OPTIGA™ TPM Application Note

Linux Trusted Keys

Devices

- 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 an OPTIGA™ TPM SLx 9670 TPM2.0 can be integrated into a Raspberry Pi® to enable the Linux Trusted and Encrypted Keys.

Trusted Keys require the availability of a TPM to function for added security, while Encrypted Keys do not depend on a TPM but it can be protected by a specified master key. A master key can be a regular user-key or a trusted-key type. Encrypted Keys can be used by some useful subsystems, e.g., encrypted file system and Extended Verification Module (EVM), both will be covered in this document.

The OPTIGA™ TPM SLx 9670 TPM2.0 uses a SPI interface to communicate with the Raspberry Pi®. The OPTIGA™ TPM SLx 9670 TPM2.0 product family with SPI interface consists of 3 different products:

- OPTIGA™ TPM SLB 9670 TPM2.0 standard security applications
- OPTIGA™ TPM SLI 9670 TPM2.0 automotive security applications
- OPTIGA™ TPM SLM 9670 TPM2.0 industrial security applications

OPTIGA™ TPM SLx 9670 TPM2.0 products are fully TCG compliant TPM products with CC (EAL4+) and FIPS certification. The OPTIGA™ TPM SLx 9670 TPM2.0 products standard, automotive, and industrial differ with regards to supported temperature range, lifetime, quality grades, test environment, qualification, and reliability to fit the target applications requirements. An overview of all Infineon OPTIGA™ TPM products can be found on Infineon's website [2][3]. More information on TPM specification can be found on Trusted Computing Group (TCG) in reference [4].

Intended audience

This document is intended for customers who want to increase the security level of their platforms using a TPM 2.0 and like to evaluate the implementation of Linux Trusted and Encrypted Keys for their target applications.

Please read the Important Notice and Warnings at the end of this document

Table of contents

List of figures

List of tables

Acronyms and Abbreviations

Acronym	Definition
DEK	Disk Encryption Key
EVM	Extended Verification Module
IMA	Integrity Measurement Architecture
KMK	Kernel Master Key
Rootfs	Root file system
TPM	Trusted Platform Module

1 Prepare Raspberry Pi®

This section describes all the steps necessary for building a Raspberry Pi® bootable SD card image.

1.1 Prerequisites

- Raspberry Pi® 4
- Flash the Raspberry Pi® OS image (2020-08-20 release from [5]) on a micro-SD card (≥8GB)
- Host machine running Ubuntu 18.04 LTS
- OPTIGA™ TPM (TPM2.0)
    - SLB 9670
    - SLI 9670
    - SLM 9670

