

OPTIGA™ TPM User Guide

OPTIGA™ TPM SLx 967y TPM2.0 on Raspberry Pi QuickStarter User Guide

Devices

- OPTIGA™ TPM SLB 9672 TPM2.0 FW 15.xx
- OPTIGA™ TPM SLB 9672 TPM2.0 FW 16.xx
- OPTIGA™ TPM SLB 9673 TPM2.0
- OPTIGA™ TPM SLB 9670 TPM2.0
- OPTIGA™ TPM SLI 9670 TPM2.0
- OPTIGA™ TPM SLM 9670 TPM2.0

About this document

Scope and purpose

This document explains how to use an OPTIGA™ TPM SLx 967y TPM2.0 with a Raspberry Pi. The same setup works on Raspberry Pi 5, 4, and 3 systems that we have tested. Note, I²C-based setups on the older Raspberry Pis use software I²C due to clock-stretching bugs. Unless noted otherwise, this document refers to Raspberry Pi 5.

The Linux® kernel version used in this document is 6.12, which is the latest version in Raspberry Pi OS at the time of writing. The kernel includes TPM 2.0 driver support, which simplifies TPM setup on Raspberry Pi platforms. This application note shows how to set up TPM 2.0 with basic Linux knowledge.

Intended audience

This document targets customers who want to increase the security of embedded platforms with TPM 2.0 and evaluate TPM software integration for their target applications.

Evaluation boards

You can follow the integration steps with one of the Infineon evaluation boards listed in the following table.

Sales Name	Orderable Part Number (OPN)	Sales Product (SP)
OPTIGA™ TPM SLB 9672 FW 15.25 RPI EB	TPM9672FW1625RPIEBTOBO1	SP006285514
OPTIGA™ TPM SLB 9673 FW 26.25 RPI EB	TPM9673FW2625RPIEBTOBO1	SP006285518

The following sections refer to these boards as the Infineon SLx 967y TPM2.0 evaluation boards. All boards with OPTIGA™ TPM SLB 9672 or OPTIGA™ TPM SLB 9673 are compatible with the setup in this guide. The same integration steps also work with OPTIGA™ TPM SLB 9670, OPTIGA™ TPM SLI 9670, and OPTIGA™ TPM SLM 9670 devices.

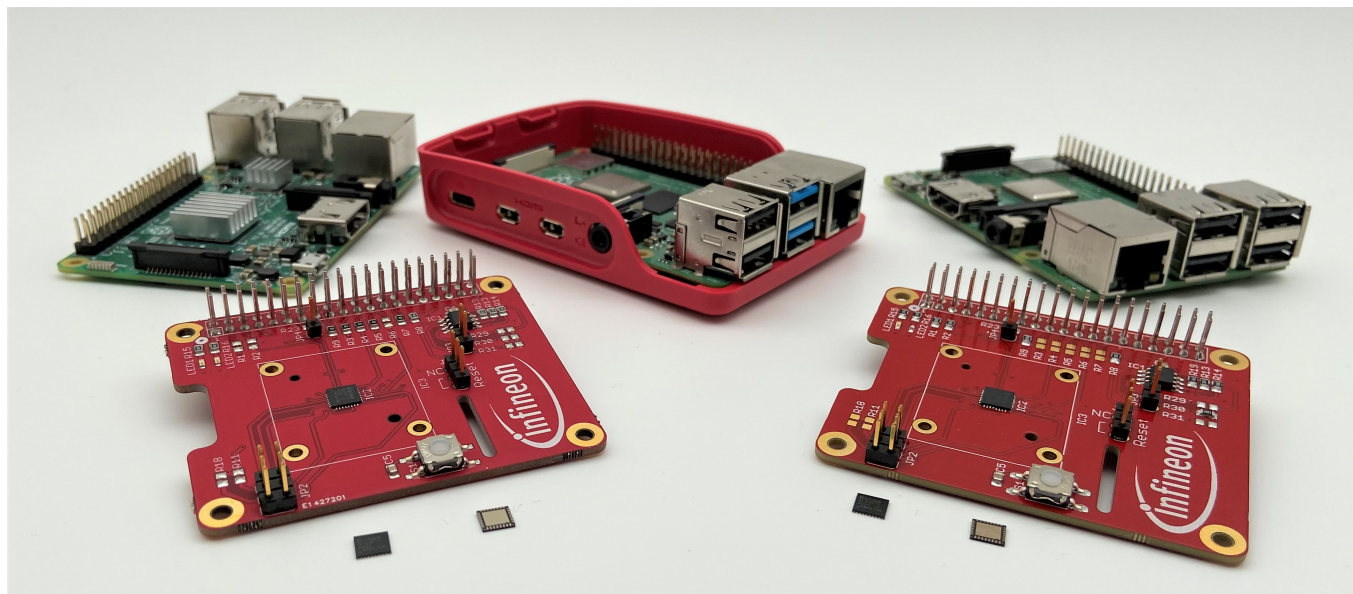
TABLE OF CONTENTS

Table of contents

1	Introduction	3
1.1	Motivation	4
1.2	Scope	4
1.3	Acronyms and abbreviations	4
2	Hardware setup	5
2.1	Infineon OPTIGA™ TPM SLB 9672 RPI Evaluation Boards	5
2.2	Infineon OPTIGA™ TPM SLB 9673 RPI Evaluation Boards	6
3	OS and driver setup	7
3.1	Raspberry Pi OS	7
4	Software setup	9
4.1	Install the Open Source TPM software stack tpm2-tss, TPM2 tools	9
5	Using the TPM with tpm2-tss and tpm2-tools	10
5.1	Software stack overview	10
5.2	Verifying TSS2 functionality	10
5.3	Using tpm2-tools	10
6	Using TPM with OpenSSL	17
6.1	Installing the software	17
6.2	Interfacing with OpenSSL via command line	17
6.3	Interfacing with OpenSSL from NGINX® web server	19
6.4	Interfacing with OpenSSL from curl web client	22
7	Using TPM with PKCS11	25
7.1	Installation and setup of the tpm2-pkcs11 module	25
7.2	Configuring a TPM token with pkcs11-tool	25
8	Using TPM for HDD encryption	29
9	Appendix	33
9.1	Troubleshooting	33
9.2	Support on other Linux distributions	36
9.3	Compiling Linux with TPM2 driver (for Raspberry Pi OS)	40
9.4	Compiling the TSS from source	43
9.5	Accessing the TPM with eltt2	44
9.6	Working with the Raspberry Pi	44

Introduction

1 Introduction



The OPTIGA™ TPM SLx 967y TPM2.0 communicates with the Raspberry Pi through SPI or I²C. The product family includes the following six products:

- OPTIGA™ TPM SLB 9672 TPM2.0 FW 15.xx standard security applications (SPI)
- OPTIGA™ TPM SLB 9672 TPM2.0 FW 16.xx embedded security applications (SPI)
- OPTIGA™ TPM SLB 9673 TPM2.0 embedded security applications (I²C)
- OPTIGA™ TPM SLB 9670 TPM2.0 standard security applications (SPI)
- OPTIGA™ TPM SLI 9670 TPM2.0 automotive qualified security applications (SPI)
- OPTIGA™ TPM SLM 9670 TPM2.0 industrial qualified security applications (SPI)

In this document, “OPTIGA™ TPM SLx 967y TPM2.0” refers to all six members of the OPTIGA™ TPM 2.0 product family.

OPTIGA™ TPM SLx 967y TPM2.0 products are TCG-compliant TPMs with CC (EAL4+) and FIPS certification. The standard, automotive, and industrial variants differ in supported temperature range, lifetime, quality grades, test environment, qualification, and reliability to match different application requirements. The embedded security variants provide additional functionality. OPTIGA™ TPM SLB 9672 and 9673 devices are also available for different operating temperature ranges. For more details, see [Infineon's website](#)¹.

Because all OPTIGA™ TPM SLx 967y TPM2.0 products use TCG-compliant SPI or I²C interfaces and software interfaces, the integration steps in this Raspberry Pi Linux environment are the same for all variants. This document also refers to the OPTIGA™ TPM SLx 967y TPM2.0 as the “TPM”. The [Trusted Computing Group \(TCG\) TPM specifications](#)² and [TCG PC Client Platform TPM Profile \(PTP\) specification](#)³ provide more background on TPM integration.

¹ <https://www.infineon.com/tpm>

² <https://trustedcomputinggroup.org/resource/tpm-library-specification/>

³ <https://trustedcomputinggroup.org/resource/pc-client-platform-tpm-profile-ptp-specification/>

Introduction

1.1 Motivation

Two of the basic principles of information security when exchanging data are confidentiality and authenticity. While confidentiality is widely understood, authenticity often does not receive the appropriate amount of attention. Authenticity is just as important as confidentiality. The highest confidentiality of sensitive data is ineffective if the data come from or go to the wrong communication partner.

A TPM adds security features to a platform that support both confidentiality and authenticity for data and communication partners.

One of the most important TPM features is internal storage for sensitive data such as cryptographic keys and secrets. The TPM hardware protects this data from unauthorized access and manipulation, which increases confidentiality. Users can also bind data to one TPM and therefore to one platform. This adds an authenticity factor because only the platform or user with the correct TPM can provide the secret required to access the protected data.

Another TPM feature is its role as a starting point for a root of trust, which helps detect unauthorized changes to platform hardware or software.

The TPM also performs cryptographic operations in hardware, which reduces exposure to attacks such as reading unencrypted sensitive data from platform memory.

1.2 Scope

This document introduces a subset of TPM functionality on Linux-based embedded platforms. The goal is to interact with TPM 2.0 and run the test scripts provided by the [tpm2-software GitHub community](https://github.com/tpm2-software)⁴.

This document covers the following topics:

- The evaluation board and its expansion header for [the Raspberry Pi](https://www.raspberrypi.org/)⁵
- TPM 2.0 usage on the Raspberry Pi

This document was generated with Raspberry Pi OS “trixie” and Linux kernel version 6.12.75+rpt-rpi-2712.

This document uses Raspberry Pi OS, a Raspberry Pi Linux distribution based on *Debian*, to describe system setup and TPM 2.0 usage. Unless noted otherwise, the document refers to specific package versions to avoid compatibility problems. The versions referenced here were publicly available from the listed links and repositories at the time of release.

1.3 Acronyms and abbreviations

Acronym	Explanation
I ² C	Inter-Integrated Circuit
OS	Operating System
PC	Personal Computer
SPI	Serial Peripheral Interface
SSH	Secure Shell
TCG	Trusted Computing Group
TPM	Trusted Platform Module
TSS	TPM Software Stack

⁴ <https://github.com/tpm2-software>

⁵ <https://www.raspberrypi.org/>

Hardware setup

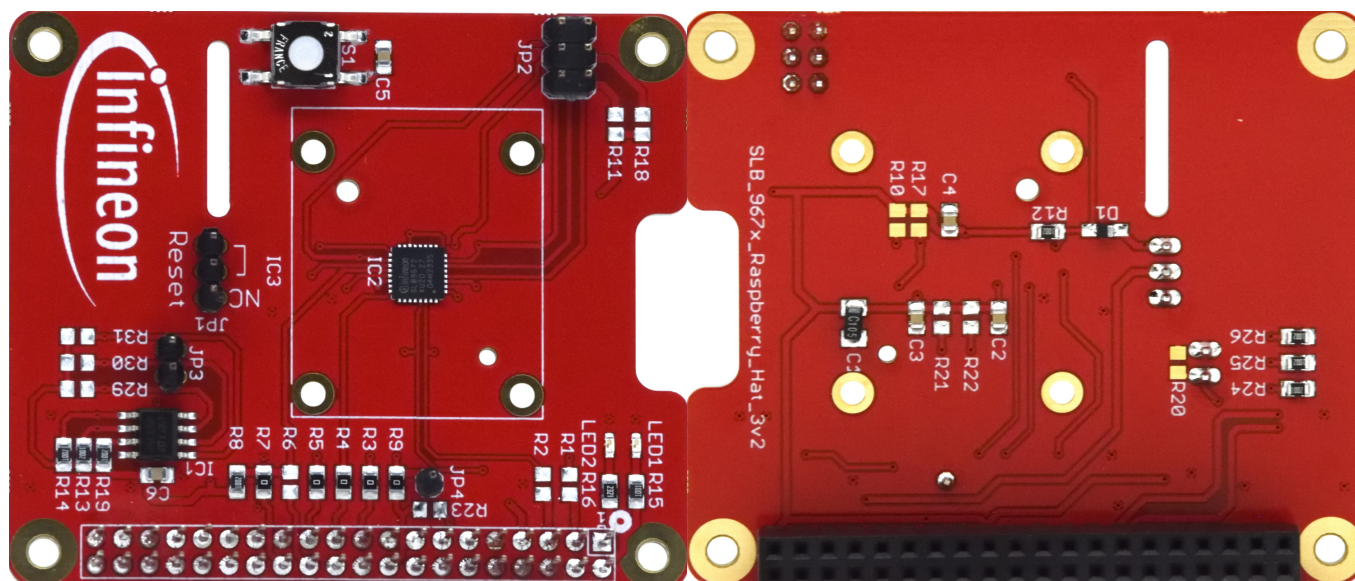
2 Hardware setup

The hardware required for the steps in this document includes:

- Developer PC: Use this platform to install the Raspberry Pi OS image by following [this guide](#)⁶. Note: The Raspberry Pi Imager now lets you set a default user and password because Raspberry Pi OS no longer enables the default `pi` user and `raspberrypi` password
- Infineon OPTIGA™ TPM SLB 967y TPM2.0 RPI evaluation board: This board contains an Infineon OPTIGA™ TPM SLx 967y TPM2.0 mounted on a hardware board that attaches to the Raspberry Pi
- Raspberry Pi:
 - Raspberry Pi board
 - MicroSD card with at least 8 GB
 - USB-C cable and power supply
- MicroSD card reader for the developer PC

