F&S Documentation “Secure Boot” for i.MX8M

Documentation for the “F&S Secure Boot” for i.MX8M

Version 1.1

(2023-02-09)

armStone efus NetDCU

PicoCOM PicoCore PicoMOD QBliss

Elektronik Systeme

About This Document

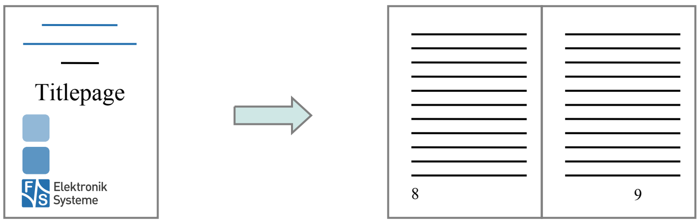
This document is intended to explain the usage of the Secure Boot on F&S Boards.

Remark

The version number on the title page of this document is the version of the document. It is not related to the version number of any software release.

How To Print This Document

This document is designed to be printed double-sided (front and back) on A4 paper. If you want to read it with a PDF reader program, you should use a two-page layout where the title page is an extra single page. The settings are correct if the page numbers are at the outside of the pages, even pages on the left and odd pages on the right side. If it is reversed, then the title page is handled wrongly and is part of the first double-page instead of a single page.



Typographical Conventions

We use different fonts and highlighting to emphasize the context of special terms:

`File names`

Menu entries

```
Board input/output
```

`Program code`

```
PC input/output
```

```
Listings
```

```
Generic input/output
```

`Variables`

History

Date	V	Platform	A,M,R	Chapter	Description	Au
2022-03-05	1.0	Linux	A	All	Derivate from ‘F&S Documentation “Secure Boot” Version 1.3’	DD
2023-02-09	1.1	Linux	M	All	Improved the Document	DD

V	Version
A,M,R	Added, Modified, Removed
Au	Author

Table of Contents

1 Introduction

This document is intended to describe the usage of the F&S Secure Boot on I.MX8M based Boards. The F&S Secure Boot package contains some binaries and precompiled images. There is also a so called F&S Secure Boot Tool. This Tool is able to:

- create certificates (used for signing an image)
- sign an image (create unique hash sum for a specific memory range)

The F&S Secure Boot Tool is a collection of shell scripts, which perform their individual task.

1.1 Boot Sequence

Figure 1 shows the typical boot sequence of the board.

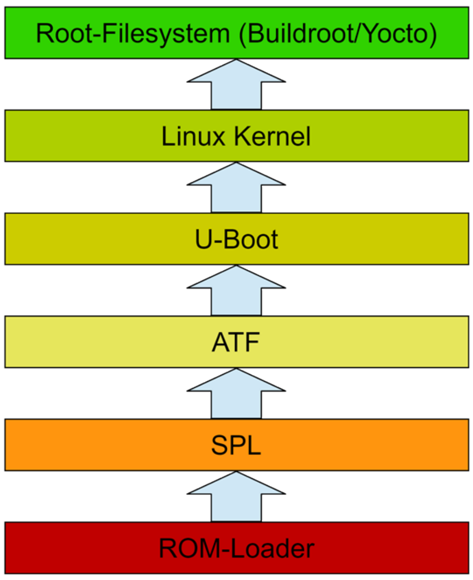


Figure 1: Boot sequence

Let us go through each step one by one, starting at the bottom.

1. The ROM-Loader is code that is located in a ROM inside of the chip. It is usually capable of loading some piece of code from a boot device (NAND flash, eMMC, SD card, or similar). However at this point of time the dynamic RAM (DRAM) is not yet initialized. The chip does not even know if there is any DRAM attached at all. So it can only load this code to some rather small internal RAM (OC-RAM). A regular U-Boot bootloader with more than 500KB will not fit in there.

This means we need a smaller step-stone loader first, which is called the SPL. This step-stone loader is loaded to OC-RAM and then executed.