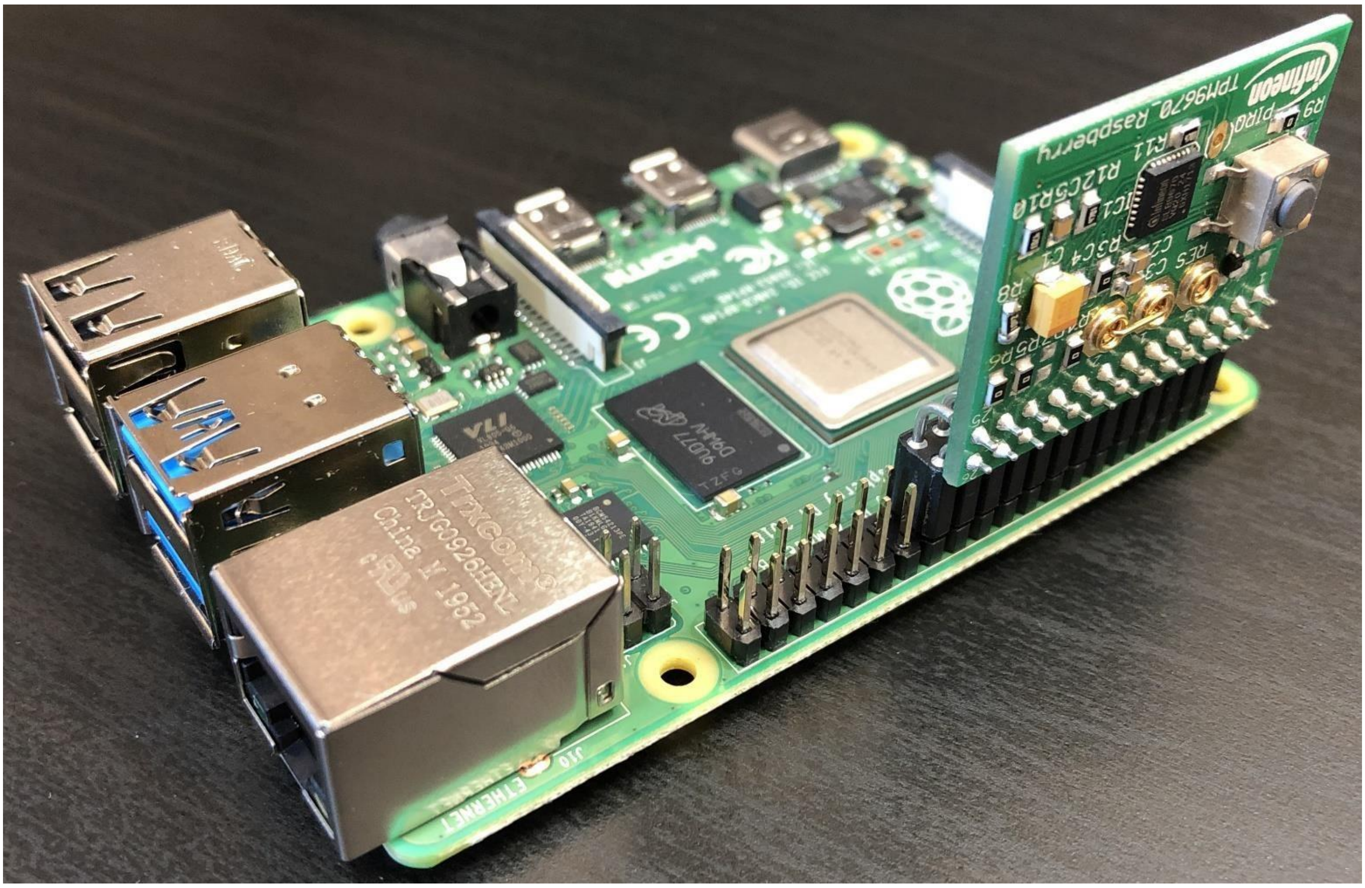

Figure 1 Infineon Iridium SLx 9670 TPM2.0 SPI Board on Raspberry Pi® 4

1.2 Kernel Build Guide

This guide is for cross-compilation only. Optionally, native-compilation guide can be found at [6].

Install required dependencies on a host machine:

Code Listing 1

```
$ sudo apt install git bc bison flex libssl-dev make libc6-dev
  libncurses5-dev libncurses5-dev
```

Download the Git repository [1]:

Code Listing 2

```
$ cd ~
$ git clone https://github.com/Infineon/linux-trusted-key-
  optiga-tpm
```

Install toolchain and set environment variable:

Code Listing 3

```
$ cd ~
$ git clone https://github.com/raspberrypi/tools
$ export PATH=$PATH:~/tools/arm-bcm2708/arm-linux-
  gnueabihf/bin
```

Download Linux kernel source:

Code Listing 4

```
$ cd ~
$ git clone -b rpi-5.4.y https://github.com/raspberrypi/linux
$ cd linux
$ git checkout raspberrypi-kernel_1.20200902-1
```

Build Linux kernel source:

Code Listing 5

```
# Prepare
$ KERNEL=kernel7l
$ make ARCH=arm CROSS_COMPILE=arm-linux-gnueabihf-
  bcm2711_defconfig

# Configure
$ make ARCH=arm CROSS_COMPILE=arm-linux-gnueabihf- menuconfig

Security options  --->
  <M> TRUSTED KEYS
  <M> ENCRYPTED KEYS

# Build
$ make -j$(nproc) ARCH=arm CROSS_COMPILE=arm-linux-gnueabihf-
  zImage modules dtbs
```

Transfer kernel modules, kernel image, and device tree blobs to a SD card (remember to set /dev/sd?1 and /dev/sd?2 accordingly, find this information with command “sudo fdisk -l”):