2.1 Infineon OPTIGA™ TPM SLB 9672 RPI Evaluation Boards

Connect the Infineon OPTIGA™ TPM SLB 9672 TPM2.0 SPI RPI evaluation board to a Raspberry Pi through its extension header. The board uses the SPI bus for data transfer:



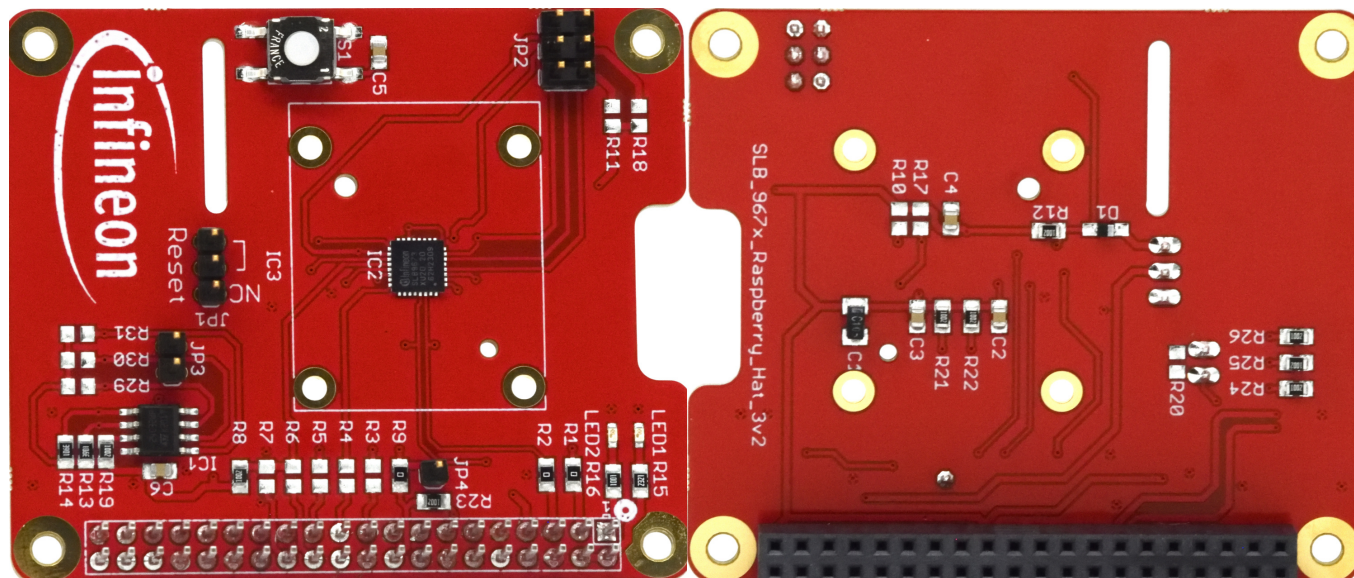
To distinguish this board from the OPTIGA™ TPM SLB 9673 evaluation board, note that resistors R1 and R2 are clear, while R3, R4, R5, and R7 are connected.

⁶ <https://www.raspberrypi.org/documentation/installation/installing-images/README.md>

Hardware setup

2.2 Infineon OPTIGA™ TPM SLB 9673 RPI Evaluation Boards

Connect the Infineon OPTIGA™ TPM SLB 9673 TPM2.0 I²C RPI evaluation board to a Raspberry Pi through its extension header. The board uses the I²C bus for data transfer:



To distinguish this board from the OPTIGA™ TPM SLB 9672 evaluation board, note that resistors R1 and R2 are connected, while R3, R4, R5, and R7 are clear.

For Raspberry Pi 5 systems using OPTIGA™ TPM SLB 9673 over I²C, see the *Enabling Device Tree for I2C* (page 37) before running TPM command examples for improved performance. By default the device tree overlay on the OPTIGA™ TPM SLB 9673 evaluation board uses a software-based I²C implementation for backwards compatibility. Older Raspberry Pi hardware I²C controllers have a [clock stretching bug](https://github.com/raspberrypi/linux/issues/4884)⁷ where the host ignores clock stretches shorter than half an I²C clock cycle, which breaks the TCG-mandatory clock stretching requirement. The Raspberry Pi 5's RP1 chip correctly supports clock stretching, enabling hardware I²C that is both significantly faster and fully TCG-compliant.

⁷ <https://github.com/raspberrypi/linux/issues/4884>

OS and driver setup

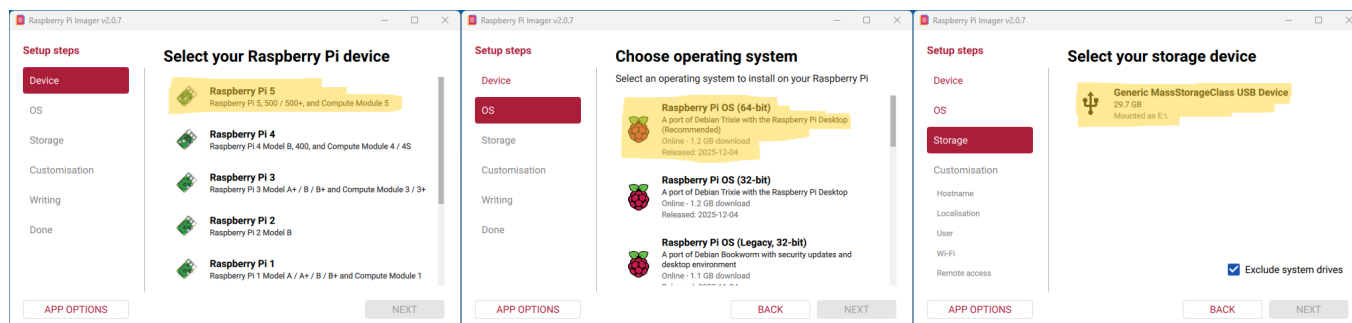
3 OS and driver setup

3.1 Raspberry Pi OS

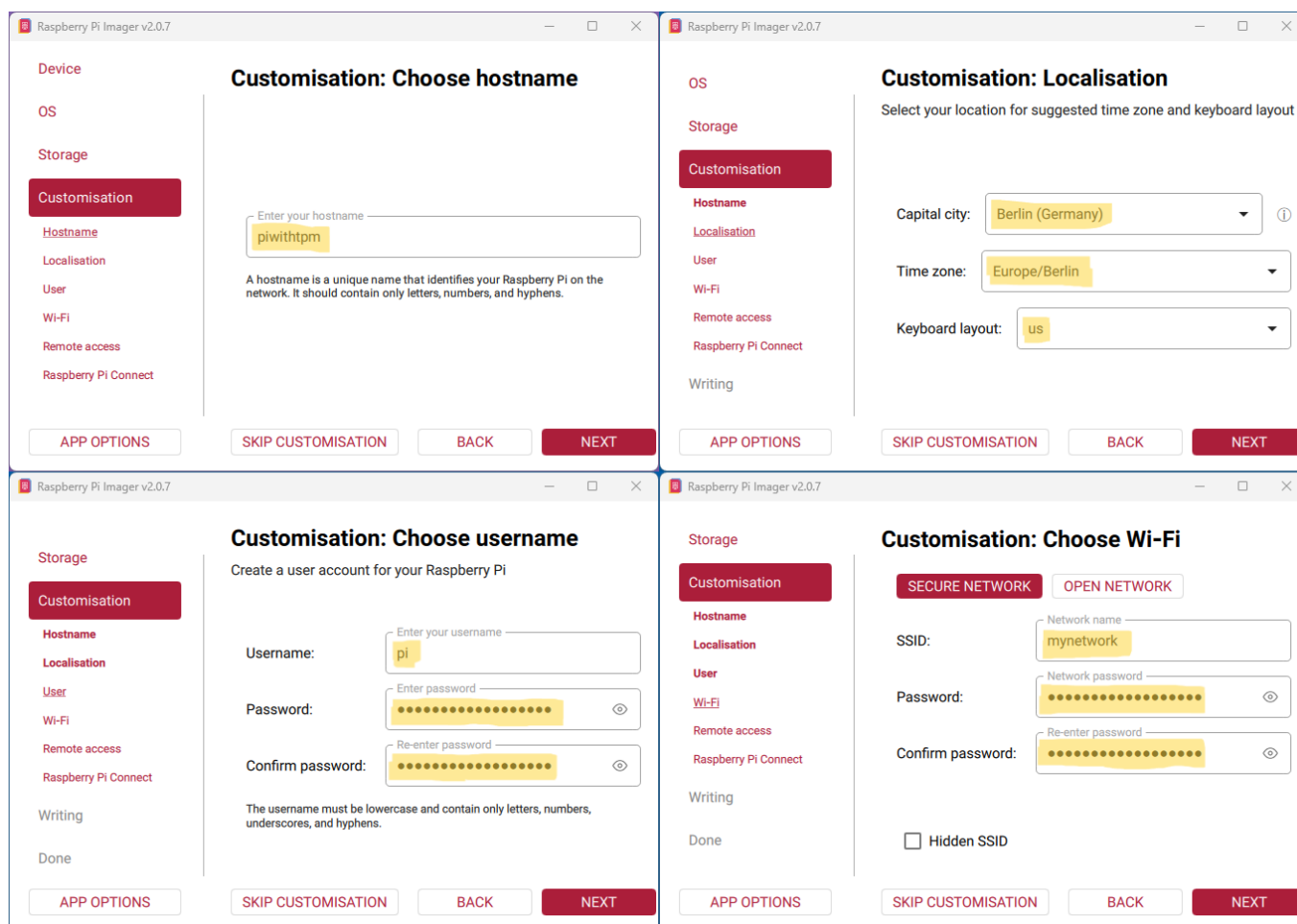
Flashing Raspberry Pi OS

Download [Raspberry Pi Imager](https://www.raspberrypi.com/software/)⁸, install, and start the application.

Select your Raspberry Pi device, the recommended OS image, and the external storage device:



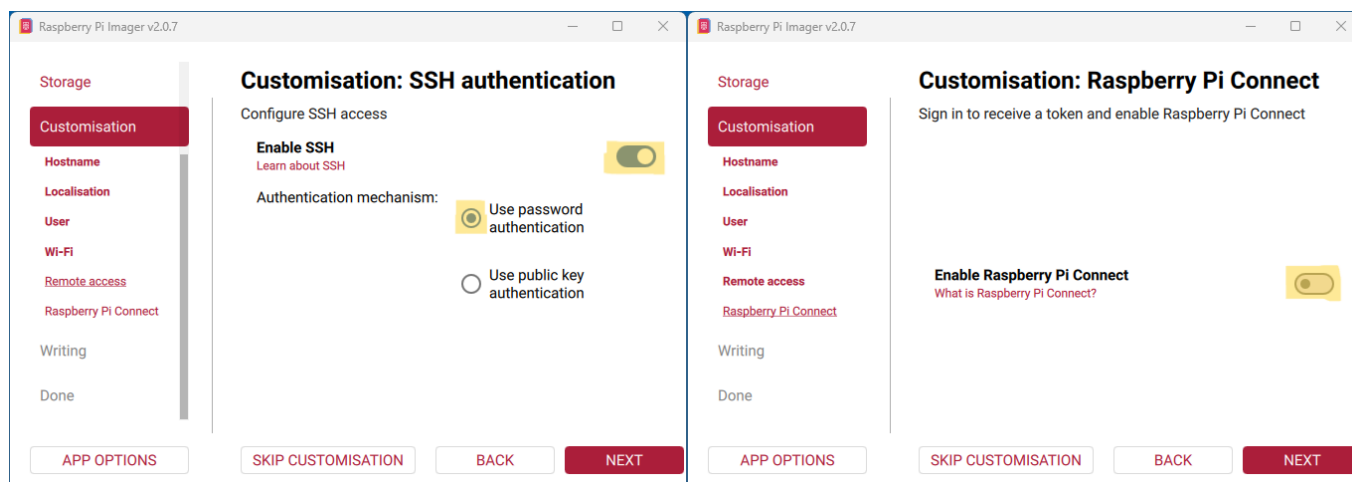
Configure the hostname, localisation, username, and password. If you plan to use wireless internet, also provide Wi-Fi credentials:



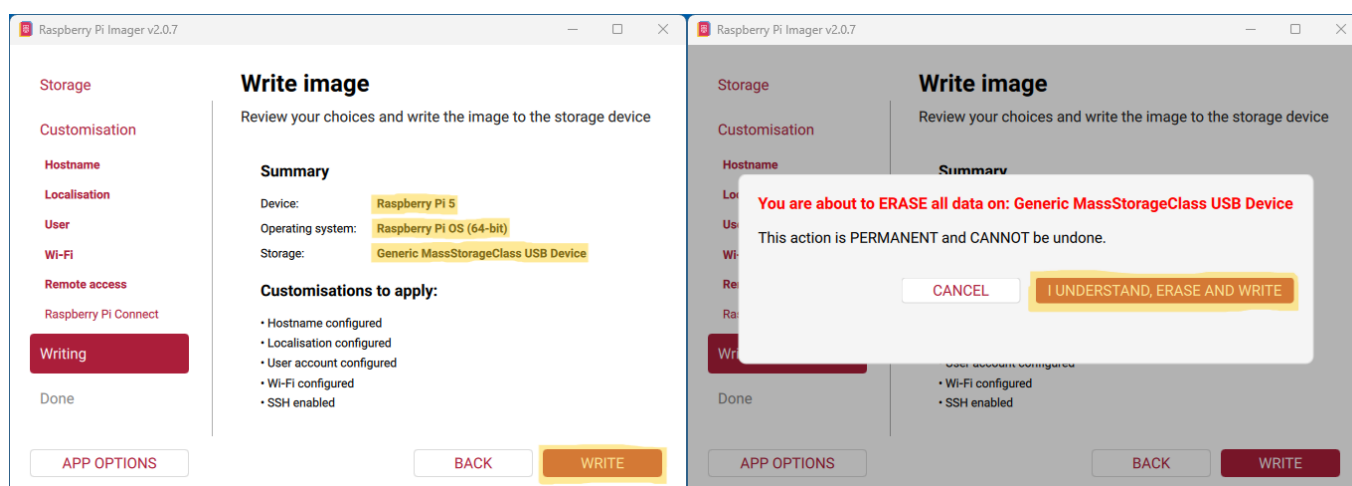
⁸ <https://www.raspberrypi.com/software/>

OS and driver setup

Configure SSH authentication for remote access. Setting up Raspberry Pi Connect is not necessary:



Verify the configuration and start writing to the storage device with the “WRITE” button. Confirm the pop-up dialog and wait for the write process to finish:



OPTIGA™ TPM SLB 9672 and SLB 9673 RPI Evaluation Boards

Insert the flashed SD card and boot the Raspberry Pi. The system should automatically load and configure the driver. After boot, check whether `/dev/tpm0` and `/dev/tpmrm0` exist:

```
1 ls -l /dev/tpm*
```

During boot, the device tree tells the Linux kernel how the host CPU and its peripherals are connected. This information is static for the base board, but device tree overlays can extend it. The OPTIGA™ TPM SLB 967y RPI evaluation boards include an EEPROM that stores these overlays. Raspberry Pi OS reads the EEPROM and applies the configuration automatically.

