# AN12312

## Secure Boot on AHAB Supported Devices

Rev. 1 — 14 August 2023 Application note

**Document Information**

| Information | Content |
|---|---|
| Keywords | Secure boot architecture, AHAB, SECO, ELE, PKI tree |
| Abstract | This application note provides a secure boot reference for AHAB supported i.MX application processors. It also provides information related to some SECO and ELE features required for secure boot. |

# 1 Introduction

This document provides a secure boot reference for AHAB supported i.MX application processors. It also provides information on the select SECO and ELE features required for secure boot.

Executing trusted and authentic code on the processor used for an application starts with securely booting the device. The i.MX family of applications processors provides this capability with the AHAB component on the security co-processor.

***Note:*** *Security Controller (SECO) for the i.MX 8/8X family and EdgeLock secure enclave (ELE) for the i.MX 8ULP/9x SoC.*

AHAB is responsible for authenticating the corresponding security co-processor firmware, which supplies the services for authenticating the images signed by the user to the System Controller (in i.MX 8/8X) or application core (in i.MX 8ULP/9x) ROM.

## 1.1 Intended audience

This document is intended for those who:

- Requires the procedure for signing a boot image.
- Designs signed software images that are used with the AHAB-enabled processor.

**Assumption**: The reader is familiar with the basics of digital signatures and public key certificates.

For a step-by-step technical guide, see the U-Boot project documentation in References.

## 1.2 Scope

This document provides the practical example which illustrates the construction of a secure boot image and to configure the target device to run securely. This is possible due to 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 8X: 8 Dual X Plus (DXP), 8 Quad X Plus (QXP)
- i.MX 8X Lite: 8 Dual X Lite (DXL)
- i.MX 8 ULP
- i.MX 93

This application note demonstrates the secure boot solution on the i.MX application processors and some SECO/EdgeLock secure enclave features for secure boot. It focuses on:

- Secure boot architecture
- Secure boot implementation
- SECO features
- EdgeLock secure enclave features
- Container authentication

## 2 Secure boot architecture

### 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. This 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 the i.MX 8/8X/8XLite 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.

On the i.MX 8ULP and i.MX 9x families, there is no SCU. An application core or real time core provides an interface with the boot media, based on the boot mode configuration. The boot core manages the process of loading the firmware and software images in different partitions of the SoC. The EdgeLock enclave authenticates the images, authorizing the execution of them.

