Freescale Semiconductor, Inc.
Application Note

Document Number: AN4581
Rev. 1, 10/2015

Secure Boot on i.MX50, i.MX53, and i.MX 6 Series using HABv4

1. Introduction

1.1. Purpose

Executing trusted and authentic code on an applications processor starts with securely booting the device. The i.MX family of applications processors provides this capability with the High Availability Boot (HAB) component of the on-chip ROM. The ROM is responsible for loading the initial program image from the boot medium. HAB enables the ROM to authenticate the program image by using digital signatures. This initial program image is usually a bootloader.

HAB provides a mechanism to establish a root of trust for the remaining software components and establishes a secure state on the i.MX IC's secure state machine in hardware.

The purpose of this application note is to provide a secure boot reference for i.MX applications processors that include HABv4. It demonstrates an example for generating a signed U-Boot image and configuring the IC to run securely.

1.2. Audience

This document is intended for those who:

- Need an example of the procedure for signing a boot image.
- Need to design signed software images to be used with a HAB-enabled processor.

Contents

It is assumed that the reader is familiar with the basics of digital signatures and public key certificates.

1.3. Scope

This document is a practical example to illustrate the construction of a secure boot image, in addition to configuring the target device to run securely. Extending the secure boot chain past the initial stage is also possible with HAB, but that is beyond the scope of this document.

This document answers the following questions:

- How is the hardware configured?
- What components are required?
- How is each of these different components generated?
- How are all these components assembled to create a signed image?

NOTE

This document covers secure boot using i.MX6. HABv4 is present on the following i.MX processors: i.MX50, i.MX53, and i.MX 6 Series. This document applies to all HABv4 processors. Secure boot features for other processors, such as i.MX25, i.MX35, and i.MX51, which use HABv3, are documented in *Secure Boot on i.MX25, i.MX35, and i.MX51 using HAB3* application note (document AN4547).

NOTE

Secure boot features for i.MX28 are documented in *Secure Boot with i.MX28 HAB v4* application note (document AN4555). i.MX28 supports HABv4, but its boot architecture is significantly different from other processors in the i.MX family.

NOTE

Secure boot using HAB is no longer supported on i.MQX27 and i.MQX31.

1.4. Definitions, Acronyms, and Abbreviations

Definitions of the terms and acronyms in this document are as follows:

- CA: Certificate Authority, the holder of a private key used to certify public keys.
- CAAM: Cryptographic Acceleration and Assurance Module, an accelerator for encryption, stream cipher, and hashing algorithms, with a random number generator and run time integrity checker.
- CMS: Cryptographic Message Syntax, a general format for data that may have cryptography applied to it, such as digital signatures and digital envelopes. HAB uses the CMS as a container holding PKCS#1 signatures.

- CSF: Command Sequence File, a binary data structure interpreted by the HAB to guide authentication operations.
- CST: Code Signing Tool, an application running on a build host to generate a CSF and associated digital signatures.
- DCD: Device Configuration Data, a binary table used by the ROM code to configure the device at early boot stage.
- HAB: High Assurance Boot, a software library executed in internal ROM on the Freescale processor at boot time which, among other things, authenticates software in external memory by verifying digital signatures in accordance with a CSF. This document is strictly limited to processors running HABv4.
- IVT: Image Vector Table.
- OS: Operating System.
- OTP: One-Time Programmable. OTP hardware includes masked ROM, and electrically programmable fuses (eFuses).
- PKCS#1: Standard specifying the use of the RSA algorithm.
- PKI: Public Key Infrastructure, a hierarchy of public key certificates in which each certificate (except the root certificate) can be verified using the public key above it.
- RSA: Public key cryptography algorithm developed by Rivest, Shamir, and Adleman.
- Accelerator (including hash acceleration) found on some processors.
- SDP: Serial Download Protocol, also called UART/USB Serial Download Mode. This allows code provisioning through UART or USB during production and development phases.
- SRK: Super Root Key, an RSA key pair which forms the start of the boot-time authentication chain. The hash of the SRK public key is embedded in the processor using OTP hardware. The SRK private key is held by the CA. Unless explicitly noted, SRK in this document refers to the public key only.
- UID: Unique Identifier, a unique value (such as a serial number) assigned to each processor during fabrication.

1.5. References

- *i.MX50 Reference Manual*, *i.MX53 Reference Manual*, and *i.MX 6Dual/6Quad Reference Manual*.
- *i.MX53 Security Reference Manual*, *i.MX 6Dual/6Quad Security Reference Manual*, and *i.MX 6Solo/6DualLite Security Reference Manual*.
- *HAB CST User Guide* available in the *Code Signing Tool* package downloadable on freescale.com. Search for IMX_CST_TOOL.