Note: On the older OPTIGA™ TPM SLx 9670 Iridium board, you must enable the driver manually. After flashing, remove and reinsert the SD card into your PC. Edit `config.txt` in the `firmware` directory, on the FAT32 partition labeled `boot`, and add the following line to the `[all]` section:

```
1 dtoverlay=tpm-slb9670
```

Software setup

4 Software setup

4.1 Install the Open Source TPM software stack tpm2-tss, TPM2 tools

The tpm2-software community provides an Open Source TPM Software Stack (TSS) implementation and additional tools.

To test this, run the following commands:

```
1 sudo apt --yes install libtss2-* tpm-udev tpm2-abrmd tpm2-tools
3 # Add the current user to group tss (with privilege to access the TPM)
4 sudo usermod --append --groups tss $USER
6 # Reboot for group change to become effective
7 sudo reboot
```

Now test the connection to the TPM:

```
1 tpm2 getcap properties-fixed
```

The output should look similar to this:

```
1 $ tpm2 getcap properties-fixed
2 TPM2_PT_FAMILY_INDICATOR:
3   raw: 0x322E3000
4   value: "2.0"
5 TPM2_PT_LEVEL:
6   raw: 0
7 TPM2_PT_REVISION:
8   raw: 0x9F
9   value: 1.59
10 TPM2_PT_DAY_OF_YEAR:
11   raw: 0xAA
12 TPM2_PT_YEAR:
13   raw: 0x7E4
14 TPM2_PT_MANUFACTURER:
15   raw: 0x49465800
16   value: "IFX"
17 TPM2_PT_VENDOR_STRING_1:
18   raw: 0x534C4239
19   value: "SLB9"
20 TPM2_PT_VENDOR_STRING_2:
21   raw: 0x36373200
22   value: "672"
23 ...
```

5 Using the TPM with tpm2-tss and tpm2-tools

The following sections describe how to start and use the TPM 2.0 on the Raspberry Pi with `tpm2-tss` and `tpm2-tools`. Be sure that you have successfully completed all setup steps as described in the previous sections. For general information about the TPM, its operational states and requirements, see the [TPM specification](#)⁹ and the [TPM 2.0 PTP specification](#)¹⁰. Execute all of the steps in this section on the Raspberry Pi (unless the text states otherwise). There are several ways to interact with the Raspberry Pi, see *Working with the Raspberry Pi* (page 44) for more information.

5.1 Software stack overview

tpm2-tss is a TPM Software Stack (TSS) for TPM 2.0. It consists of software stack libraries and provides functions such as:

- Simplifying TPM command execution by generating and processing TPM data structures with cryptographic functions
- Marshalling and unmarshalling TPM commands to and from raw byte streams
- Supporting the establishment and management of secure communication from the host processor to the TPM chip through transport sessions

tpm2-abrmd is the TPM Access Broker and Resource Manager Daemon. Its purpose is to:

- Provide resource management for the TPM, for example for sessions and the generation and storage of cryptographic keys
- Multiplex TPM access so that different applications and separate users can use TPM functions independently

Alternatively, the Linux kernel TPM driver exposes a basic subset of these operations through `/dev/tpmrm0`. Its limitation is that it does not work well with long-lived authorization sessions.

tpm2-tools is a set of executables that issue TPM 2.0 commands from the user shell. The tpm2-tools call the corresponding TSS2 function, which sends commands to the resource manager. The resource manager converts these commands to raw bytes and sends them to the TPM driver in the kernel.

The [tpm2-software community website](#)¹¹ provides a non-comprehensive list of software support.

5.2 Verifying TSS2 functionality

If you completed the previous steps, TSS2 and the tools should now work. Test them with the following command:

```
1 tpm2 getcap properties-fixed
```

If this command fails, see the *Troubleshooting* (page 33) section in the appendix.

5.3 Using tpm2-tools

`man tpm2`¹² provides a complete overview of all TPM tools, with explanations for every command and its parameters. You can also retrieve the list of commands and documentation on each command from the tools manpages directly:

```
1 man tpm2
2 man tpm2 getcap
```

⁹ <https://trustedcomputinggroup.org/resource/tpm-library-specification/>

¹⁰ <https://trustedcomputinggroup.org/resource/pc-client-platform-tpm-profile-ptp-specification/>

¹¹ <https://tpm2-software.github.io/software>

¹² <https://github.com/tpm2-software/tpm2-tools/blob/master/man/tpm2.1.md#arguments>

Using the TPM with tpm2-tss and tpm2-tools

Encrypt and Decrypt a File using RSA 2048

Using asymmetric cryptography to exchange data securely between two peers is the basis for many applications. Anyone who wants to send encrypted data to the owner of a private key can use the corresponding public key. Asymmetric algorithms usually encrypt only small amounts of data because the calculations require significant processing power. Typical uses include a symmetric key for later bulk encryption or configuration data for IoT applications. This section shows how to encrypt and decrypt a file with TPM 2.0 on a Raspberry Pi. In a real-world scenario, a remote device may perform the encryption before sending confidential data to the Raspberry Pi. Non-secure hardware can also perform the encryption because the plaintext, public key, and ciphertext do not reveal the private key. As a result, you can use the TPM as an encryption service, but you do not have to.

The following commands execute the encryption and decryption use case:

- `tpm2 createprimary`
- `tpm2 create`
- `(tpm2 loadexternal)`
- `(tpm2 rsaencrypt)`
- `tpm2 load`
- `tpm2 rsadecrypt`

The command `tpm2 createprimary` creates and loads a Primary Object under one of the Primary Seeds. To specify which seed to use, set the parameter `--hierarchy` to one of o: Owner, p: Platform, e: Endorsement, n: Null. In this example, we create the Primary Object in the Endorsement Hierarchy. The command line option `--hash-algorithm` defines the hash algorithm used for computing the name of the object (0x000B: SHA256), `--key-algorithm` defines the algorithm associated with this object (0x0001: RSA) and `--key-context` defines the file to save the object context (`primary_ctx`). In the code below, you provide the hierarchy, hash algorithm and object algorithm as input, and the `tpm2 createprimary` function returns the object reference (`--key-context primary_ctx`). Subsequent code then uses this reference as input.

```
1 tpm2 createprimary --hierarchy e --hash-algorithm sha256 --key-algorithm \
  rsa2048 --key-context primary_ctx
2 # Output file: primary_ctx
```

The `tpm2 create` command creates an encryption and decryption key and returns both the public and private portion of the key to the application. The command stores both parts as files (`key_pub`, `key_priv`) in the folder where you execute it. The TPM encrypts the private part with the public part of the parent key, so only the TPM can decrypt and use it.

```
1 tpm2 create --hash-algorithm sha256 --key-algorithm rsa2048 \
  --parent-context primary_ctx --public key_pub --private key_priv
2 # Input file: primary_ctx - Output files: key_pub, key_priv
```

Encryption requires only the public key, which you can load on any TPM without a password and without referencing the parent key of the private key. Due to resource limitations of the TPM, it stores the keys externally to the chip, with the public portion in plain format and the private portion encrypted. This allows creating and storing an unlimited amount of keys with a single TPM 2.0 without any security drawback. The only disadvantage is that you must load every key into the TPM before using it for a cryptographic operation. Persistent storage inside the TPM for frequently used keys can simplify and accelerate this mechanism. Since performance is not a requirement for this sample use case, we do not store the keys permanently inside the TPM and load them beforehand. The command below loads the encryption key (`key_pub`) into the TPM. It takes a parameter defining the hierarchy where to load the encryption key, the public key itself, and returns the reference for the subsequent encryption process:

Using the TPM with tpm2-tss and tpm2-tools

```
1 tpm2 loadexternal --hierarchy n --public key_pub --key-context \
   rsaencrypt_key_ctx
2 # Input file: key_pub - Output file: rsaencrypt_key_ctx
```

To encrypt data with the TPM, provide a filename for the plain-text input data (plaintext_data) and a filename for the encrypted data (cipher_data).

```
1 echo "Hello World" >plaintext_data
2 # Output file: plaintext_data

4 tpm2 rsaencrypt --key-context rsaencrypt_key_ctx --output cipher_data \
   plaintext_data
5 # Input files: rsaencrypt_key_ctx, plaintext_data - Output file: cipher_data
```

Use the command `tpm2 load` to load objects into the TPM when a certain command needs both the public and private part present inside the TPM. A decryption operation involving a private key requires this. The decryption process will use a key context (rsadecrypt_key_ctx) for referencing the private key inside the TPM. The `tpm2 load` command creates this handle as follows:

```
1 tpm2 load --parent-context primary_ctx --public key_pub --private key_priv \
   --key-context rsadecrypt_key_ctx
2 # Input files: primary_ctx, key_pub, key_priv - Output file: \
   rsadecrypt_key_ctx
```

In addition to the key context (rsadecrypt_key_ctx), the decryption command needs the previously created file with cipher-text data (cipher_data) and optionally a file name for the output data (decrypted_data) as command line parameters. After executing this command, the content of decrypted_data or the terminal output is the same as the plaintext_data.

```
1 tpm2 rsadecrypt --key-context rsadecrypt_key_ctx cipher_data
2 # Input files: rsadecrypt_key_ctx cipher_data
3 # Optional output to file: --output decrypted_data
```

The corresponding terminal output should look as follows:

```
1 $ tpm2 createprimary --hierarchy e --hash-algorithm sha256 --key-algorithm \
   rsa2048 --key-context primary_ctx
2 name-alg:
3   value: sha256
4   raw: 0xb
5 attributes:
6   value: \
   fixedtpm|fixedparent|sensitivedataorigin|userwithauth|restricted|decrypt
7   raw: 0x30072
8 type:
9   value: rsa
10  raw: 0x1
11 exponent: 65537
12 bits: 2048
13 scheme:
14   value: null
15   raw: 0x10
```

Using the TPM with tpm2-tss and tpm2-tools

```
16 scheme-halg:
17   value: (null)
18   raw: 0x0
19 sym-alg:
20   value: aes
21   raw: 0x6
22 sym-mode:
23   value: cfb
24   raw: 0x43
25 sym-keybits: 128
26 rsa: b5602c2838730a108984ab7eb9886f98ec11c2bfe44a300f013baf639897bae1cf8d...
27 $ tpm2 create --hash-algorithm sha256 --key-algorithm rsa2048 \
   --parent-context primary_ctx --public key_pub --private key_priv
28 name-alg:
29   value: sha256
30   raw: 0xb
31 attributes:
32   value: fixedtpm|fixedparent|sensitivedataorigin|userwithauth|decrypt|sign
33   raw: 0x60072
34 type:
35   value: rsa
36   raw: 0x1
37 exponent: 65537
38 bits: 2048
39 scheme:
40   value: null
41   raw: 0x10
42 scheme-halg:
43   value: (null)
44   raw: 0x0
45 sym-alg:
46   value: null
47   raw: 0x10
48 sym-mode:
49   value: (null)
50   raw: 0x0
51 sym-keybits: 0
52 rsa: cafd9cb68c5279f2010e99f37b69820aed75a3f64e4be759db8f9f69687f0c2e...
53 $ tpm2 loadexternal --hierarchy n --public key_pub --key-context \
   rsaencrypt_key_ctx
54 name: 000bdee578e96855a778ae680159aa304ebe56a8cbfe2580c7a7b08278a3cb57204b
55 $ echo "Hello World" >plaintext_data
56 $ tpm2 rsaencrypt --key-context rsaencrypt_key_ctx --output cipher_data \
   plaintext_data
57 $ tpm2 load --parent-context primary_ctx --public key_pub --private \
   key_priv --key-context rsadecrypt_key_ctx
58 name: 000bdee578e96855a778ae680159aa304ebe56a8cbfe2580c7a7b08278a3cb57204b
59 $ tpm2 rsadecrypt --key-context rsadecrypt_key_ctx cipher_data
60 Hello World
```

Using the TPM with tpm2-tss and tpm2-tools

Sign and Verify using ECC 256

You can create a signature with a private key for any data. The only precondition is that you can create a hash value of the data, which may be a file or a message. If so, the hash function output serves as the input for the signing operation. The receiver can then use the signature to check both the authenticity and integrity of data that the owner of the private key sent. Therefore, the receiver needs to have the public key to successfully verify the signature. In terms of the TPM, execute the following commands to sign a file and verify the signature.

