F&S Documentation “Secure Boot”

Documentation for the “F&S Secure Boot”

Version 1.3
(2020-02-10)

About This Document

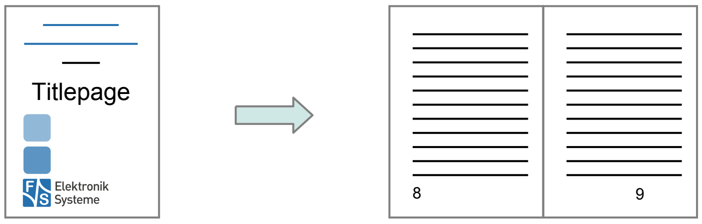
This document is intended to explain the usage of the Secure Boot on F&S Boards.

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
2017-01-27	1.0	Linux	A	All	Initial document generation	FB
2017-12-07	1.1	Linux	M	All	Modify different chapters because of reworking the user tool	PJ
2019-01-31	1.2	Linux	M	All	Modify different chapters because of reworking the F&S Secure Boot Tool	PJ
2020-02-10	1.3	Linux	M	All	Update chapters to new code and reorder the structure	PJ

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. 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)
- encrypt an image (encrypt a specific memory range with a key)
- create a JTAG response key

The F&S Secure Boot Tool is a collection of shell scripts, which perform their individual task (i.e. create certificates, sign an image, encrypt an image or create a JTAG response key).

1.1 Boot Sequence

Figure 1 shows the typical boot sequence of the board.

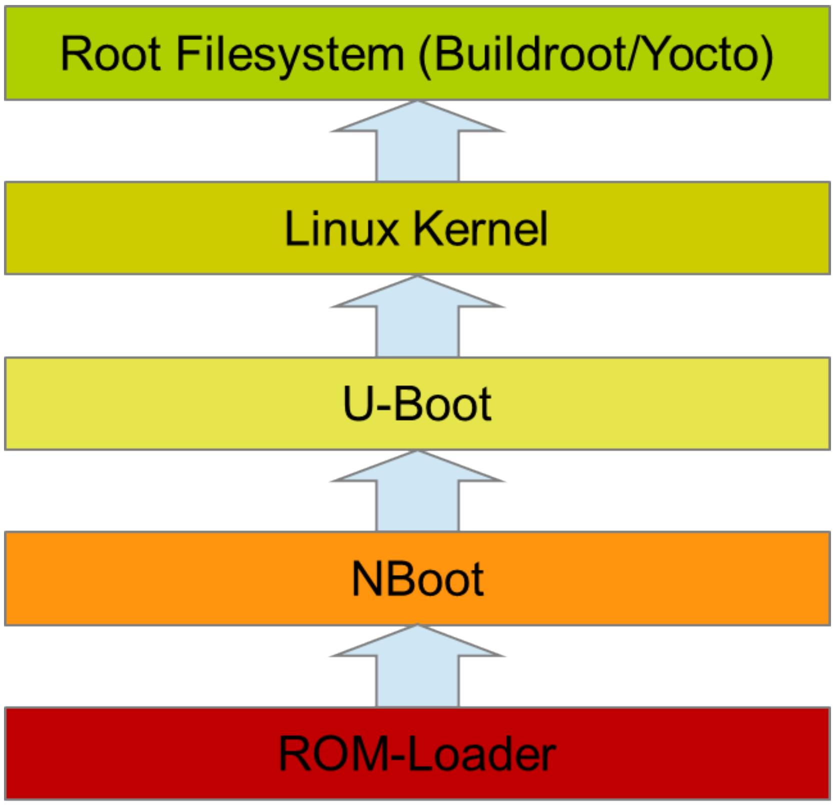


Figure 1: Boot sequence

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, SD card, or similar). However at this point of time the dynamic RAM (D-RAM) is not yet initialized. The chip does not even know if there is any D-RAM attached at all. So it can only load this code to some rather small internal RAM (IRAM). 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 NBoot on F&S systems. This step-stone loader is loaded to IRAM and then executed
2. NBoot (short for NAND-Boot) is a rather small step-stone bootloader. It detects and activates the D-RAM and then loads the main bootloader U-Boot. NBoot is the same for boards with Linux and Windows Embedded Compact. Only from then on, the boot sequence differs. As long as NBoot is on the board, the Linux system can be re-installed at any time without the need of special software or hardware tools. When erasing the memory, NBoot is supposed to stay on the board.
3. 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.
4. The Linux kernel provides the Linux operating system including the device drivers. It will activate the peripherals and finally mounts the root filesystem.
5. 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.