2. Overview

HAB authentication is based on public key cryptography using the RSA algorithm in which image data is signed offline using a series of private keys. The resulting signed image data is then verified on the i.MX processor using the corresponding public keys. This key structure is known as a PKI tree. Super Root Keys, or SRK, are components of the PKI tree. HAB relies on a table of the public SRKs to be hashed and placed in fuses on the target. The *Freescale Code Signing Tool*, CST, is used in this guide to generate the HABv4 signatures for images using the PKI tree data and SRK table. On the target, HAB evaluates the SRK table included in the signature by hashing it and comparing the result to the SRK fuse values. If the SRK verification is successful, this establishes the root of trust, and the remainder of the signature can be processed to authenticate the image. Detailed information for CST and HAB can be found in their respective user guides included in the CST package.

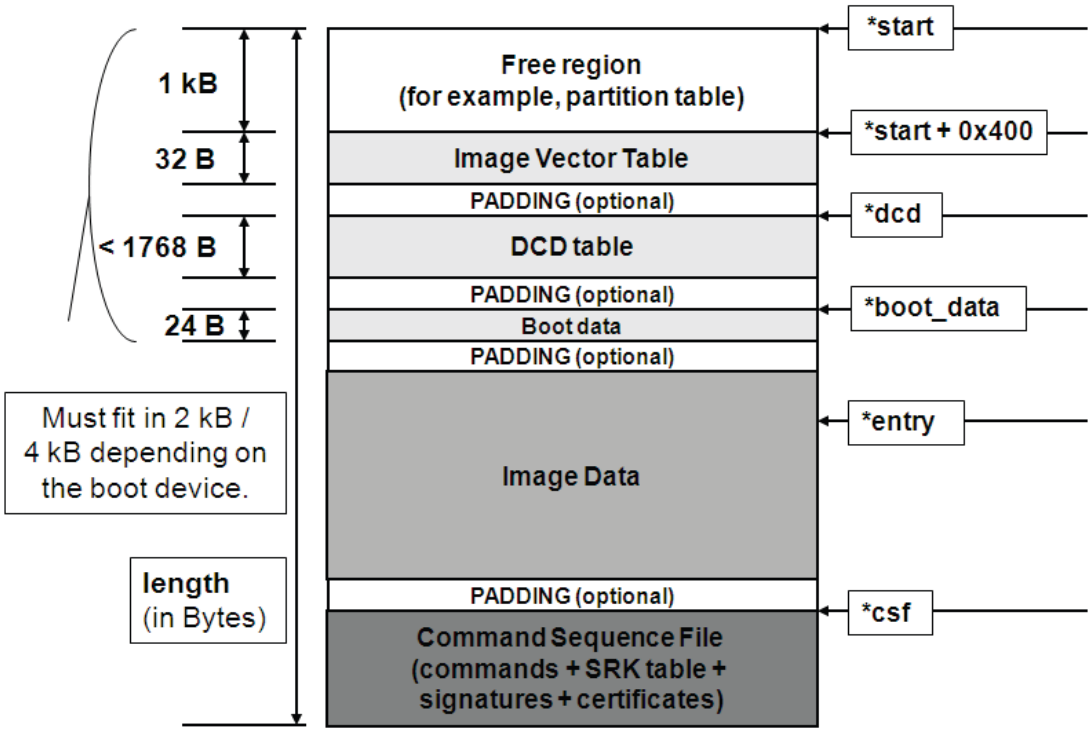

Figure 1. Typical memory layout of a signed image

NOTE

HAB requires that the IVT, initial bye of boot data, DCD table, and first word of the image must all be covered by a digital signature.

3. Code signing example

The following sections detail the step by step process to securely boot an i.MX 6 Series part with HABv4. After completing the steps, the image will have a valid HABv4 signature attached and the part will be “closed”. Once “closed”, the part will only execute signed images.

3.1. Generate PKI tree

CST includes scripts for generating a PKI tree and SRK table. The PKI tree for this example is detailed in this figure.

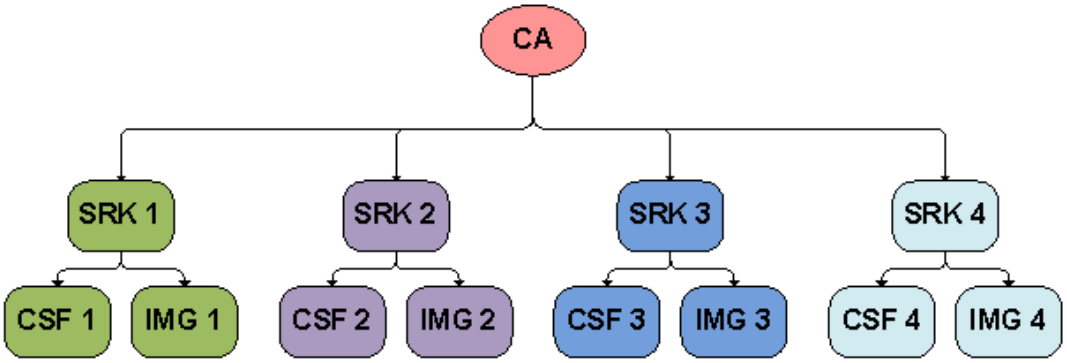

Figure 2. HABv4 PKI tree

Our example uses the first SRK as the root of trust. The CSF 1 and IMG 1 keys are used to sign the CSF data and the image respectively. CST contains a script in the "keys" directory that generate the above PKI tree.

