AN12312

Secure Boot on i.MX 8 and i.MX 8X Families using AHAB

Rev. 0 — May 2019

Application Note

1 Introduction

Contents

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 Advanced High Assurance Boot (AHAB) component on the on-chip ROM and the Security Controller (SECO) Firmware. The AHAB (only supported on i.MX 8 and i.MX 8X families) in ROM is responsible for authenticating the SECO firmware (NXP signed), which will supply the services for authenticating the images signed by the user to the System Controller ROM.

The purpose of this application note is to provide a secure boot reference for i.MX 8 and i.MX 8X families applications processors that include AHAB. It also provides information on select SECO features required for secure boot.

1.2 Intended audience

This document is intended for those who:

- Need an explanation about the procedure for signing a boot image.
- Need to design signed software images to be used with an AHAB-enabled processor.

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

For a step-by-step technical guide, please refer to the U-Boot project documentation (see References on page 3).

1.3 Scope

This document is a practical example to illustrate the construction of a secure boot image and to configure the target device to run securely, which is possible because of the AHAB and the Code Signing Tool (CST).

This document targets the Secure Boot feature on the following applications processors from the i.MX family:

- i.MX 8 – 8 Dual Max (DM), 8 Quad Max (QM).
- i.MX 8 X – 8 Dual X Plus (DXP), 8 Quad X Plus (QXP).

This application note only demonstrates the secure boot solution on the i.MX 8 and 8X processors, as well as some SECO features for secure boot. It focuses on:

- Secure boot architecture.
- Secure boot implementation.
- SECO features.
- Container authentication.

1.4 Definitions, acronyms, and abbreviations

Table 1. Definitions, acronyms and abbreviations on page 2 lists the terms and acronyms used in this document.

Table 1. Definitions, acronyms and abbreviations

Abbreviation	Full name	Remarks
AES	Advanced Encryption Standard	—
AHAB	Advanced 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 processor running AHAB.
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 algorithm, with a random number generator and run time integrity checker
CMS	Cryptographic Message Syntax	A general formal for data that may have cryptography applied to it, such as digital signatures and digital envelopes. AHAB uses the CMS as a container to hold the PKCS#1 signatures.
CSF	Command Sequence File	A binary data structure interpreted by AHAB 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 stages.
OS	Operating System	—
OTP	One-Time Programmable	The OTP hardware includes the masked ROM and electrically programmable fuses (eFuses).
PKCS#1	—	A standard that specifies the use of the RSA algorithm
PKI	Public Key Infrastructure	A hierarchy of public key certificates in which each certificate (except for the root certificate) can be verified using the public key above it.
RSA	—	Public key cryptography algorithm developed by Rivest, Shamir and Adleman
SA	Signature Authority	The holder of the private key used to sign software components.
SCFW	SCU FirmWare	—
SDP	Serial Download Protocol	Also call UART/USB serial download mode. It allows code provisioning through UART or USB during the production and development phases.

Table continues on the next page...

Table 1. Definitions, acronyms and abbreviations (continued)

SECO	SEcurity COntroller	—
SPL	Secondary Program Loader	—
SRK	Super Root Key	A 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, the SRK acronym (in this document) refers to the public key only.

1.5 References

- i.MX 8 Reference Manual and Security Reference Manual
- AHAB CST User Guide available in the Code Signing Tool package downloadable on nxp.com. Search for CST_TOOL
- U-Boot technical guides and examples, available in *doc/imx/ahab* repository of the U-Boot project, on the *imx_v2018.03_4.14.78GA* release branch (initial release)

2 Overview

2.1 AHAB secure boot architecture

The AHAB secure boot feature relies on digital signatures to prevent unauthorized software execution during the device boot sequence. In case a malware takes control of the boot sequence, sensitive data, services and network can be impacted.

The AHAB authentication is based on public key cryptography in which image data is signed offline using one or more private keys. The resulting signed image data is then verified on the i.MX processor using the corresponding public keys. The public keys are included in the final binary and the SRK Hash is programmed in the SoC fuses for establishing the root of trust.