With the F&S Secure Boot package you are able to sign/encrypt the following stages:

- NBoot
- U-Boot
- Linux Kernel

1.2 Secure NBoot

The secure NBoot is only available as a binary. The source code will not published. Below there is a table which commands are in NBoot available.

NBoot VS (Secure) Commands	
Menu	
\r	re-display menu
\n	re-display menu
?	Show some of the most important commands
E	Erase whole flash, excluded NBoot
r	Reboot hardware
d	Serial Download of bootloader
D	Serial Download of bootloader
f[2]	Save image to flash
l[2]	Loading installed bootloader image (UBoot/EBoot)
L[2]	Loading installed bootloader image (NBoot)
x[2]	execute image
c	Load EBoot/U-Boot from SD card (not implemented yet)
B	Show bad blocks
N	Load NBoot from SD card (not implemented yet)
.	Show fuses menu
Fuses Menu	
1[1]	Blow fuses to boot from NAND (only allowed if board is in open mode)
7[1]	Blow fuses Serial Number (only allowed if board is in open mode) and lock serial number
8[1]	Enable Watchdog (only allowed if board is in open mode)
R[3]	Revoke SRK
s[1]	dump secure fuses: Security Configuration, SRK-Revoke, JTAG Mode, *SRK-HASH
S[1]	Set SRK-HASH (only allowed if board is in open mode) and lock SRK

J[1]	Set JTAG mode fuse, if necessary JTAG responsible key too and lock JTAG
I[1]	HAB Status and Event logging
B[1]	Set Boot Security Settings (only allowed if board is in open mode)

Table 1: NBoot VS commands

[1] only allowed if board is in open mode

[2] Images will be verified

[3] If in closed mode only available if CSF is set with feature [Unlock] – Engine = OCOTP, Features = SRK Revoke

1.3 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 one of the following NAND partitions will be written with the “nand write” command, the images will be verified. Partitions: UBoot, Kernel, FDT and Images.
- If the boot/bootz/bootm/bootd command is executed the images will be verified.

Supported Image types:

- UBoot
- zImage
- legacy Image (uImage)
- FIT-Images
- Device-Trees

It is necessary that only the outer images are signed not the inner. So if you are using e.g. an FIT-Image, the images inside of the FIT-Image are not allowed to be signed, only the FIT-Image itself. The same applies to the legacy image.

1.3.1 Build Secure U-Boot

The following example refers to the architecture fsimx6.

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

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

After that setup the correct secure defconfig.

```
[fs-dev@localhost u-boot-2018.03-fssecure-V<year>.<month>]$ make
fsimx6_secure_boot_defconfig
```

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

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

The output image is called **u-boot.bin** and can be signed now.

Note:

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

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 "**3.2.0**". 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-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	Script to unpack F&S Secure Boot source packages to a build directory
binaries/	Images to be used with the board directly
nbootimx6_secure_<v>.bin	NBoot fsimx6 secure bootloader image
nbootimx6sx_secure_<v>.bin	NBoot fsimx6sx secure bootloader image
nbootimx6ul_secure_<v>.bin	NBoot fsimx6ul secure bootloader image
u-boot-secure-fsimx6.bin	U-Boot fsimx6 secure image without padding
u-boot-secure-fsimx6sx.bin	U-Boot fsimx6sx secure image without padding
u-boot-secure-fsimx6ul.bin	U-Boot fsimx6ul secure image without padding
/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
doc/	Documentation
FS_Secure_Boot.pdf	F&S Secure Boot document

Table 2: 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
```

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.

If you prefer to do the installation by hand, well, the script more or less executes the following commands, just with some more checks and directory switching.