- From the "keys" directory, execute:

```
./hab4_pki_tree.sh

    ++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
    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
Enter key length in bits for PKI tree: 2048
Enter PKI tree duration (years): 10
How many Super Root Keys should be generated? 4
```

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

After completing the questions, the PKI tree is created. This example tree creates a new CA, uses 2048 bit keys, lasts for 10 years (HAB does not consider the duration), and has 4 SRKs. For all i.MX series HABv4 enabled parts except the i.MX6SX, the last question regarding the "CA flag" in the SRK must be answered as "y". The resulting private keys are placed in the keys directory of the CST, and the corresponding X.509 certificates are placed in the crts directory.

NOTE

The i.MX6SX HABv4 revision contains a new feature that allows the user to have the SRK authenticate the CSF and image data. The feature supplies the user with a faster boot time, at the cost of a less robust signature. Unless boot time is critical, it is recommended the SRK have the CA flag, and the CSF and IMG keys are used to validate their respective data.

For more details on key generation for CST, see the *HAB CST User Guide*.

3.2. Generate SRK table

The SRK table is required by CST. It is a table of the Public SRKs. To generate an SRK table, CST provides the srktool, which requires X.509v3 public key certificates as inputs for the SRKs. This tool creates the SRK table and an SRK fuse table. The fuse table contains a hash value of the SRK table, and is programmed to the SRK fuses on the target. The srktool is capable of outputting the fuse table in different formats to align with different fuse controllers used on various parts. i.MX 6 Series parts use OCOTP and the format is 32 fuses per word, so set the output format to "-f 1" . The following shows how to generate an SRK table with four keys for this example.

- From the "crts" directory execute:

```
../linux64/srktool -h 4 -t SRK_1_2_3_4_table.bin -e SRK_1_2_3_4_fuse.bin -d sha256 -c
./SRK1_sha256_2048_65537_v3_ca_crt.pem,./SRK2_sha256_2048_65537_v3_ca_crt.pem,./SRK3_sha256_2048_65537_v
3_ca_crt.pem,./SRK4_sha256_2048_65537_v3_ca_crt.pem -f 1
```

3.3. Fuse programming

Enabling the secure boot features of the device requires programming fuses on the part. A Fuse Map for the specific part should always be obtained and referenced to ensure the correct fuse locations are being programmed.

3.3.1. SRK Fuses

The SRK fuse values are generated by `srktool` when the SRK table was assembled in the previous section. Be careful when programming these values, as this data is the basis for the root of trust. An error in SRK results in a part that does not boot.

- Display the fuse value using hexdump utility with formatting to display 32-bit values in the correct byte order for programming using U-Boot fuse commands. From the “crts” directory execute:

```
hexdump -e '/4 "0x"' -e '/4 "%X""\n"' SRK_1_2_3_4_fuse.bin0x20593752
0x6ACE6962
0x26E0D06C
0xFC600661
0x1240E88F
0x209F144
0x831C8117
0x1190FD4D
```

- Program the fuse values using the “fuse” command, available in i.MX U-Boot.

```
fuse prog <bank> <word> <value>  ← The <bank> and <word> values are sourced from the Fuse Map.

fuse prog 3 0 0x20593752
fuse prog 3 1 0x6ACE6962
fuse prog 3 2 0x26E0D06C
fuse prog 3 3 0xFC600661
fuse prog 3 4 0x1240E88F
fuse prog 3 5 0x0209F144
fuse prog 3 6 0x831C8117
fuse prog 3 7 0x1190FD4
```

3.3.2. RNG Trim fuses

HAB provides two options for managing the hardware RNG available in CAAM. HAB can initialize the RNG, or defer the initialization for the CAAM Operating System driver to manage.

RNG trim fuses provide HAB with a value to program in CAAM. This value setting causes a delay so that sufficient entropy can be generated on the chip. This ensures that the RNG self-test passes during RNG initialization. If the self-test fails, HAB does not allow the device to continue booting if it is “Closed”. Only HAB sources the RNG Trim Fuse value. The CAAM driver has to perform a similar RNG trim configuration, but the values it uses are built into the driver software.

NOTE

Choose one of two methods, or the chip will not boot when the device is “Closed”.

3.3.2.1 Option 1 – Set the RNG Trim in Fuse

The Fuse setting is essentially a delay. Increasing the value increases boot time. The recommended safe value for ensuring the self-test passes is 0x10. Smaller values may work on some parts, but not all. The delay required to pass the test could also vary based on temperature.

- Program the RNG Trim fuse value using the “fuse” command available in i.MX U-Boot.

```
fuse prog 1 0 0x00100000
```

3.3.2.2 Option 2 – Defer RNG Instantiation for Post HAB Software (Recommended Option)

Deferring the RNG instantiation is done by adding a command to the CSF signature data. This command informs HAB to skip the instantiation. As of all CST versions 2.3 and greater, this command is included in the CSF signature by default, unless it is overridden by the CSF description file. Any operations requiring the RNG are not available to software until it is initialized, such as encryption and blob generation. This does not affect HAB-signed or encrypted boot features. Until the device is closed, HAB does not instantiate the RNG at boot.

