AN4581

i.MX Secure Boot on HABv4 Supported Devices

Rev. 4 — June 2020

Application Note

Contents

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 Assurance Boot version 4 (HABv4) component of the on-chip ROM. The ROM is responsible for loading the initial program image from the boot medium. HABv4 enables the ROM to authenticate the program image by using digital signatures. This initial program image is usually a bootloader.

HABv4 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 signed images 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.

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

For MPU step-by-step technical guides, please refer to the U-Boot project documentation (see References).

1.3 Scope

This document is intended to illustrate the construction of a secure boot image, in addition to configuring the target device to run securely.

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?
- How to extend the secure boot chain past the initial stage?

NOTE

This document covers secure boot in all i.MX HABv4 enabled devices. Secure boot features for i.MX 28 are documented in *Secure Boot with i.MX28 HAB Version 4* (document AN4555). i.MX 28 supports HABv4, but its boot architecture is significantly different from other processors in the i.MX family.

i.MX 50 and i.MX 53 family devices are no longer recommended for new designs.

Secure boot features for other processors, such as i.MX 25, i.MX 35, and i.MX 51, which use HABv3, are documented in *Secure Boot on i.MX 25, i.MX 35, and i.MX 51 using HABv3* (document AN4547).

Secure boot using HAB is no longer supported on i.MX 27 and i.MX 31.

Encrypted boot is not included in this document. For further details, refer to *Encrypted Boot on HABv4 and CAAM Enabled Devices* (document AN12056).

1.4 Definitions, acronyms, and abbreviations

Table 1 describes the definitions of terms and acronyms in this document.

Table 1. Definition of terms and acronyms

Terms or acronyms	Definitions	Remarks
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 NXP 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	A vector table that consists of pointers to image start address, signature address, etc.
OS	Operating System	—
OTP	One-Time Programmable	OTP hardware includes masked ROM, and electrically programmable fuses (eFuses).
PKCS#1	Public Key Cryptography Standards	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	Rivest–Shamir–Adleman	Public key cryptography algorithm developed by Rivest, Shamir, and Adleman.

Table continues on the next page...

Table 1. Definition of terms and acronyms (continued)

Terms or acronyms	Definitions	Remarks
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.MX 50, i.MX 53, i.MX 6, i.MX 7, and i.MX 8M family processors reference and security reference manuals.
- HABv4 CST User Guide and HABv4 API Guide are available in the Code Signing Tool release package downloadable at http://www.nxp.com. Search for `IMX_CST_TOOL`.
  - There are two CST packages available at http://www.nxp.com to download. The newer version is applicable to only HABv4 devices.
- U-Boot technical guides in the *doc/imx/habv4* folder of the U-Boot project. Available starting from imx_v2018.03_4.14.78_ga release branch.
- *HABv4 RVT Guidelines and Recommendations* (document AN12263).

2 Overview

High Assurance Boot (HAB) authentication is based on public key cryptography using the RSA or ECDSA algorithms 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. HABv4 relies on a table of the public SRKs to be hashed and placed in fuses on the target. The i.MX 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, HABv4 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, the root of trust is established, and the remainder of the signature can be processed to authenticate the image. Detailed information for CST and HABv4 can be found in their respective user guides included in the CST package.

Figure 1 illustrates the secure boot process overview.