Code Listing 6

```
$ mkdir mnt
$ mkdir mnt/fat32
$ mkdir mnt/ext4
$ sudo umount /dev/sd?1
$ sudo umount /dev/sd?2
$ sudo mount /dev/sd?1 mnt/fat32
$ sudo mount /dev/sd?2 mnt/ext4
$ sudo env PATH=$PATH make ARCH=arm CROSS_COMPILE=arm-linux-gnueabihf- INSTALL_MOD_PATH=mnt/ext4 modules_install
$ sudo cp mnt/fat32/$KERNEL.img mnt/fat32/$KERNEL-backup.img
$ sudo cp arch/arm/boot/zImage mnt/fat32/$KERNEL.img
$ sudo cp arch/arm/boot/dts/*.dtb mnt/fat32/
$ sudo cp arch/arm/boot/dts/overlays/*.dtb* mnt/fat32/overlays/
$ sudo cp arch/arm/boot/dts/overlays/README mnt/fat32/overlays/
$ sudo umount mnt/fat32
$ sudo umount mnt/ext4
$ sync
```

1.3 Enable TPM

Insert the flashed SD card and boot the Raspberry Pi®.

Open the kernel config in an editor:

Code Listing 7

```
$ sudo nano /boot/config.txt
```

Insert the following lines to enable SPI and TPM.

Code Listing 8

```
dtoverlay=tpm-slb9670
```

Save the file and exit the editor.

Reboot the Raspberry Pi® and check if TPM is activated.

Code Listing 9

```
$ ls /dev | grep tpm
tpm0
tpmrm0
```

1.4 Install TPM Software

Install the following software on the Raspberry Pi®:

Table 1 TPM 2.0 software

Software	Link	Version
tpm2-tss	https://github.com/tpm2-software/tpm2-tss	3.0.3

tpm2-tools	https://github.com/tpm2-software/tpm2-tools	5.0

Install dependencies:

Code Listing 10

```
$ sudo apt update
$ sudo apt -y install autoconf-archive libcmocka0 libcmocka-dev procps iproute2 build-essential git pkg-config gcc libtool automake libssl-dev uthash-dev autoconf doxygen libgcrypt-dev libjson-c-dev libcurl4-gnutls-dev uuid-dev pandoc
```

Install TPM software stack:

Code Listing 11

```
$ cd ~
$ git clone https://github.com/tpm2-software/tpm2-tss.git
$ cd tpm2-tss
$ git checkout 3.0.3
$ ./bootstrap
$ ./configure
$ make -j$(nproc)
$ sudo make install
$ sudo ldconfig
```

Install TPM tools:

Code Listing 12

```
$ cd ~
$ git clone https://github.com/tpm2-software/tpm2-tools.git
$ cd tpm2-tools
$ git checkout 5.0
$ ./bootstrap
$ ./configure
$ make -j$(nproc)
$ sudo make install
$ sudo ldconfig
```

2 Linux Trusted and Encrypted Keys

This section describes how to setup trusted and encrypted keys on a Raspberry Pi®.

2.1 TPM Initialization

Authorize non-privileged access to TPM device nodes:

Code Listing 13

```
$ sudo chmod a+rw /dev/tpmrm0
```

Clear the TPM:

Code Listing 14

```
$ tpm2_clear -c p
```

Create primary key and store it as persistent key. The auth value must set to 40 hexadecimal characters for keyctl to work:

Code Listing 15

```
$ tpm2_createprimary -c primary.ctx -G ecc -p
  hex:0123456789abcdef0123456789abcdef01234567
$ tpm2_evictcontrol -C o -c primary.ctx 0x81000001
```

2.2 Create Trusted Key

Create a TPM-protected kernel master key (KMK):

Code Listing 16

```
$ sudo modprobe trusted
$ keyctl add trusted kmk "new 32 keyhandle=0x81000001
  keyauth=0123456789abcdef0123456789abcdef01234567" @s
$ keyctl show @s
```

2.3 Create Disk Encryption Key

Create a KMK-protected disk encryption key (DEK). The description must set to 16 hexadecimal characters for ecryptfs to work:

Code Listing 17

```
$ sudo modprobe encrypted-keys
$ keyctl add encrypted 0123456789abcdef "new ecryptfs
  trusted:kmk 64" @s
$ keyctl show @s
```

2.4 Create EVM Key

Create a KMK-protected EVM key:

Code Listing 18

```
$ sudo modprobe encrypted-keys
$ keyctl add encrypted evm-key "new trusted:kmk 32" @s
$ keyctl show @s
```

2.5 Make Keys Persistent

Create a target directory:

Code Listing 19

```
$ mkdir ~/keys
```

Backup the KMK:

Code Listing 20

```
$ keyctl pipe `keyctl search @s trusted kmk` > ~/keys/kmk.blob
```

Backup the DEK:

Code Listing 21

```
$ keyctl pipe `keyctl search @s encrypted 0123456789abcdef` > ~/keys/dek.blob
```

Backup the EVM key:

Code Listing 22

```
$ keyctl pipe `keyctl search @s encrypted evm-key` > ~/keys/evm-key.blob
```

Reboot the Raspberry Pi® and load the keys:

Code Listing 23

```
$ sudo modprobe trusted
$ keyctl add trusted kmk "load `cat ~/keys/kmk.blob` keyhandle=0x81000001 keyauth=0123456789abcdef0123456789abcdef01234567" @s
$ sudo modprobe encrypted-keys
$ keyctl add encrypted 0123456789abcdef "load `cat ~/keys/dek.blob`" @s
$ keyctl add encrypted evm-key "load `cat ~/keys/evm-key.blob`" @s
$ keyctl show @s
```

3 KEYCTL Cheat Sheet

Keyctl is a tool to allow user-space programs to perform key manipulation on a Linux system. Table 2 shows a short list of useful commands.

Table 2 Keyctl cheat sheet

Command	Description
Code Listing 24 `$ keyctl clear @s`	Unlink all keys attached to a session keyring.
Code Listing 25 `$ keyctl unlink <key>`	Unlink a specific key from all keyrings.
Code Listing 26 `$ keyctl show @s`	Display current session's keyrings/keys.
Code Listing 27 `$ keyctl add user kmk "password" @s`	Create a plain key with following properties: • Key type: user • Description: kmk (kernel master key) • Data: password • Keyring: session keyring
Code Listing 28 `$ keyctl add trusted kmk "new 32 keyhandle=0x81000001 keyauth=<auth-value>" @s`	Create a TPM-protected key with following properties: • Key type: trusted • Description: kmk • Data: 32 bytes long random numbers • Sealed by TPM key handle: 0x81000001 • TPM key auth value: 40 bytes hexadecimal • Keyring: session keyring
Code Listing 29 `$ keyctl add encrypted enc-key "new user:kmk 32" @s`	Encrypted keys are created from kernel generated random numbers and are encrypted/decrypted using a specified master key. In this example, the key is created with following properties: • Key type: encrypted • Description: enc-key • Data: 32 bytes long random numbers • Master key: kmk • Keyring: session keyring
Code Listing 30 `$ keyctl search @s user kmk`	Search a keyring (@s) for a key of a particular type (user) and description (kmk). If discovered, the ID of the key will be shown.
Code Listing 31 `$ keyctl print <key>`	Print the data field of a key.

Code Listing 32 `$ keyctl pipe <key> > key.blob`	Backup a key.
Code Listing 33 `` $ keyctl add <key-type> <description> "load `cat ~/key.blob`" @s ``	Load a backup key. This method only works with trusted and encrypted key types.

4 Encrypted File System

This section describes how DEK from section 2.3 can be used to setup an encrypted file system.

Install ecryptfs:

Code Listing 34

```
$ sudo apt install ecryptfs-utils
```

Create a target directory:

Code Listing 35

```
$ mkdir ~/vault
```

Mount the directory as an ecryptfs file system:

Code Listing 36

```
$ sudo mount -i -t ecryptfs -o
  ecryptfs_sig=0123456789abcdef,ecryptfs_fnek_sig=0123456789ab
  cdef,ecryptfs_cipher=aes,ecryptfs_key_bytes=32 ~/vault
  ~/vault
```

Create a file in the vault directory:

Code Listing 37

```
$ echo "secret" > ~/vault/data
$ xxd ~/vault/data
00000000: 7365 6372 6574 0a secret.
```