The CSF configuration file is discussed in later sections of this document. There are also examples in the CSF File Example Appendix. Deferring RNG instantiation is done by adding the following command to the CSF description file:

```
[Unlock]
Engine = CAAM
Features = RNG
```

3.4. Configure and build U-Boot to support secure boot

The i.MX U-Boot supports secure boot configurations and provides access to HAB APIs exposed by the ROM vector table. The support is enabled by defining CONFIG_SECURE_BOOT in the board configuration header file. When built with this configuration, U-Boot provides extra functions for HAB, one of those being access to HAB status logs. It also correctly pads the U-Boot image to an appropriate size so the CSF signature data generated by CST can be concatenated to image. Beyond the scope of this document, it also enables U-Boot to utilize HAB to verify other software, such as the Linux® OS kernel for example.

- Uncomment or add CONFIG_SECURE_BOOT to the board configuration header.

```
#define CONFIG_SECURE_BOOT
```

3.5. Create the CSF Description file

The CSF contains all the commands that the ROM executes during the secure boot. These commands instruct HAB on which memory areas of the image to authenticate, which keys to install and use, what data to write to a particular register, and so on. In addition, the necessary certificates and signatures involved in the verification of the image, as well as the SRK table, are attached to the CSF binary signature.
When creating the CSF description file, it is important to keep in mind that commands in the binary CSF follow the order in which they appear in the CSF description file. Ordering of commands within the CSF description file is significant only to the following extent:

- The Header command must precede any other command.

- The Install SRK command must precede the Install CSFK command.
- The Install CSFK must precede the Authenticate CSF command.
- Install SRK, Install CSFK and Authenticate CSF commands must appear exactly once in a CSF description file.
- A verification index in an Authenticate Data command must appear as the target index in a previous Install Key command.

Command Sequence File for the example:

```
#Illustrative Command Sequence File Description
[Header]
Version = 4.1
Hash Algorithm = sha256
Engine = ANY
Engine Configuration = 0
Certificate Format = X509
Signature Format = CMS

[Install SRK]
File = “../crts/SRK_1_2_3_4_table.bin”
Source index = 0          # Index of the key location in the SRK table to be installed

[Install CSFK]
# Key used to authenticate the CSF data
File = “../crts/CSF1_1_sha256_2048_65537_v3_usr_crt.pem”

[Authenticate CSF]

[Install Key]
# Key slot index used to authenticate the key to be installed
Verification index = 0
# Target key slot in HAB key store where key will be installed
Target Index = 2
# Key to install
File= ”../crts/IMG1_1_sha256_2048_65537_v3_usr_crt.pem”

[Authenticate Data]
# Key slot index used to authenticate the image data
Verification index = 2
#        Address    Offset  Length    Data File Path
Blocks = 0x877fb000 0x000   0x48000   “/home/user/path_to_u-boot_dir/u-boot.imx”
```

The Authenticate Data command Blocks line contains 3 values and the file containing the data being signed. The first value is the address on the target where HAB expects the signed image data to begin. The second and third values are related to the image file that is reference by the data file path. The second value is the offset into the file where CST will begin signing. The third value is length in bytes of the data to sign starting from the offset. It is also required that the IVT and DCD regions are signed. HAB will verify the DCD and IVT fall in an authenticated region. The CSF will not successfully authenticate unless all commands are successful and all required regions are signed.

For more detailed information about the CSF commands, reference the CST User's Guide. It may also be helpful to review the "Program Image" Chapter in the chip's reference manual to understand the contents of a boot image.

3.6. Generate the CSF binary signature

The CSF binary signature is generated from the CSF input file by the Code Signing Tool.

- From the "linux 32 or linux64" directory, call CST with the CSF input file:

```
./cst --o csf-uboot.bin < csf-uboot
```

3.7. Attach CSF Signature to U-Boot image

The CSF binary data needs to be concatenated to the image.

- Use the cat command to attach the CSF binary to the end of the image:

```
cat u-boot.imx csf-uboot.bin > u-boot-signed.imx
```

3.8. Verify HAB successfully authenticates the signed image

The next step is to verify that the signature attached to U-Boot is successfully processed without issues. HAB generates events when processing the commands if it encounters issues. One of the utilities enabled with the CONFIG_SECURE_BOOT switch in U-Boot is the "hab_status" command. This command displays any events that were generated. This command is executed by typing "hab_status" at the u-boot prompt.

```
=> hab_status
```

3.9. Secure the device

After the device successfully boots a signed image without generating any HAB events, it is safe to secure, or "close", the device. This is the last step in the process, and is completed by blowing the SEC_CONFIG[1] fuse bit. Refer to the fuse map for the part before configuring the fuse to ensure its location is correct. Once the fuse is blown, the chip does not load in image that has not been signed using the correct PKI tree.