2. SPL (short for Second Program Loader) is a rather small step-stone bootloader. It reads the Board-Configuration, detects and activates the DRAM and then loads the main bootloader - the U-Boot - and the Hardware Manager - the ATF - into DRAM and executes it.
3. The ATF (short for ARM trusted Firmware) is designed by ARM itself, as a hardware resource manager that runs on a higher execution level than the regular user software. It initializes itself and executes the main Bootloader. It is not OS specific, only from then on, the boot sequence differs.
4. The U-Boot is the main bootloader. It is used to load and execute the Linux kernel and the device tree. It is also used to install the Linux images and to configure the boot procedure.
5. The Linux kernel provides the Linux operating system including the device drivers. It will activate the peripherals and finally mounts the root filesystem.
6. The root filesystem contains the code for the init process and all the Userland software. The init process will finalize the boot process, for example it will start the background services, the GUI and probably the main application.

During the setup of the system users may notice that they are not installing an SPL or ATF Image but an N-Boot Image. The i.MX8M NBoot is an Image Container including for example the SPL and the ATF. This “Native-Boot” is not the same as the I.MX6 “NAND-Boot”. But it still has similarities like being loaded before the U-Boot.

With the F&S Secure Boot package you are able to sign the following Images:

- N-Boot
- U-Boot
- Linux Kernel
- Linux Devicetree

Note:

F&S Elektronik Systeme tested the F&S Secure Boot Tool and the CST on a Fedora 27 machine. We do not test other linux distributions neither does the F&S Secure Boot Tool run under Windows.

1.2 Secure U-Boot

The secure U-Boot is available as a binary and in source code. You can find it in the release archive.

The U-Boot has implemented the authentication mechanism. Below is listed when an image will be authenticated.

- If the Secure Images are written with the “fsimage save” command, the images will be verified.
- If the boot command is executed the images will be verified.

1.2.1 Build Secure U-Boot

To build the Secure U-Boot change to extracted U-Boot folder

```
[fs-dev@localhost ~]$ cd u-boot-2020.04-fssecure-imx8m-
V<year>.<month>
```

After then setup the correct secure defconfig.

```
[fs-dev@localhost u-boot-2020.04-fssecure-imx8m-
V<year>.<month>]$ make fsimx<arch>_secure_boot_defconfig
```

Now the correct defconfig is setup and we can build the U-Boot image.

```
[fs-dev@localhost u-boot-2020.04-fssecure-imx8m-
V<year>.<month>]$ make
```

The output image is called **u-boot.fs**

Note:

F&S has implemented the verification process to U-Boot, but that doesn´t mean the whole U-Boot is secure. The access e.g. to write the memory and registers are further granted. So the user has to check his requirements and setup the U-Boot by himself. We recommend preventing access to the U-Boot.

1.3 Secure N-Boot

The secure N-Boot is available as a binary and in source code. You can find it in the release archive.

The Security Feature of the N-Boot are integrated into the SPL. The SPL will try to authenticate the following parts of the N-Boot.

- Board Configuration File
- D-RAM Settings
- D-RAM Timings
- ATF

Additionally it will try to verify the U-Boot.

1.3.1 Build Secure N-Boot

Building the Secure N-Boot yourself requires more steps than the secure U-Boot. Since the N-Boot is made of several Images. Most of these files are already part of the u-boot repository. Some files, especially the firmware files need to be build by yourself.

1.3.1.1 Getting the DRAM Firmware

To get the DRAM Firmware, it must be downloaded from NXPs firmware release. For this open an Terminal and write

```
[fs-dev@localhost ~]$ wget
http://www.freescale.com/lgfiles/NMG/MAD/YOCTO/firmware-imx-
8.10.1.bin
```

This will download the firmware as an executable binary. First you need to change its permissions, then execute the file.

```
[fs-dev@localhost ~]$ chmod 755 firmware-imx-8.10.1.bin
```

```
[fs-dev@localhost ~]$./firmware-imx-8.10.1.bin
```

Now there is a directory with the same name, enter the following path and copy all files to the NXP-Firmware directory.

```
[fs-dev@localhost ~]$ cd firmware-imx-8.10.1/firmware/ddr/synopsys
[fs-dev@localhost synopsys]$ cp * <u-boot source>/board/F+S/NXP-
Firmware/
```

1.3.1.2 Building the ATF

To build the ATF change to extracted ATF folder

```
[fs-dev@localhost ~]$ cd atf-f+s-V<year>.<month>
```

And execute

```
[fs-dev@localhost atf-f+s-V<year>.<month>]$ make bl31 PLAT=<arch>
```

Where <arch> can be imx8mm, imx8mn or imx8mp.