Unmount the file system and notice the difference, filename and the content are both encrypted now:

Code Listing 38

```
$ sudo umount ~/vault
$ ls ~/vault
ECRYPTFS_FNEK_ENCRYPTED.FWY...
$ xxd ~/vault/ECRYPTFS_FNEK_ENCRYPTED.FWY...
 00000000: 0000 0000 0000 0007 b245 eee8 8ec4 591d .........E....Y.
 00000010: 0300 000a 0000 1000 0002 8c2d 0409 0301 ...........-....
 00000020: 0000 0000 0000 0000 605a ec8c 8901 0c03 ........`Z......
 00000030: 6fa9 8896 e23c 8fa3 ca63 ed0a 7de9 9859 o....<...c..}..Y
 00000040: 2dd5 6938 d1a9 81d0 36ed 1662 085f 434f -.i8....6..b._CO
 00000050: 4e53 4f4c 4500 0000 0001 2345 6789 abcd NSOLE.....#Eg...
 ...
```

Mount the directory as an ecryptfs file system to regain access to the file.

5 Platform Integrity Protection

This section describes how an evm-key from section 2.4 can be used to setup Linux IMA and EVM subsystems to protect the integrity of a platform.

The IMA subsystem is responsible for verifying the integrity of file contents. This is possible by appraising a file’s measurement against a “good” value stored as a file’s extended attribute (e.g., security.ima).

The EVM subsystem is responsible for verifying the integrity of security-sensitive extended attributes with the help of EVM key to prevent offline tempering of files.

5.1 Kernel Re-build

First time Git setup, insert your username and email.

Code Listing 39

```
$ git config --global user.name "your name"
$ git config --global user.email your-email@example.com
```

Apply the following patch and skip 5.1.1; otherwise, for detailed steps please follow 5.1.1.

Code Listing 40

```
$ cd linux
$ git am ~/linux-trusted-key-optiga-tpm/patches/code-listing-
  39-40-41.patch
```

5.1.1 Kernel Source Change

Modify line 142 of file security/integrity/ima/ima_policy.c. The change restricts IMA/EVM appraisal to files that are owned and access by root only.

Code Listing 41

```
static struct ima_rule_entry default_appraise_rules[]
         __ro_after_init = {
        {.action = DONT_APPRAISE, .fsmagic = PROC_SUPER_MAGIC,
         .flags = IMA_FSMAGIC},
        {.action = DONT_APPRAISE, .fsmagic = SYSFS_MAGIC,
         .flags = IMA_FSMAGIC},
        {.action = DONT_APPRAISE, .fsmagic = DEBUGFS_MAGIC,
         .flags = IMA_FSMAGIC},
        {.action = DONT_APPRAISE, .fsmagic = TMPFS_MAGIC,
         .flags = IMA_FSMAGIC},
        {.action = DONT_APPRAISE, .fsmagic = RAMFS_MAGIC,
         .flags = IMA_FSMAGIC},
        {.action = DONT_APPRAISE, .fsmagic =
         DEVPTS_SUPER_MAGIC, .flags = IMA_FSMAGIC},
        {.action = DONT_APPRAISE, .fsmagic = BINFMTFS_MAGIC,
         .flags = IMA_FSMAGIC},
        {.action = DONT_APPRAISE, .fsmagic = SECURITYFS_MAGIC,
         .flags = IMA_FSMAGIC},
        {.action = DONT_APPRAISE, .fsmagic = SELINUX_MAGIC,
         .flags = IMA_FSMAGIC},
```

Code Listing 41

```
                {.action = DONT_APPRAISE, .fsmagic = SMACK_MAGIC,
                 .flags = IMA_FSMAGIC},
                {.action = DONT_APPRAISE, .fsmagic = NSFS_MAGIC,
                 .flags = IMA_FSMAGIC},
                {.action = DONT_APPRAISE, .fsmagic = EFIVARFS_MAGIC,
                 .flags = IMA_FSMAGIC},
                {.action = DONT_APPRAISE, .fsmagic =
                 CGROUP_SUPER_MAGIC, .flags = IMA_FSMAGIC},
                {.action = DONT_APPRAISE, .fsmagic =
                 CGROUP2_SUPER_MAGIC, .flags = IMA_FSMAGIC},
                {.action = APPRAISE, .fowner = GLOBAL_ROOT_UID,
                 .fowner_op = &uid_eq,
                 .uid = GLOBAL_ROOT_UID, .uid_op = &uid_eq, .flags =
                 IMA_FOWNER | IMA_UID},
        };
```