- Program the fuse values using the "fuse" command, available in i.MX U-Boot.

```
fuse prog 0 6 0x2
```

4. Troubleshooting

4.1. HAB Events

HAB generates events when it encounters issues. These events are writer to a region in OCRAM to supply feedback to assist in debugging. The location in OCRAM varies based on the i.MX Series part.

As previously mentioned, U-Boot supples the “hab_status” command to read these events and feed them to the console.

During development, it is recommended that the events are checked before the device is “closed”. Once an image is signed with a signature that does not generate events during load, the signed image should boot on a “closed” device without issues. This should be the goal for development, since trying to debug on a “closed” platform requires the use of JTAG or the USB serial download protocol to acquire the event debug information.

4.2. SRK Authentication for i.MQX50 and i.MQX53 in Open configuration

On i.MX50 and i.MX53 devices, it is required to blow the SRK even for Open configuration when developing a secure product. On these devices HAB enforces SRK authentication, even for Open configuration. This means that if the SRK fuses are not properly provisioned, the Install SRK CSF command fails and HAB stops processing the CSF.

5. Revision history

The table provides a revision history for this application note.

Table 1. Revision history

Revision Number	Date	Substantive Changes
0	10/2012	Initial public release.
1.0	06/2015	Reorganized sections and removed duplicate information available in CST/HAB user guides. Added sections for Fast Authentication and the RNG Trim Added CSF Examples

Appendix A. SRK Revocation on i.MX 6 Series

The i.MX 6 Series supports revocation of SRKs. The SRK table generated by the `srktool` of the CST may contain up to four separate public keys. Only one SRK may be selected at boot time through an Install SRK CSF command. In the case where one of the first three SRKs are compromised, it is possible to revoke that SRK on i.MX 6 Series devices. There are three SRK Revoke fuse bits that map to the first three SRK table indexes. An SRK key is revoked by burning the corresponding bit in the SRK_REVOKE eFuse field. This figure illustrates an example.

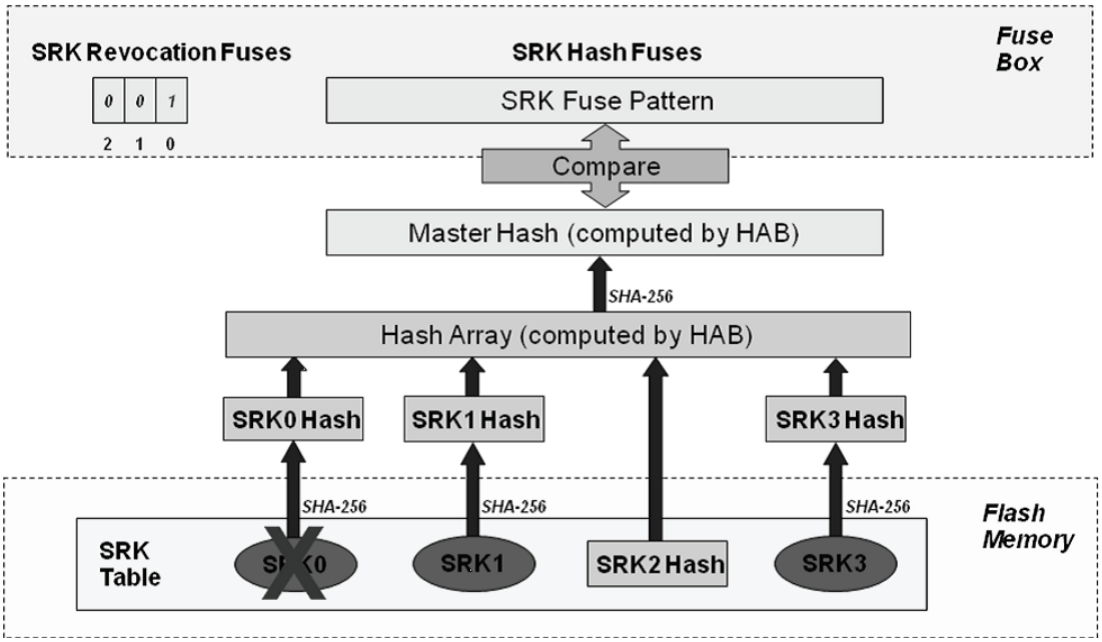

Figure 3. SRK Revocation using HABv4

In this example, an SRK table with four public keys has been generated. To revoke SRK0 from a bootloader, or another stage after the boot ROM, it is necessary to blow the SRK_REVOKE[0] eFuse. However, in the Closed configuration, HAB, by default, sets the SRK_REVOKE_LOCK sticky bit in the OCOTP controller to write protect this eFuse field. To instruct HAB not to lock the SRK_REVOKE field requires the use of the Unlock CSF command, with the command flag indicating to unlock the SRK_REVOKE field. Including this command in a CSF signature allows the SRK0 fuse to be blown by a trusted bootloader or runtime image. Below is an example CSF command that unlocks the SRK_REVOKE eFuse field, allowing U-Boot or a later stage to burn the fuse.

```
[Unlock]
Engine = OCOTP
Features = SRK Revoke
```

For more detailed information about CSF commands, reference the *CST User's Guide*.