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

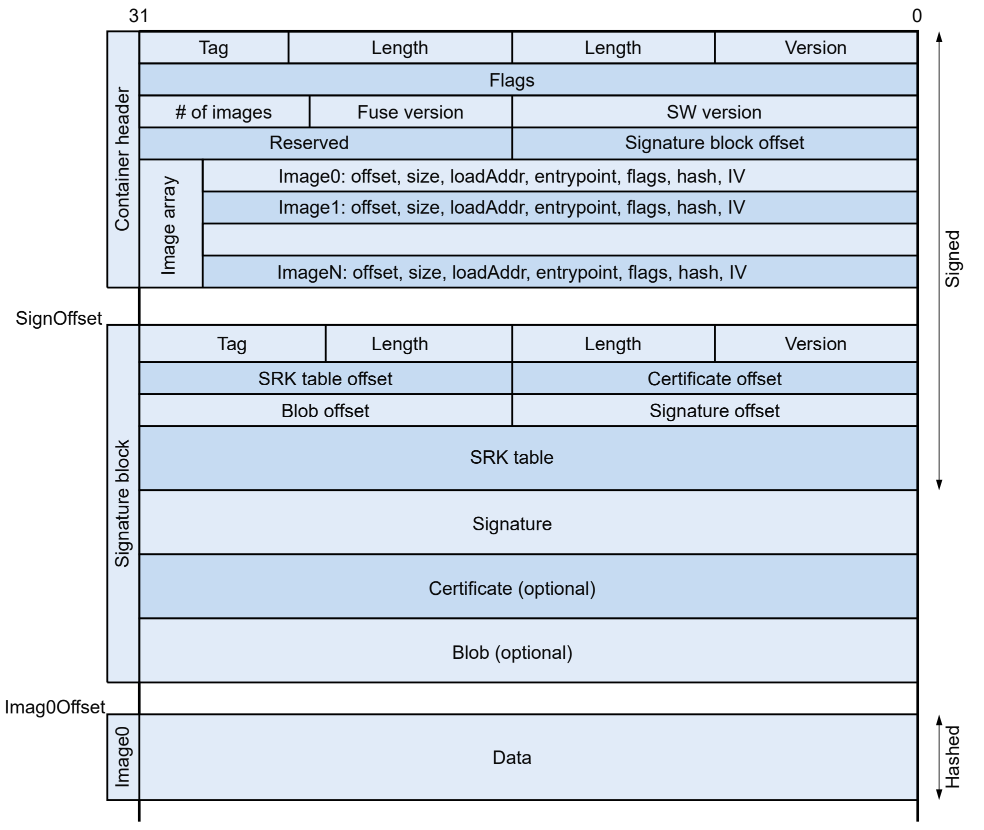

**Figure 1. AHAB secure boot architecture**

## 2.2 Secure boot flow

### 2.2.1 i.MX 8/8x family

The i.MX 8/8x boot sequence involves SCU ROM, SCU firmware, SECO ROM, and SECO firmware, due to its multicore architecture.

Figure 2 illustrates the secure boot flow overview.