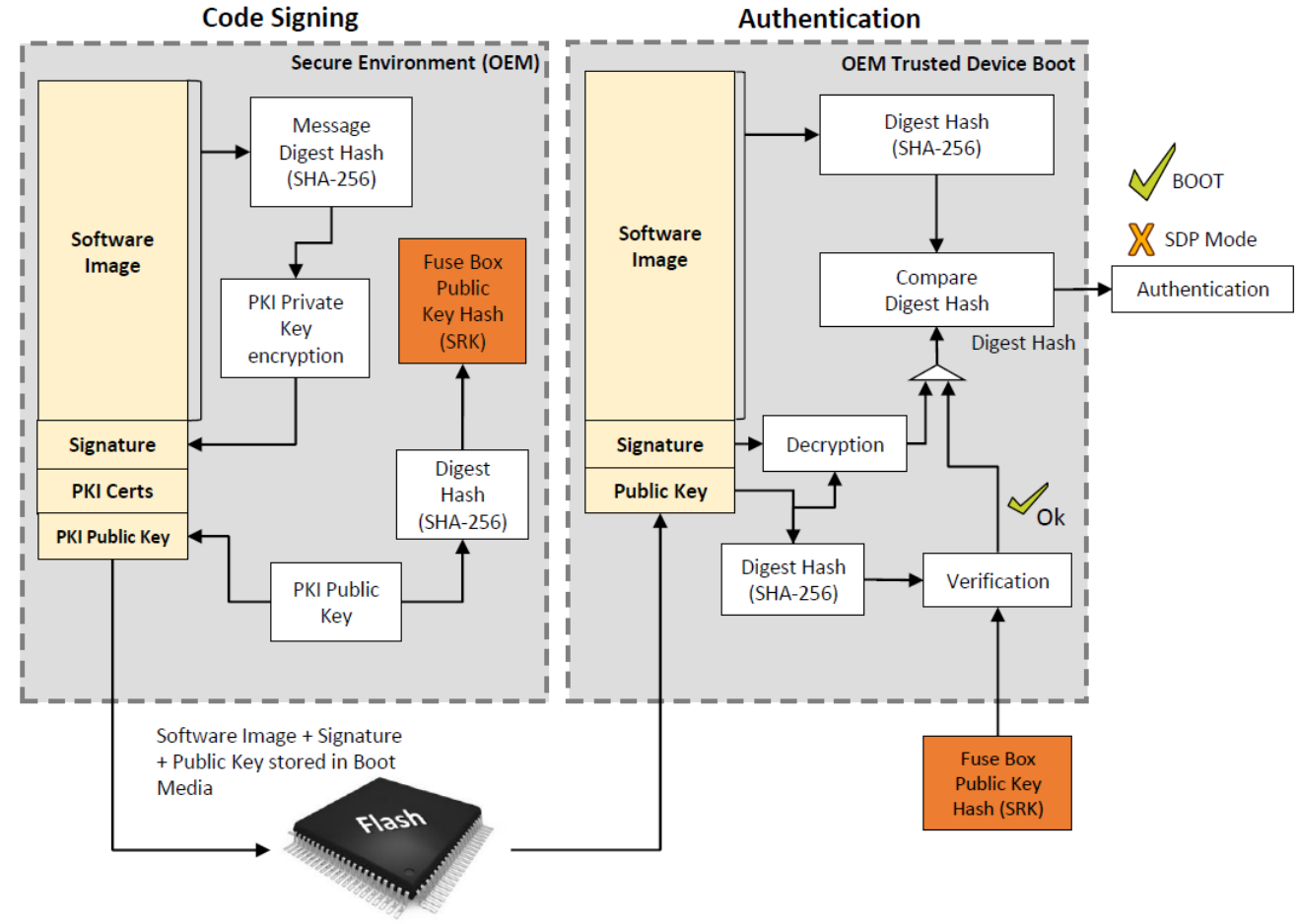

Figure 1. Secure boot process overview

The main features supported by the HABv4 are:

- Authentication of software loaded from any boot device supported, including the Serial Download Protocol (SDP).
- Authenticated USB download fail-over on any security failure.
- Interface with the SNVS to guarantee the system security state.
- Initialization of security components.
- X.509 public key certificate support.
- CMS signature format support.
- 1024, 2048, 3072, and 4096 bits RSA keys length support.
- p256, p384, p521 ECC keys support.
- SHA-256 message digest operations accelerated by hardware.
- Manufacturing protection private key generation in supported devices.
- Super Root Key (SRK) revocation support.

NOTE

ECDSA support is available in i.MX8MP A1 silicon rev. Application Processor.

Details on how to use HABv4 via SDP can be found in UUU documentation.

3 Boot flow

The operational system may consist of several independent images with each one launching the next image, the system designer must ensure every software component is authentic to prevent unauthorized software execution during the device boot sequence. In case a malware takes control of the boot flow, sensitive data, services and network can be impacted.

The secure boot process starts with ROM authenticating the first image in the boot flow which is typically a bootloader such as U-Boot, Second Program Loader (SPL) or a custom implementation. Once the root of trust is established the HABv4 API can be leveraged to authenticate additional images, extending the secure boot chain.

The boot sequence may vary according to the system software implementation, the examples included in this document are based in NXP BSP implementation.

NOTE

Details about HABv4 API can be found in *HABv4 RVT Guidelines and Recommendations* (document AN12263) and in HABv4 API document in CST release.

3.1 Boot flow in i.MX 6, i.MX 7 and i.MX RT 10xx series devices.

Most common bootloader implementation for i.MX 6, i.MX 7 and i.MX RT 10xx series devices consist in a single binary, the root of trust is typically extended at bootloader level to others software components.

Figure 2 illustrates a typical boot flow sequence on i.MX 6 and i.MX 7 series devices.

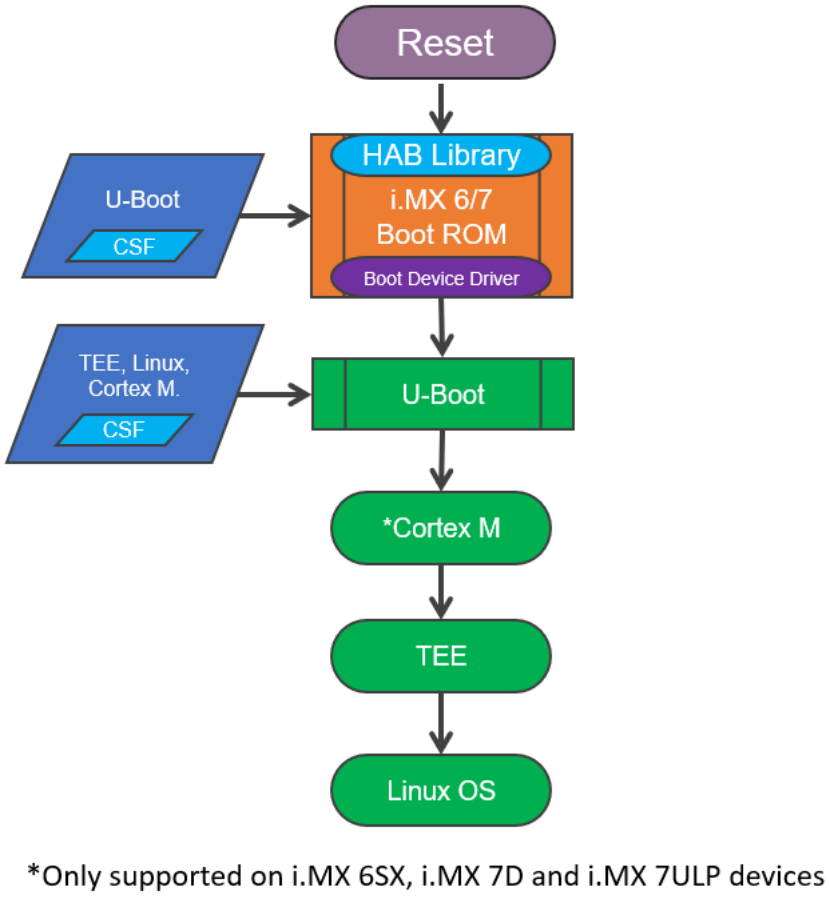

Figure 2. Secure boot flow on i.MX 6 and i.MX 7 series devices

Similarly, on i.MX RT 10xx family devices, the bootloader or final software implementation consist in a single binary. Figure 3 illustrates a typical boot flow.