```
mkdir <build-dir>
tar xf u-boot-2018.03-fssecure-V<year>.<month>.tar.bz2
tar xf fs-secure-boot-tool-V<year>.<month>.tar.bz2
```

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 and **/**.

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

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.

2.5 Encrpyt/Sign 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 Images
- UBoot Images
- zImages
- uImages
- FIT-Images
- Device Tree Images

2.5.1 Signed Images

Here is a small overview which parts of the images will be signed by default.

Image Types	Signed areas	
	Offset	Length
NBoot Images[1]	0x0 0x14	0x10 end of Image
UBoot Images	0x0	end of Image
zImages	0x0	end of Image
uImages	0x0	end of Image
FIT-Images	0x0	end of Image
Device Tree	0x0	end of Image

[1] There are 4 Bytes in NBoot which will dynamically setup while transferring it. So that´s the reason why these 4 Bytes can´t be signed.

2.5.2 Encrypted Images

Here is a small overview which parts of the images will be signed by default.

Image Types	Signed areas		Encrypted areas	
	Offset	Length	Offset	Length
NBoot Images[1][2]	0x0 0x14	0x10 0x41c	0x430	end of Image
UBoot Images[3]	0x0	0x80	0x80	end of Image
zImages[4]	0x0	0x68	0x68	end of Image
uImages[5]	0x0	0x80	0x80	end of Image
FIT-Images[5]	0x0	0x68	0x68	end of Image
Device Tree[5]	0x0	0x68	0x68	end of Image

[1] There are 4 Bytes in NBoot which will dynamically setup while transferring it. So that´s the reason why these 4 Bytes can´t be signed.

[2] The signing goes from beginning of the Image to the IVT and Boot Data Header. Everything else is encrypted.

[3] The signing goes from beginning of the Image to the Magic Number of the UBoot. Everything else is encrypted.

[4] The signing goes from beginning of the Image to the Magic Number of the zImage. Everything else is encrypted.

[5] The signing goes from beginning of the Image to the whole header of the Image. Everything else is encrypted.

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 'enter' 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 create once 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', 'enter' and 'd', 'enter'. You will be reeted with an enumeration of CPU architectures. Select the architecture that suits your board by pressing '1', '2', '3', '4' or '5' and 'enter'.

```
[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!
s
Do you want to use the default or custom settings (d/c)?
d
1) FSIMX6SDQ
2) FSIMX6SX
3) FSIMX6UL
4) FSIMX6ULL
5) FSVYBRID
#? 1
```

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

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


Signed image (uboot_sdq_signed.nb0) successfully generated.
```

Now the signed image (in this example the signed U-Boot “`uboot_sdq_signed.nb0`”) 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 Encrypting an image

At the moment we only support OTPMK as master key.

Execute the “`generate.sh`” script. After that enter 'e', 'enter' and 'd', 'enter'. You will be greeted with an enumeration of CPU architectures. Select the architecture you have on your board by pressing '1', '2', '3', '4' or ‘5’ and 'enter'.

```
[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)?
d
1) FSIMX6SDQ
```

```
2) FSIMX6SX
3) FSIMX6UL
4) FSIMX6ULL
5) FSVYBRID
#? 1
```

Now you need to enter the path to the image (**absolute path!**) you want to encrypt and press 'enter'. After you have done this, you need to enter the key length (128/192/256 bits).

```
Enter image path (absolute path!)
/home/fs-dev/u-boot.bin
Enter key length (128/192/256):
256
Insert key
```

The F&S Secure Boot Tool asks you for your key “Insert key”. Before we can do this, we need to generate the key (so called dek blob) on the board. The private key called `dek.bin` was created by the tool and automatically copied to the `/tftpboot` folder.

After entering the U-Boot, transfer the “dek.bin” to the board using tftp and generate the key by running:

```
U-Boot 2014.07-ga8061e6 (Dec 07 2017 - 09:25:44) for F&S


CPU:      Freescale i.MX6DL rev1.3 at 792 MHz
RESET:    POR
Board:    efusA9 Rev. 1.20 (LAN, eMMC, 2x DRAM)
DRAM:     512 MiB
NAND:     256 MiB
MMC:      FSL_SDHC: 0, FSL_SDHC: 1
In:       serial
Out:      serial
Err:      serial
Net:      FEC [PRIME]
USB EHCI 1.00
```

```
Hit any key to stop autoboot: 0
efusA9 # tftp dek.bin
Using FEC device
TFTP from server 10.0.0.109; our IP address is 10.0.0.252
Filename 'dek.bin'.
Load address: 0x10800000
Loading:
  *
  2.9 KiB/s
done
Bytes transferred = 32 (0x20)
efusA9 # dek_blob $loadaddr 0x10800100 256
SEC: RNG instantiated

Encapsulating provided DEK to form blob
DEK Blob
8100584166552000CC526D82F83EB075FF0228052EA1B3E2E502F436B142CDE30F
08053DD00C4397DCCB3E2DBC350260A46BC3EDC07E15A5538DFA9FFAF9C6B64902
35D332CFEE6B8E9B9B4F22B5C7F2CA081466553C03EB
efusA9 #
```