The ATF Image will be under build/<arch>/release/bl31.bin. Copy it into your u-boot directory to board/F+S/NXP-Firmware/bl31-<arch>.bin.

Now go to the u-boot directory

```
[fs-dev@localhost ~]$ cd u-boot-2018.03-fssecure-V<year>.<month>
```

After setup the correct secure defconfig if you did not do it already.

```
[fs-dev@localhost u-boot-2018.03-fssecure-V<year>.<month>]$ make
fsimx<arch>_secure_boot_defconfig
```

Now the correct defconfig is setup and we can build the N-Boot image.

```
[fs-dev@localhost u-boot-2018.03-fssecure-V<year>.<month>]$ make
nboot
```

1.4 Images with OP-TEE Support

To use op-tee the ATF needs to be recompiled. Change into its directory,

```
[fs-dev@localhost ~]$ cd atf-f+s-V<year>.<month>
```

It is advised to clean the ATF source code before rebuilding it, since some changes may not be applied when rebuilding from an unclean source. For this write

```
[fs-dev@localhost atf-f+s-V<year>.<month>]$ make distclean
```

And rebuild with

```
[fs-dev@localhost atf-f+s-V<year>.<month>]$ make bl31 PLAT=<arch>
SPD=opteed
```

Copy the new bl31.bin like shown into chapter 1.3.1.2.

Now change into the op-tee directory.

```
[fs-dev@localhost ~]$ cd op-tee-f+s-V<year>-<month>
```

There you build the optee binary with the command

```
[fs-dev@localhost op-tee-f+s-V<year>-<month>]$ make PLAT=imx
PLATFORM_FLAVOUR=<arch> DDR_SIZE=0x<amount in hexadecimal>
```

Arch can be fsimx8mm, fsimx8mn and fsimx8mp. DDR_SIZE can be omitted but is not advisable since the system relies on the DRAM size provided by op-tee.

Copy the resulting Image from op-tee-f+s-V<year>-<month>/out/arm-plat-imx/core/tee.bin to the u-boot to board/F+S/NXP-Firmware/tee-<arch>.bin. Where arch can be fsimx8mm, fsimx8mn and fsimx8mp.

Now go into the U-Boot directory

```
[fs-dev@localhost ~]$ cd u-boot-2018.03-fssecure-V<year>.<month>
```

After set the secure boot defconfig with op-tee support

```
[fs-dev@localhost u-boot-2018.03-fssecure-V<year>.<month>]$ make
fsimx<arch>_secure_boot_optee_defconfig
```

Now the correct defconfig is setup and we can build the U-Boot and N-Boot image. The get the Images with op-tee support.

```
[fs-dev@localhost u-boot-2018.03-fssecure-V<year>.<month>]$ make
```

And build the N-Boot

```
[fs-dev@localhost u-boot-2018.03-fssecure-V<year>.<month>]$ make
nboot
```

And U-Boot

```
[fs-dev@localhost u-boot-2018.03-fssecure-V<year>.<month>]$ make
```

Please make sure you installed both images before resetting the board.

2 Secure Boot on F&S Boards

2.1 Configure your host computer

In order to configure your computer properly (in order to use F&S software) please refer to the "*Linux on F&S Boards*" guide provided by F&S Elektronik Systeme. After you have done this, continue with this guide.

It is crucial that you install the packages "GNU objcopy" version 2.15 or higher" and "OpenSSL" 1.10 or higher provided from Fedora standard repositories.

2.2 Get the tools and packages

First of all, download the F&S Secure Boot package from the F&S website and get the Code Signing Tool (CST) from NXPs website. The tested version of the CST is "**3.3.1**". We recommend you to use this version, older or newer versions may also work but are not tested.

2.3 Release Content

These tar archive is compressed with bzip2. So to see the files, you first have to unpack the archive.

```
tar xvf fs-secure-boot-fsimx<arch>-V<year>.<month>.tar.bz2
```

This will create a directory `fs-secure-boot-<YEAR>.<MONTH>` that contains all the files of the release, except the CST.