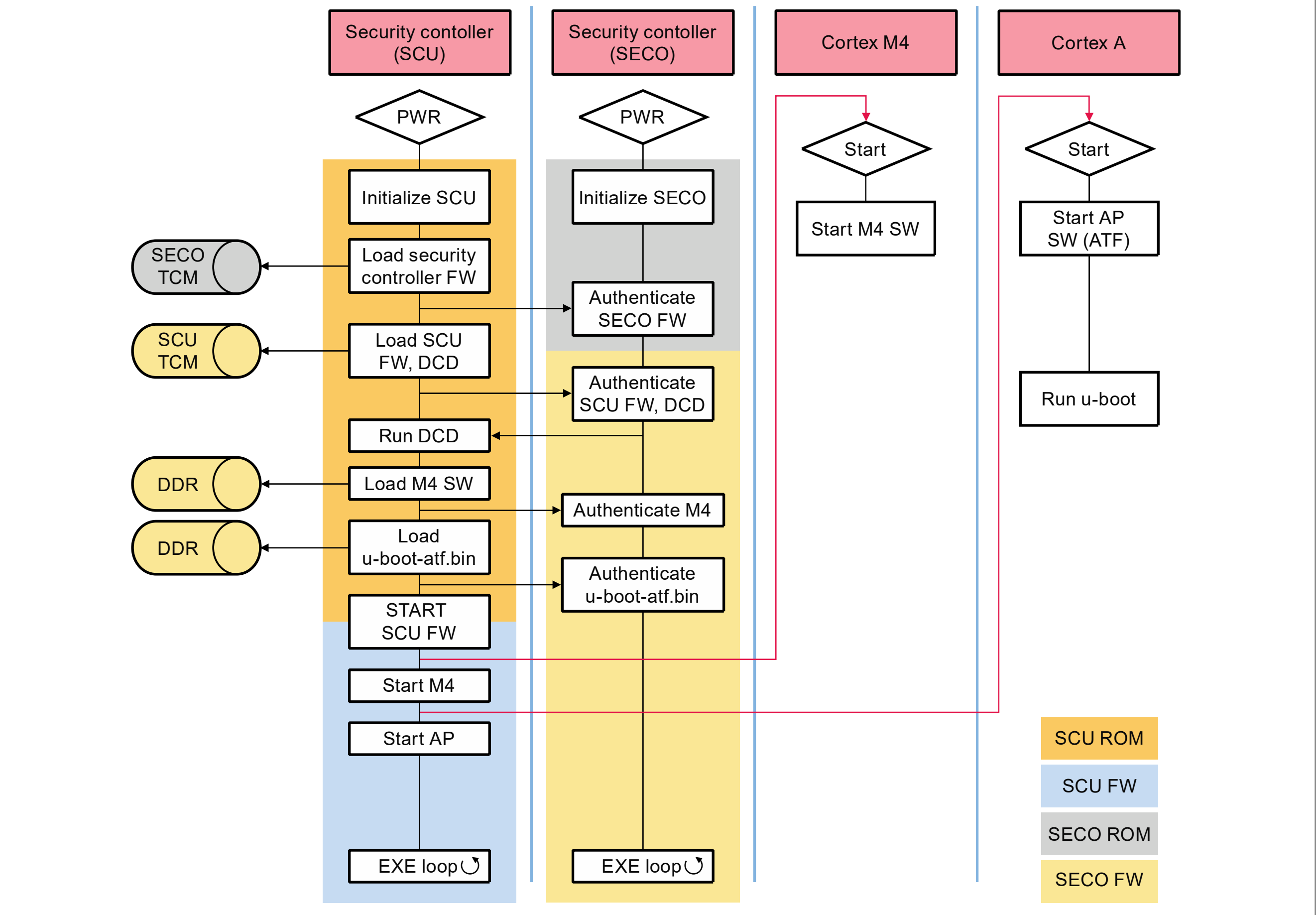

**Figure 2. Secure boot flow overview**

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 firmware (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 firmware which is signed using the NXP key.
4. The SCU ROM loads the second container from the boot media, this container must contain at least the SCU firmware, which is signed using the OEM keys.
5. The SCU ROM loads the SCU firmware to the SCU TCM, a message is sent via MU requesting the SECO firmware to authenticate the SCU firmware 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 firmware to authenticate the M4 image.

8. The SCU ROM requests the SECO firmware to authenticate the AP image.
9. The SCU firmware is initialized and starts the Arm Cortex-M and Cortex-A cores.
10. From this point, the 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 ).

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

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

Compared to i.MX8/8X, the i.MX8XL device has a new V2X subsystem to accelerate/offload V2X related cryptographic operations.

The I.MX8XL boot flow is as follows:

1. At reset, the SCU ROM and SECO ROM start execution.
2. The SCU ROM reads the boot configuration and loads the SECO firmware (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 firmware which is signed using the NXP key.
4. The SCU ROM loads the second container from the boot media, this container must contain at least the SCU firmware, which is signed using the OEM keys.
5. The SCU ROM loads the SCU firmware to the SCU TCM, a message is sent via MU requesting the SECO firmware to authenticate the SCU firmware and DCD table.
6. The SCU boot ROM loads the V2X firmwares and optional V2X patches to the intermedia loading space in DDR if the boot device is not FlexSPI NOR.
7. The SECO firmware informs V2X ROM to load/authenticate the V2X.
8. The SCU ROM configures the DDR and loads the M4 and AP images to their respective load addresses.
9. The SCU ROM requests the SECO firmware to authenticate the M4 image.
10. The SCU ROM requests the SECO firmware to authenticate the AP image.
11. The SCU firmware is initialized and starts the Arm Cortex-M and Cortex-A cores.
12. From this step, the 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. For more information, see Section 4.4.2.