Appendix B. Fast Authentication (>HAB 4.1.2 only)

HAB 4.1.2 introduces the Fast Authentication feature. It provides the option to use the SRK to verify the CSF data and Image data directly, instead of using the CSF and IMG keys. This reduces the number of key pair authentications that must occur during the ROM/HAB boot stage. The typical boot time for an image smaller than 1MB can be reduced from 25ms to 12ms.

B.1. Fast Authentication public key infrastructure

Since the SRK is used to verify the CSF and Image data, the CSF and IMG keys are not generated by `hab4_pki_tree.sh`. The SRK is generated without a CA flag since it is no longer used to authenticate keys, but is used to verify data. The figure below gives an example of a PKI tree for fast authentication that is generated by the Freescale Code Signing Tools.

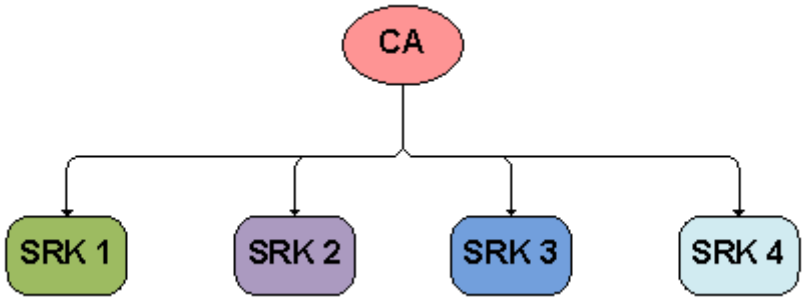

Figure 4. HABv4 Enable devices Fast Authentication PKI Tree

- Generating the Fast Authentication PKI Tree:

```
./hab4_pki_tree.sh                                    ~/work/freescale/secure-boot/cst-2.3.0/keys

    ++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
    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
Enter key length in bits for PKI tree: 2048
Enter PKI tree duration (years): 10
How many Super Root Keys should be generated? 4
Do you want the SRK certificates to have the CA flag set? (y/n)?: n
```

Appendix C. HAB4 CSF Examples

C.1. Example 1 – Signing multiple image regions

- Defines a version 4.1 CSF description.
- Overrides default engine ANY with DCP in Authenticate Data command.
- Defines three separate blocks from the image for signing.

```
#Illustrative Command Sequence File Description
[Header]
Version = 4
Hash Algorithm = sha256
Engine = ANY
Engine Configuration = 0
Certificate Format = X509
Signature Format = CMS

[Install SRK]
File = “../crts/SRK_1_2_3_4_table.bin”
Source index = 0

[Install CSFK]
File = “../crts/CSF1_1_sha256_2048_65537_v3_usr_crt.pem”

[Authenticate CSF]

[Install Key]
Verification index = 0
Target Index = 2
File= ”../crts/IMG1_1_sha256_2048_65537_v3_usr_crt.pem”

[Authenticate Data]
# Key slot index used to authenticate the image data
Verification index = 2
Engine = DCP
Blocks = 0xf8009400 0x400  0x40   “MCUROM-OCRAM-ENG_img.bin”, \
        0xf8009440 0x440  0x40   “MCUROM-OCRAM-ENG_img.bin”, \
         0xf800a000 0x1000 0x8000 “MCUROM-OCRAM-ENG_img.bin”,
```

```
# Note the “\” for line continuation in the Block definitions
```

C.2. Example 2 – Fast Authentication (HAB 4.1.2 only)

- Defines a version 4 CSF description.
- Uses Install NOCAK command SRK key for signature verification.
- Lists single block from image for signing.

```
#Illustrative Command Sequence File Description
[Header]
Version = 4.1
Hash Algorithm = sha256
Engine = ANY
Certificate Format = X509
Signature Format = CMS

[Install SRK]
File = “../crts/SRK_1_2_3_4_table.bin”
Source index = 0

[Install NOCAK]
File = “../crts/SRK1_sha256_2048_65537_v3_usr_crt.pem”

[Authenticate CSF]


[Authenticate Data]
# Key slot index 0 used to authenticate the image data
Verification index = 0
Blocks = 0x877fb00 0x0  0x48000   “u-boot.imx”
```

C.3. Example 3 – Unlock RNG

- Adds the Unlock SRK Revoke command after the Authenticate CSF command.

```
#Illustrative Command Sequence File Description
[Header]
Version = 4.1
Hash Algorithm = sha256
Engine Configuration = 0
Certificate Format = X509
Signature Format = CMS

[Install SRK]
File = “../crts/SRK_1_2_3_4_table.bin”
Source index = 0
```

```
[Install CSFK]
File = “../crts/CSF1_1_sha256_2048_65537_v3_usr_crt.pem”

[Authenticate CSF]

[Unlock]
Engine = CAAM
Features = RNG

[Install Key]
Verification index = 0
Target Index = 2
File= ”../crts/IMG1_1_sha256_2048_65537_v3_usr_crt.pem”

[Authenticate Data]
# Key slot index used to authenticate the image data
Verification index = 2
Blocks = 0x877fb00 0x0  0x48000   “u-boot.imx”
```