Since IMA/EVM appraisal can be triggered in the early boot process; therefore, TPM and SPI must be made available before IMA/EVM. In general, a kernel module can be prioritized by configuring it from loadable to built-in. However, some modules do require additional changes, in this instance, the SPI subsystem. Edit the following line in drivers/clk/bcm/clk-bcm2835.c:

Code Listing 42

```
postcore_initcall(__bcm2835_clk_driver_init);
```

Replace it with:

Code Listing 43

```
subsys_initcall(__bcm2835_clk_driver_init);
```

5.1.2 Kernel Re-configure

Apply the following configuration:

Code Listing 44

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

Device Drivers --->
  [*] SPI support --->
    <*> BCM2835 SPI controller

Device Drivers --->
  Character devices --->
    -*- TPM Hardware Support --->
      <*> TPM Interface Specification 1.3 Interface / TPM 2.0
          FIFO Interface - (SPI)

Security options  --->
  <*> TRUSTED KEYS
  -*- ENCRYPTED KEYS
  [*] Enable different security models
```

Code Listing 44

```
-*- Enable the securityfs filesystem
[*] Integrity subsystem
[*]   Integrity Measurement Architecture(IMA)
[*]     Appraise integrity measurements
[*]   ima_appraise boot parameter
[*] EVM support
[*] FSUUID (version 2)
```

Build and transfer the kernel image according to section 1.2.

Boot the Raspberry Pi® and edit the kernel boot parameters:

Code Listing 45

```
$ sudo nano /boot/cmdline.txt
```

Append the following to the existing line:

Code Listing 46

```
ima_policy=tcb
```

Reboot the Raspberry Pi® and check if IMA is activated. The return value must be greater than 1.

Code Listing 47

```
$ sudo cat /sys/kernel/security/ima/runtime_measurements_count
1517
```

5.2 Install Extended Attributes Utility

Install the utility on the Raspberry Pi®:

Code Listing 48

```
$ sudo apt install attr
```

Check if the installation is ok:

Code Listing 49

```
$ getfattr --version
getfattr 2.4.48
```

5.3 IMA Appraisal

The IMA appraisal feature enables a local system to perform file integrity validation.

5.3.1 Setup

Before IMA appraisal can be enabled, the filesystem must be labelled with extended attribute “security.ima”. This can be done by editing the kernel boot parameter:

Code Listing 50

```
ima_policy=appraise_tcb ima_appraise=fix
```

Reboot the Raspberry Pi® and check if the IMA labelling is working by looking for the extended attribute "security.ima" using the utility getfattr.

Code Listing 51

```
$ sudo su -c "echo 'hello world' > data"
$ getfattr -de hex -m - data
# file: data
security.ima=0x0122596363b3de40b06f981fb85d82312e8c0ed511
```

Execute the following to label the file system. Only files that meet the policies (Code Listing 41) will be labelled. This process will take some time (up to 20mins):

Code Listing 52

```
$ sudo su -c "time find / -fstype ext4 -type f -uid 0 -exec dd
  if='{}' of=/dev/null count=0 status=none \;"
```

Now, activate IMA appraisal by editing the kernel boot parameter:

Code Listing 53

```
ima_policy=appraise_tcb ima_appraise=enforce
```

5.3.2 Verify

Reboot the Raspberry Pi® and check if the IMA is still working by repeating Code Listing 51.

To demonstrate an IMA appraisal failure, create a root owned file and permit a non-root user to edit the file (this is for testing only). Since user actions do not trigger the appraisal due to the specified policies; therefore, the extended attribute is not updated. Consequently, appraisal will fail, and the file will become inaccessible in root.

Code Listing 54