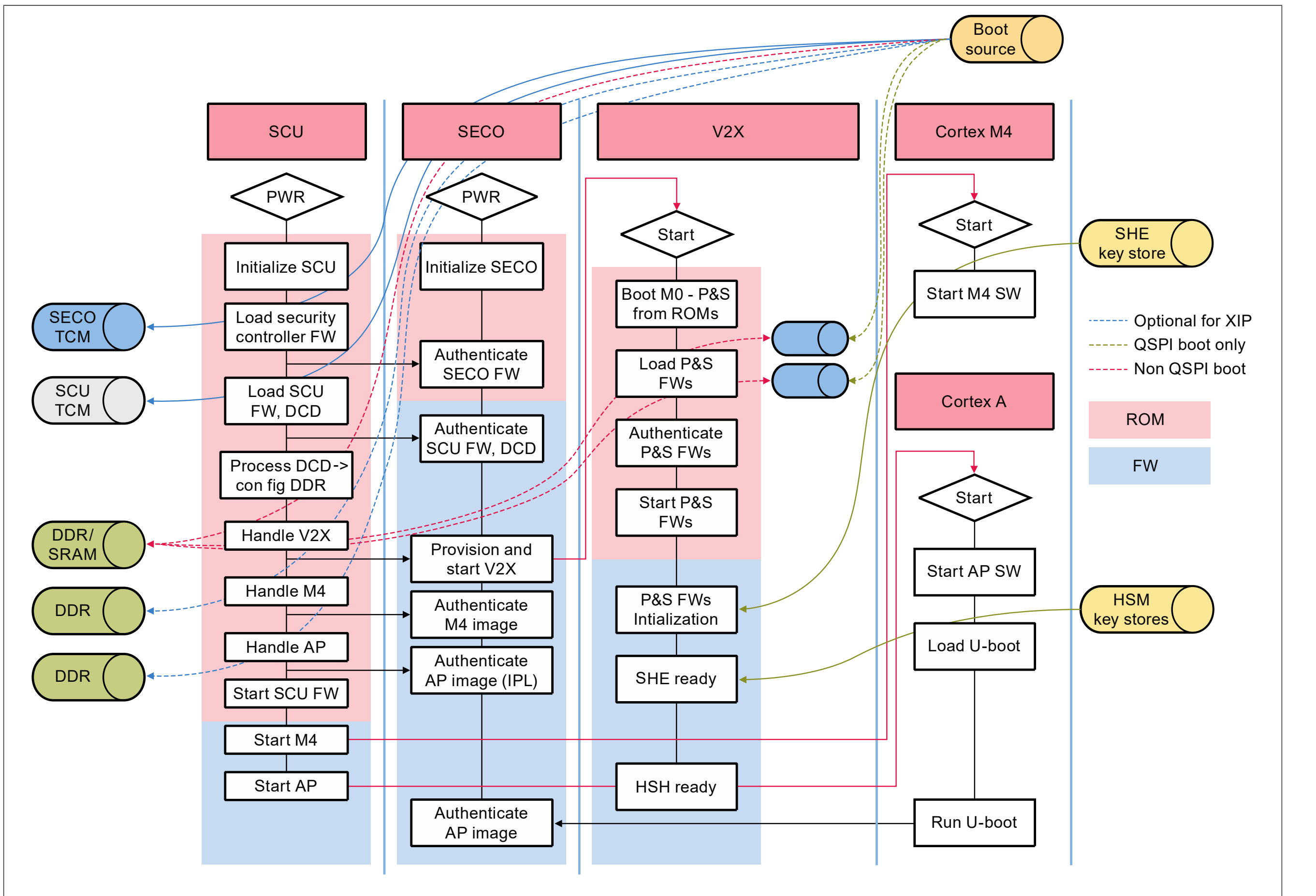

**Figure 3. I.MX8XL boot flow**

### 2.2.2 i.MX 8ULP/9x family

i.MX 8ULP leverages the same boot architecture from i.MX7ULP that relies on separate Boot ROM for Application Domain and Real Time Domain to support different boot modes: Single, dual, and low power boot modes. However with the inclusion of uPower, the i.MX8ULP adds the dedicated uPower ROM to manage the initial power-up sequence.

Figure 4 introduces the dual boot mode as an example to give a high-level boot flow diagram, which comprises other boot modes.