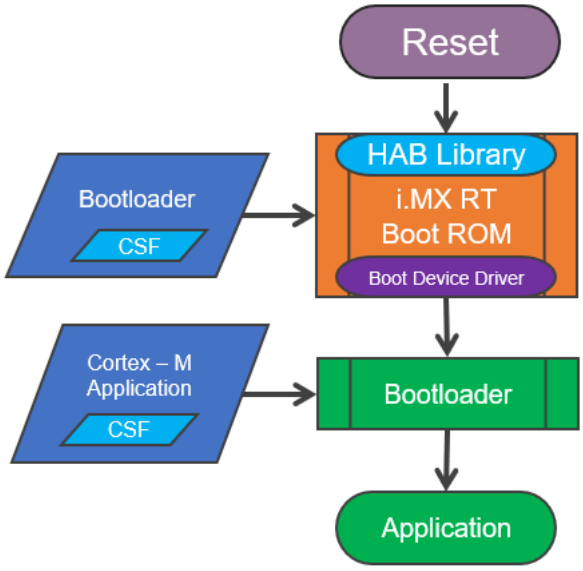

Figure 3. Secure boot flow on i.MX RT 10xx series devices

In certain MPU heterogeneous devices the Cortex®-M has an independent ROM and HABv4 implementation. This is the case for i.MX 7ULP family device. In case booting in dual memory boot mode the two processors on i.MX 7ULP boot independently of each other and software images are independently authenticated at ROM level using their own SRK keys. System designers must ensure all boot software components are authentic, which can be done by extending the root of trust whenever is possible.

Figure 4 illustrates a typical i.MX 7ULP dual memory boot, Cortex-A7 boots from SDCard or eMMC and Cortex-M4 boots from QSPI flash.

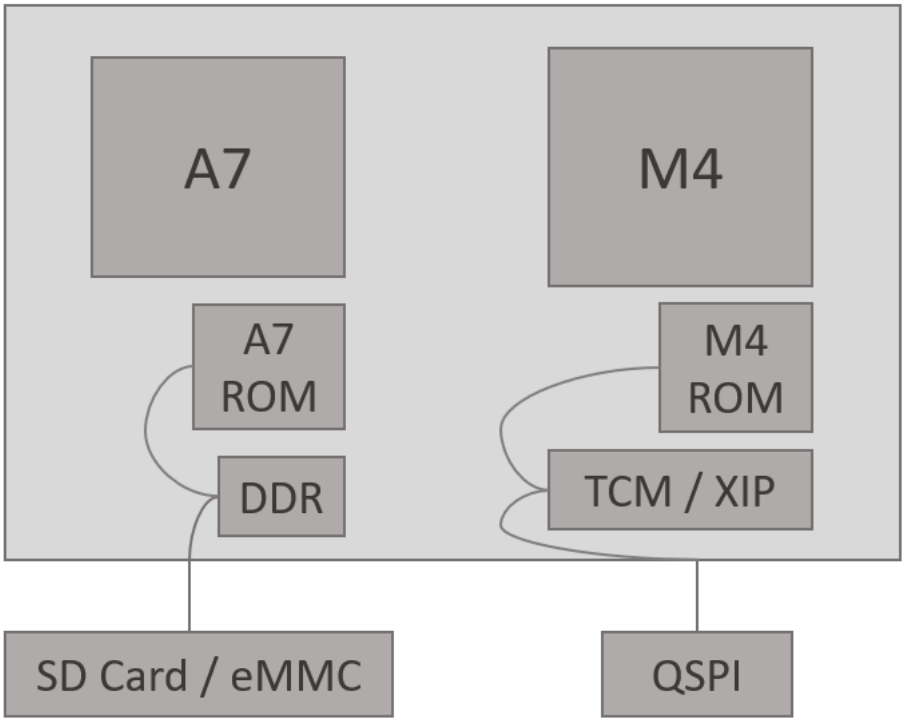

Figure 4. i.MX 7ULP dual memory boot

3.2 Boot flow in i.MX 8M family devices.

The boot flow on i.MX 8M family devices are slightly different when compared with i.MX 6, i.MX 7, and i.MX RT 10xx series device. The boot flow changes are due to the different architecture, multiple firmware and software components required to initialize the device.