On i.MX 8 and i.MX 8X families, the SCU is responsible to interface with the boot media, managing the process of loading the firmware and software images in different partitions of the SoC. The SECO is responsible to authenticate the images, authorizing the execution of them.

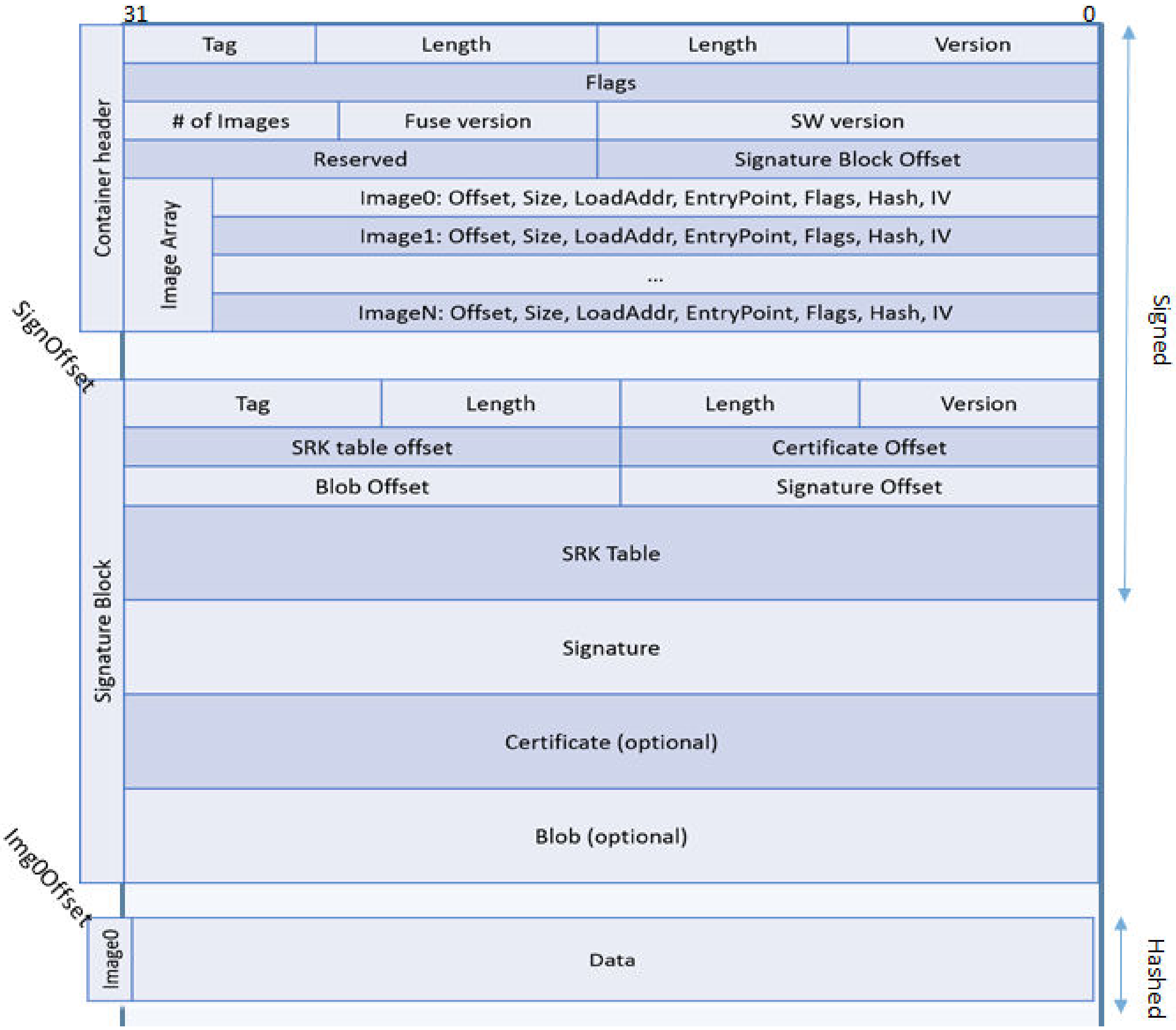


Figure 1. Container layout

The SRK Table is generated with the SRK Tool, provided with the Code Signing Tool (CST).

