simple illustration, but in a real-world scenario, the NV index can have a more complex authorization value: it can for instance be bound to the TPM platform configuration registers (PCRs), so that the AES key is only released if the TPM device is in the expected state.

Note: OpenSSL is a registered trademark owned by the OpenSSL Software Foundation.

8.4.3 Extending a PCR

The TPM platform configuration registers (PCRs) are 2 banks of hash values (by default SHA-256 and SHA-1, or SHA-256 and SHA-384, depending on the firmware version of the TPM device). They cannot be directly written, but they can be extended with another hash value. The PCRs can be used to provide a “measure” of a file or a system.

The objective of this use case is to extend the fifth SHA-256 PCR with the *mydata.txt* file, which was created in the RSA encryption and decryption use case.

Start by reading the TPM PCRs with the following command:

```
# tpm2 pcrread
```

Now, hash the data:

```
# tpm2 hash --hex -g sha256 mydata.txt
```

The `TPM2_Hash` command returns the digest that must be used to extend the PCR.

```
# tpm2 pcrextend 5:sha256=[returned hash in hex form]
```

Note: To avoid copying and pasting the returned hash, use both tools in a single Linux terminal command:

```
# tpm2 pcrextend 5:sha256=`tpm2 hash --hex -g sha256 mydata.txt`
```

When the TPM PCRs are read once again, the value of the fifth SHA-256 PCR must have changed.

```
# tpm2 pcrread
```

The `TPM2_PCR_Event` command is the equivalent of a combined `TPM2_Hash` and `TPM2_PCR_Extend` operation, which means that the data file can be used directly with this command. By executing the command line below, the sixth PCRs of both banks are extend with the `mydata.txt` file:

```
# tpm2 pcrevent mydata.txt 6
```

Check the PCR values once more with the `TPM2_PCR_Read` command:

```
# tpm2 pcrread
```

Note: The fifth and sixth PCRs of the SHA-256 bank have the same values. This is logical, because even though the process was slightly different, they are extended with the same data.

9 Higher-level applications

In the [TPM2-TSS] repository, some extra projects exist for specific use cases. This section presents two of these projects.

9.1 OpenSSL engine

The *OpenSSL*® engine project is accessible from the GitHub website (see [OpenSSL-engine]). OpenSSL is often used for cryptographic operations: for instance, to generate cryptographic keys. However, generating keys requires generating strong random numbers, and OpenSSL is purely software. Software implementations sometimes have weaker random number generation algorithms than {hardware + software} implementations.

Therefore, a functionality exists in OpenSSL to allow the use of a hardware device as a cryptographic engine. If an engine is specified, then the cryptographic operations are computed by the hardware device instead of the OpenSSL software.

The TPM2 OpenSSL engine project offers the possibility of using the TPM device as a cryptographic engine.

9.1.1 Installing the TPM2 OpenSSL engine project

Use the following commands to install the TPM2 OpenSSL engine project:

```
# sudo apt install expect
# cd /home/pi/Downloads
# git clone -b v1.1.0 https://github.com/tpm2-software/tpm2-tss-engine(1)
# cd tpm2-tss-engine
# ./bootstrap
# ./configure
# make
# sudo make install
# sudo ldconfig
```

9.1.2 RSA encryption and decryption with the TPM2 OpenSSL engine project

The objective of this use case is to encrypt and decrypt a data file with an RSA key pair.

Step 1. Create the RSA key pair using the TPM device.

```
# cd /home/pi/
# tpm2tss-genkey -a rsa -s 2048 rsa_key
```

The `rsa_key` file contains the private key.

Step 2. Retrieve the public portion of the key:

```
# openssl rsa -engine tpm2tss -inform engine -in rsa_key -pubout -outform pem -out
rsa_key.pub
```

The `rsa_key.pub` file contains the public portion of the key.

Step 3. Encrypt a data file using the public key. The data file can be `mydata.txt`, created earlier on.

```
# openssl pkeyutl -pubin -inkey rsa_key.pub -in mydata.txt -encrypt -out
openssl_encrypted_data.txt
```

Step 4. Decrypt the encrypted data using the private portion of the RSA key:

```
# openssl pkeyutl -engine tpm2tss -keyform engine -inkey rsa_key -decrypt -in
openssl_encrypted_data.txt -out openssl_decrypted_data.txt
```

9.1.3 Generating a self-signed x509 certificate

The objective of this short use case is to generate a self-signed certificate for an RSA key. This means that the certificate is not signed by a certificate authority (CA) but directly by the issuer of the RSA key.