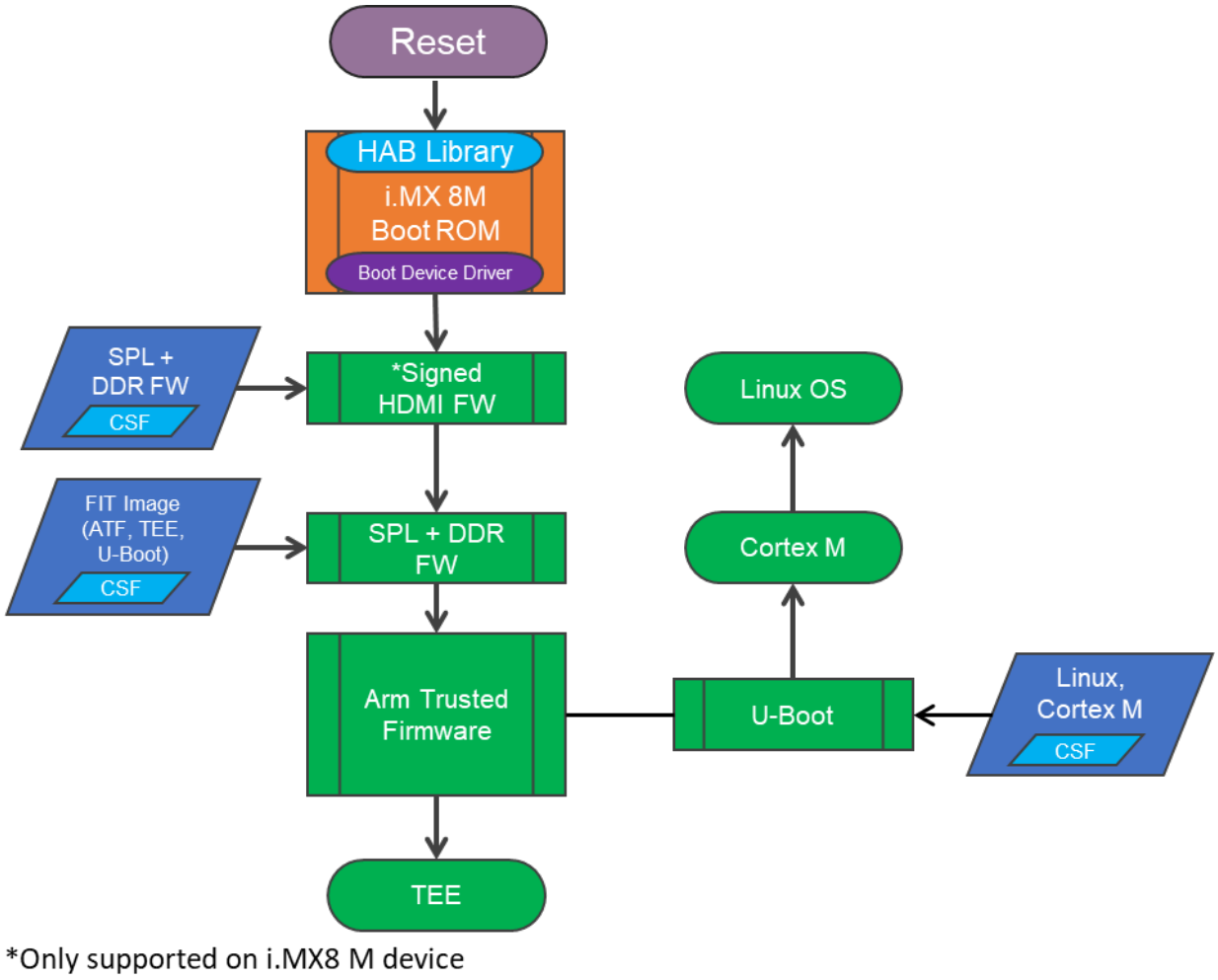

Figure 5. Secure boot flow on i.MX 8 M family devices

Only on i.MX8 M device the HDMI or DisplayPort firmware is the first image to boot on the device. The ROM code loads the HDCP keys at the beginning of primary boot process and locks the HDMI controller prior to loading any boot images to guarantee the integrity and security of HDCP key material. This firmware is signed and distributed by NXP and is always authenticated regardless of security configuration. If not required by the application, the HDMI controller can be disabled by programming the HDMI_DISABLE fuse and firmware authentication is not required anymore.

The next images are not signed by NXP and system designers must ensure all boot software components are authenticated by extending the root of trust whenever needed.

In NXP BSP implementation the Second Program Loader (SPL) and DDR firmware are loaded and authenticated by the ROM code. Once the DDR is available, the SPL code loads all the images included in the FIT structure to their specific execution addresses. The HABv4 APIs are called to extend the root of trust, authenticating the U-Boot, Arm® Trusted Firmware (ATF) and TEE (optional).

Additionally, the Linux Kernel and Cortex-M images can be authenticated at bootloader level.

4 Image layout

The software image to be programmed into the boot media must be properly constructed to be used by HABv4. For example, it must contain a Command Sequence File (CSF). The CSF is a binary data structure interpreted by the HABv4 to guide

authentication process, which is generated by the Code Signing Tool. The CSF structure contains the commands, SRK table, signatures, and certificates. For further details, see Code signing example.

The HABv4 relies in an Image Vector Table (IVT) to identify the CSF location. System designers must ensure that the IVT, initial byte of boot data, DCD table (if included), and first word of the image are covered by a digital signature.

NOTE

The **Program Image** chapter in the SoC reference manual provides more details on the content of a boot image.

Figure 6 illustrates a typical memory layout of a primary boot image.

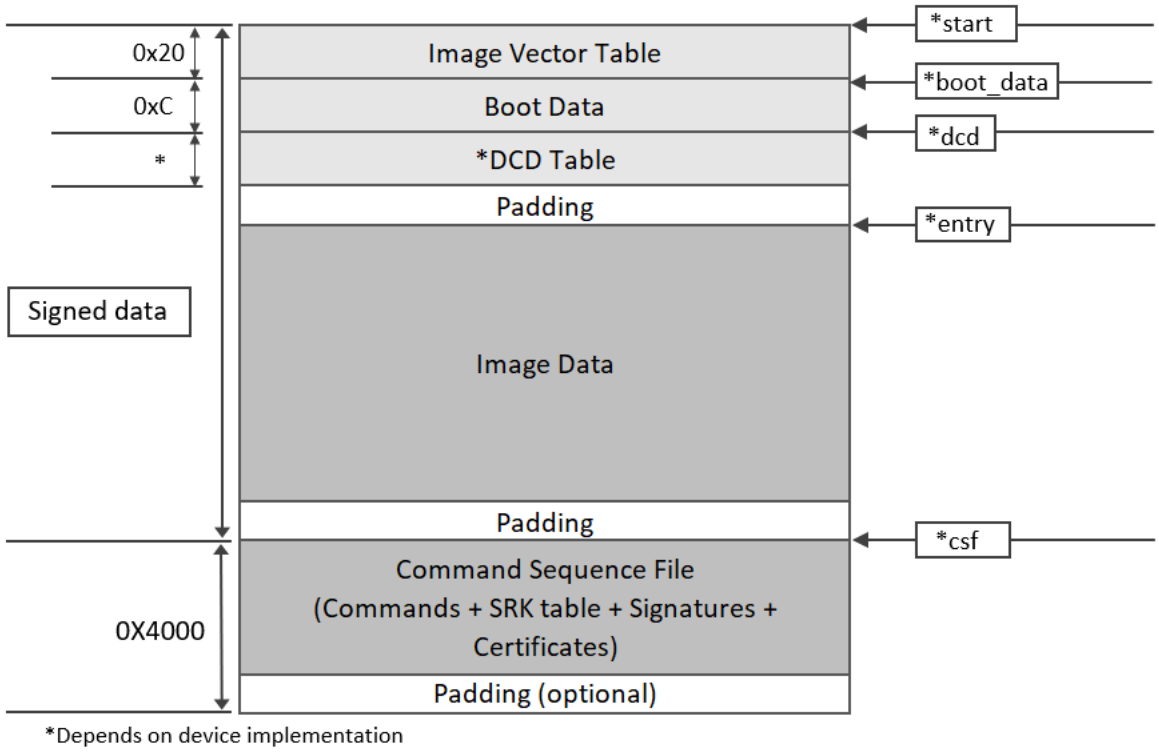

Figure 6. Typical memory layout of a signed image

Additional boot images do not require DCD and boot data fields. In this case, HABv4 only requires the IVT and first word of the image to be covered by a digital signature.