- `tpm2 createprimary`
- `tpm2 create`
- `tpm2 load`
- `tpm2 sign`
- `(tpm2 loadexternal)`
- `(tpm2 verifysignature)`

The first three steps (`createprimary`, `create`, and `load`) are almost the same as in the previous section. Both use cases require an asymmetric key pair, either for encryption and decryption or for signing and verification. The main difference here is changing `--key-algorithm` from `rsa2048` to `ecc` and using `eccsigning_key_ctx` as the context file name. Note that the primary key algorithm does not matter in this case because the TPM encrypts the child key symmetrically. The asymmetric algorithm specified here is only relevant for encrypted import operations.

```

1 tpm2 createprimary --hierarchy e --hash-algorithm sha256 --key-algorithm \
   ecc --key-context primary_ctx
2 # Output file: primary_ctx

4 tpm2 create --hash-algorithm sha256 --key-algorithm ecc --parent-context \
   primary_ctx --public key_pub --private key_priv
5 # Input file: primary_ctx - Output files: key_pub, key_priv

7 tpm2 load --parent-context primary_ctx --public key_pub --private key_priv \
   --key-context eccsigning_key_ctx
8 # Input files: primary_ctx, key_pub, key_priv - Output file: \
   eccsigning_key_ctx

```

Before signing, create an input file:

```

1 echo "Sign me" > input_data
2 # Output file: input_data

```

The `tpm2 sign` command uses the signing key (`--key-context eccsigning_key_ctx`) and creates a signature, saving it in the signature file. Before signing the file, it creates a digest from the `input_data` using SHA256.

```

1 tpm2 sign --key-context eccsigning_key_ctx --signature signature_file \
   input_data
2 # Input files: eccsigning_key_ctx, input_data - Output files: signature_file

```

Now, use the command `tpm2 verifysignature` to verify the signature created above. You do not need to reload the public key into the TPM because the previous `load` command already loaded it. If the system reboots or you verify the signature on a different system, load the public key into the TPM before verifying the signature. Use the command `tpm2 loadexternal` as described in the previous section.

Using the TPM with tpm2-tss and tpm2-tools

```

1 tpm2 verifysignature --key-context eccsigning_key_ctx --message input_data \
  --signature signature_file
2 # Input files eccsigning_key_ctx, input_data, signature_file

```

The corresponding terminal output should look as follows:

```

1 $ tpm2 createprimary --hierarchy e --hash-algorithm sha256 --key-algorithm \
  ecc --key-context primary_ctx
2 name-alg:
3   value: sha256
4   raw: 0xb
5 attributes:
6   value: \
  fixedtpm|fixedparent|sensitivedataorigin|userwithauth|restricted|decrypt
7   raw: 0x30072
8 type:
9   value: ecc
10  raw: 0x23
11 curve-id:
12   value: NIST p256
13   raw: 0x3
14 kdfa-alg:
15   value: null
16   raw: 0x10
17 kdfa-halg:
18   value: (null)
19   raw: 0x0
20 scheme:
21   value: null
22   raw: 0x10
23 scheme-halg:
24   value: (null)
25   raw: 0x0
26 sym-alg:
27   value: aes
28   raw: 0x6
29 sym-mode:
30   value: cfb
31   raw: 0x43
32 sym-keybits: 128
33 x: eca6c97c4b43231176b043488830632da9a3f7baefaa73898deff6fd896663b5
34 y: 3033c24be8d34534d519eea7b9bf2baaa8324676af2d8bd4e3de75a817ee9eaa
35 $ tpm2 create --hash-algorithm sha256 --key-algorithm ecc --parent-context \
  primary_ctx --public key_pub --private key_priv
36 name-alg:
37   value: sha256
38   raw: 0xb
39 attributes:
40   value: fixedtpm|fixedparent|sensitivedataorigin|userwithauth|decrypt|sign
41   raw: 0x60072

```

Using the TPM with tpm2-tss and tpm2-tools

```
42 type:
43   value: ecc
44   raw: 0x23
45 curve-id:
46   value: NIST p256
47   raw: 0x3
48 kdfa-alg:
49   value: null
50   raw: 0x10
51 kdfa-halg:
52   value: (null)
53   raw: 0x0
54 scheme:
55   value: null
56   raw: 0x10
57 scheme-halg:
58   value: (null)
59   raw: 0x0
60 sym-alg:
61   value: null
62   raw: 0x10
63 sym-mode:
64   value: (null)
65   raw: 0x0
66 sym-keybits: 0
67 x: 451a818210f3fe9c908e028c1f4cd496d6f5d69499b9a9e868e8daa6188c5110
68 y: c8419b4ef7083bce1665bc309c08c7d1270139c65b4aabe68e525f10887575c4
69 $ tpm2 load --parent-context primary_ctx --public key_pub --private \
   key_priv --key-context eccsigning_key_ctx
70 name: 000bebf1d29ee27c17fe5cbadd30738ed91f595f96f707b09ac4d964347554aecb52
71 $ echo "Sign me" > input_data
72 $ tpm2 sign --key-context eccsigning_key_ctx --signature signature_file \
   input_data
73 $ tpm2 verifysignature --key-context eccsigning_key_ctx --message \
   input_data --signature signature_file
```

6 Using TPM with OpenSSL

The `tpm2-openssl` project¹³ implements a provider that extends OpenSSL 3.0.0 and above with TPM functionality. To use it, load the `tpm2` provider into the OpenSSL context to handle certain tasks on behalf of the `openssl` command-line tool.

Note: For the now end-of-life OpenSSL 1.1.1, the `tpm2-tss-engine` project¹⁴ provided an alternative implementation. New projects should use OpenSSL 3.x or 4.x with the `tpm2-openssl` provider.

6.1 Installing the software

First, install the `tpm2-openssl` software module:

```
1 sudo apt --yes install tpm2-openssl
```

In older Raspberry Pi OS versions there is a bug in the packaged `tpm2-openssl` provider. For a workaround consider the *Compiling the TPM provider* (page 44) section in the appendix.

6.2 Interfacing with OpenSSL via command line

When using the command line interface of OpenSSL, activate the provider by adding the argument `-provider tpm2` to the command line. For some commands, also add `-provider base`. OpenSSL needs this when it performs an operation that the `tpm2` provider does not support, such as calculating a hash using SHA2-512.

To prepare the test scenario, generate a key pair. The following example uses an ECC P256 key pair. This key exists as a child of the default SRK (Storage Root Key) template, which is a primary key in the storage hierarchy. The tool encodes the corresponding public and private key blobs in TSS2 PRIVATE KEY format and stores them as PEM in `testkey.priv`. Several Open Source projects, including GNUTLS and OpenConnect, support this format.

From this key pair, export the public key to the standard PUBLIC KEY format, which works without a TPM and `tpm2-openssl` module.

```
1 # Generate ECC P256 key pair as PEM
2 openssl genpkey -provider tpm2 -provider base -propquery '?provider=tpm2' \
  -algorithm EC -pkeyopt group:P-256 -out testkey.priv
3 # Output file testkey.priv

5 # Verify and display the key
6 openssl ec -provider tpm2 -provider base -in testkey.priv -check -text -noout
7 # Input file testkey.priv

9 # Export the public key into separate file for verifier
10 openssl pkey -provider tpm2 -provider base -in testkey.priv -passin pass: \
  -pubout -out testkey.pub
11 # Input file testkey.priv - Output file testkey.pub
```

Now use the key pair in `testkey.priv` to sign the document `testdata`. The command stores the signature in `testdata.sig`.

```
1 # Prepare data to be signed
2 echo -n "abcde12345abcde12345abcde12345ab" > testdata
```

¹³ <https://github.com/tpm2-software/tpm2-openssl>

¹⁴ <https://github.com/tpm2-software/tpm2-tss-engine>

Using TPM with OpenSSL

```

3 # Output file testdata
5 # Sign the data
6 openssl pkeyutl -provider tpm2 -provider base -sign -inkey testkey.priv -in \
  testdata -out testdata.sig
7 # Input files testkey.priv, testdata - Output file testdata.sig

```

You can then verify this signature without the tpm2-openssl module, using the previously exported public key testkey.pub.

```

1 openssl pkeyutl -verify -pubin -inkey testkey.pub -in testdata -sigfile \
  testdata.sig
2 # Input files testkey.pub, testdata, testdata.sig

```

The corresponding terminal output should look as follows:

```

1 $ openssl genpkey -provider tpm2 -provider base -propquery '?provider=tpm2' \
  -algorithm EC -pkeyopt group:P-256 -out testkey.priv
2 $ openssl ec -provider tpm2 -provider base -in testkey.priv -check -text \
  -noout
3 read EC key
4 Private-Key: (EC P-256, TPM 2.0)
5 pub:
6     04:a4:0c:b5:c8:20:61:88:8e:e6:2e:cc:a2:a1:9a:
7     34:43:ad:83:65:59:28:d9:15:69:0e:d9:aa:29:4f:
8     bf:0e:fb:df:c0:89:b6:e6:a3:91:52:c8:78:41:c8:
9     64:9f:18:3b:3e:96:49:27:1b:53:28:0e:07:14:a4:
10    f1:e0:e0:86:c5
11 ASN1 OID: prime256v1
12 Object Attributes:
13     fixedTPM
14     fixedParent
15     sensitiveDataOrigin
16     userWithAuth
17     decrypt
18     sign / encrypt
19 EC Key valid.
20 $ openssl pkey -provider tpm2 -provider base -in testkey.priv -passin pass: \
  -pubout -out testkey.pub
21 $ echo -n "abcde12345abcde12345abcde12345ab" > testdata
22 $ openssl pkeyutl -provider tpm2 -provider base -sign -inkey testkey.priv \
  -in testdata -out testdata.sig
23 $ openssl pkeyutl -verify -pubin -inkey testkey.pub -in testdata -sigfile \
  testdata.sig
24 Signature Verified Successfully

```

Note: After upgrading the Linux kernel, interface changes can break the tpm2-openssl provider. Should the sign operation fail with the following error message, follow the instructions in the *Compiling the TPM provider* (page 44) section:

```

1 $ openssl pkeyutl -provider tpm2 -provider base -sign -inkey testkey.priv \
  -in testdata -out testdata.sig

```

Using TPM with OpenSSL

```

2  WARNING:esys:src/tss2-esys/api/Esys_Sign.c:311:Esys_Sign_Finish() Received \
    TPM Error
3  ERROR:esys:src/tss2-esys/api/Esys_Sign.c:105:Esys_Sign() Esys Finish \
    ErrorCode (0x000001d5)
4  Public Key operation error
5  E08EB376FF7F0000:error:4000000F:tpm2::cannot sign::-1:469 \
    tpm:parameter(1):structure is the wrong size
6  E08EB376FF7F0000:error:030000EA:digital envelope \
    routines:EVP_PKEY_sign:provider signature \
    failure:../crypto/evp/signature.c:1021:ECDSA sign:

```

More examples can be found in the `tpm2-openssl` test suite¹⁵.

6.3 Interfacing with OpenSSL from NGINX® web server

The following example shows how to use the TPM in a web server to protect long-term identity keys. In this use case, the TPM creates and stores the identity key used during the SSL/TLS handshake. During the handshake, this key proves the identity of the server. The client and server also negotiate session secrets during the same handshake. OpenSSL holds these secrets in its process context at the same isolation level as the data sent through the SSL/TLS channel.

Preparing keys and certificates

First, generate a new key pair and a self-signed certificate. The `openssl` command asks for the certificate information. Press <enter> on each line to use the example values shown in the [...] brackets.

```

1  openssl req -provider tpm2 -provider base -propquery '?provider=tpm2' -x509 \
    -new -newkey ec -pkeyopt group:P-256 -keyout httpserver-key.pem -out \
    httpserver-crt.pem
2  # Output files: httpserver-key.pem, httpserver-crt.pem

```

Installing and configuring NGINX

First, install the NGINX web server on the system:

```

1  # Install the NGINX web server
2  sudo apt --yes install nginx

```

Then edit the NGINX configuration file:

```

1  sudo nano /etc/nginx/sites-enabled/default
2  # Exit the editor using CTRL+x -> y

```

In the server section, add the lines that enable SSL/TLS communication, which start with `listen`. Then add the lines that define the paths to the certificate and key pair used for authentication, which start with `ssl_`:

```

1  server {
2      # SSL configuration
3      #
4      listen 443 ssl default_server;
5      listen [::]:443 ssl default_server;
6      ssl_certificate /home/pi/httpserver-crt.pem;
7      ssl_certificate_key /home/pi/httpserver-key.pem;

```

¹⁵ <https://github.com/tpm2-software/tpm2-openssl/tree/master/test>

Using TPM with OpenSSL

Also update the global OpenSSL configuration to make the `tpm2-openssl` provider module globally available. Only then can the NGINX web server correctly parse the `TSS2_PRIVATE_KEY` in the key pair file.

```
1 sudo nano /etc/ssl/openssl.cnf
2 # Exit the editor using CTRL+x -> y
```

In the `openssl_init` section, add the providers section. In the provider section, keep the default provider active and declare the `tpm2-openssl` provider module:

```
1 openssl_conf = openssl_init

3 [openssl_init]
4 providers = provider_sect

6 [provider_sect]
7 default = default_sect
8 tpm2provider = tpm2provider_sect

10 [default_sect]
11 activate = 1

13 [tpm2provider_sect]
14 activate = 1
15 module = /usr/lib/aarch64-linux-gnu/openssl-modules/tpm2.so
```

After restarting NGINX, the web server should be ready to receive connections:

```
1 sudo systemctl restart nginx
```

To test the status of NGINX, query systemd:

```
1 sudo systemctl status nginx
```