They often use a common naming scheme:

`<package>-<YEAR>.<MONTH>.<extension>`

With the following meaning:

<package>..................................	The name of the package (e.g. fs-secure-boot).
<year>.<month>.........................	The year and month of the release (e.g. 2019.02)
<extension>.............................	The extension of the package (e.g. .bin, .tar.bz2, etc.). Please note that some file types do not have an extension, for example the zImage file of the Linux kernel.

The following table lists the files that you get after unpacking the release archive.

Directory	Description
/	Top Directory
Readme.txt	Release information
setup-fs-secure-boot.sh	Script to unpack F&S Secure Boot source packages to a build directory
binaries/	Images to be used with the board directly
nboot-secure-fsimx8mm.fs	Secure N-Boot for bootloader fsimx8mm
nboot-secure-fsimx8mn.fs	Secure N-Boot for bootloader fsimx8mn
nboot-secure-fsimx8mp.fs	Secure N-Boot for bootloader fsimx8mp
u-boot-secure-fsimx8mm.fs	Secure U-Boot Image for fsimx8mm.
u-boot-secure-fsimx8mn.fs	Secure U-Boot Image for fsimx8mn.
u-boot-secure-fsimx8mp.fs	Secure U-Boot Image for fsimx8mp.
/sources	Configurations and sources
u-boot-2018.03-fssecure-V<year>.<month>.tar.bz2	U-Boot secure source code
fs-secure-boot-tool-V<year>.<month>.tar.bz2	F&S Secure Boot Tool source code
fs-atf-V<year>.<month>.tar.bz2	Hardware manager, necessary for the N-Boot
fs-optee-os-V<year>.<month>.tar.bz2	Op-tee code in case, op-tee has to be used
doc/	Documentation
FS_Secure_Boot_i.MX8M.pdf	F&S Secure Boot document

Table 1: Content of the created release directory

2.4 Install Content

The source code packages are located in the `sources` subdirectory of the release archive. We will assume that you want to create a separate build directory where you extract the source code and build all the software. The easiest way is to extract U-Boot and F&S Secure Boot Tool next to each other, so that the top directories of their source trees are siblings.

We have prepared a shell script called `setup-fs-secure-boot` that does this installation automatically. Just call it when you are in the top directory of the release and give the name of the build directory as argument.

```
cd <release-dir>
./setup-fs-secure-boot <build dir>
```

Add option `--dry-run` if you want to check first what this command will do. Then only a list of actions will be output but no actual changes will take place. For further information simply call

```
./setup-fs-secure-boot -help
```

Note:

Option `--dry-run` was introduced in release fs-secure-boot-tool-2019.02. In older versions you also had to create the target build directory by hand before calling the script.

After you got both tools, make sure to export an environment variable called `CST_DIR`. This variable is necessary because here you setup the path to the corresponding CST directory. Be sure that the path to the directory ends with a '/'.

```
[fs-dev@localhost ~]$ export CST_DIR=/home/fs-dev/release/
[fs-dev@localhost ~]$ echo $CST_DIR
/home/fs-dev/release/
```

2.5 Signable Images

F&S support different kind of images which can be signed or encrypted. Here is a list which images can be signed or encrypted:

- NBoot Image
  - SPL
  - Board Configuration File
  - ATF
  - DRAM-Firmware
  - DRAM-Timings
- U-Boot

- Linux Kernel
- Linux Devicetree

3 F&S Secure Boot Tool usage

In this chapter the general usage of the F&S Secure Boot Tool is illustrated.

3.1 Creating Certificates

First of all we need to open a terminal (preferably bash). After the terminal started navigate to the F&S Secure Boot Tool directory using the “cd” command.

```
cd "path_to_fs-secure-boot-tool-V<year>.<month>"
```

When you entered the directory, execute the “`generate.sh`” script.

```
./generate.sh
```

Press 'c' and 'return' in order to create the certificates. After that, you have to enter a serial number and a passphrase. Use the recommended setting for the certificates:

- create a new CA (certificate authority) key
- key length in bits for PKI tree is 2048
- PKI (Public Key Infrastructure) tree duration is 5 years (doesn't matter if it expires)
- SRKs (Super Root Keys) are generated as CA keys

Creating certificates:

```
[fs-dev@localhost fs-secure-boot-tool-V<year>.<month>]$
./generate.sh
Do you want to create certificates, create jtag response key, sign
an image or encrypt an image (c/j/s/e)?
Note: When an image is encrypted it´s also signed!
c


Please enter a serial number (8 digits!):12345678
Please enter a passphrase (no blanks!):FS_Elektronik
```

Note:

These are sample inputs, you want to use your own specifications.