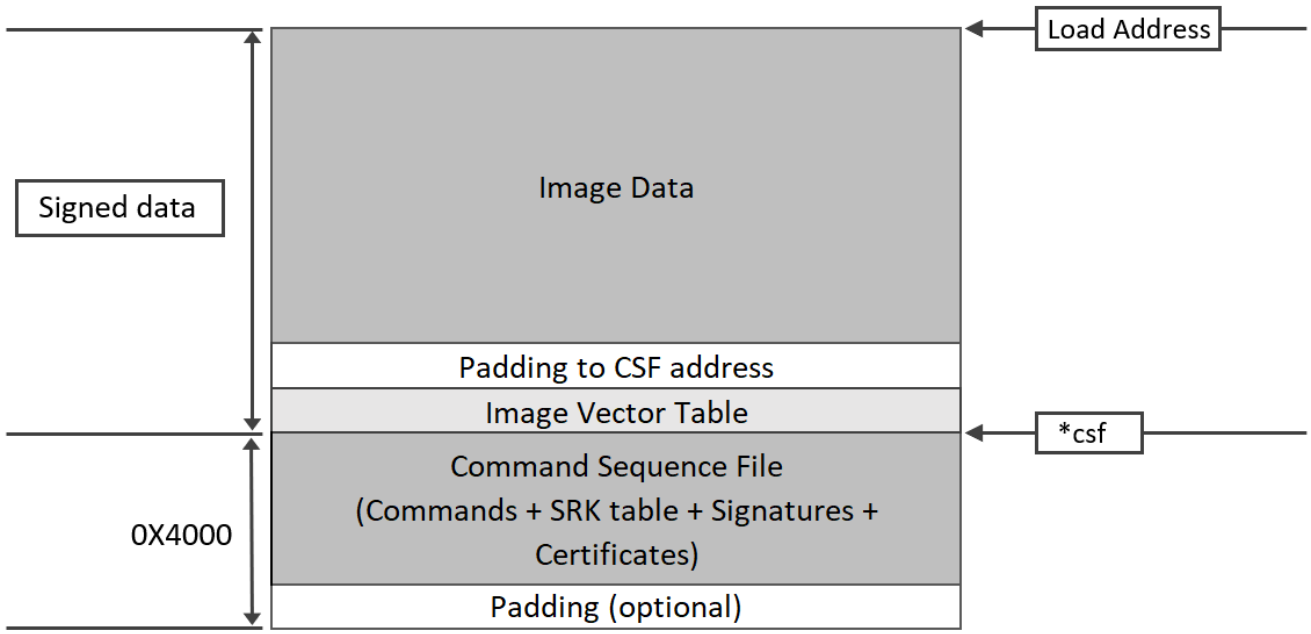

Figure 7. Typical memory layout of a signed additional boot image

A more complex structure containing multiple boot as Flattened Image Tree (FIT) is also possible. System designers must ensure their CSF file covers all images in their respective execution address.

The image layout example below is a typical bootloader for i.MX 8MQ device containing a signed HDMI FW, SPL, DDR FW and FIT image. In this case, two different CSF are required to completely sign the image, not including the signed HDMI FW.

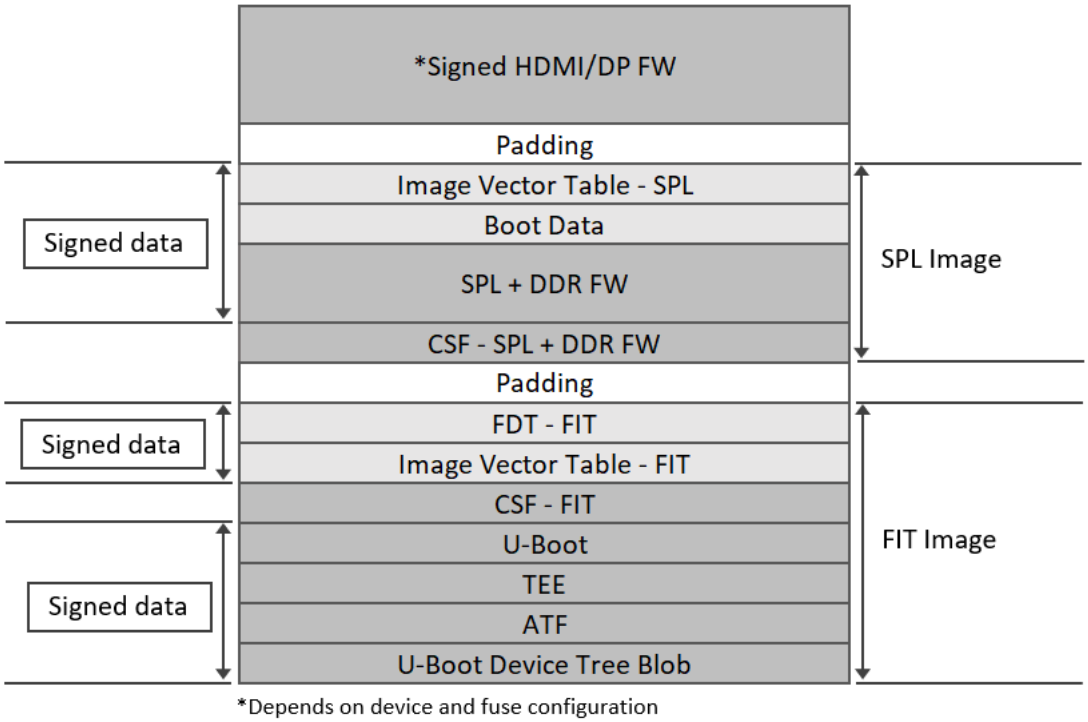

Figure 8. Typical memory layout of a signed flash.bin image

5 Code signing example

The following sections detail the process to securely boot a target with HABv4. For MPU devices, step-by-step technical guides are available in U-Boot project documentation. (See References).

5.1 Generating PKI tree

The CST package includes PKI tree generation scripts under keys directory. 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. HABv4 supports up to four SRKs in a signed image. By default, for each SRK a CSF key and IMG key are generated.