C.4. Example 4 – Unlock SRK Revoke Fuse Bank

- Adds the Unlock SRK Revoke command after Authenticate CSF command.

```
#Illustrative Command Sequence File Description
[Header]
Version = 4.1
Hash Algorithm = sha256
Engine Configuration = 0
Certificate Format = X509
Signature Format = CMS

[Install SRK]
File = “../crts/SRK_1_2_3_4_table.bin”
Source index = 0

[Install CSFK]
File = “../crts/CSF1_1_sha256_2048_65537_v3_usr_crt.pem”

[Authenticate CSF]

[Unlock]
Engine = OCOTP
Features = SRK Revoke

[Install Key]
Verification index = 0
```

```
Target Index = 2
File= ”../crts/IMG1_1_sha256_2048_65537_v3_usr_crt.pem”

[Authenticate Data]
# Key slot index used to authenticate the image data
Verification index = 2
Blocks = 0x877fb00 0x0  0x48000   “u-boot.imx”
```

Appendix D. HAB Version/Chip matrix

Table 2. Chip matrix

Chip	HAB Version	Encrypted boot supported by ROM
MX28 TO1.0	HAB 4.0.1	Yes - SigmaTel legacy encrypted boot
MX28 TO2.0	HAB 4.0.4	Yes - SigmaTel legacy encrypted boot
MX508 TO1.0	HAB 4.0.4	No
MX508 TO1.1	HAB 4.0.4	No
MX53 TO1.0	HAB 4.0.2	No
MX53 TO2.0	HAB 4.0.4	No
MX6 D/Q TO1.0	HAB 4.1.0	Yes - via HAB*
MX6 D/Q TO1.1	HAB 4.1.1	Yes - via HAB*
MX6 D/Q TO 1.2	HAB 4.1.1	Yes - via HAB*
MX6 D/Q TO 1.3	HAB 4.1.1	Yes - via HAB*
MX6 DL/S TO1.0	HAB 4.1.1	Yes – via HAB*
MX6 SL TO 1.0	HAB 4.1.1	No
MX6 SX TO x.x	HAB 4.1.2	Yes – via HAB*

Appendix E. Freescale manufacturing tool

E.1. Signing code downloadable with the manufacturing tool

The Freescale manufacturing tool can be used to download and execute code when in Close configuration mode. Note that this section is only valid when using the manufacturing tool with the i.MX 6 Series.

These are the steps to download the code:

- Parse the file to load in order to find the IVT and its DCD table pointer.
- If there is a DCD table, it is loaded to the address 0x00910000 in the OCRAM with the SDP command, DCD_WRITE. The DCD table must always be signed, which implies that this area in OCRAM must be signed.
- The pointer to the DCD table is cleared in the IVT in order to prevent the HAB library from processing the DCD table again during the authentication process. There is no need to re-initialize some memory, such as DDR3, when it already contains valid data.
- The code is loaded to the boot_data address defined in the boot image structure.

It is necessary to consider the below two points in the signature process:

- The CSF description file should contain a command to sign the DCD table, located at the address, 0x00910000.

  A typical CSF authenticate data command is provided below:

```
[Authenticate Data]
    Verification index = 2
    Blocks = 0x10800400 0x400 0x2BC00 "u-boot-pad.bin"
```

  For example, the new command is as follows:

```
[Authenticate Data]
    Verification index = 2
    Blocks = 0x10800400 0x400 0x2BC00 "u-boot-pad.bin", \
    Blocks = 0x00910000 0x430 0x2E0 "u-boot-pad.bin"
```

  The second parameter is the offset of the DCD table in the binary file, and the third parameter is the size of the table. These parameters can vary according to the memory layout defined by the user.

  Since the IVT is modified when downloading to the target, so the code must be signed with a cleared DCD pointer. However, the code must be provided with a valid pointer to allow the manufacturing tool to locate the DCD table.

  For example, a script can be used to store the DCD address, which needs to be erased before creating the CSF binary file in Section 5.4 “Generating the Command Sequence File (CSF) Binary File”. After that, the DCD address can be restored to continue the steps that generate the final signed binary.

  Here is an example of bash script used to generate the signed code:

```
#!/bin/bash
PROG_NAME=my_code

# ${PROG_NAME} padded up to 0x2C000 where the CSF will be added later
objcopy -I binary -O binary --pad-to 0x2C000 --gap-fill=0xff ${PROG_NAME}.bin
${PROG_NAME}_padded.bin

# DCD address must be cleared for signature, as mfgtool will clear it.
./mod_4_mfgtool.sh clear_dcd_addr ${PROG_NAME}_padded.bin

# generate the signatures, certificates, … in the CSF binary
../linux/cst --o ${PROG_NAME}_csf.bin < ${PROG_NAME}.csf

# DCD address must be set for mfgtool to localize the DCD table.
./mod_4_mfgtool.sh set_dcd_addr ${PROG_NAME}_padded.bin
```

```
# gather ${PROG_NAME} + its CSF
cat ${PROG_NAME}_padded.bin ${PROG_NAME}_csf.bin > ${PROG_NAME}_tmp.bin

# padding to get a file with size like specified in the IVT
objcopy -I binary -O binary --pad-to 0x22000 --gap-fill=0xff ${PROG_NAME}_tmp.bin
${PROG_NAME}_signed.bin

# remove temporary file
rm ${PROG_NAME}_tmp.bin
```