Creating the SRKs (super root keys) as CA (certificate authority) certificates is recommended. Not all NXP CPUs including HAB (high assurance boot) support SRKs as non CA certificates. Furthermore using a different key length isn't recommended. NBoot doesn't support a greater value yet.

Also the **Elliptic Curve Cryptography** and **fast authentication tree** isn't tested yet.

After this, the “`hab4_pki_tree.sh`” script developed by NXP is called. Please refer to the “*CST_UG.pdf*” for more information about this script and about the signing/encryption process. Below is an example configuration.

```
    +++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
    This script is a part of the Code signing tools for Freescale's
    High Assurance Boot.  It generates a basic PKI tree.  The PKI
    tree consists of one or more Super Root Keys (SRK), with each
    SRK having two subordinate keys:
         + a Command Sequence File (CSF) key
         + Image key.
    Additional keys can be added to the PKI tree but a separate
    script is available for this.  This this script assumes openssl
    is installed on your system and is included in your search
    path.  Finally, the private keys generated are password
    protectedwith the password provided by the file key_pass.txt.
    The format of the file is the password repeated twice:
         my_password
         my_password
    All private keys in the PKI tree are in PKCS #8 format will be
    protected by the same password.

    +++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
Do you want to use an existing CA key (y/n)?: n
Do you want to use Elliptic Curve Cryptography (y/n)?: n
Enter key length in bits for PKI tree: 2048
Enter PKI tree duration (years): 5
How many Super Root Keys should be generated? 4
```

```
Do you want the SRK certificates to have the CA flag set? (y/n)?:
y

+++++++++++++++++++++++++++++++++++++
+ Generating CA key and certificate +
+++++++++++++++++++++++++++++++++++++


++++++++++++++++++++++++++++++++++++++++
+ Generating SRK key and certificate 1 +
++++++++++++++++++++++++++++++++++++++++


++++++++++++++++++++++++++++++++++++++++
+ Generating CSF key and certificate 1 +
++++++++++++++++++++++++++++++++++++++++


++++++++++++++++++++++++++++++++++++++++
+ Generating IMG key and certificate 1 +
++++++++++++++++++++++++++++++++++++++++


++++++++++++++++++++++++++++++++++++++++
+ Generating SRK key and certificate 2 +
++++++++++++++++++++++++++++++++++++++++


++++++++++++++++++++++++++++++++++++++++
+ Generating CSF key and certificate 2 +
++++++++++++++++++++++++++++++++++++++++


++++++++++++++++++++++++++++++++++++++++
+ Generating IMG key and certificate 2 +
++++++++++++++++++++++++++++++++++++++++



++++++++++++++++++++++++++++++++++++++++
+ Generating SRK key and certificate 3 +
++++++++++++++++++++++++++++++++++++++++



++++++++++++++++++++++++++++++++++++++++
+ Generating CSF key and certificate 3 +
++++++++++++++++++++++++++++++++++++++++



++++++++++++++++++++++++++++++++++++++++
+ Generating IMG key and certificate 3 +
++++++++++++++++++++++++++++++++++++++++



++++++++++++++++++++++++++++++++++++++++
+ Generating SRK key and certificate 4 +
++++++++++++++++++++++++++++++++++++++++



++++++++++++++++++++++++++++++++++++++++
+ Generating CSF key and certificate 4 +
++++++++++++++++++++++++++++++++++++++++



++++++++++++++++++++++++++++++++++++++++
+ Generating IMG key and certificate 4 +
++++++++++++++++++++++++++++++++++++++++


Certificate(s) sucessfully created
```

```
checksum is: 4278
srkhash.txt is ready!
```

After the certificates are generated, the F&S Secure Boot Tool outputs a text file called `srkhash.txt`. This file contains a checksum and the hash value (which are meant to be burned into the fuses).

Note:

You can only once create certificates. If you have done something wrong delete the CST and unpack it again.