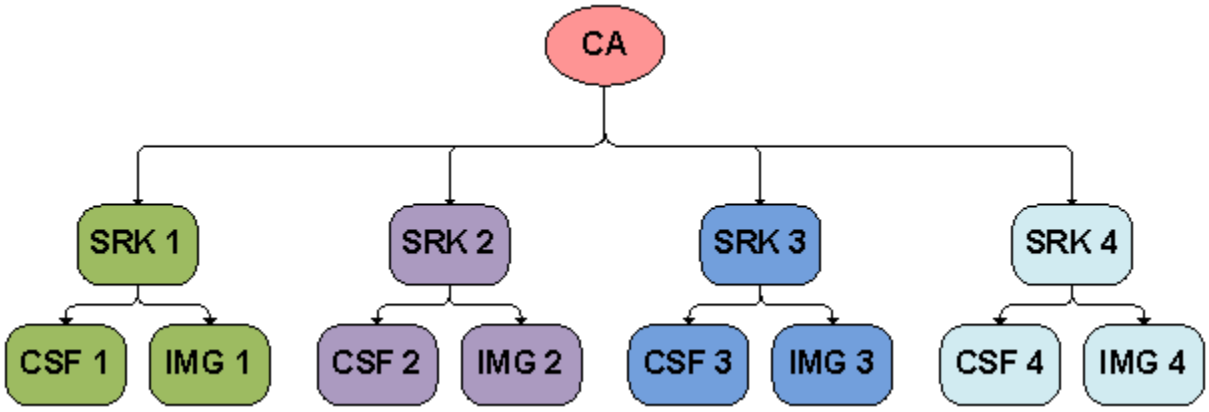

Figure 9. HABv4 PKI tree

HABv4 allows users to store up to five public keys to the key indexes. CST packages provides a script to generate additional keys under keys directory. HABv4 requires SRK in key slot 0, CSF key in slot 1 and IMG keys in keys slots 2 - 4.

NOTE

Only one SRK can be used per reset cycle. All keys must be generated from the same SRK.

5.1.1 Generating PKI tree for fast authentication

HAB v4.1.2 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 1 MB can be reduced from 25 ms to 12 ms.

Unless boot time is critical, it is recommended that the SRK have the CA flag, and the CSF and IMG keys used to validate their respective data. The fast authentication feature supplies the user with a faster boot time, at the cost of a less robust signature.

Figure 10 gives an example of a PKI tree for fast authentication that is generated by the NXP Code Signing Tools.

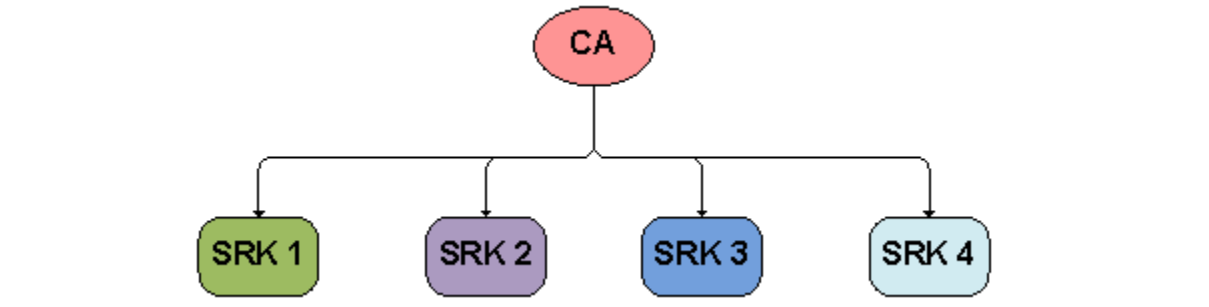

Figure 10. HABv4 enable devices fast authentication PKI tree

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

5.2 Generating SRK table

The SRK table is required by CST. It is a table of the Public SRKs. On the target device during the authentication process the HABv4 code verifies the hash of the SRK Table against the SoC `SRK_HASH[255:0]` fuses. If the verification succeeds the root of trust is established and the HABv4 code can progress with the image authentication.

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.

5.3 Creating the CSF description file

The CSF contains all the commands that the ROM executes during the secure boot. These commands instruct HABv4 on which memory areas of the image to authenticate, which keys to install and use, what data to write to a 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, remember 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 extents:

- 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 example:

```
#Illustrative Command Sequence File Description [Header]
Version = 4.2
Hash Algorithm = sha256 Engine = CAAM
Engine Configuration = 0
Certificate Format = X509
Signature Format = CMS

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

[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    "<path_to_image>"
```

The authenticate data command blocks line contains three values and the file containing the data being signed. The first value is the address on the target where HABv4 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. HABv4 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. CSF examples are also available in U-Boot documentation (see References).

5.4 Generating CSF binary signature

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

- From the *linux32, linux64 or mingw32* bin directory, call `cst` with the CSF input file:

```
./cst --o csf.bin --i csf.txt
```

5.5 Verifying HAB successfully authenticates the signed image

During development, users should check the events before the device is *closed*. Once an image is signed with a signature that does not generate events during loading, the signed image should be able to 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.

HABv4 generates events when it encounters issues. These events are written to a region in OCRAM to provide users feedback to assist in debugging. The location in OCRAM varies based on the i.MX series part.

The `hab_rvt.report_status()` function is used to obtain HABv4 audit events, details can be found in *HABv4 RVT Guidelines and Recommendations* (document AN12263).

5.5.1 SRK HASH and HAB events in open mode

Depending on the HAB version you have different behaviors for the SRK Hash fuses in open mode. For more details in the HAB version, see HAB versions and differences.

Table 2. SRK HASH and HAB events

HAB version	HAB SRK HASH check	Comments
HAB 4.1.0 and prior	Yes	HAB checks SRK Hash in open mode, must program SRK Hash fuses.
HAB 4.1.1	No	HAB does not check SRK Hash in open mode, make sure SRK's are programmed correctly in SRK fuses before closing the device.
HAB 4.1.2 and newer	Only if SRK fuses is not 0.	HAB checks SRK Hash in open mode. SRK Fuses = 0 leads to no HAB events due to SRK hash check.

NOTE

On i.MX 50 and i.MX 53 devices, it is required to program the SRK fuses 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.

5.6 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.

5.6.1 SRK fuses

The SRK fuse values are generated by `srktool` when SRK table is assembled. 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.

5.6.2 Closing 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 programming 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.

5.6.3 RNG trim fuses

HABv4 provides two options for managing the hardware RNG available in CAAM. HABv4 can initialize the RNG, or defer the initialization for the CAAM Operating System driver to manage.