Step 1. Create an RSA key:

```
# tpm2tss-genkey -a rsa -s 2048 rsa_key2
```

Step 2. Use OpenSSL to create a self-signed certificate.

```
# openssl req -new -x509 -engine tpm2tss -key rsa_key2 -keyform engine -out
self_signed_certificate.txt
```

Step 3. Provide the information required to create the certificate.

The certificate can be read in the `self_signed_certificate.txt` file.

9.1.4 Signing data with a private key stored inside the TPM and verifying the signature with OpenSSL

The objective of this use case is to generate an elliptic cryptographic curve (ECC) key with OpenSSL, and to load it into the TPM device with the `TPM2_LoadExternal` command; then, to hash a data file and to sign the resulting digest using the ECC key; and finally, to use OpenSSL to verify the signed data.

Step 1. Generate the ECC key, and retrieve its public portion from the `ecc.pub` file:

```
# openssl ecparam -name prime256v1 -genkey -noout -out ecc.priv
# openssl ec -in ecc.priv -pubout -out ecc.pub
```

Step 2. Load this key into the TPM device.

```
# tpm2 loadexternal -G ecc -r ecc.priv -c loaded_ecc_key
```

Step 3. Hash the data before signature.

```
# tpm2 hash -g sha256 -o my_hashed_data.txt mydata.txt
```

Step 4. Sign the hashed data using the ECC key.

Signing a hash means encrypting the hash with a private key.

```
# tpm2 sign -c loaded_ecc_key -g sha256 -d my_hashed_data.txt -f plain -o
signed_data.txt
```

Step 5. Use OpenSSL to verify the signature.

This means that, on one side, the `mydata.txt` file is hashed while, on the other, the signature is decrypted using the public key. If these two digests match, the signature is valid.

```
# openssl dgst -verify ecc.pub -keyform pem -sha256 -signature signed_data.txt
mydata.txt
Verified OK
```

If a different data file is verified (for instance, a `mydata2.txt` file, different from `mydata.txt`), then the verification fails.

```
# openssl dgst -verify ecc.pub -keyform pem -sha256 -signature signed_data.txt
mydata2.txt
Verification failure
```

9.2 PKCS#11

The PKCS#11 (Public-Key Cryptography Standard) is a standard used in communicating with hardware security devices (HSMs) and smartcards. These secure devices are used as cryptographic tokens: they can store biometric data, passwords or keys (used in signatures) that are used in authentication.

In the [PKCS#11] project, the TPM device is used as the cryptographic token.

9.2.1 Installing PKCS#11

To install the TPM PKCS#11 project, execute the following commands:

```
# sudo apt install libsqlite3-dev libyaml-dev
# sudo pip3 install pyyaml
# sudo pip3 install pyasn1-modules
# cd /home/pi/Downloads
# git clone -b 1.6.0 https://github.com/tpm2-software/tpm2-pkcs11(1)
# cd tpm2-pkcs11
# ./bootstrap
# sudo mkdir /etc/tpm2-pkcs11
# sudo chmod 777 /etc/tpm2-pkcs11
# ./configure --with-storedir='/etc/tpm2-pkcs11'
# make
# sudo make install
# sudo ldconfig
```

Then, set the `TPM2_PKCS11_STORE` environment variable; the application uses it to find the location of the key store.

```
# export TPM2_PKCS11_STORE=/etc/tpm2-pkcs11
```

Integrate this last command into the `.bashrc` file, so that it is executed each time a new terminal is opened. Write it at the end of the file that it is opened by the following command:

```
# nano /home/pi/.bashrc
```

9.2.2 Generating a key accessible by the PKCS#11 standard

The objective of this example is three-fold:

1. To create a key using the `tpm2-pkcs11` tool.
2. To illustrate how to access the key via the PKCS#11 standard.
3. To show how to generate a certificate signing request (CSR) for the key.

If the resource manager is used, it must be started before the commands are sent.

Use the following sequence:

Step 1. Initialize the key store:

```
# cd /home/pi/Downloads/tpm2-pkcs11/tools
# ./tpm2_ptool init
```

The above command returns the ID of the store. In the following commands, the `pid` argument is 1. If the returned ID is different, that value must be used.