The key in the red box is the key we need to attach to the image. This is done by marking the key and copy it into the terminal where the F&S Secure Boot Tool is currently running and hit 'enter'.

```
Enter key length (128/192/256):
256
Insert key
8100584166552000CC526D82F83EB075FF0228052EA1B3E2E502F436B142CDE30F
08053DD00C4397DCCB3E2DBC350260A46BC3EDC07E15A5538DFA9FFAF9C6B64902
35D332CFEE6B8E9B9B4F22B5C7F2CA081466553C03EB
Encrypted image (uboot_sdq_enc_otpmk.nb0) successfully generated.
```

Note:

Your generated key looks different! This is just a sample key. Furthermore from time to time the key won't generate on the board (means there is no key outputted). The best practice is to perform a hardware reset and try again.

Below is the full example output log again.

```
[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)?

d

1) FSIMX6SDQ

2) FSIMX6SX

3) FSIMX6UL

4) FSIMX6ULL

5) FSVYBRID

#? 1

Enter image path (absolute path!)

/home/fs-dev/u-boot.bin

Enter key length (128/192/256):

256

Insert key

8100584166552000CC526D82F83EB075FF0228052EA1B3E2E502F436B142CDE30F
08053DD00C4397DCCB3E2DBC350260A46BC3EDC07E15A5538DFA9FFAF9C6B64902
35D332CFEE6B8E9B9B4F22B5C7F2CA081466553C03EB

Encrypted image (uboot_sdq_enc_otpmk.nb0) successfully generated.
```

Now, the encrypted image is ready and is placed in the subdirectory `bin/` of your current directory.

Note:

Since the CST from NXP uses the "/dev/random" device, it can take some time until the images are encrypted.

3.4 Creating a JTAG response key

If you want to setup the JTAG mode to secure then you have to create a JTAG response key. Execute the “generate.sh” script. After that enter ‘j’, ‘enter’ and then enter the JTAG response key 7 Bytes and Little-Endian.

```
[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!

j

Please enter the response key (7 Bytes) (Little-Endian) and
confirm it with enter

123456789abcde

checksum is: 840

jtag_resp_key.txt is ready!
```

Now the JTAG response key called `jtag_resp_key.txt` is created and is placed in the current directory.

3.5 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:

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.
```

Encrypting an image

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

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
Enter key length (128/192/256):
256
Insert key
8100584166552000CC526D82F83EB075FF0228052EA1B3E2E502F436B142CDE30F
08053DD00C4397DCCB3E2DBC350260A46BC3EDC07E15A5538DFA9FFAF9C6B64902
35D332CFEE6B8E9B9B4F22B5C7F2CA081466553C03EB


Encrypted image (my_output_binary.bin_enc_otpmk) successfully
generated.
```

3.6 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

--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*" and 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. We assume that our architecture is *fsimx6*.

- nbootimx6_secure_<v>.bin
- u-boot-secure-fsimx6.bin

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.

4.3 Burn SRK Hash

In order to burn the SRK hash value into the fuses, you first need to enter the NBoot.

After you entered the NBoot press '.' to show the hidden menu. In this hidden menu there is an entry called "Set SRK-HASH". You enter this function by pressing 'S'.

Now you must enter the hash value. This is simply done by transferring the "srkhash.txt" to the board (after you pressed 'S'!). The hash sum will be printed in order to verify it. After you verified the hash sum, press 'y'. Now the hash sum is burned into the fuses which means, that when we close the board (done in the next step) the ROM functions uses this value to verify an image.