RNG trim fuses provide HABv4 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, HABv4 does not allow the device to continue booting if it is *Closed*. Only HABv4 sources the RNG Trim Fuse value. The CAAM driver needs to perform a similar RNG trim configuration, but the values it uses are built into the driver software.

NOTE

On HAB v4.2.0 and later the RNG is not instantiated by default, for older versions choose one of two methods, or the chip will not boot when the device is *Closed*.

5.6.3.1 Option 1 – Deferring RNG instantiation for post HAB software (recommended)

Deferring the RNG instantiation is done by adding a command to the CSF signature data. This command informs HABv4 to skip the instantiation. As of all CST versions 2.3 and greater, this command is included in the CSF signature by default if CAAM engine is defined, unless it is overridden by the CSF description file. When deferring the RNG instantiation any operation that requires the RNG is not available during ROM boot, if necessary RNG can be instantiated in a later boot stage in bootloader or kernel level. Although encryption and blob generation are not available, the HAB-signed or encrypted boot features are not affected.

The CSF configuration file is discussed later and you can find examples in U-Boot documentation. Deferring RNG instantiation is done by adding the following line in CSF Header:

```
[Header]
Engine = CAAM
```

In case that Engine must be set to other configuration value, RNG can be deferred by using the Unlock command:

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

5.6.3.2 Option 2 – Setting RNG trim in fuse

The `RNG_TRIM[7:0]` 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.

5.7 Securing the device

Additionally, the following fuses must be programmed to completely secure your device. Note that this operation is irreversible, and some features used for development may not be available after the following commands.

- `SRK_LOCK`: Lock for `SRK_HASH[255:0]` fuses.
- `DIR_BT_DIS`: Disable direct external memory boot.
- `SJC_DISABLE`: Disable the secure JTAG controller module.
- `JTAG_SMODE`: Set JTAG security mode to no debug mode `0x11`.
- `JTAG_HEO`: Disallows HAB JTAG enabling.
- `BOOT_CFG_LOCK`: Lock on BOOT related fuses.

NOTE

For a full list of OEM programmable fuses, please contact your local NXP representative.

5.8 Extending the root of trust

The High Assurance Boot (HABv4) code located in the on-chip ROM provides an Application Programming Interface (API) making it possible to call back into the HABv4 code for authenticating additional images.

The HABv4 API functions are accessible through the ROM Vector Table (RVT). For further details, see *HABv4 RVT Guidelines and Recommendations* (document AN12263).

NXP recommends using the `hab_rvt.authenticate_image()` API function whenever possible. This ensures all the proper authentication steps are performed. For HAB versions 4.2.0 and newer it is highly recommended to use `authenticate_image_no_dcd()` API function instead.

NOTE

The `hab_rvt.authenticate_image()` API function is intended to authenticate additional boot images in a post-ROM stage, initial boot images are supposed to be authenticate only once by the initial ROM code. It's highly recommended to ensure the IVT DCD pointer is Null prior to calling HABv4 authenticate function.

6 Known limitations

HABv4-related issues and limitations are documented at i.MX support community. See *HABv4 known limitations and guidelines* community page. Please contact your local NXP representative for more details.

7 Revision history

Table 3. Revision history

Revision number	Date	Substantive changes
0	10/2012	Initial release
1	06/2015	• Updated CST command lines to version 2.3.3 and added Windows example. • Reorganized sections and removed duplicate information available in CST/HAB user guides. • Added sections for Fast Authentication and the RNG Trim. • Added CSF Examples.

Table continues on the next page...

Table 3. Revision history (continued)

Revision number	Date	Substantive changes
2	05/2018	• Updated Figure 6. • Updated CST command lines to version 2.3.3 and added Windows example. • Updated 3.3.2 RNG trim fuses. • Updated scripts and CSF examples. • Added a note about ERR010449. • Added reference for i.MX7 devices. • Added Securing the device. • Added Appendix E Extracting U-boot data for CSF • Added HAB versions and differences. • Added Extending the root of trust.
3	06/2018	• Document restructured as step-by-step content is now included in U-Boot guides (see References). • Added reference for i.MX RT 10xx, i.MX 7ULP, i.MX 8M, i.MX 8M Mini and i.MX 8M Nano devices. • Added Figure 1. • Added Boot flow. • Added Image layout. • Added Field return enablement on HABv4.
4	06/2020	• Make information common for i.MX 8M family. • Updated Table 1. • Added information for ECDSA support in i.MX 8M Plus. • Updated Figure 5. • Updated HAB versions and differences. • Added information for i.MX 7ULP SRK revocation.

A HAB versions and differences

For details on HAB versions and differences document, please contact your local NXP representative.

B SRK revocation on HABv4

The HABv4 supported devices includes 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 or more SRKs are compromised, it is possible to revoke that SRK. There are up to four SRK revoke fuse bits that map to the SRK table indexes. An SRK key is revoked by burning the corresponding bit in the `SRK_REVOKE[3:0]` eFuse field. Figure 11 illustrates an example.