2.2 Secure boot flow

Due to the multicore architecture, the i.MX 8 boot sequence involves SCU ROM, SCU FirmWare, SECO ROM, and SECO FW.

Figure 2 on page 5 illustrates the secure boot flow overview.

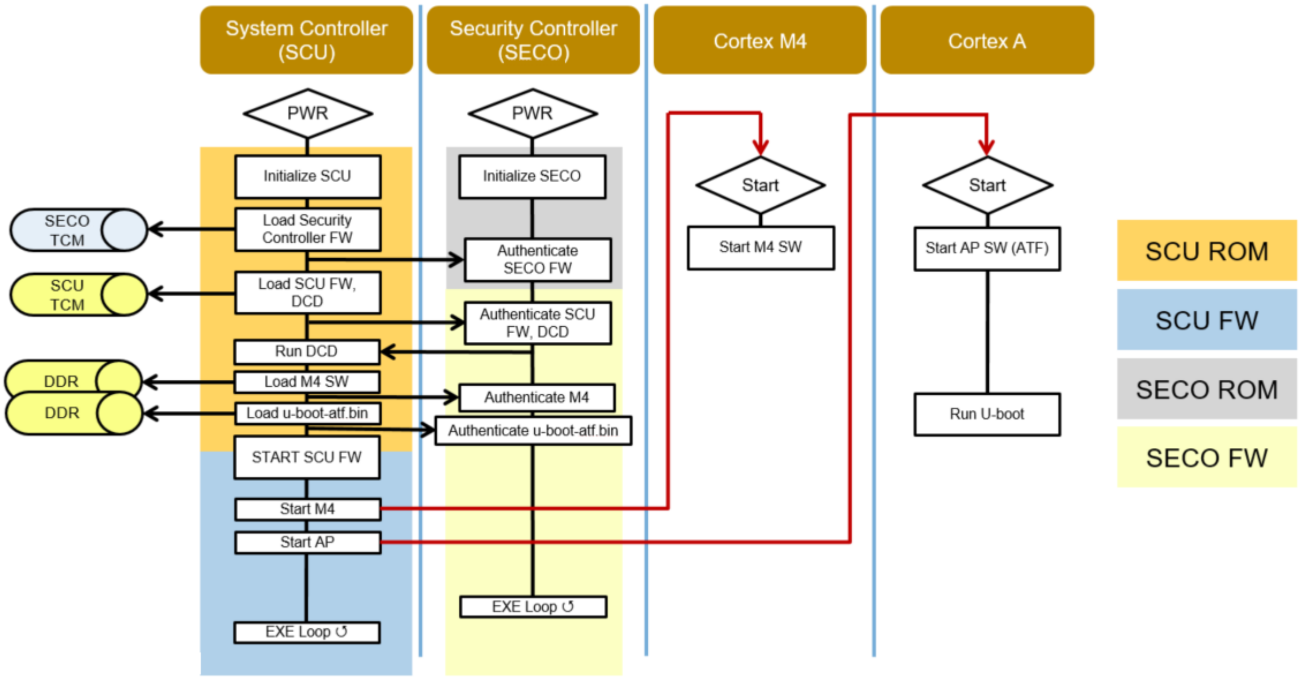


Figure 2. Secure boot flow overview

The i.MX8 and i.MX8x boot flow is as follows.

1. At reset, the SCU ROM and SECO ROM both start execution.
2. The SCU ROM reads the boot configuration and loads the SECO FW (first container) from the boot media to the SECO TCM.
3. A message is sent by the SCU ROM via MU requesting the SECO ROM to authenticate the SECO FW which is signed using NXP key.
4. The SCU ROM loads the second container from the boot media, this container must contain at least the SCFW which is signed using the OEM keys.
5. The SCU ROM loads the SCFW to the SCU TCM, a message is sent via MU requesting the SECO FW to authenticate the SCU FW and DCD table.
6. The SCU ROM configures the DDR and loads the M4 and AP images to their respective load addresses.
7. The SCU ROM requests the SECO FW to authenticate the M4 image.
8. The SCU ROM requests the SECO FW to authenticate the AP image.
9. The SCU FW is initialized and starts the Arm® Cortex®-M and Cortex-A cores.
10. From this point additional containers can be loaded and authenticated by Cortex-M and Cortex-A cores, the software must interface with SCU by calling the sc_misc_seco_authenticate() API function (see SECO authentication service via SCU API on page 11).