Note:

When you burned 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.

```
Please select action
'd'  -> Serial download of bootloader
'E'  -> Erase flash
'B'  -> Show bad blocks
Use NetDCUUsbLoader for USB download

Select fuse settings
'1'  -> Boot from NAND (64 pages)
'S'  -> Set SRK-HASH
'B'  -> Set Boot Security Settings
'J'  -> Set JTAG mode
's'  -> dump secure fuses
'l'  -> HAB events
'R'  -> Revoke SRK
THIS IS A PERMANENT CHANGE
please enter the Hash-Value and confirm it with enter

CRC-Check successful!
The following fuses will be blown:
  address 0x18 value 0x12e61558
  address 0x19 value 0xd4023941
  address 0x1a value 0xf69b56a7
  address 0x1b value 0x3457b2b4
  address 0x1c value 0xe86dad80
  address 0x1d value 0xe755cc48
  address 0x1e value 0xf2ca8c01
  address 0x1f value 0x3655f33c
  address 0x00 value 0x00004000

Enter 'y' to proceed, any other key to exit

Blowing fuses
address 0x18 value 0x12e61558 ...blown
address 0x19 value 0xd4023941 ...blown
address 0x1a value 0xf69b56a7 ...blown
address 0x1b value 0x3457b2b4 ...blown
address 0x1c value 0xe86dad80 ...blown
address 0x1d value 0xe755cc48 ...blown
address 0x1e value 0xf2ca8c01 ...blown
address 0x1f value 0x3655f33c ...blown
address 0x00 value 0x00004000 ...blown
Finished blowing fuses
```

4.4 Set JTAG mode

An important and powerful interface is JTAG. JTAG is by default activated so everybody can use this interface to debug the system. That’s why JTAG is a Security vulnerability if you are using Secure Boot. You have the possibility to disabled the JTAG interface or using JTAG secure, which means you can use the JTAG interface with a private key. It is necessary to setup the JTAG mode before you enable the Secure Boot, because afterwards there is a write protect on the fuses so can´t change the bits for JTAG.

Disable JTAG:

Disable JTAG is done by entering the hidden menu again and press 'J' after that you need to press 'd' and confirm with 'y'. The board has now disabled the JTAG interface.

```
Please select action
‘d’  -> Serial download of bootloader
‘E’  -> Erase flash
‘B’  -> Show bad blocks
Use NetDCUUsbLoader for USB download

Select fuse settings
‘1’  -> Boot from NAND (64 pages)
‘S’  -> Set SRK-HASH
‘B’  -> Set Boot Security Settings
‘J’  -> Set JTAG mode
‘s’  -> dump secure fuses
```

```
'l'  -> HAB events
'R'  -> Revoke SRK


THIS IS A PERMANENT CHANGE!


please enter 's' to set secure JTAG or 'd' to set disable JTAG
debugging
The following fuses will be blown:
  address 0x06 value 0x00100000
  address 0x06 value 0x00c00000


Enter 'y' to proceed, any other key to exit


Blowing fuses
  address 0x06 value 0x00100000
  address 0x06 value 0x00c00000
Finished blowing fuses
```

Secure JTAG:

Secure JTAG is done by entering the hidden menu again and press 'J' after that you need to press 's' and enter the response key which you have created with the User Tool. After that confirm it with 'y'. The JTAG interface is now set to secure JTAG.

```
Please select action
'd'  -> Serial download of bootloader
'E'  -> Erase flash
'B'  -> Show bad blocks
Use NetDCUUsbLoader for USB download


Select fuse settings
'1'  -> Boot from NAND (64 pages)
'S'  -> Set SRK-HASH
'B'  -> Set Boot Security Settings
'J'  -> Set JTAG mode
's'  -> dump secure fuses
```

```
'l'  -> HAB events
'R'  -> Revoke SRK

THIS IS A PERMANENT CHANGE!

please enter 's' to set secure JTAG or 'd' to set disable JTAG
debugging
please enter the response key and confirm it with enter
CRC-Check successful!
The following fuses will be blown:
  address 0x20 value 0x789abcde
  address 0x21 value 0x00123456

Enter 'y' to proceed, any other key to exit

Blowing fuses
  address 0x20 value 0x789abcde
  address 0x21 value 0x00123456
Finished blowing fuses
The following fuses will be blown:
  address 0x06 value 0x00100000
  address 0x06 value 0x00c00000

Enter 'y' to proceed, any other key to exit

Blowing fuses
  address 0x06 value 0x00100000
  address 0x06 value 0x00c00000
Finished blowing fuses
```

4.5 Enable Secure Boot

Before you enable Secure Boot be sure you have installed a correctly signed image. Once burned this fuse you are not able to revoke the fuse. Also you have to check the JTAG mode. By default the JTAG mode is open. So be sure you setup JTAG secure or disable the interface. If not anybody are able to connect the board to JTAG and debug the system.