The corresponding terminal outputs should look as follows:

```
1 $ openssl req -provider tpm2 -provider base -propquery '?provider=tpm2' \
   -x509 -new -newkey ec -pkeyopt group:P-256 -keyout httpserver-key.pem \
   -out httpserver-crt.pem
2 -----
3 You are about to be asked to enter information that will be incorporated
4 into your certificate request.
5 What you are about to enter is what is called a Distinguished Name or a DN.
6 There are quite a few fields but you can leave some blank
7 For some fields there will be a default value,
8 If you enter '.', the field will be left blank.
9 -----
10 Country Name (2 letter code) [AU]:
11 State or Province Name (full name) [Some-State]:
12 Locality Name (eg, city) []:
13 Organization Name (eg, company) [Internet Widgits Pty Ltd]:
14 Organizational Unit Name (eg, section) []:
15 Common Name (e.g. server FQDN or YOUR name) []:
16 Email Address []:
```

Using TPM with OpenSSL

```
17 $ sudo apt --yes install nginx
18 Reading package lists... Done
19 Building dependency tree... Done
20 Reading state information... Done
21 The following package was automatically installed and is no longer required:
22   libpisp0.0.1
23 Use 'sudo apt autoremove' to remove it.
24 The following additional packages will be installed:
25   nginx-common
26 Suggested packages:
27   fcgiwrap nginx-doc
28 The following NEW packages will be installed:
29   nginx nginx-common
30 0 upgraded, 2 newly installed, 0 to remove and 1 not upgraded.
31 Need to get 606 kB of archives.
32 After this operation, 1,708 kB of additional disk space will be used.
33 Get:1 http://deb.debian.org/debian bookworm/main arm64 nginx-common all \
34     1.22.1-9 [112 kB]
35 Get:2 http://deb.debian.org/debian bookworm/main arm64 nginx arm64 1.22.1-9 \
36     [494 kB]
37 Fetched 606 kB in 0s (2,788 kB/s)
38 Preconfiguring packages ...
39 Selecting previously unselected package nginx-common.
40 (Reading database ... 148291 files and directories currently installed.)
41 Preparing to unpack .../nginx-common_1.22.1-9_all.deb ...
42 Unpacking nginx-common (1.22.1-9) ...
43 Selecting previously unselected package nginx.
44 Preparing to unpack .../nginx_1.22.1-9_arm64.deb ...
45 Unpacking nginx (1.22.1-9) ...
46 Setting up nginx-common (1.22.1-9) ...
47 Created symlink /etc/systemd/system/multi-user.target.wants/nginx.service \
48     -> /lib/systemd/system/nginx.service.
49 Setting up nginx (1.22.1-9) ...
50 Upgrading binary: nginx.
51 Processing triggers for man-db (2.11.2-2) ...
52 $ sudo nano /etc/nginx/sites-enabled/default
53 $ sudo nano /etc/ssl/openssl.cnf
54 $ sudo systemctl restart nginx
55 $ sudo systemctl status nginx
56 - nginx.service - A high performance web server and a reverse proxy server
57   Loaded: loaded (/lib/systemd/system/nginx.service; enabled; preset: \
58   enabled)
59   Active: active (running) since Thu 2023-11-16 10:43:24 CET; 7s ago
60     Docs: man:nginx(8)
61   Process: 3086 ExecStartPre=/usr/sbin/nginx -t -q -g daemon on; \
62   master_process on; (code=exited, status=0/SUCCESS)
63   Process: 3089 ExecStart=/usr/sbin/nginx -g daemon on; master_process \
64   on; (code=exited, status=0/SUCCESS)
65   Main PID: 3092 (nginx)
```

Using TPM with OpenSSL

```

60     Tasks: 5 (limit: 3912)
61     CPU: 120ms
62     CGroup: /system.slice/nginx.service
63     |-3092 "nginx: master process /usr/sbin/nginx -g daemon on; \
master_process on;"
64     |-3093 "nginx: worker process"
65     |-3094 "nginx: worker process"
66     |-3095 "nginx: worker process"
67     +-3096 "nginx: worker process"

69 Nov 16 10:43:23 piwithtpm systemd[1]: Starting nginx.service - A high \
performance web server and a reverse proxy server...
70 Nov 16 10:43:24 piwithtpm systemd[1]: Started nginx.service - A high \
performance web server and a reverse proxy server.

```

Testing the connection to NGINX

To test the connection to NGINX, run the following command:

```
1 curl -v --insecure https://localhost
```

The corresponding terminal outputs should look as follows:

```

1 $ curl -v --insecure https://localhost
2 * Trying 127.0.0.1:443...
3 * Connected to localhost (127.0.0.1) port 443 (#0)
4 ...
5 * Server certificate:
6 * subject: C=AU; ST=Some-State; O=Internet Widgits Pty Ltd
7 ...
8 <h1>Welcome to nginx!</h1>
9 ...

```

6.4 Interfacing with OpenSSL from curl web client

The `tpm2-openssl` provider module also supports client authentication in an SSL/TLS connection. To test this, the web server must offer client authentication to the web client.

Enabling client certificate authentication in NGINX

First, configure the NGINX web server to allow connecting clients to provide client authentication.

```

1 sudo nano /etc/nginx/sites-enabled/default
2 # Exit the editor using CTRL+x -> y

```

When editing this file, make sure you enable the SSL/TLS options. In the following listing the values for a TPM enabled web server remain active. Further, add the line `ssl_verify_client optional_no_ca;` to the server section.

```

1 server {
2     # SSL configuration
3     # ensure working ssl configuration is configured, e.g. the previous
4     listen 443 ssl default_server;

```

Using TPM with OpenSSL

```

5      listen [::]:443 ssl default_server;
6      ssl_certificate /home/pi/httpserver-crt.pem;
7      ssl_certificate_key /home/pi/httpserver-key.pem;

9      # Add the following line
10     ssl_verify_client optional_no_ca;

```

Finally, restart NGINX to start accepting client certificates.

```

1  sudo systemctl restart nginx

```

Preparing the client

For client authentication, create a key pair and a certificate file similar to the server-side certificate. Fill in the requested certificate fields with the example values by pressing <enter>.

```

1  openssl req -provider tpm2 -provider base -propquery '?provider=tpm2' -x509 \
    -new -newkey ec -pkeyopt group:P-256 -keyout httpclient-key.pem -out \
    httpclient-crt.pem
2  # Output files: httpclient-key.pem httpclient-crt.pem

```

Also ensure the file `openssl.cnf` enables the `tpm2-openssl` provider similar to the server-side example. If you have not already done this, edit the file accordingly.

```

1  sudo nano /etc/ssl/openssl.cnf
2  # Exit the editor using CTRL+x -> y

```

In the `openssl_init` section, add the providers section. In the provider section, keep the default provider active and declare the `tpm2-openssl` provider module:

```

1  openssl_conf = openssl_init

3  [openssl_init]
4  providers = provider_sect

6  [provider_sect]
7  default = default_sect
8  tpm2provider = tpm2provider_sect

10 [default_sect]
11 activate = 1

13 [tpm2provider_sect]
14 activate = 1
15 module = /usr/lib/aarch64-linux-gnu/openssl-modules/tpm2.so

```

Testing the communication

To test the communication, use `curl` again. No visible output indicates the successful transfer of the client certificate, however. Therefore, enable logging of the TSS by prepending `TSS2_LOG=esys+trace` to the command invocation. In the log output, you can observe the communication with the TPM, which includes a call to `Esys_Sign`. During this call, the TPM uses the private key to authenticate to the server.

Using TPM with OpenSSL

```
1 curl --insecure -v --key httpclient-key.pem --cert httpclient-crt.pem \  
  https://localhost  
2 # Input files: httpclient-key.pem httpclient-crt.pem  
  
4 # With log information; look for Esys_Sign  
5 TSS2_LOG=esys+trace curl --insecure -v --key httpclient-key.pem --cert \  
  httpclient-crt.pem https://localhost
```

The corresponding terminal outputs should look as follows:

```
1 $ sudo nano /etc/nginx/sites-enabled/default  
2 $ sudo systemctl restart nginx  
3 $ openssl req -provider tpm2 -provider base -propquery '?provider=tpm2' \  
  -x509 -new -newkey ec -pkeyopt group:P-256 -keyout httpclient-key.pem \  
  -out httpclient-crt.pem  
4 -----  
5 You are about to be asked to enter information that will be incorporated  
6 into your certificate request.  
7 What you are about to enter is what is called a Distinguished Name or a DN.  
8 There are quite a few fields but you can leave some blank  
9 For some fields there will be a default value,  
10 If you enter '.', the field will be left blank.  
11 -----  
12 Country Name (2 letter code) [AU]:  
13 State or Province Name (full name) [Some-State]:  
14 Locality Name (eg, city) []:  
15 Organization Name (eg, company) [Internet Widgits Pty Ltd]:  
16 Organizational Unit Name (eg, section) []:  
17 Common Name (e.g. server FQDN or YOUR name) []:  
18 Email Address []:  
19 $ curl --insecure -v --key httpclient-key.pem --cert httpclient-crt.pem \  
  https://localhost  
20 * Trying 127.0.0.1:443...  
21 * Connected to localhost (127.0.0.1) port 443 (#0)  
22 ...  
23 <h1>Welcome to nginx!</h1>  
24 ...
```

7 Using TPM with PKCS11

The TPM can also serve as a “virtual smartcard”. For this, the project [tpm2-pkcs11](https://github.com/tpm2-software/tpm2-pkcs11)¹⁶ implements the PKCS11 interface that smart cards typically use.

7.1 Installation and setup of the tpm2-pkcs11 module

Install this module and the pkcs11-tool directly from the Raspberry Pi OS repositories:

```
1 sudo apt --yes install libtpm2-pkcs11* opensc
```

The tpm2-pkcs11 module requires a directory to store the token data. Create the directory and ensure only the user has permission to write to it:

```
1 # Create the directory
2 sudo mkdir -p /tmp/tpm2-pkcs11

4 # Give the user ownership so it can write to it
5 sudo chown $USER:$USER /tmp/tpm2-pkcs11

7 # Restrict permissions to read/write only for the owner
8 sudo chmod 700 /tmp/tpm2-pkcs11
```

Tell the pkcs11-tool where this directory is located by exporting the TPM2_PKCS11_STORE variable in your current session. Run this command before running any pkcs11-tool commands:

```
1 export TPM2_PKCS11_STORE=/tmp/tpm2-pkcs11
```

7.2 Configuring a TPM token with pkcs11-tool

In its initial state, the tpm2-pkcs11 module offers a single uninitialized token slot. Initialize this slot to create a first token (i.e. virtual smartcard) with a name (i.e. label) and a security officer PIN (i.e. so-pin, the administrator account). To support several virtual smart cards on a single system, the tpm2-pkcs11 module also creates a new uninitialized token after each initialization. This means an uninitialized token always remains present, supporting arbitrary numbers of smart cards:

```
1 # List all tokens
2 pkcs11-tool --module \
  /usr/lib/aarch64-linux-gnu/pkcs11/libtpm2_pkcs11.so.1.9.0 \
  --list-token-slots

4 # Initialize the token in slot 0 with label "mytoken" and so-pin "1234"
5 pkcs11-tool --module \
  /usr/lib/aarch64-linux-gnu/pkcs11/libtpm2_pkcs11.so.1.9.0 --slot-index=0 \
  --init-token --label="mytoken" --so-pin="1234"

7 # List all tokens; including the newly created token
8 pkcs11-tool --module \
  /usr/lib/aarch64-linux-gnu/pkcs11/libtpm2_pkcs11.so.1.9.0 \
  --list-token-slots
```

¹⁶ <https://github.com/tpm2-software/tpm2-pkcs11>

Using TPM with PKCS11

After initializing the token, initialize the user PIN. All further operations use this PIN:

```
1 # Initialize the user pin to "5678" using the previously set so-pin "1234"
2 pkcs11-tool --module \
  /usr/lib/aarch64-linux-gnu/pkcs11/libtpm2_pkcs11.so.1.9.0 --slot-index=0 \
  --init-pin --so-pin="1234" --login --pin="5678"
```

In the next step, create a cryptographic key pair based on ECC P256 with the label “mykey”. Initially the list of objects is empty. After generating the key pair, the public key becomes publicly visible. To also see the private key, provide a login:

```
1 # List objects (empty list)
2 pkcs11-tool --module \
  /usr/lib/aarch64-linux-gnu/pkcs11/libtpm2_pkcs11.so.1.9.0 --slot-index=0 \
  --list-objects

4 # Create a keypair labeled "mykey"
5 pkcs11-tool --module \
  /usr/lib/aarch64-linux-gnu/pkcs11/libtpm2_pkcs11.so.1.9.0 --slot-index=0 \
  --login --pin="5678" --label="mykey" --keypairgen --usage-sign --key-type \
  EC:prime256v1

7 # List objects (only public key visible)
8 pkcs11-tool --module \
  /usr/lib/aarch64-linux-gnu/pkcs11/libtpm2_pkcs11.so.1.9.0 --slot-index=0 \
  --list-objects

10 # List objects after a login (also private key visible)
11 pkcs11-tool --module \
  /usr/lib/aarch64-linux-gnu/pkcs11/libtpm2_pkcs11.so.1.9.0 --slot-index=0 \
  --list-objects --login --pin 5678
```

To use the new key pair on the virtual smart card, export the public key. Then sign a file containing testdata. You can then verify the signature using the `openssl` command and the exported public key.

```
1 # Export the public key
2 pkcs11-tool --module \
  /usr/lib/aarch64-linux-gnu/pkcs11/libtpm2_pkcs11.so.1.9.0 --read-object \
  --slot-index=0 --label="mykey" --type pubkey --output-file pubkey.der
3 # Output file pubkey.der

5 # Sign a file
6 pkcs11-tool --module \
  /usr/lib/aarch64-linux-gnu/pkcs11/libtpm2_pkcs11.so.1.9.0 --slot-index=0 \
  --login --pin 5678 --sign --input-file testdata --output-file \
  testdata.sig --mechanism ECDSA-SHA256 --signature-format openssl
7 # Input file testdata - Output file testdata.sig

9 # Convert DER to PEM for OpenSSL
10 openssl pkey -inform DER -pubin -in pubkey.der -out pubkey.pem -pubin
11 # Input file pubkey.der - Output file pubkey.pem
```