After each authentication, SECO FW returns a success or failure status to SCU.

If SCU receives a fail response from SECO FW authentication while attempting to boot from the primary boot source, the SCU will attempt to boot from the secondary boot source (if any).

If SCU receives a fail response from SECO FW authentication while attempting to boot from the secondary boot source, the SCU will got into recovery mode.

If the SCU receives a fail response for the second container, the SCU will enter the recovery mode.

3 Secure boot implementation

3.1 Overview

The boot image is composed of different layers, as shown in Figure 3 on page 6.

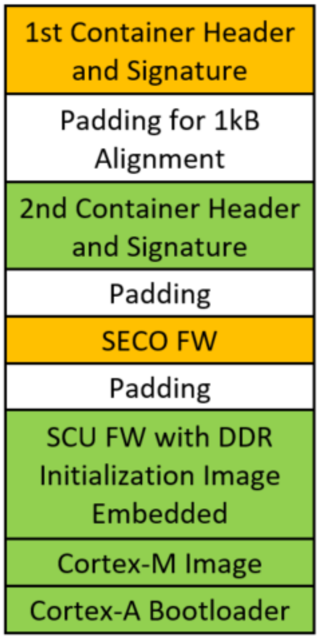

1st Container Header and Signature
Padding for 1kB Alignment
2nd Container Header and Signature
Padding
SECO FW
Padding
SCU FW with DDR Initialization Image Embedded
Cortex-M Image
Cortex-A Bootloader

Figure 3. Secure boot image layout

The boot image contains two containers, one for the SECO firmware (AHAB), and one for the SCFW, the ATF, U-Boot and M4 Image. They are preceded by their headers. The first one, containing the SECO firmware image, is padded to `0x1000` to fix the start address of the second one, which can contain one or multiple images.

NOTE

The only required images for the device are the SECO FW and the SCFW. The Cortex-A or Cortex-M images may or may not be part of it depending on the OEMs system design.

In contrast with the secure boot process used in the HABv4 architecture, there is no need for CSF in this architecture. The CST is responsible to handle the signature block, as shown in Figure 4 on page 7.