Step 2. Add a token to the store.

```
# ./tpm2_ptool addtoken --pid=1 --sopin=mysopin --userpin=myuserpin --label=mylabel
```

Step 3. Add an RSA key:

```
# ./tpm2_ptool addkey --algorithm=rsa2048 --label=mylabel --userpin=myuserpin
```

Step 4. In the PKCS#11 standard, a key is accessed via its URL. First, use the P11Tool program to obtain the URL of the token:

```
# sudo apt install gnutls-bin
# p11tool --list-token-urls
```

The command requires the user pin: enter *myuserpin*.

The command returns a list of token URLs. The URL corresponding to the previously created token has the `label=mylabel` attribute. The URL must look like this:

```
pkcs11:model=;manufacturer=STMicro;serial=0000000000000000;token=mylabel
```

Step 5. Use the token URL to obtain the key URL with the following command:

```
p11tool --list-
all ”pkcs11:model=;manufacturer=STMicro;serial=0000000000000000;token=mylabel”
```

The key URL printed in the terminal must look like this:

```
pkcs11:model=%01%01%04;manufacturer=STMicro;serial=0000000000000000;token=
mylabel;id=%xx%xx%xx%xx=%xx%xx%xx%xx=%xx%xx%xx%xx=%xx%xx%xx%xx;type=public
```

Step 6. Use the key URL to generate a CSR (certificate signing request) using OpenSSL.

```
# sudo apt install libengine-pkcs11-openssl
# openssl req -new -engine pkcs11 -keyform engine -key
"pkcs11:model=%01%01%04;manufacturer=STMicro;serial=0000000000000000;
token=mylabel;id=%xx%xx%xx%xx%xx%xx%xx%xx%xx%xx%xx%xx%xx%xx%xx%xx;type=public" -out
mycsr.csr
```

The user pin must be entered once again, as well as some information used in the CSR.

Step 7. Sign the resulting CSR, `mycsr.csr`, with a CA key.

The resulting certificate can be used to authenticate the TPM device, via the cryptographic token, for wireless connections.

Appendix A Glossary

Table 27. List of acronyms

Acronym	Meaning
AES	Advanced encryption standard
B	Byte
CA	Certificate authority
CC	Common criteria
CSR	Certificate signing request
ECC	Elliptic cryptographic curve
HDMI	High-definition multimedia interface
HSM	Hardware security device
I²C	Inter-integrated circuit
NV	Nonvolatile
OpenSSL®	Full-featured toolkit for the transport layer security (TLS) and secure sockets layer (SSL) protocols.
OS	Operating system
PCR	Platform configuration register
PKCS	Public-key cryptography standard
RM	Resource manager
RSA	Public-key signature algorithm (Ron Rivest, Adi Shamir and Leonard Adleman)
SD card	Secure digital card
SHA	Secure hash algorithm
SPI	Serial peripheral interface
TAB/RM	TPM access broker and resource manager
TPM	Trusted platform module
TSS	TPM software stack
URL	Secure hash algorithm
USB	Universal serial bus

Revision history

Table 28. **Document revision history**

Date	Revision	Changes
16-Sep-2021	1	Initial release.
06-Apr-2022	2	Added: • Table 3. Compliant open-source references • Figure 6. TPM software stack illustration • Section 7.1.4 Compiling with the kernel RM Updated: • Section Introduction • Table 2. List of open-source software resources • Section 2 Raspberry Pi • Section 3 STPM4RasPI connector board • Section 4.1 Raspberry Pi OS • Section 4.2 Hardware setup • Section 4.2 Hardware setup • Section 5.1 SPI/I2C activation • Section 5.2 Downloading the Linux kernel • Section 5.3.1 Drivers • Section 5.3.2 Device tree • Section 5.3.3 Linux kernel configuration • Section 5.3.4 Compiling and installing the kernel • Section 6 Low-level communication • Section 6.1 Linux terminal I^2C commands • Section 6.2 Python scripts • Section 6.3 C language scripts • Section 7 TPM software stack (TSS) • Section 7.1 Installing the TSS • Section 7.1 Installing the TSS • Section 7.1.5 Compiling with the separate TAB/RM • Section 8 TPM2.0 tools • Section 8.3 Installing the TPM2.0 tools • Section 8.4 Example use cases • Section 8.4.1 RSA encryption and decryption • Section 8.4.2 NV indexes • Section 8.4.3 Extending a PCR • Section 9.1.4 Signing data with a private key stored inside the TPM and verifying the signature with OpenSSL • Section 9.2.1 Installing PKCS#11 • Section 9.2.2 Generating a key accessible by the PKCS#11 standard

Contents

List of tables

List of figures