To set the board configuration to “closed”. This is done by entering the hidden menu again and press 'B' after that you need to press 'c' and confirm with 'y'. The board is now in closed configuration and will only accept correct signed/encrypted images.

```
Please select action
‘d’ -> Serial download of bootloader
‘E’ -> Erase flash
‘B’ -> Show bad blocks
Use NetDCUUsbLoader for USB download

Select fuse settings
‘1’ -> Boot from NAND (64 pages)
‘S’ -> Set SRK-HASH
‘B’ -> Set Boot Security Settings
‘J’ -> Set JTAG mode
‘s’ -> dump secure fuses
‘l’ -> HAB events
‘R’ -> Revoke SRK
testing NBoot …
found CSF
successfully authenticated

THIS IS A PERMANENT CHANGE!

HAB STATUS: Open
please enter ‘c’ to set HAB-Mode: Closed
The following fuses will be blown:
  address 0x06 value 0x00000002
  address 0x08 value 0x00100000
```

```
    address 0x06 value 0x00060000
    address 0x00 value 0x0000000c

Enter 'y' to proceed, any other key to exit

Blowing fuses
address 0x06 value 0x00000002 ...blown
address 0x08 value 0x00100000 ...blown
address 0x06 value 0x00060000 ...blown
address 0x00 value 0x0000000c ...blown
Finished blowing fuses
```

4.6 Dek blob creation

If you want to create an encrypted image, it is required that you have installed secure U-Boot.

After entering the U-Boot, transfer the key from User Tool “`dek.bin`” to the board e.g. using TFTP.

```
U-Boot 2014.07-ga8061e6 (Dec 07 2017 - 09:25:44) for F&S


CPU:      Freescale i.MX6DL rev1.3 at 792 MHz
RESET:    POR
Board:    efusA9 Rev. 1.20 (LAN, eMMC, 2x DRAM)
DRAM:     512 MiB
NAND:     256 MiB
MMC:      FSL_SDHC: 0, FSL_SDHC: 1
In:       serial
Out:      serial
Err:      serial
Net:      FEC [PRIME]
USB EHCI 1.00
Hit any key to stop autoboot: 0
efusA9 # tftp dek.bin
Using FEC device
```

```
TFTP from server 10.0.0.109; our IP address is 10.0.0.252
Filename 'dek.bin'.
Load address: 0x10800000
Loading:
  *
  2.9 KiB/s
done
Bytes transferred = 32 (0x20)
efusA9 #
```

Now you can call the dek_blob function in U-Boot. The dek_blob function expect 3 parameters

1. Source RAM address which points to the key
2. Destination RAM address which points to the dek blob
3. Key length, is the length which you have entered during the F&S Secure Boot Tool execution.

Now generate the key by running:

```
efusA9 # dek_blob $loadaddr 0x10800100 256
SEC: RNG instantiated

Encapsulating provided DEK to form blob
DEK Blob
8100584166552000CC526D82F83EB075FF0228052EA1B3E2E502F436B142CDE30F
08053DD00C4397DCCB3E2DBC350260A46BC3EDC07E15A5538DFA9FFAF9C6B64902
35D332CFEE6B8E9B9B4F22B5C7F2CA081466553C03EB
efusA9 #
```

The key in the red box is the output key and is known as dek blob.

5 Getting started

The first step you need to do is create certificates and the resulting `srkhash.txt` (for burning the fuses).

5.1 Signed Boot

1. Execute the F&S Secure Boot Tool and sign a secure NBoot image.
2. Transfer this NBoot image to the board and save it to NAND flash.
3. After that check if any HAB event occurs.
4. Setup JTAG mode.
5. Burn the “closed” fuse to activate Secure Boot.
6. Power off the board for a few seconds and Power it on again.
   a. If you see the NBoot menu the images is correctly signed and Secure Boot is working.
   b. If you don’t see anything there was something and the image cannot be verified. So the board is now broken.

5.2 Encrypted Boot

Preconditions:

Before you do an encrypted Boot be sure you have done a Signed Boot and it was successfully.

Also you must have installed a signed secure NBoot and a signed secure UBoot.

1. Now execute the F&S Secure Boot Tool and encrypt an image. If the line “Insert key” prompts you have to create the dek blob on the Board.
2. Insert created dek blob to F&S Secure Boot Tool.
3. Transfer this image to the board and save it to NAND flash.
4. If no HAB Event occurs everything was successful.