Here is an example mod_4_mfgtool.sh script used to handle the DCD address:

```
#!/bin/bash

# DCD address must be cleared for signature, as mfgtool will clear it.
if [ "$1" == "clear_dcd_addr" ]; then
  # store the DCD address
  dd if=$2 of=dcd_addr.bin bs=1 count=4 skip=1036
  # generate a NULL address for the DCD
  dd if=/dev/zero of=zero.bin bs=1 count=4
  # replace the DCD address with the NULL address
  dd if=zero.bin of=$2 seek=1036 bs=1 conv=notrunc
fi
# DCD address must be set for mfgtool to localize the DCD table.
if [ "$1" == "set_dcd_addr" ]; then
  # restore the DCD address with the original address
  dd if=dcd_addr.bin of=$2 seek=1036 bs=1 conv=notrunc
  rm zero.bin
fi
```

E.2. Freescale manufacturing tool i.MX 6 Series script

```
<LIST name="MX6Q Sabre-lite SRK Hash" desc="SRK hash fuse programming">
    <CMD type="find" body="Recovery" timeout="180"/>
    <CMD type="boot" body="Recovery" file ="u-boot-mx6q-sabrelite.bin" >Loading uboot.</CMD>
    <CMD type="load" file="uImage" address="0x10800000"
       loadSection="OTH" setSection="OTH" HasFlashHeader="FALSE" >Doing Kernel.</CMD>
    <CMD type="load" file="initramfs.cpio.gz.uboot" address="0x10C00000"
       loadSection="OTH" setSection="OTH" HasFlashHeader="FALSE" >Doing Initramfs.</CMD>
    <CMD type="jump" > Jumping to OS image. </CMD>
    <CMD type="find" body="Updater" timeout="180"/>
    <!-- ***** Caution - running this xml script with the fuse burning commands uncommented
         ***** in the Mfg tool permanently burns fuses. Once completed this operation cannot
         ***** be undone!
    -->
    <CMD type="push" body="$ echo 0xfdf28547 > /sys/fsl_otp/HW_OCOTP_SRK0">Burn Word 0 of SRK
hash field in OTP </CMD>
    <CMD type="push" body="$ echo 0x270d6ac6 > /sys/fsl_otp/HW_OCOTP_SRK1">Burn Word 1 of SRK
hash field in OTP </CMD>
```

```
    <CMD type="push" body="$ echo 0xee44ad7b > /sys/fsl_otp/HW_OCOTP_SRK2">Burn Word 2 of SRK
hash field in OTP </CMD>
    <CMD type="push" body="$ echo 0x058b0724 > /sys/fsl_otp/HW_OCOTP_SRK3">Burn Word 3 of SRK
hash field in OTP </CMD>
    <CMD type="push" body="$ echo 0x49da1948 > /sys/fsl_otp/HW_OCOTP_SRK4">Burn Word 4 of SRK
hash field in OTP </CMD>
    <CMD type="push" body="$ echo 0xb4374a3f > /sys/fsl_otp/HW_OCOTP_SRK5">Burn Word 5 of SRK
hash field in OTP </CMD>
    <CMD type="push" body="$ echo 0xffefed48 > /sys/fsl_otp/HW_OCOTP_SRK6">Burn Word 6 of SRK
hash field in OTP </CMD>
    <CMD type="push" body="$ echo 0x4247c04f > /sys/fsl_otp/HW_OCOTP_SRK7">Burn Word 7 of SRK
hash field in OTP </CMD>
    <CMD type="push" body="$ cat /sys/fsl_otp/HW_OCOTP_SRK0"/>
    <CMD type="push" body="$ cat /sys/fsl_otp/HW_OCOTP_SRK1"/>
    <CMD type="push" body="$ cat /sys/fsl_otp/HW_OCOTP_SRK2"/>
    <CMD type="push" body="$ cat /sys/fsl_otp/HW_OCOTP_SRK3"/>
    <CMD type="push" body="$ cat /sys/fsl_otp/HW_OCOTP_SRK4"/>
    <CMD type="push" body="$ cat /sys/fsl_otp/HW_OCOTP_SRK5"/>
    <CMD type="push" body="$ cat /sys/fsl_otp/HW_OCOTP_SRK6"/>
    <CMD type="push" body="$ cat /sys/fsl_otp/HW_OCOTP_SRK7"/>
</LIST>
</UCL>
```

This script can also be extended to burn the SRK Hash lock fuse.

How to Reach Us:

Home Page:
freescale.com

Web Support:
freescale.com/support

Document Number: AN4581
Rev. 1
10/2015