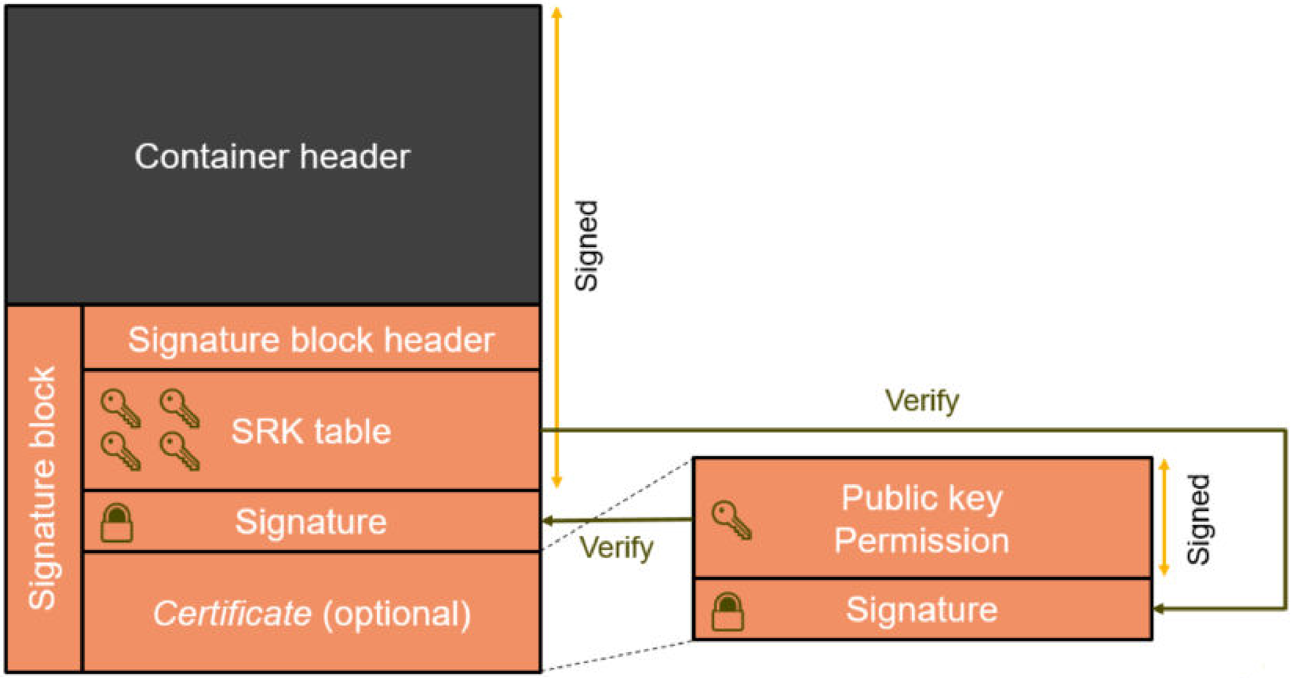


Figure 4. Signature block

The container signature is verified against the SGK key certificate, which is then verified against the SRK table.

If the subordinate key is not used, the container signature is directly verified against the SRK keys.

3.2 Protocols

There are some changes required to the usual image building process to get a secure boot image.

- Generate the final binary image which must be in a container format. Offsets must be calculated or copied from the build log to create the associated CSF description file.
- Use CST as described by the *mx8_mx8x_secure_boot.txt* document available in U-Boot source, updating the offsets of the *doc/imx/ahab/csf_examples/csf_boot_image.txt* file.
- Generate a PKI tree (see Generating a PKI tree on page 7).
- Program SRK HASH fuses on the target.
- Check for SECO events (for more details, see Verifying/Decoding SECO events on page 9).
- Close the device (OEM mode).

3.3 Generating a PKI tree

CST includes scripts for generating a PKI tree and SRK table. Figure 5 on page 8 shows an example of the PKI tree.

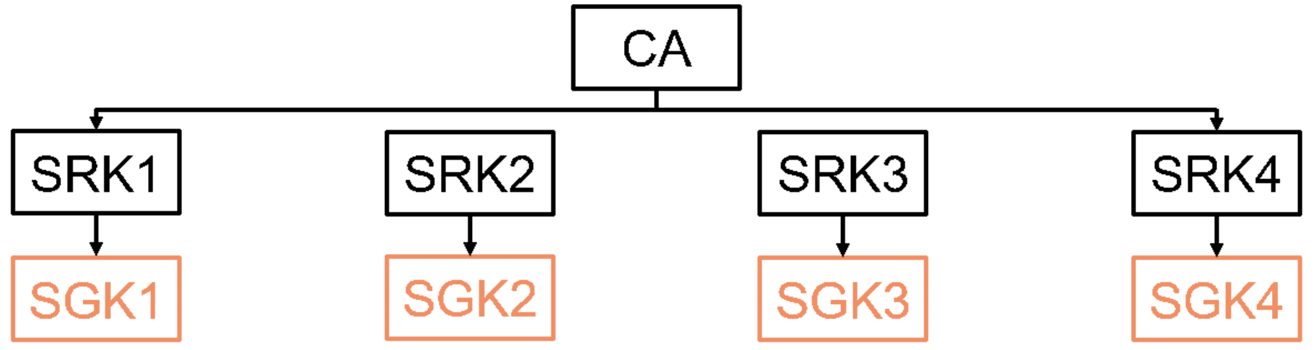


Figure 5. PKI tree example

With:

- Super Root Key (SRK): always 4 SRKs
- SGK: Optional Image Keys (Certificate) which can be used to verify the signature included in the final signed image.