```
$ sudo rm data
$ sudo su -c "echo 'hello world' > data"
$ getfattr -de hex -m - data
# file: data
security.ima=0x0122596363b3de40b06f981fb85d82312e8c0ed511
$ sudo cat data
hello world
$ sudo chmod a+w data
$ echo "world hello" >> data
$ getfattr -de hex -m - data
# file: data
security.ima=0x0122596363b3de40b06f981fb85d82312e8c0ed511
$ sudo cat data
cat: data: Permission denied
```

5.4 EVM Appraisal

With IMA appraisal enabled, EVM appraisal can be introduced to protect the IMA extended attribute using a KMK-protected EVM key.

For EVM appraisal to work, KMK and EVM key must be loaded before EVM activation. This is where initramfs comes in, it provides a miniature filesystem just enough to populate the keyring and activate EVM before rootfs gets mounted.

5.4.1 Setup

Download the Git repository [1] on the Raspberry Pi®:

Code Listing 55

```
$ cd ~
$ git clone https://github.com/Infineon/linux-trusted-key-optiga-tpm
```

An initramfs image can be created using the utility initramfs-tools [7].

After completing section 2.4 & 2.5, both KMK and EVM key are available at directory ~/keys.

Create a hook script “ima-hook” (is also available in ~/linux-trusted-key-optiga-tpm/initramfs-tools) in the directory /etc/initramfs-tools/hooks/. Hook scripts are used to indicate what files to be included in an initramfs image. These scripts will not be included in the image itself.

Code Listing 56

```
#!/bin/sh
PREREQ=""
prereqs()
{
   echo "$PREREQ"
}

case $1 in
prereqs)
   prereqs
   exit 0
   ;;
esac

. /usr/share/initramfs-tools/hook-functions
# Begin real processing below this line

# Copy executables we need to initramfs
copy_exec /bin/keyctl /usr/bin

# Copy other files to initramfs
mkdir -p $DESTDIR/etc/keys
cp /home/pi/keys/kmk.blob $DESTDIR/etc/keys
cp /home/pi/keys/evm-key.blob $DESTDIR/etc/keys

exit 0
```

Create a boot script “ima-boot” (is also available in ~/linux-trusted-key-optiga-tpm/initramfs-tools) in the directory /etc/initramfs-tools/scripts/local-top/. Boot script as its name suggests is executed at initramfs boot time to load KMK and EVM keys before enabling EVM. Boot scripts will be included in the initramfs image.

Code Listing 57

```
#!/bin/sh
PREREQ=""
prereqs()
{
   echo "$PREREQ"
}

case $1 in
prereqs)
   prereqs
   exit 0
   ;;
esac

echo "initramfs keys loading start..." > /dev/kmsg 2>&1 < /dev/console

# import KMK and EVM key
keyctl add trusted kmk "load `cat /etc/keys/kmk.blob` keyhandle=0x81000001 keyauth=0123456789abcdef0123456789abcdef01234567" @u
keyctl add encrypted evm-key "load `cat /etc/keys/evm-key.blob`" @u

# change evm-key permission to grant full access to processor and owner
EVM_KEY=`keyctl show @u | grep evm-key | sed "s/ *//" | sed "s/ .*//"`
KMK_KEY=`keyctl show @u | grep kmk | sed "s/ *//" | sed "s/ .*//"`
keyctl setperm $EVM_KEY 0x3f000000
keyctl setperm $KMK_KEY 0x3f000000
echo "initramfs, kmk and evm-key loaded successfully: " > /dev/kmsg
keyctl show > /dev/kmsg

# mount securityfs
mount -n -t securityfs securityfs /sys/kernel/security

# enable EVM
echo "1" > /sys/kernel/security/evm
```

Make both scripts executable:

Code Listing 58

```
$ sudo chmod a+x /etc/initramfs-tools/hooks/ima-hook
$ sudo chmod a+x /etc/initramfs-tools/scripts/local-top/ima-boot
```

Disable IMA appraisal by editing the kernel boot parameters. The file is not accessible from within Raspberry Pi® OS due to IMA appraisal is in enforcement mode; instead, access it externally with a microSD card reader.

Code Listing 59

```
ima_policy=appraise_tcb ima_appraise=fix evm=fix
```

Boot the Raspberry Pi® then create an initramfs image in the directory /boot:

Code Listing 60