In this boot mode, the M33 core and A35 core boot independently of each other, except for any software-imposed handshaking inserted explicitly to synchronize the process. Under normal operation, each of these two processors boots using the Secure Boot algorithm, performed by the EdgeLock secure enclave. The EdgeLock secure enclave verifies the authenticity of the codes in external memories and then allows them to execute. For other boot modes, refer to *i.MX8ULP Reference Manual*.

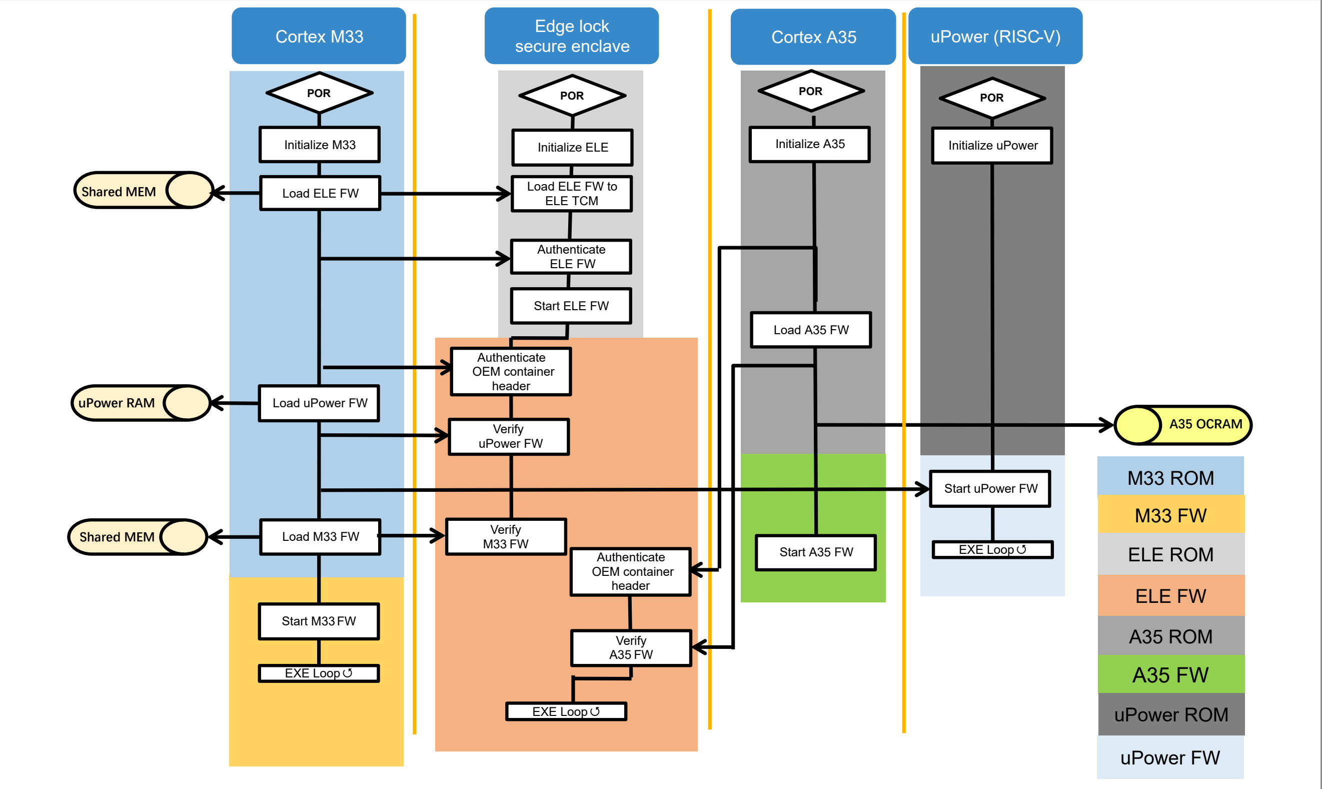

**Figure 4. Dual boot mode**

In this dual boot flow:

- On the Cortex-M33 side:
  1. Cortex-M33 ROM reads the data from the selected boot device and loads the ELE firmware (if available), uPower firmware, Cortex-M33 firmware image and request EdgeLock secure enclave to authenticate these firmware images and these firmwares to run on each core.
  2. In Cortex-M33 firmware, you may further decide whether and when to load DSP Fusion firmware or DSP HiFi4 firmware and when to use the Edge lock secure enclave to authenticate the images and start these firmwares on HiFi4 and Fusion DSP cores.
- On the Cortex-A35 side:
  1. Cortex-A35 ROM reads the data from the selected boot device and loads the first stage bootloader of Cortex-A35 firmware image directly into on-chip RAM in the AP domain and then requests the EdgeLock secure enclave to authenticate the image and start the first bootloader of Cortex-A35.
  2. In this bootloader (for example, SPL), you can further load and authenticate the next stage container and images (like ATF/uboot) in this bootloader.

i.MX93 supports two types of boot: single boot and low power boot.

Figure 5 introduces the typical single boot flow as an example. For more information, refer to *i.MX8ULP Reference Manual*.

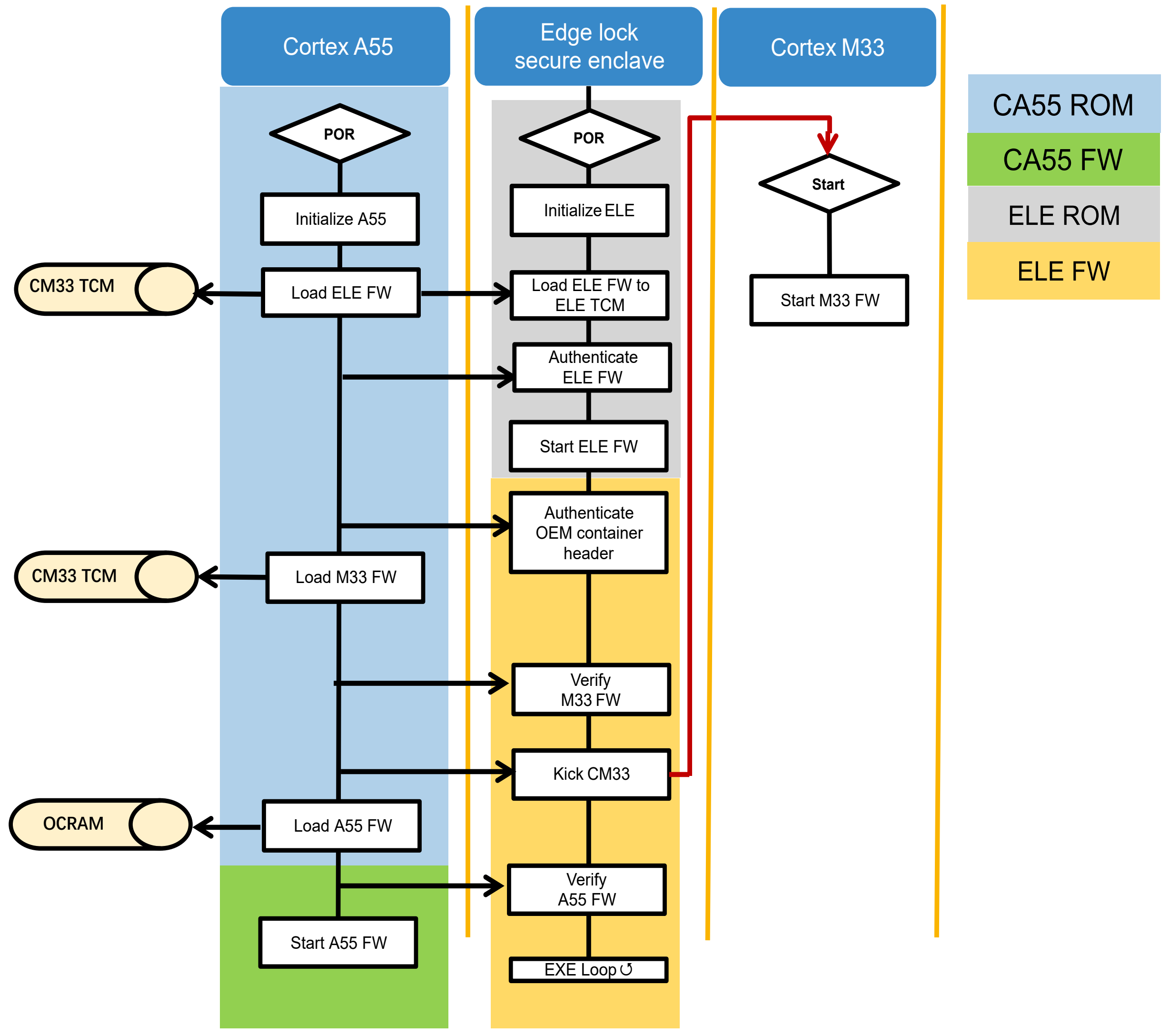

**Figure 5. Single boot flow**

During the single boot:

1. The Cortex-A55 reads the data from the selected boot device. It loads all containers in the selected boot image set one by one, all images in each container (EdgeLock secure enclave firmware, Cortex-M33 firmware, A55 firmware, and so on) one by one, and requests the EdgeLock secure enclave ROM/firmware to authenticate the containers and verify the images one by one.
2. EdgeLock enclave authenticates the ELE firmware and run the firmware after it is verified successfully.