To generate a P384 EEC PKI tree on CST v3.1.0, run the ahab_pki_tree.sh script located in the *keys* repository of the CST:

```
$ ./ahab_pki_tree.sh
…
Do you want to use an existing CA key (y/n)?: n
Do you want to use Elliptic Curve Cryptography (y/n)?: y
Enter length for elliptic curve to be used for PKI tree:
Possible values p256, p384, p521:  p384
Enter the digest algorithm to use: sha384
Enter PKI tree duration (years): 10
Do you want the SRK certificates to have the CA flag set? (y/n)?: n
```

To also generate the SGKs, you must set the CA flags when generating the SRK certificates:

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

3.4 Examples

Technical step-by-step examples have been added to U-Boot documentation. The following documents are dedicated to a Secure Boot example for i.MX 8 and 8X device families:

- uboot/doc/imx/ahab/guides/mx8_mx8x_secure_boot.txt
- uboot/doc/imx/ahab/csf_examples/csf_boot_image.txt

Additional documents can be found on these repositories, as AHAB introduction, SPL Secure Boot and other CSF examples. Are also mentioned/detailed into these documents:

- SRK tool.
- SRK hash and SRK fuses.
- CSF description file and CST tool (generic command and examples).
- Closing the device / changing lifecycle.

4 SECO features

This section describes SECO interfaces useful for secure boot. For more details about functions definitions, please refer to the SECO API Reference Guide.

4.1 Getting chip info

The SCU API provides an interface to retrieve SECO chip info (monotonic counter, lifecycle and UID): sc_misc_seco_chip_info(…).

4.2 Lifecycle

The Lifecycle defines the security state of the device. Access to some features are limited by the state, for instance writing fuses. Refer to the Security Reference Manual for more details about Lifecycle.

The SCU API provides two interfaces to manage lifecycle:

- sc_misc_seco_forward_lifecycle(…): This function is used to update the lifecycle of the device from **NXP Closed** to **OEM Closed**.
- sc_misc_seco_return_lifecycle(…): This function updates the lifecycle from **OEM Closed** to **Partial Field Return**.

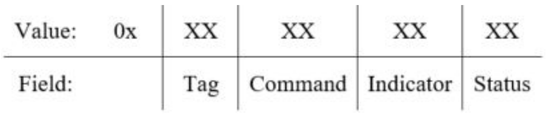
Changing the lifecycle to **Partial Field Return** requires a message signed by OEM SRK.

4.3 Verifying/Decoding SECO events

A new SECO service is available to retrieve any singular event that has occurred since the firmware as started: sc_seco_get_event().

The events are stored in a fixed sized buffer, therefore new occurring events are lost when the capacity of the buffer is exceeded. The event buffer is systematically returned in full, whatever the number of events stored.

About the returned event format, see Figure 6 on page 9.



Value:	0x	XX	XX	XX	XX
Field:		Tag	Command	Indicator	Status

Figure 6. Returned event format

For the command field, the expected value at this step is `0x87` (ID for AHAB_AUTH_CONTAINER_REQ).

About the indicator field, see Table 2. Indicators on page 9.

Table 2. Indicators

Value	Indicator	Description
`0xEE`	AHAB_NO_AUTHENTICATION_IND	Container required to skip the authentication.
`0xF0`	AHAB_BAD_SIGNATURE_IND	Bad signature.

Table continues on the next page...

Table 2. Indicators (continued)

`0xF1`	AHAB_BAD_HASH_IND	Bad hash.
`0xF9`	AHAB_INVALID_KEY_IND	The key in the container is invalid.
`0xFA`	AHAB_BAD_KEY_HASH_IND	The key hash verification does not match OTP.

The two main events you might face are:

- `0x0087EE00`: Your container image has not been signed yet.
- `0x0087FA00`: The container image has been signed with wrong key which does not match the OTP SRK hashes fused on the target.

NOTE

The event `0x0087FA00` may also be displayed in case the SRK fuses are not programmed yet.

4.4 Authenticating OS containers (and other ones)

4.4.1 Overview

More containers can be authenticated, for instance in the case of boot image generated by SPL targets. It contains three containers, as shown in Figure 7 on page 10.