When you want to use several keys for different products then you have to create several CST folder to handle the keys.

3.2 Signing an image

Again, execute the “`generate.sh`” script. After that press 's', 'return' and 'd', 'enter'. You will be greeted with an enumeration of CPU architectures. Select the architecture that suits your board by entering the number it is in the list and confirm by pressing 'enter'.

```
[fs-dev@localhost fs-secure-boot-tool-V<year>.<month>]$
./generate.sh
Enter image path (absolute path!)
/home/fs-dev/u-boot.bin
Do you want to create certificates, create jtag response key, sign
an image or encrypt an image (c/j/s/e)?
Note: When an image is encrypted it's also signed!
s
Do you want to use the default or custom settings (d/c)?
d
1) FSIMX6SDQ  3) FSIMX6UL   5) FSVYBRID
2) FSIMX6SX   4) FSIMX6ULL  6) FSIMX8MM
#? 6
```

Now you need to return the path (**absolute path!**) to the image you want to sign and press 'enter'.

```
Enter image path (absolute path!)
/home/developer/git/uboot_sec/nboot-fsimx8mm-2021.06.fs

Found NBoot
-> Found SPL
-> Found Board Configs: #########################
-> Found Firmware
--> Found DRAM settings: ##########
--> Found ATF
Found EXTRA
Signed image (nboot_8mm_signed.fs) successfully generated.
```

Now the signed image (in this example the signed U-Boot “`nboot_8mm_signed.fs`”) is created and is placed in the subdirectory `bin/` of your current directory.

```
[fs-dev@localhost fs-secure-boot-tool-V<year>.<month>]$ ls
bin/                      encryptImage.sh*          README.txt
boot_raw                  generate.sh*              release/
boot_raw_edited           History-F+S-UserTool.txt  signImage.sh*
createCertificates.sh*    jtag.sh*                  srkhash.txt
cst-3.2.0.tgz             makeAction.sh*
```

Repeat this with all images (except the root file system).

3.3 Using custom settings

Creating certificates

Since we at F&S Elektronik Systeme can't distribute the CST (developed from NXP) we can't feature default settings for the certification generation. But we highly recommend using the following settings:

- create a new CA (certificate authority) key
- key length in bits for PKI tree is 2048
- PKI (public key infrastructure) tree duration is 5 years (doesn't matter if it expires)
- SRKs (super root keys) are generated as CA keys

Note:

Creating the SRKs as CA certificates is recommended. Not all NXP CPUs including HAB support SRKs as non CA certificates. Furthermore using a different key length isn't recommended. NBoot doesn't support a greater value yet. Furthermore NBoot doesn't support keys larger than 2048 yet.

Signing an image

The steps are fairly the same as when you use the default settings. The only thing that changes is that enter an output name and the load address (in hexadecimal).

Note:

Custom Settings do not work for the I.MX8M NBoot.

Note:

The load address is entered without the leading '0x'.

```
 [fs-dev@localhost fs-secure-boot-tool-V<year>.<month>]$
./generate.sh

Do you want to create certificates, create jtag response key, sign
an image or encrypt an image (c/j/s/e)?

Note: When an image is encrypted it´s also signed!

e

Do you want to use the default or custom settings (d/c)?

c

Enter output name:

my_output_binary.bin

Enter Loadaddress (Hex):

Note: Type your Loadaddress without 0x at the beginning!

10800000

1) FSIMX6SDQ

2) FSIMX6SX

3) FSIMX6UL

4) FSIMX6ULL

5) FSVYBRID

#? 1
```

```
Enter image path (absolute path!)
/home/fs-dev/u-boot.bin

Signed image (my_output_binary.bin) successfully generated.
```

3.4 Advanced F&S Secure Boot Tool usage

Instead of calling “genereate.sh”, you can call “makeAction.sh” and give this script the transfer parameters.

Available parameters:

--output	output name
--cpu	CPU architecture
--action	create certificates, sign image or encrypt image
--image-path	absolute image path
--key-length	key length in bits
--master-key	master key otpmk
--jtag-resp-key	jtag responsible key

--output=...

Any string can be entered as an output name.

--cpu=...

Possible parameters:

1	FSIMX6SDQ
2	FSIMX6SX
3	FSIMX6UL
4	FSIMX6ULL
5	FSVYBRID
6	FSIMX8MM

--action=...