Using TPM with PKCS11

```

13 # Verify the signature
14 openssl dgst -sha256 -verify pubkey.pem -signature testdata.sig testdata
15 # Input files pubkey.pem, testdata.sig, testdata

```

The corresponding terminal outputs should look as follows:

```

1  $ pkcs11-tool --module \
   /usr/lib/aarch64-linux-gnu/pkcs11/libtpm2_pkcs11.so.1.9.0 \
   --list-token-slots
2  Available slots:
3  Slot 0 (0x1):
4   token state:  uninitialized
5  $ pkcs11-tool --module \
   /usr/lib/aarch64-linux-gnu/pkcs11/libtpm2_pkcs11.so.1.9.0 --slot-index=0 \
   --init-token --label="mytoken" --so-pin="1234"
6  Using slot with index 0 (0x1)
7  Token successfully initialized
8  $ pkcs11-tool --module \
   /usr/lib/aarch64-linux-gnu/pkcs11/libtpm2_pkcs11.so.1.9.0 \
   --list-token-slots
9  Available slots:
10 Slot 0 (0x1): mytoken
11 token label      : mytoken
12 token manufacturer : Infineon
13 token model      : SLB9672
14 token flags      : login required, rng, token initialized, PIN \
   initialized
15 hardware version  : 1.59
16 firmware version  : 15.22
17 serial num       : 0000000000000000
18 pin min/max      : 0/128
19 Slot 1 (0x2):
20 token state:  uninitialized
21 $ pkcs11-tool --module \
   /usr/lib/aarch64-linux-gnu/pkcs11/libtpm2_pkcs11.so.1.9.0 --slot-index=0 \
   --init-pin --so-pin="1234" --login --pin="5678"
22 Using slot with index 0 (0x1)
23 User PIN successfully initialized
24 $ pkcs11-tool --module \
   /usr/lib/aarch64-linux-gnu/pkcs11/libtpm2_pkcs11.so.1.9.0 --slot-index=0 \
   --list-objects
25 Using slot with index 0 (0x1)
26 $ pkcs11-tool --module \
   /usr/lib/aarch64-linux-gnu/pkcs11/libtpm2_pkcs11.so.1.9.0 --slot-index=0 \
   --login --pin="5678" --label="mykey" --keypairgen --usage-sign --key-type \
   EC:prime256v1
27 Using slot with index 0 (0x1)
28 Key pair generated:
29 Private Key Object; EC
30 label:  mykey

```

Using TPM with PKCS11

```

31  Usage:      decrypt, sign
32  Access:     sensitive, always sensitive, never extractable, local
33  Allowed mechanisms: ECDSA, ECDSA-SHA1, ECDSA-SHA256, ECDSA-SHA384, ECDSA-SHA512
34  Public Key Object; EC  EC_POINT 256 bits
35  EC_POINT:   04410461f8f0482ba24d6594bffd40540762c3b48f640bc6294ddd60a...
36  EC_PARAMS:  06082a8648ce3d030107
37  label:      mykey
38  Usage:      encrypt, verify
39  Access:     local
40  $ pkcs11-tool --module \
    /usr/lib/aarch64-linux-gnu/pkcs11/libtpm2_pkcs11.so.1.9.0 --slot-index=0 \
    --list-objects
41  Using slot with index 0 (0x1)
42  Public Key Object; EC  EC_POINT 256 bits
43  EC_POINT:   04410461f8f0482ba24d6594bffd40540762c3b48f640bc6294ddd60a...
44  EC_PARAMS:  06082a8648ce3d030107
45  label:      mykey
46  Usage:      encrypt, verify
47  Access:     local
48  $ pkcs11-tool --module \
    /usr/lib/aarch64-linux-gnu/pkcs11/libtpm2_pkcs11.so.1.9.0 --slot-index=0 \
    --list-objects --login --pin 5678
49  Using slot with index 0 (0x1)
50  Public Key Object; EC  EC_POINT 256 bits
51  EC_POINT:   04410461f8f0482ba24d6594bffd40540762c3b48f640bc6294ddd60a...
52  EC_PARAMS:  06082a8648ce3d030107
53  label:      mykey
54  Usage:      encrypt, verify
55  Access:     local
56  Private Key Object; EC
57  label:      mykey
58  Usage:      decrypt, sign
59  Access:     sensitive, always sensitive, never extractable, local
60  Allowed mechanisms: ECDSA, ECDSA-SHA1, ECDSA-SHA256, ECDSA-SHA384, ECDSA-SHA512
61  $ pkcs11-tool --module \
    /usr/lib/aarch64-linux-gnu/pkcs11/libtpm2_pkcs11.so.1.9.0 --read-object \
    --slot-index=0 --label="mykey" --type pubkey --output-file pubkey.der
62  Using slot with index 0 (0x1)
63  $ pkcs11-tool --module \
    /usr/lib/aarch64-linux-gnu/pkcs11/libtpm2_pkcs11.so.1.9.0 --slot-index=0 \
    --login --pin 5678 --sign --input-file testdata --output-file \
    testdata.sig --mechanism ECDSA-SHA256 --signature-format openssl
64  Using slot with index 0 (0x1)
65  Using signature algorithm ECDSA-SHA256
66  $ openssl pkey -inform DER -pubin -in pubkey.der -out pubkey.pem -pubin
67  $ openssl dgst -sha256 -verify pubkey.pem -signature testdata.sig testdata
68  Verified OK

```

Using TPM for HDD encryption

8 Using TPM for HDD encryption

First, install the cryptsetup package. You can use it to create the encrypted drive.

```
1 sudo apt --yes install cryptsetup systemd-cryptsetup
```

systemd-cryptenroll uses tpm2-tss, which calls OpenSSL to generate ephemeral keys when it establishes a session with the TPM. If you globally activate the tpm2-openssl provider as described in the *Using TPM with OpenSSL* (page 17) section, the provider intercepts these key generation calls. The provider tries to perform key generation on the TPM before OpenSSL finishes establishing the session. This sequence causes systemd-cryptenroll to fail. If you use tpm2-tss versions older than 4.2.0, such as on Raspberry Pi OS, deactivate the tpm2-openssl provider. Open /etc/ssl/openssl.cnf and make the following modifications:

```
1 sudo nano /etc/ssl/openssl.cnf
2 # Exit the editor using CTRL+x -> y
```

In the provider section, set activate = 0 for the tpm2-openssl provider module in /etc/ssl/openssl.cnf:

```
1 [provider_sect]
2 default = default_sect
3 tpm2provider = tpm2provider_sect

5 [default_sect]
6 activate = 1

8 [tpm2provider_sect]
9 # Change the following line
10 activate = 0
11 module = /usr/lib/aarch64-linux-gnu/openssl-modules/tpm2.so
```

For simplicity, this tutorial uses a virtual disk instead of a physical disk partition. It creates a 50 MB file and maps it to the /dev/loop1 loopback device. This device then behaves like a regular disk partition such as /dev/sda3.

```
1 # Create an empty 50MB file
2 dd if=/dev/zero of=virtual-disk.img bs=1M count=50
3 # Output file virtual-disk.img

5 # Create a loopback device
6 sudo losetup /dev/loop1 virtual-disk.img
7 # Input file virtual-disk.img - Output pseudo-file /dev/loop1
```

Next, initialize the virtual disk partition as an encrypted device. First, format and open it. Then create the file system and mount the partition at /virtual-disk/.

```
1 # Initialize the encrypted device
2 sudo cryptsetup -y luksFormat /dev/loop1
3 # Input pseudo-file /dev/loop1

5 # Open the encrypted device
6 sudo cryptsetup open /dev/loop1 virtual-disk
7 # Input pseudo-file /dev/loop1 - Output pseudo-file /dev/mapper/virtual-disk

9 # Create a file system
```

Using TPM for HDD encryption

```
10 sudo mkfs.ext4 /dev/mapper/virtual-disk
11 # Input pseudo-file /dev/mapper/virtual-disk

13 # Create a directory for mounting
14 sudo mkdir -p /virtual-disk
15 # Output directory /virtual-disk

17 # Mount the filesystem
18 sudo mount /dev/mapper/virtual-disk /virtual-disk/
19 # Input pseudo-file /dev/mapper/virtual-disk - Input/Output directory \
    /virtual-disk
```

Now, create a file with arbitrary content to demonstrate an encrypted file.

```
1 # Create a file for testing
2 echo "This is a testfile" | sudo tee /virtual-disk/testfile
3 # Output file /virtual-disk/testfile
```

Then, unmount the encrypted disk and close the encrypted container.

```
1 # Unmount the filesystem
2 sudo umount /virtual-disk

4 # Close the encrypted drive
5 sudo cryptsetup close virtual-disk
6 # Output pseudo file removed /dev/mapper/virtual-disk
```

Now add TPM support as an additional authentication slot. The password from the beginning remains valid.

```
1 # Add TPM support to encrypted container
2 sudo systemd-cryptenroll --tpm2-device=auto --tpm2-with-pin=true /dev/loop1

4 # To replace an existing TPM slot, add --wipe-slot=tpm2
5 # Omit --tpm2-with-pin=true if you do not want to require a PIN.
```

To use the encrypted partition, run the following commands:

```
1 # Open the encrypted device
2 sudo systemd-cryptsetup attach virtual-disk /dev/loop1
3 # Input pseudo-file /dev/loop1 - Output pseudo-file /dev/mapper/virtual-disk
4 # Note, we are using systemd-cryptsetup instead of cryptsetup in this step

6 # Mount the file system
7 sudo mount /dev/mapper/virtual-disk /virtual-disk/
8 # Input pseudo-file /dev/mapper/virtual-disk - Input/Output directory \
    /virtual-disk

10 # Dump the content of the test file
11 cat /virtual-disk/testfile
12 # Input file /virtual-disk/testfile
```

Note: If you want the encrypted partition to open during boot, add the required information to `/etc/crypttab` and include the TSS libraries in the initial ramdisk.

Using TPM for HDD encryption

To unmount the file system, close the encrypted mapping, and detach the loop device, run:

```
1 sudo umount /virtual-disk
2 sudo /usr/lib/systemd/systemd-cryptsetup detach virtual-disk
3 sudo losetup --detach /dev/loop1
```

The expected result from the above command is as follows:

```
1 $ dd if=/dev/zero of=virtual-disk.img bs=1M count=50
2 50+0 records in
3 50+0 records out
4 52428800 bytes (52 MB, 50 MiB) copied, 0.0169446 s, 3.1 GB/s
5 $ sudo losetup /dev/loop1 virtual-disk.img
6 $ sudo cryptsetup -y luksFormat /dev/loop1

8 WARNING!
9 =====
10 This will overwrite data on /dev/loop1 irrevocably.

12 Are you sure? (Type 'yes' in capital letters): YES
13 Enter passphrase for /home/pi/virtual-disk.img:
14 Verify passphrase:
15 $ sudo cryptsetup open /dev/loop1 virtual-disk
16 Enter passphrase for /home/pi/virtual-disk.img:
17 $ sudo mkfs.ext4 /dev/mapper/virtual-disk
18 mke2fs 1.47.2 (1-Jan-2025)
19 Creating filesystem with 34816 1k blocks and 8720 inodes
20 Filesystem UUID: dcacd9b0-7839-4367-a752-3935ebaaf866
21 Superblock backups stored on blocks:
22     8193, 24577

24 Allocating group tables: done
25 Writing inode tables: done
26 Creating journal (4096 blocks): done
27 Writing superblocks and filesystem accounting information: done

29 $ sudo mkdir -p /virtual-disk
30 $ sudo mount /dev/mapper/virtual-disk /virtual-disk/
31 $ echo "This is a testfile" | sudo tee /virtual-disk/testfile
32 This is a testfile
33 $ sudo umount /virtual-disk
34 $ sudo cryptsetup close virtual-disk
35 $ sudo systemd-cryptenroll --tpm2-device=auto --tpm2-with-pin=true /dev/loop1
36 Please enter current passphrase for disk /dev/loop1: *****
37 Please enter TPM2 PIN: ****
38 Please enter TPM2 PIN (repeat): ****
39 TPM2 device supports SHA256 PCR bank but none of the selected PCRs are \
   valid! Firmware apparently did not initialize any of the selected PCRs. \
   Proceeding anyway with SHA256 bank. PCR policy effectively unenforced!
40 New TPM2 token enrolled as key slot 1.
41 $ sudo systemd-cryptsetup attach virtual-disk /dev/loop1
```

Using TPM for HDD encryption

```
42 Please enter LUKS2 token PIN: ****
43 $ sudo mount /dev/mapper/virtual-disk /virtual-disk/
44 $ cat /virtual-disk/testfile
45 This is a testfile
46 $ sudo umount /virtual-disk
47 $ sudo /usr/lib/systemd/systemd-cryptsetup detach virtual-disk
48 $ sudo losetup --detach /dev/loop1
```

If you leave the `tpm2-openssl` provider enabled, `systemd-cryptenroll` shows the following error message:

```
1 $ sudo systemd-cryptenroll --tpm2-device=auto --tpm2-with-pin=true /dev/loop1
2 Please enter current passphrase for disk /dev/loop1: ****
3 Please enter TPM2 PIN: ****
4 Please enter TPM2 PIN (repeat): ****
5 TPM2 device supports SHA256 PCR bank but none of the selected PCRs are \
   valid! Firmware apparently did not initialize any of the selected PCRs. \
   Proceeding anyway with SHA256 bank. PCR policy effectively unenforced!
6 ERROR:esys_crypto:src/tss2-esys/esys_crypto_oss.c:986:iesys_cryptoss_get_ecdh_point()
   ErrorCode (0x00070001) Get ephemeral key
7 ERROR:esys:src/tss2-esys/esys_iutil.c:566:iesys_compute_encrypted_salt() \
   During computation of ECC public key. ErrorCode (0x00070001)
8 ERROR:esys:src/tss2-esys/api/Esys_StartAuthSession.c:226:Esys_StartAuthSession_Async()
   Error in parameter encryption. ErrorCode (0x00070001)
9 ERROR:esys:src/tss2-esys/api/Esys_StartAuthSession.c:113:Esys_StartAuthSession() \
   Error in async function ErrorCode (0x00070001)
10 Failed to seal to TPM2: State not recoverable
```

Appendix

9 Appendix

9.1 Troubleshooting

/dev/tpm0 does not appear

The OPTIGA™ TPM SLB 9673 RPI evaluation board requires Linux kernel version 6.0 or later. Check the version with:

```
1 uname --kernel-release
```

Then update the kernel to the latest version with:

```
1 sudo rpi-update
```

TPM return code decoding

The TPM Library specification Part 2, Section 6.6.3 “TPM_RC Values”¹⁷ documents TPM return codes. The TCG TSS 2.0 Overview and Common Structures Specification, Section 3.4 tss2_common.h Common Return Codes¹⁸ documents TSS return codes.