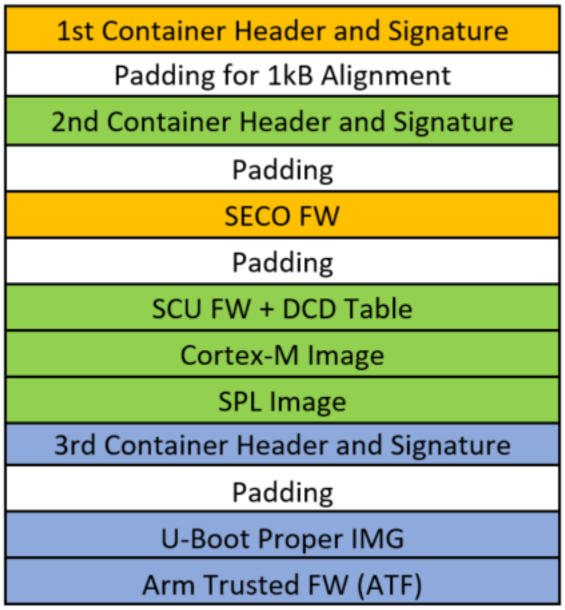

1st Container Header and Signature
Padding for 1kB Alignment
2nd Container Header and Signature
Padding
SECO FW
Padding
SCU FW + DCD Table
Cortex-M Image
SPL Image
3rd Container Header and Signature
Padding
U-Boot Proper IMG
Arm Trusted FW (ATF)

Figure 7. Secure boot image with SPL layout

First and second containers are authenticated at SCU ROM level, whereas the third container is authenticated at SPL level.

4.4.2 Authentication with code signing tool

The first method to authenticate OS containers is similar to the signing method of secure boot described in Secure boot implementation on page 6. Once the OS container image has been generated, update the CSF example provided by U-Boot documentation with the offset values returned by the make command, then execute the CST with this file to sign the OS image. The new generated file is the OS signed image.

NOTE

A step-by-step guide to authenticate the OS container as described here is available in the *mx8_mx8x_secure_boot.txt* guide from U-Boot documentation.

4.4.3 SECO authentication service via SCU API

It is also possible to authenticate an image with the `sc_misc_seco_authenticate(…)` function of the SCU API.

It is used to authenticate an SECO image or issue a security command. The `addr` parameter often points to a container, but it can also be some data (or even unused) for some commands.

NOTE

It is recommended to load the container header in CAAM Secure Memory (SM) instead of OCRAM as SECO does not have direct access to OCRAM and OCRAM does not have the same access permission control that CAAM SM does.

4.5 SRK revocation

It is possible to revoke an SRK (for instance if the SRK1 public key must be replaced to SRK2). If the SRK revoke bit is set, the SECO firmware will set the fuse for that, but only after successful authentication of the header that contains the SRK revocation command and the receipt of the **COMMIT** command with the corresponding argument. It can be performed with the Code Signing Tool and the SCU API.

NOTE

An SRK being used by the current container cannot be revoked

As only one SRK may be selected at boot time through an Install SRK CSF command, users must ensure that the CSF is updated accordingly. It is possible to revoke only the first three SRKs, by burning the corresponding bit in the `SRK_REVOKE [2:0]` eFuse field.

4.6 Known limitations

There are some known limitations:

- In i.MX 8 QXP B0, the container header size is limited to 4 kB. For the maximum number of images supported depending on the cryptographic algorithm, see Maximum number of images supported on page 12.
- The SECO FW ignores image integrity failure if detected in open mode.
- On *L4.14.78_1.0.0GA* release, there is a 1K SPL limitation for the container header size.

A Maximum number of images supported

Table 3. Maximum number of images supported

	Maximum number of images supported			
	No Blob	**Blob of AES key 128-bit**	**Blob of AES key 192-bit**	**Blob of AES key 256-bit**
ECC P256	28	28	28	28
+ Cert	27	27	26	26
ECC P384	27	27	26	26
+ Cert	25	25	25	25
ECC P521	26	25	25	25
+ Cert	23	23	23	23
RSA 2k	21	20	20	20
+ Cert	16	16	16	16
RSA 3k	16	15	15	15
+ Cert	9	9	9	9
RSA 4k	11	10	10	10
+ Cert	2	2	2	2

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: May 2019
Document identifier: AN12312