Possible parameters:

c	create
s	sign
e	encrypt
j	JTAG response key

--image-path=...

Absolute path to the image

--key-length=...

Possible parameters:

128	128 bit key
192	192 bit key
256	256 bit key

--master-key=...

Possible parameters:

o	master key is otpmk

--jtag-resp-key=...

Possible parameters:

hex-value	jtag responsible key 7 Bytes, Little-Endian

4 Setting up a new board

4.1 Preconditions

You read the documentation “*Linux on F&S Boards*”, installed all the necessary packages and set up a tftp service. Furthermore you downloaded the Code Signing Tool from NXP.com and the F&S Secure Boot Tool from F&S Elektronik Systeme. It should be mentioned that you also need the N- and U-Boot with security features.

4.2 Install secure images

The first step is to install the secure images, which corresponds to the architecture which you are using.

- n-boot-secure-<arch>.fs
- u-boot-secure-<arch>.fs

To install these images on your board please take a look to the document “Linux on F&S Boards”. You can find this document on our homepage.

Note:

You should both Images at the same time. Both the U-Boot and the N-Boot may not work as expected if the Board is reset while only one of them has been saved into flash.

4.3 Install signed Images

First you have to install Images with Secure Boot features as explained in chapter 4.2. A regular N-Boot and U-Boot will not work.

1. Execute the F&S Secure Boot Tool and sign a secure N-Boot and U-Boot image.
2. Transfer the images to the board and save it to flash.
3. Reset the board
4. After that check if any HAB event occurs, by writing “hab_status” in U-Boot.

If you see no hab Events you can continue to chapter 4.4, if you see hab events you should not continue until you found the problem, else you could damage your board.

4.4 Burn SRK Hash

Note:

When you burn the hash value to the fuses be very careful, since this value can't be changed. A wrong value means that the board won't boot anymore when it's closed.

In order to burn the SRK hash value into the fuses, you first need to enter the Secure U-Boot. Now you must enter the hash value. This is simply done by substituting the values below with those of your “srkhash.txt”. First you need to format the values with a “hexdump” like it is shown below. The “srkhash.txt” is located under the “crts/” directory of your code signing tool.

```
[fs-dev@localhost fs-secure-boot-tool-V<year>.<month>]$ hexdump -e
'/4 "0x"' -e '/4 "%X""\n"' < SRK_1_2_3_4_fuse.bin
```

Now you need to input each of the values carefully into their fuse individually.

```
# fuse prog -y 6 0 0x12e61558
# fuse prog -y 6 1 0xd4023941
# fuse prog -y 6 2 0xf69b56a7
# fuse prog -y 6 3 0x3457b2b4
# fuse prog -y 7 0 0xe86dad80
# fuse prog -y 7 1 0xe755cc48
# fuse prog -y 7 2 0xf2ca8c01
# fuse prog -y 7 3 0x3655f33c
```

4.5 Enable Secure Boot

Now that the hash is burned into the device it is ready to be closed. To do this another fuse needs to be burned.

```
# fuse prog 1 3 0x02000000
```

Reset the Board and wait a few seconds.

Note:

If you don’t see anything there was something wrong and the image cannot be verified. So the board is now broken. If the Hash was burned into the fuses correctly, it may be possible to recover the Board with a manufacturer tool and correctly signed Image.