6 Flowcharts

6.1 Create certificates and burn SRK hash

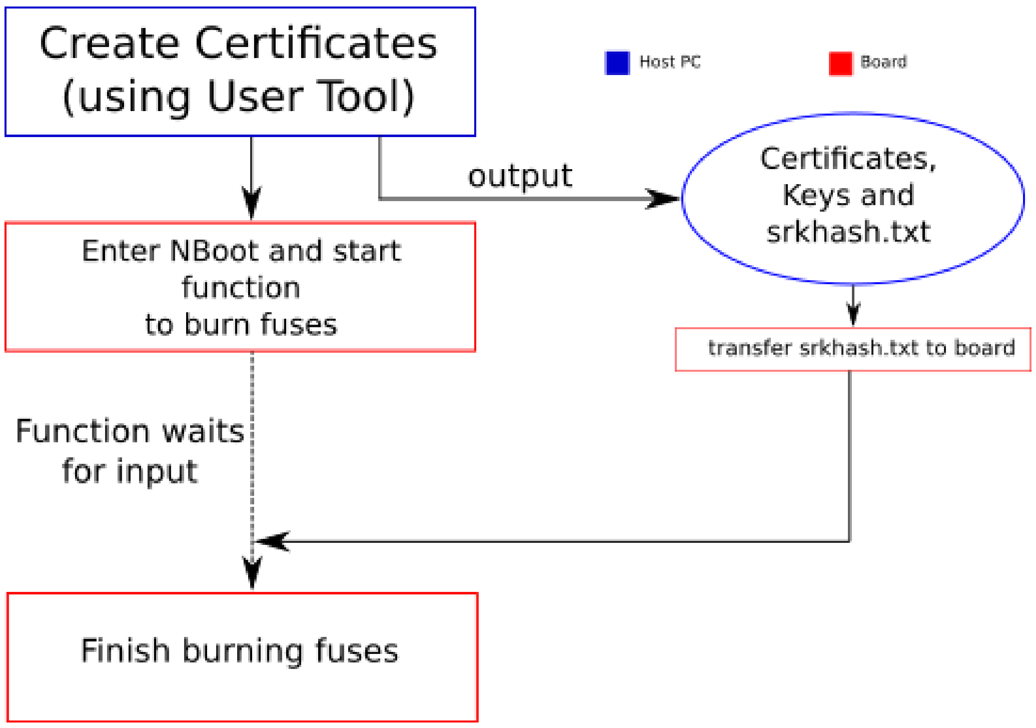

Create Certificates (using User Tool)
Host PC
Board
output
Certificates, Keys and srkhash.txt
Enter NBoot and start function to burn fuses
transfer srkhash.txt to board
Function waits for input
Finish burning fuses

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!

6.2 Sign an image

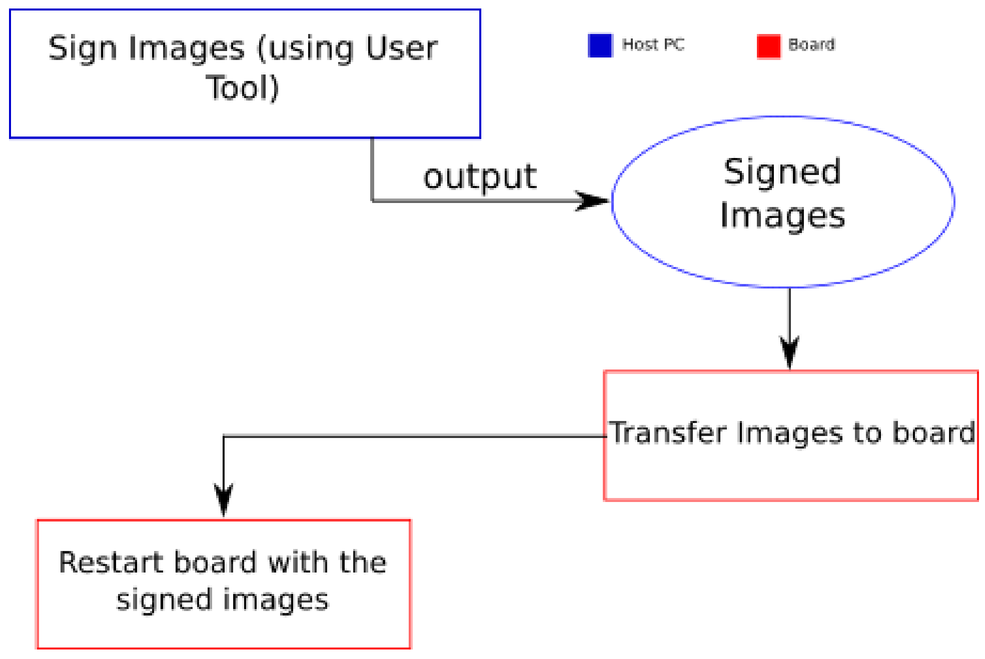

Sign Images (using User Tool)
Host PC
Board
output
Signed Images
Transfer Images to board
Restart board with the signed images

Figure 3: Signing images

Note:

You have to sign your images only once.