```
$ sudo update-initramfs -c -k $(uname -r)
```

Verify the image by checking if keys and boot script are copied over:

Code Listing 61

```
$ mkdir ~/initramfs
$ cd ~/initramfs
$ zcat /boot/initrd.img-5.4.51-v7l+ | cpio -idmv
$ ls etc/keys/
evm-key.blob  kmk.blob
$ ls scripts/local-top/
ima-boot  ORDER
```

Insert the following lines to kernel config:

Code Listing 62

```
initramfs initrd.img-5.4.51-v7l+
```

Reboot the Raspberry Pi® and check if EVM is activated:

Code Listing 63

```
$ sudo cat /sys/kernel/security/evm
1
```

Execute the following to label the file system. Only files that meet the policies (Code Listing 41) will be labelled. This process will take some time (up to 20mins):

Code Listing 64

```
$ sudo su -c "time find / -fstype ext4 -type f -uid 0 -exec dd
  if='{}' of=/dev/null count=0 status=none \;"
```

Activate IMA and EVM appraisal by editing the kernel boot parameter to the following without including "evm=fix":

Code Listing 65

```
ima_policy=appraise_tcb ima_appraise=enforce
```

Reboot the Raspberry Pi®.

5.4.2 Verify

After completing section 5.4.1, a new extended attribute “security.evm” can be observed by:

Code Listing 66

```
$ sudo rm data
$ sudo su -c "echo 'hello world' > data"
$ getfattr -de hex -m - data
# file: data
security.evm=0x0284513b42309bf4f084203396853df0edb5c9a1bf
security.ima=0x0122596363b3de40b06f981fb85d82312e8c0ed511
```

To demonstrate the effect of EVM, disable early activation of EVM by editing the kernel config (Code Listing 67) and editing the kernel boot parameters (Code Listing 68). Again, they are not accessible from within Raspberry Pi® OS due to IMA & EVM appraisal are in enforcement mode; instead, access them externally with a microSD card reader.

Code Listing 67

```
#initramfs initrd.img-5.4.51-v7l+
```

Code Listing 68

```
ima_policy=appraise_tcb ima_appraise=fix evm=fix
```

Boot the Raspberry Pi® and check if EVM is disabled:

Code Listing 69

```
$ sudo cat /sys/kernel/security/evm
0
```

To demonstrate an EVM appraisal failure, execute the following to corrupt the EVM attribute. Observe EVM attribute is not changed despite the group owner change (IMA attribute keep track of file content. EVM attribute keep track of file metadata, e.g., ownership, IMA attribute, …).

Code Listing 70

```
$ sudo chgrp pi data
$ getfattr -de hex -m - data
# file: data
security.evm=0x0284513b42309bf4f084203396853df0edb5c9a1bf
security.ima=0x0122596363b3de40b06f981fb85d82312e8c0ed511
```

Activate EVM by editing the kernel config:

Code Listing 71

```
initramfs initrd.img-5.4.51-v7l+
```

Activate EVM appraisal by editing the kernel boot parameter to the following without including “evm=fix”:

Code Listing 72

```
ima_policy=appraise_tcb ima_appraise=enforce
```

Reboot the Raspberry Pi® and observe the file is no longer accessible:

Code Listing 73

```
$ sudo cat data
cat: data: Permission denied
```

References

[1] https://github.com/Infineon/linux-trusted-key-optiga-tpm

[2] https://www.infineon.com/cms/en/product/evaluation-boards/iridium9670-tpm2.0-linux/

[3] http://www.infineon.com/tpm

[4] https://trustedcomputinggroup.org/resource/tpm-main-specification/

[5] https://downloads.raspberrypi.org/raspios_armhf/images/raspios_armhf-2020-08-24/2020-08-20-raspios-buster-armhf.zip

[6] https://www.raspberrypi.org/documentation/linux/kernel/building.md

[7] https://wiki.debian.org/initramfs-tools

Revision history

Reference	Description
Revision 1.0, 2021-01-21	
all	Initial version

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

Edition 2021-01-21

Published by

Infineon Technologies AG

81726 Munich, Germany

Do you have a question about this document?

Email: **csscustomerservice@infineon.com**

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

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

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

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

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