5 Flowcharts

5.1 Create certificates and burn SRK hash

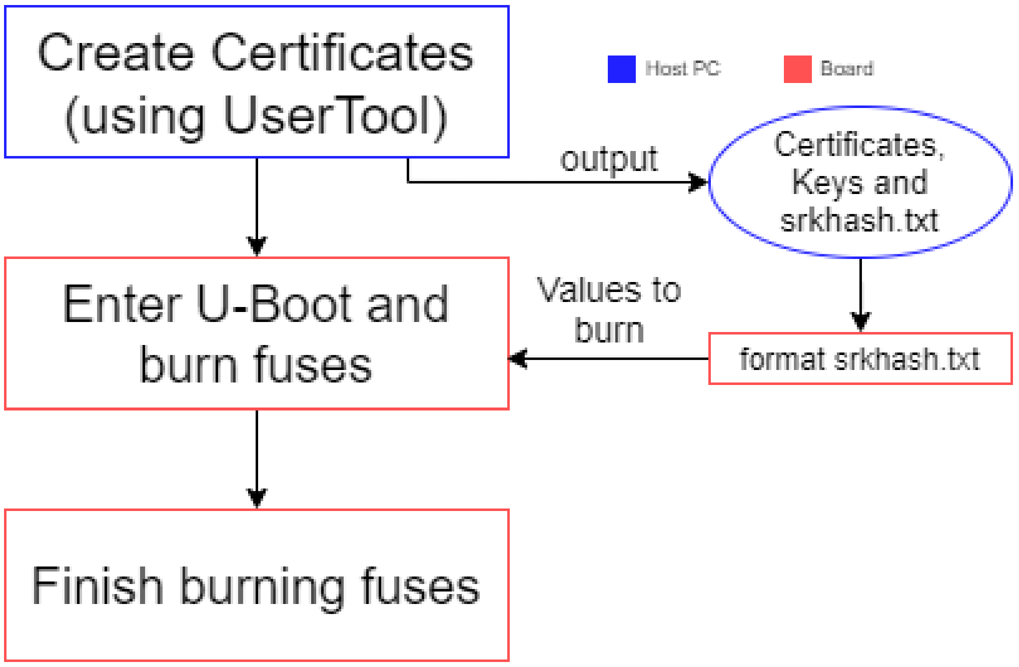

Create Certificates (using UserTool)
Host PC
Board
output
Certificates, Keys and srkhash.txt
Enter U-Boot and burn fuses
Values to burn
format srkhash.txt
Finish burning fuses

Figure 2: Creating certificates and blow fuses (new board)

Note:

You have to create your certificates only once! Furthermore you have to blow the fuses on every board!

5.2 Sign an image

Flowcharts

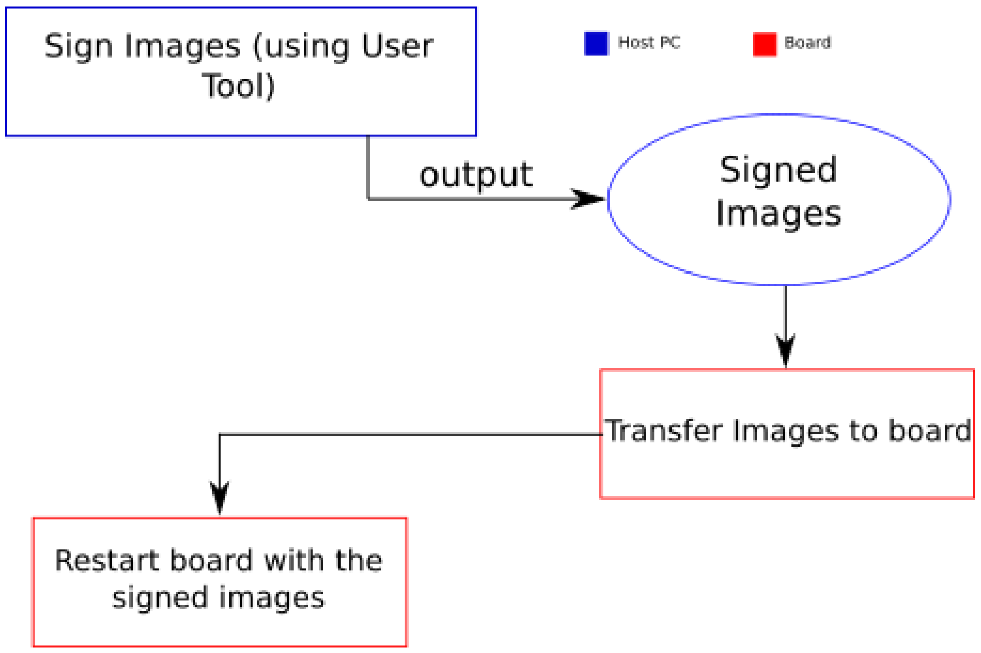

Sign Images (using User Tool)
Host PC
Board
output
Signed Images
Transfer Images to board
Restart board with the signed images

Figure 3: Signing images

Note:

You have to sign your images only once.

List of Figures

List of Tables

Listings

Important Notice