The Linux tool `tpm2_rc_decode`¹⁹, which is part of `tpm2-tools`²⁰, decodes these errors and also includes TSS error codes.

There is also a Windows Store tool called TPM Return Code Decoder²¹, but it does not decode TSS error codes. Note: Ignore responses that contain “(1.2)” because those refer to TPM 1.2. Only use lines marked “(2.0)”.

Permission problems

If you receive an error similar to this:

```
1 ERROR:tcti:src/tss2-tcti/tctildr-dl.c:254:tctildr_get_default() No standard \
  TCTI could be loaded
2 ERROR:tcti:src/tss2-tcti/tctildr.c:416:Tss2_TctiLdr_Initialize_Ex() Failed \
  to instantiate TCTI
3 ERROR: Could not load tcti, got: "(null)"
```

This usually indicates that the current user has no permission to access the TPM.

Check the permissions of the device file:

```
1 ls -l /dev/tpm*
```

The result should look like this:

```
1 $ ls -l /dev/tpm*
2 crw-rw---- 1 tss root 10, 224 Aug 26 12:42 /dev/tpm0
3 crw-rw---- 1 tss tss 236, 65536 Aug 26 12:42 /dev/tpmrm0
```

If the owner of these devices is `root`, install the `tpm-udev` package and restart the system.

Next check that the current user is a member of the `tss` group:

¹⁷ https://trustedcomputinggroup.org/wp-content/uploads/TCG_TPM2_r1p59_Part2_Structures_pub.pdf

¹⁸ https://trustedcomputinggroup.org/wp-content/uploads/TSS_Overview_Common_v1_r10_pub09232021.pdf

¹⁹ https://tpm2-tools.readthedocs.io/en/latest/man/tpm2_rc_decode.1/

²⁰ <https://github.com/tpm2-software/tpm2-tools>

²¹ <https://apps.microsoft.com/store/detail/tpm-return-code-decoder/9NBLGGH4V8NP>

Appendix

```
1 whoami
2 id --groups --name $USER
```

The result for the pi username should look like this:

```
1 $ whoami
2 pi
3 $ id --groups --name $USER
4 ... tss ...
```

To add a user to the group tss, run:

```
1 sudo usermod --append --groups tss $USER
```

Also note that a change of group memberships only becomes effective after logging out and back in again.

Driver-related problems

Examples include a failed driver load, a missing /dev/tpm0 device file, or a TPM that is not detected. Verify that the TPM is properly connected. Check if your kernel has TPM support by running the following command:

```
1 zgrep --ignore-case tpm /proc/config.gz
```

The output should look like this:

```
1 CONFIG_HW_RANDOM_TPM=m
2 CONFIG_TCG_TPM=m
3 CONFIG_TCG_TIS_SPI=m
```

For the OPTIGA™ TPM SLB 9673, the output must also contain:

```
1 CONFIG_TCG_TIS_I2C=m
```

You may need to run `modprobe configs` first to create the /proc/config.gz archive. A software upgrade can switch the system to the wrong kernel. In that case, reinstall the self-compiled kernel.

- Check that you are using the correct kernel (it should be your self-built kernel) by running `uname -a` or `cat /proc/version`
- The driver is loaded. You can try to load it manually with `sudo modprobe tpm_tis_spi` to determine whether the automatic driver load mechanism is failing or whether the driver does not run at all

Login via console

If you experience problems with SSH communication, connect to the Raspberry Pi using a serial UART connection. Edit the following two files:

```
1 #/boot/firmware/config.txt
3 [all]
4 # Enable a Linux console
5 enable_uart=1
```

Appendix

```
1 #/boot/firmware/cmdline.txt
3 #ensure the following statements are included
4 console=serial0,115200 console=tty1
```

Boot logs

To retrieve early boot logs from the Raspberry Pi, enable UART as follows:

```
1 #/boot/firmware/config.txt
3 [all]
4 # Enable a Linux console
5 enable_uart=1
6 # Enable boot log messages from ROM code
7 uart_2ndstage=1
8 # Disable Bluetooth(R), since it shares PINs with UART
9 dtoverlay=disable-bt

1 #/boot/firmware/cmdline.txt
2 # remove quiet and splash
```

Linux logs

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

Checking kernel messages:

```
1 dmesg | grep tpm
2 grep tpm /var/log/messages
3 grep tpm /var/log/syslog
```

Checking whether the TPM device driver has been loaded:

```
1 ls /dev/tpm*
```

Check the Linux kernel device tree:

```
1 find /proc/device-tree/ | grep -i -e tpm -e slb
```

Disable automatic device tree loading

Disable the automatic configuration of the device tree with the TPM driver information by editing the following file:

```
1 #/boot/firmware/config.txt
3 [all]
4 force_eeprom_read=0
```

Appendix

SPI low-level analysis using libsigrok

The Sigrok project with its pulse-view front-end for a large number of logic analysers has a pending contribution that enables analysis of the TPM communication protocol. Retrieve it from the corresponding [pull request](#)²².

9.2 Support on other Linux distributions

Ubuntu 20.04

On Ubuntu 20.04, find the configuration file at `/boot/firmware/config.txt`.

Also, Ubuntu 20.04 does not contain the driver module in its initrd. To enable it, run:

```
1 echo tpm_tis_spi | sudo tee -a /etc/initramfs-tools/modules
2 sudo update-initramfs -k all -c
```

Enabling Device Tree for SPI

Install the device tree compiler:

```
1 sudo apt --yes install device-tree-compiler
```

Create a Device Tree with the following content, e.g. `tpm-slb9670.dts`:

```
1 /dts-v1/;
3 / {
4     compatible = "brcm,bcm2835";
6     fragment@0 {
7         target = <0xffffffff>;
9         __overlay__ {
10             status = "okay";
11         };
12     };
14     fragment@1 {
15         target = <0xffffffff>;
17         __overlay__ {
18             status = "disabled";
19         };
20     };
22     fragment@2 {
23         target = <0xffffffff>;
25         __overlay__ {
26             #address-cells = <0x01>;
27             #size-cells = <0x00>;
```

²² <https://github.com/sigrokproject/libsigrokdecode/pull/88>

Appendix

```

28         slb9670: slb9670@1 {
29             compatible = "infineon,slb9672", \
30             "infineon,slb9670", "tcg,tpm_tis-spi";
31             reg = <0x01>;
32             #address-cells = <0x01>;
33             #size-cells = <0x00>;
34             spi-max-frequency = <32000000>;
35             status = "okay";
36             phandle = <0x01>;
37         };
38     };
39
40     __symbols__ {
41         slb9670 = "/fragment@2/__overlay__/slb9670@1";
42     };
43
44     __fixups__ {
45         spi0 = "/fragment@0:target:0\0/fragment@2:target:0";
46         spidev1 = "/fragment@1:target:0";
47     };
48 };

```

Compile the device tree

```
1 dtc -@ -I dts -O dtb -o tpm-slb9670.dtbo tpm-slb9670.dts
```

For testing, load this dtbo directly:

```
1 sudo dtoverlay ./tpm-slb9670.dtbo
```

Copy the device tree to the boot directory:

```
1 sudo cp tpm-slb9670.dtbo /boot/firmware/overlays
```

Open the config.txt file in an editor.

```
1 sudo nano /boot/firmware/config.txt
2 # Exit the editor using CTRL+x -> y
```

In this file insert following two lines (slb9670 also works for OPTIGA™ TPM SLB 9672):

```
1 dtparam=spi=on
2 dtoverlay=tpm-slb9670
```

Enabling Device Tree for I2C

Install the device tree compiler:

```
1 sudo apt --yes install device-tree-compiler
```

Note: I²C clock stretching is mandatory for TCG-compliant TPM communication. When porting to platforms other than the Raspberry Pi, verify that the host I²C controller correctly supports clock stretching before enabling hardware I²C.

Appendix

On older Raspberry Pis, the hardware I²C controller has a [clock stretching bug](https://github.com/raspberrypi/linux/issues/4884)²³: it ignores clock stretches shorter than half an I²C clock cycle, violating this requirement. To work around this, disable the hardware I²C peripheral and use a software-based I²C implementation instead.

On Raspberry Pis with the RP1 chip, like the Raspberry Pi 5, the hardware I²C peripheral correctly supports clock stretching. We recommend using hardware I²C on these platforms for improved performance, full TCG compliance, and efficient hardware offloading. Create a device tree with the following content, e.g. `tpm-slb9673.dts`:

```

1 // Minimal TPM SLB9673 overlay for Raspberry Pi 5
2 // Native Hardware I2C1 on GPIO2 (SDA) / GPIO3 (SCL) @ 1MHz

4 /dts-v1/;
5 /plugin/;

7 / {
8     compatible = "brcm,bcm2712";

10     /* enable the I2C1 Hardware Controller and set frequencies */
11     fragment@0 {
12         target-path = "/axi/pcie@120000/rp1/i2c@74000";
13         __overlay__ {
14             clock-frequency = <1000000>;
15             status = "okay";
16         };
17     };

19     /* TPM Device Node on Hardware I2C1 */
20     fragment@1 {
21         target-path = "/axi/pcie@120000/rp1/i2c@74000";
22         __overlay__ {
23             #address-cells = <1>;
24             #size-cells = <0>;

26             slb9673: slb9673@2e {
27                 compatible = "infineon,slb9673",
28                             "tcg,tpm-tis-i2c";
29                 reg = <0x2e>;
30                 i2c-transfer-timeout-ms = <5000>;
31                 status = "okay";
32             };
33         };
34     };
35 };

```

For older Raspberry Pi revisions, use the following device tree overlay. Create a device tree with the following content, e.g. `tpm-slb9673.dts`:

```

1 /*
2  * Device Tree overlay for the Infineon OPTIGA(tm) TPM SLB 9673 Trusted \
   Platform Module

```

²³ <https://github.com/raspberrypi/linux/issues/4884>

Appendix

```

3  * add-on boards, which can be used as a secure key storage and hwrng.
4  */

6  /dts-v1/;
7  /plugin/;

9  / {
10     compatible = "brcm,bcm2835", "brcm,bcm2708", "brcm,bcm2709";

12     /* The hardware I2C controller ignores clock stretches shorter than half
13        an I2C clock cycle (https://github.com/raspberrypi/linux/issues/4884).
14        Since clock stretching is mandatory for TCG-compliant I2C TPM
15        communication, disable hardware I2C and use software I2C instead. */
16     fragment@0 {
17         target = <&i2c_arm>;
18         __overlay__ {
19             status = "disabled";
20         };
21     };

23     fragment@1 {
24         target-path = "/";
25         __overlay__ {
26             i2c1: i2c-gpio@1 {
27                 #address-cells = <1>;
28                 #size-cells = <0>;
29                 compatible = "i2c-gpio";
30                 gpios = <&gpio 2 6>, /* SDA GPIO_OPEN_DRAIN */
31                       <&gpio 3 6>; /* CLK GPIO_OPEN_DRAIN */
32                 clock-frequency = <4000000>;
33                 i2c-gpio,delay-us = <0x01>;
34                 status = "okay";
35             };
36         };
37     };

39     /* Add the TPM */
40     fragment@2 {
41         target = <&i2c1>;
42         __overlay__ {
43             #address-cells = <1>;
44             #size-cells = <0>;

46             slb9673: slb9673@2e {
47                 compatible = "infineon,slb9673", "tcg,tpm-tis-i2c";
48                 reg = <0x2e>;
49                 status = "okay";
50             };
51         };

```

Appendix

```
52     };  
53 };
```

Compile the device tree:

```
1 dtc -@ -I dts -O dtb -o tpm-slb9673.dtb tpm-slb9673.dts
```

Copy the device tree to the boot directory:

```
1 sudo cp tpm-slb9673.dtb /boot/firmware/overlays
```

Open the config.txt file in an editor.

```
1 sudo nano /boot/firmware/config.txt  
2 # Exit the editor using CTRL+x -> y
```

In this file insert the following line:

```
1 dtoverlay=tpm-slb9673
```

9.3 Compiling Linux with TPM2 driver (for Raspberry Pi OS)

This is only necessary for system that do not already have the driver compiled as module or statically.