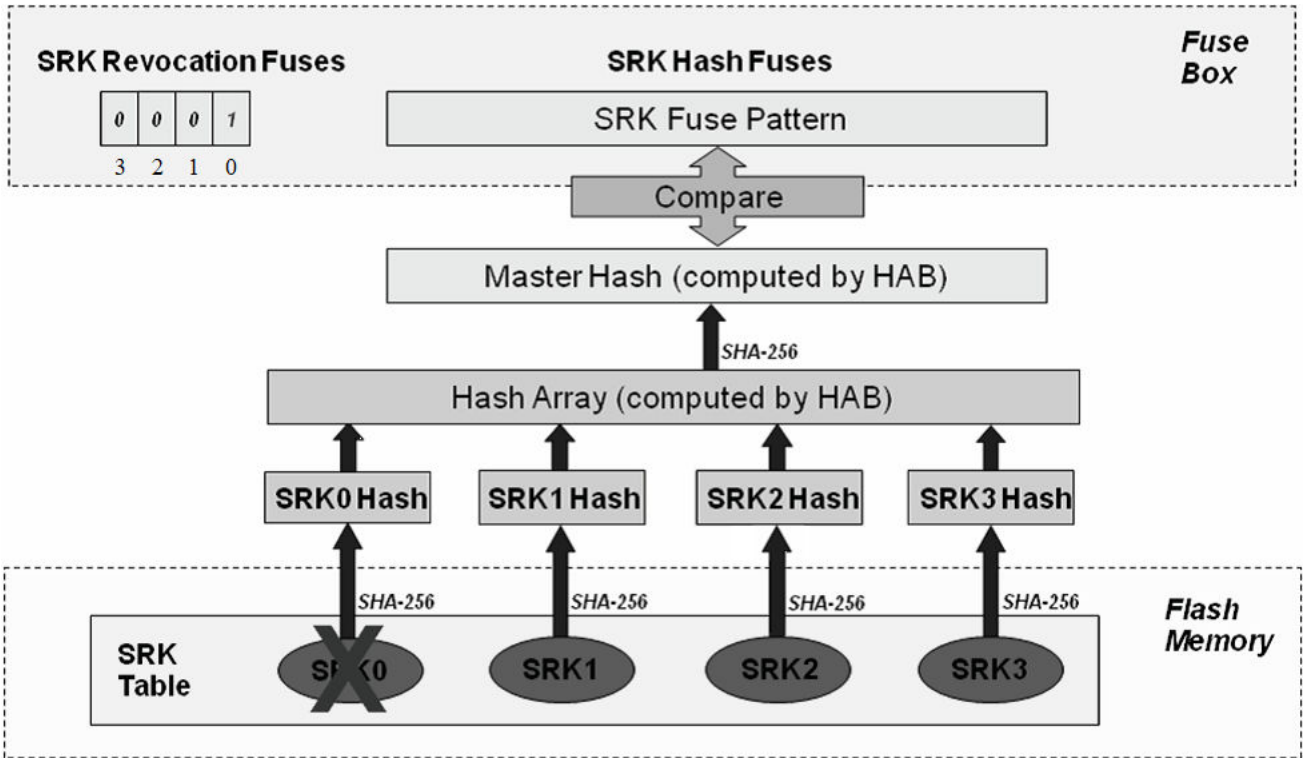

Figure 11. SRK Revocation using HABv4

NOTE

There are only three SRK revoke fuse bits on i.MX 6, 7 and RT10xx series devices, and only the first three SRKs can be revoked.

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.

For more details about `SRK_REVOKE` fuses, please check Table 4.

Table 4. SRK revocation

SRK key	Source index	`SRK_REVOKE[3:0]`
SRK0	0	0001
SRK1	1	0010

Table continues on the next page...

Table 4. SRK revocation (continued)

SRK key	Source index	SRK_REVOKE[3:0]
SRK2	2	0100
SRK3	3	1000

In closed configuration, HABv4, by default, set the SRK_REVOKE_LOCK sticky bit in the OCOTP controller to write protect this eFuse field. To instruct HABv4 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 any of the SRK fuses to be blown by a trusted bootloader or runtime image. Below is an example CSF command that unlocks the SRK_REVOKE eFuse field, allowing bootloader or a later stage to burn the fuse.

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

For more detailed information about CSF commands, refer to *CST User's Guide*.

NOTE

The SRK Revocation does not modify the SRK Hash values, only SRK_REVOKE fuse has to be programmed.

In i.MX 7ULP, SRK revocation is not possible in Single boot mode. Please refer to Known limitations.

C Field return enablement on HABv4

Manufacturers usually enable debugging restrictions to protect sensitive data in their end product design, Fuse programming advises the following fuses to be programed to completely secure the device:

- SEC_CONFIG
- DIR_BT_DIS
- JTAG_SMODE
- SJC_DISABLE

These debugging restrictions also constrain the legitimate debugging of the field return devices with suspected faults. The chip includes a field return feature to enable the legitimate debugging, including the possibility for NXP to run the test modes on returned parts.

The field return feature is based in FIELD_RETURN efuse configuration, once programmed the part is:

- Reverted the value of the DIR_BT_DIS fuse.
- JTAG restrictions are disabled.
- Reverted to open mode, unsigned images are allowed to be executed.

NOTE

Field return configuration is required for NXP to run the test patterns even on non-secure products, for further details and fuse addresses please contact your local NXP representative.

Before leaving the BootROM the FIELD_RETURN_LOCK bit is set to prevent any unintended programming operation. To instruct HABv4 not to lock the FIELD_RETURN bit the use of the Unlock CSF command is required. Including this command in a CSF signature allows the FIELD_RETURN fuse to be blown by a trusted bootloader or runtime image.

The Unlock field return command also requires a 64-bit unique identifier (UID), this value is unique per device and stored in SoC fuses. Below is an example CSF command that unlocks the field return feature, allowing bootloader or a later stage to burn the fuse.

```
[Unlock]
Engine = OCOTP
Features = Field Return
UID = 0x01, 0x23, 0x45, 0x67, 0x89, 0xab, 0xcd, 0xef
```

How To Reach Us

Home Page:

nxp.com

Web Support:

nxp.com/support

Information in this document is provided solely to enable system and software implementers to use NXP products. There are no express or implied copyright licenses granted hereunder to design or fabricate any integrated circuits based on the information in this document. NXP reserves the right to make changes without further notice to any products herein.

NXP makes no warranty, representation, or guarantee regarding the suitability of its products for any particular purpose, nor does NXP assume any liability arising out of the application or use of any product or circuit, and specifically disclaims any and all liability, including without limitation consequential or incidental damages. “Typical” parameters that may be provided in NXP data sheets and/or specifications can and do vary in different applications, and actual performance may vary over time. All operating parameters, including “typicals,” must be validated for each customer application by customer's technical experts. NXP does not convey any license under its patent rights nor the rights of others. NXP sells products pursuant to standard terms and conditions of sale, which can be found at the following address: nxp.com/SalesTermsandConditions.

While NXP has implemented advanced security features, all products may be subject to unidentified vulnerabilities. Customers are responsible for the design and operation of their applications and products to reduce the effect of these vulnerabilities on customer’s applications and products, and NXP accepts no liability for any vulnerability that is discovered. Customers should implement appropriate design and operating safeguards to minimize the risks associated with their applications and products.

For more information, please visit: http://www.nxp.com

For sales office addresses, please send an email to: salesaddresses@nxp.com

Date of release: June 2020

Document identifier: AN4581

arm