6.3 Encrypt an image

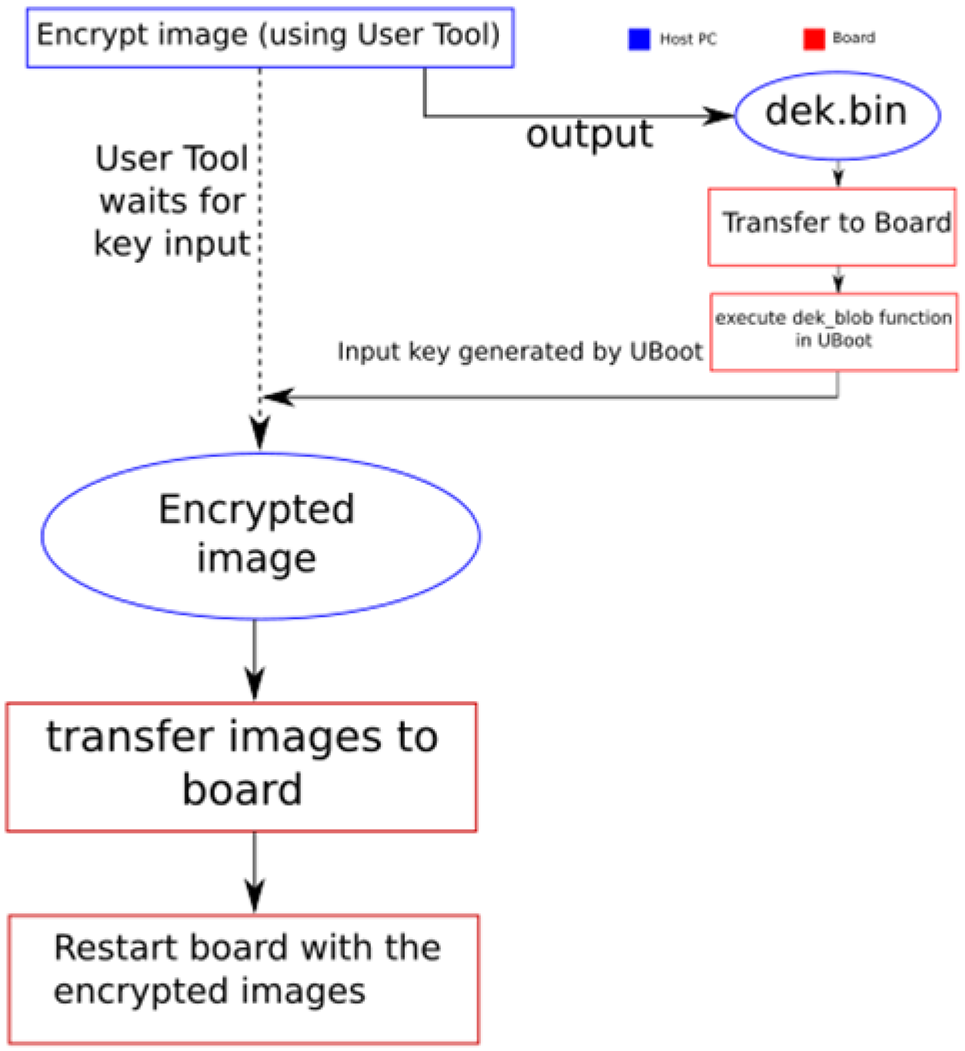

Encrypt image (using User Tool)
Host PC
Board
dek.bin
output
User Tool waits for key input
Transfer to Board
execute dek_blob function in UBoot
Input key generated by UBoot
Encrypted image
transfer images to board
Restart board with the encrypted images

Figure 4: Encrypting images

List of Figures

List of Tables

Listings

Important Notice