Installing the build dependencies

```
1 sudo apt --yes install git bc bison flex libssl-dev make
```

Downloading the Linux kernel source code

```
1 # git clone --depth=1 --branch <branch> \  
   https://github.com/raspberrypi/linux with <branch> replaced; currently \  
   this is rpi-6.18.y  
2 git clone --depth=1 https://github.com/raspberrypi/linux  
3 cd linux
```

Configuring the Linux kernel

Raspberry Pi 3, 3+, 4, 400, Zero 2 W in 64 Bit:

```
1 export KERNEL=kernel8  
2 make bcm2711_defconfig
```

Raspberry Pi 4, 400 in 32 Bit:

```
1 export KERNEL=kernel7l  
2 make bcm2711_defconfig
```

Raspberry Pi 3, 3+, Zero 2 W in 32 Bit:

```
1 export KERNEL=kernel7  
2 make bcm2709_defconfig
```

Appendix

Checking the Linux kernel configuration for OPTIGA™ TPM SLB 9672 and SLx 9670 (SPI)

Check whether the driver is enabled.

```
1 grep -e CONFIG_TCG_TIS .config
```

This should result in either of these results:

```
1 CONFIG_TCG_TIS_SPI=m
2 CONFIG_TCG_TIS_SPI=y
```

If instead you see the following, enable the driver in the kernel's configuration:

```
1 # CONFIG_TCG_TIS_SPI is not set
```

Checking the Linux kernel configuration for OPTIGA™ TPM SLB 9673 (I²C)

Check whether the driver is enabled.

```
1 grep -e CONFIG_TCG_TIS .config
```

This should result in either of these results:

```
1 CONFIG_TCG_TIS_I2C=m
2 CONFIG_TCG_TIS_I2C=y
```

If instead you see the following, enable the driver in the kernel's configuration:

```
1 # CONFIG_TCG_TIS_I2C is not set
```

Building the Linux kernel

For 64-Bit systems use:

```
1 make -j4 Image.gz modules dtbs
```

For 32-Bit systems use:

```
1 make -j4 zImage modules dtbs
```

Installing the Linux kernel

For 64-Bit systems use:

```
1 sudo make modules_install
2 sudo cp /boot/$KERNEL.img /boot/$KERNEL-backup.img
3 sudo cp arch/arm64/boot/Image.gz /boot/$KERNEL.img
4 sudo cp arch/arm64/boot/dts/broadcom/*.dtb /boot/
5 sudo cp arch/arm64/boot/dts/overlays/*.dtb* /boot/firmware/overlays/
6 sudo cp arch/arm64/boot/dts/overlays/README /boot/firmware/overlays/
```

For 32-Bit systems use:

Appendix

```

1 sudo make modules_install
2 sudo cp /boot/$KERNEL.img /boot/$KERNEL-backup.img
3 sudo cp arch/arm/boot/zImage /boot/$KERNEL.img
4 sudo cp arch/arm/boot/dts/*.dtb /boot/
5 sudo cp arch/arm/boot/dts/overlays/*.dtb* /boot/firmware/overlays/
6 sudo cp arch/arm/boot/dts/overlays/README /boot/firmware/overlays/

```

Note: If the newly built kernel will not boot, choose the old kernel by adding one of the following lines to the file `config.txt` on the FAT32 boot partition:

```

1 image=kernel8-backup.img
2 image=kernel7l-backup.img
3 image=kernel7-backup.img

```

Cross-Compiling

To cross-compile, additionally install the following packages:

```

1 sudo apt --yes install crossbuild-essential-armhf crossbuild-essential-arm64

```

Use the following for 64-Bit:

```

1 KERNEL=kernel8
2 make ARCH=arm64 CROSS_COMPILE=aarch64-linux-gnu- bcm2711_defconfig
3 make ARCH=arm64 CROSS_COMPILE=aarch64-linux-gnu- Image modules dtbs
4 make ARCH=arm64 CROSS_COMPILE=aarch64-linux-gnu- INSTALL_MOD_PATH=modInst \
  modules_install

6 #mount the boot and root partitions to /mnt/fat32 and /mnt/ext4
7 sudo cp mnt/fat32/$KERNEL.img mnt/fat32/$KERNEL-backup.img
8 sudo cp arch/arm64/boot/Image mnt/fat32/$KERNEL.img
9 sudo cp arch/arm64/boot/dts/broadcom/*.dtb mnt/fat32/
10 sudo cp arch/arm64/boot/dts/overlays/*.dtb* mnt/fat32/overlays/
11 sudo cp arch/arm64/boot/dts/overlays/README mnt/fat32/overlays/
12 sudo cp -ar modInst/* /mnt/ext4

```

Use the following for 32-Bit Pi4, 400, CM4:

```

1 KERNEL=kernel7l
2 make ARCH=arm CROSS_COMPILE=arm-linux-gnueabihf- bcm2711_defconfig
3 make ARCH=arm CROSS_COMPILE=arm-linux-gnueabihf- zImage modules dtbs
4 make ARCH=arm CROSS_COMPILE=arm-linux-gnueabihf- INSTALL_MOD_PATH=modInst \
  modules_install

6 #mount the boot and root partitions to /mnt/fat32 and /mnt/ext4
7 sudo cp mnt/fat32/$KERNEL.img mnt/fat32/$KERNEL-backup.img
8 sudo cp arch/arm/boot/zImage mnt/fat32/$KERNEL.img
9 sudo cp arch/arm/boot/dts/*.dtb mnt/fat32/
10 sudo cp arch/arm/boot/dts/overlays/*.dtb* mnt/fat32/overlays/
11 sudo cp arch/arm/boot/dts/overlays/README mnt/fat32/overlays/
12 sudo cp -ar modInst/* /mnt/ext4

```

Appendix

Use the following for 32-Bit Pi2, 3, 3+, Zero 2 W, CM3, CM3+:

```

1 KERNEL=kernel7
2 make ARCH=arm CROSS_COMPILE=arm-linux-gnueabihf- bcm2709_defconfig
3 make ARCH=arm CROSS_COMPILE=arm-linux-gnueabihf- zImage modules dtbs
4 make ARCH=arm CROSS_COMPILE=arm-linux-gnueabihf- INSTALL_MOD_PATH=modInst \
  modules_install

6 #mount the boot and root partitions to /mnt/fat32 and /mnt/ext4
7 sudo cp mnt/fat32/$KERNEL.img mnt/fat32/$KERNEL-backup.img
8 sudo cp arch/arm/boot/zImage mnt/fat32/$KERNEL.img
9 sudo cp arch/arm/boot/dts/*.dtb mnt/fat32/
10 sudo cp arch/arm/boot/dts/overlays/*.dtb* mnt/fat32/overlays/
11 sudo cp arch/arm/boot/dts/overlays/README mnt/fat32/overlays/
12 sudo cp -ar modInst/* /mnt/ext4

```

If you are cross-compiling from WSL you cannot access the ext4 file system yet. Therefore, add the modules into a tar archive and transfer them via the /boot partition:

```

1 (cd modInst/lib/modules && tar cjf ../../../../modInst.tar.bz2 ./*)

```

Then copy the \$KERNEL.img all *.dtb* files and modInst.tar.bz2 to a subfolder of the /boot mount (we will call this folder kernelUpdate). On the Raspberry Pi copy everything to the target locations:

```

1 KERNEL=kernel8 # or KERNEL=kernel7l or KERNEL=kernel7

3 (cd /lib/modules && sudo tar xjf /boot/kernelUpdate/modInst.tar.bz2 && sudo \
  chown -R root:root .)

5 sudo cp /boot/$KERNEL.img /boot/$KERNEL-backup.img
6 sudo cp /boot/kernelUpdate/$KERNEL.img /boot/$KERNEL.img
7 sudo cp /boot/kernelUpdate/*.dtb /boot/
8 sudo cp /boot/kernelUpdate/overlays/*.dtb* /boot/firmware/overlays/
9 sudo cp /boot/kernelUpdate/overlays/README /boot/firmware/overlays/

```

9.4 Compiling the TSS from source

Compiling tpm2-tss from source

The latest version of the tpm2-tss (the project for the libtss2-* packages) can be found in the [GitHub repository](https://github.com/tpm2-software/tpm2-tss)²⁴. This package provides a middleware for interaction with the TPM based on a C API. Documentation can be found on [readthedocs](https://tpm2-tss.readthedocs.io)²⁵.

Compiling tpm2-tools from source

The latest version of the tpm2-tools can be found in the [GitHub repository](https://github.com/tpm2-software/tpm2-tools)²⁶. This package provides a set of command line utilities for interacting with the TPM via the tpm2-tss. Documentation can be found on [readthedocs](https://tpm2-tools.readthedocs.io)²⁷.

²⁴ <https://github.com/tpm2-software/tpm2-tss>

²⁵ <https://tpm2-tss.readthedocs.io>

²⁶ <https://github.com/tpm2-software/tpm2-tools>

²⁷ <https://tpm2-tools.readthedocs.io>

Appendix

Compiling TPM support for OpenSSL from source

TPM support for OpenSSL can be found [here for 1.1.x²⁸](#) and [here for 3.x.x²⁹](#). These packages provide either an OpenSSL engine or an OpenSSL provider implementation that uses the TPM.

Compiling the TPM provider

Due to a bug in the official *Debian* package on older Raspberry Pi OS releases, you cannot install the `tpm2-openssl` package directly. Instead, compile the module manually:

```
1 # Install the dependencies of tpm2-openssl
2 sudo apt --yes install build-essential autoconf autoconf-archive libtool
3 # Retrieve the latest version of the source code
4 git clone --depth=1 https://github.com/tpm2-software/tpm2-openssl.git
5 cd tpm2-openssl
6 # Build the module
7 ./bootstrap
8 ./configure
9 make -j
10 # Install the module
11 sudo make install
```

9.5 Accessing the TPM with eltt2

The [Embedded Linux TPM Toolbox 2 \(ELTT2\)³⁰](#) is an Open Source tool provided by Infineon to enable basic TPM interaction.

You can use it as follows:

```
1 git clone --depth=1 https://github.com/Infineon/eltt2.git
2 cd eltt2
3 make
4 sudo ./eltt2 -g
```

For further information and other commands, refer to the [eltt2 README³¹](#).

9.6 Working with the Raspberry Pi

To interact with the Raspberry Pi, either attach a monitor and keyboard directly or use SSH. This section describes both options.

Direct interaction

To interact with the Raspberry Pi directly, attach a USB keyboard to one of its USB ports and connect a monitor through HDMI. This lets you view the complete boot process.

²⁸ <https://github.com/tpm2-software/tpm2-tss-engine>

²⁹ <https://github.com/tpm2-software/tpm2-openssl>

³⁰ <https://github.com/Infineon/eltt2>

³¹ <https://github.com/Infineon/eltt2#2-usage-of-eltt2>

Appendix

Connecting via SSH

To connect to the Raspberry Pi over Ethernet, use SSH. SSH also lets you open multiple simultaneous terminal sessions with the same user or different users. SSH is active by default. If you disabled SSH, reinstall it by running the following command from a terminal with an attached keyboard and monitor, as described in *Direct interaction* (page 44):

```
1 sudo apt --yes install ssh
```

To avoid warnings about missing system locales, especially if your PC uses locales that are not installed on the Raspberry Pi, disable the corresponding SSH option. Comment out the option in `/etc/ssh/ssh_config` on the developer PC:

```
1 nano /etc/ssh/ssh_config
2 # Exit the editor using CTRL+x -> y
```

Now comment out the `SendEnv` line, it should look like this:

```
1 # SendEnv LANG LC_*
```

To use SSH, connect both the developer PC and the Raspberry Pi to the same network. If your LAN has a DHCP server, both boards obtain an IP address automatically during boot. Otherwise, you must configure an IP address for each device manually.

For SSH to work, the IP addresses of the developer PC and the Raspberry Pi must be in the same subnet. For IPv4, this means the first three blocks must match. 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, whether dedicated or integrated into a router, handles this automatically. Keep this requirement in mind when assigning IP addresses manually.

Depending on your network configuration, the Raspberry Pi may also publish its host name on the LAN. You can retrieve the host name by running `hostname` on the Raspberry Pi. In that case, use SSH with the following command on the developer PC:

```
1 ssh [username]@[hostname]
```

On older Raspberry Pi systems, the default user is `pi`, the default host name is `raspberrypi`, and the default password for `pi` is `raspberrypi`. In that case, the SSH command is:

```
1 ssh pi@raspberrypi
```

If you cannot open a connection or your network does not support host-name resolution, use the IP address instead. You can get the IP address by running `ifconfig` on the Raspberry Pi. Then use the following SSH command on the developer PC:

```
1 ssh [username]@[IP Address]
```

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

```
1 exit
```

Exchanging Files

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

Appendix

Direct Copy on microSD card

You can insert the microSD card into a card reader attached to the developer PC. Ubuntu then mounts the partitions automatically so that you can copy files to them. Mounting the Raspberry Pi microSD card works reliably only on Linux-based operating systems because other operating systems may not be able to mount the ext4 partition or more than one partition on the card.

Copying via SCP

SCP uses SSH to copy files between two machines on a network. It requires a working SSH client on the local machine and a running SSH server on the remote machine. The syntax is as follows:

```
1 scp [source] [target]
```

[source] and [target] can be a normal file path for local files or must use the following format for remote files:
[remote username]@[remote hostname or IP address]:[remote file path]

Trademarks

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

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

Linux® is the registered trademark of Linus Torvalds in the U.S. and other countries.

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

Raspberry Pi is a trademark of Raspberry Pi Ltd.

This project is not affiliated with, produced by, or endorsed by the Debian project.

This project is not affiliated with, produced by, or endorsed by OpenSSL.

Edition May 29, 2026
Published by
Infineon Technologies AG
81726 Munich, Germany

© 2026 Infineon Technologies AG.
All Rights Reserved.

Do you have a question about this document?
Email: dsscusterservice@infineon.com

IMPORTANT NOTICE

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

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

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

WARNINGS

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

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