3. After the Cortex-M33 firmware is loaded and verified, the Cortex-A55 ROM fills the INITSVTOR register of Cortex-M33 with the start address and sends a MU message to the EdgeLock secure enclave to release the Cortex-M33 and start from the address.
4. After loading all the containers and images and authentication of all the containers is passed and all the images are verified, the Cortex-A55 ROM jumps to the Cortex-A55 firmware.
5. In Cortex-A55 firmware, you may further decide when to load the next stage Cortex-A55 container/images and when to authenticate them by using the EdgeLock secure enclave.

***Note:*** *ELE firmware is optional in i.MX8ULP and i.MX93.*

# 3 Secure boot implementation

Figure 6 shows the boot image for i.MX8X, composed of different layers.

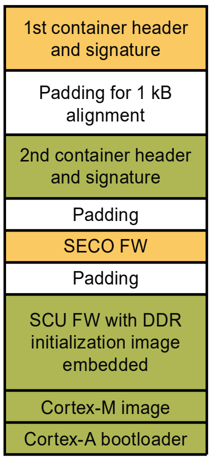
1st container header and signature
Padding for 1 kB alignment
2nd container header and signature
Padding
SECO FW
Padding
SCU FW with DDR initialization image embedded
Cortex-M image
Cortex-A bootloader

**Figure 6. Secure boot image layout**

The boot image contains two containers, one for the SECO firmware (AHAB), and one for the SCU firmware, the ATF, U-Boot and M4 Image. They are preceded by their headers. The first one, containing the SECO firmware image, is padded to 0x400 bytes 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 firmware and the SCU firmware. The Cortex-A or Cortex-M images are optional.*

The boot image layout for i.MX8ULP and i.MX93 are different between each type of boot modes, below figure is for the image layout of Single boot mode. The boot image contains three containers, one for the ELE firmware (AHAB), and one for the uPower firmware/Cortex-M33 image/Cortex-A35 SPL image, and one for the ATF image and uboot image. The first one, containing the ELE firmware image, is padded to 0x400 bytes to fix the start address of the second one, which can contain one or multiple images.

Compared to i.MX8ULP, i.MX93 does not have uPower firmware in its second container.

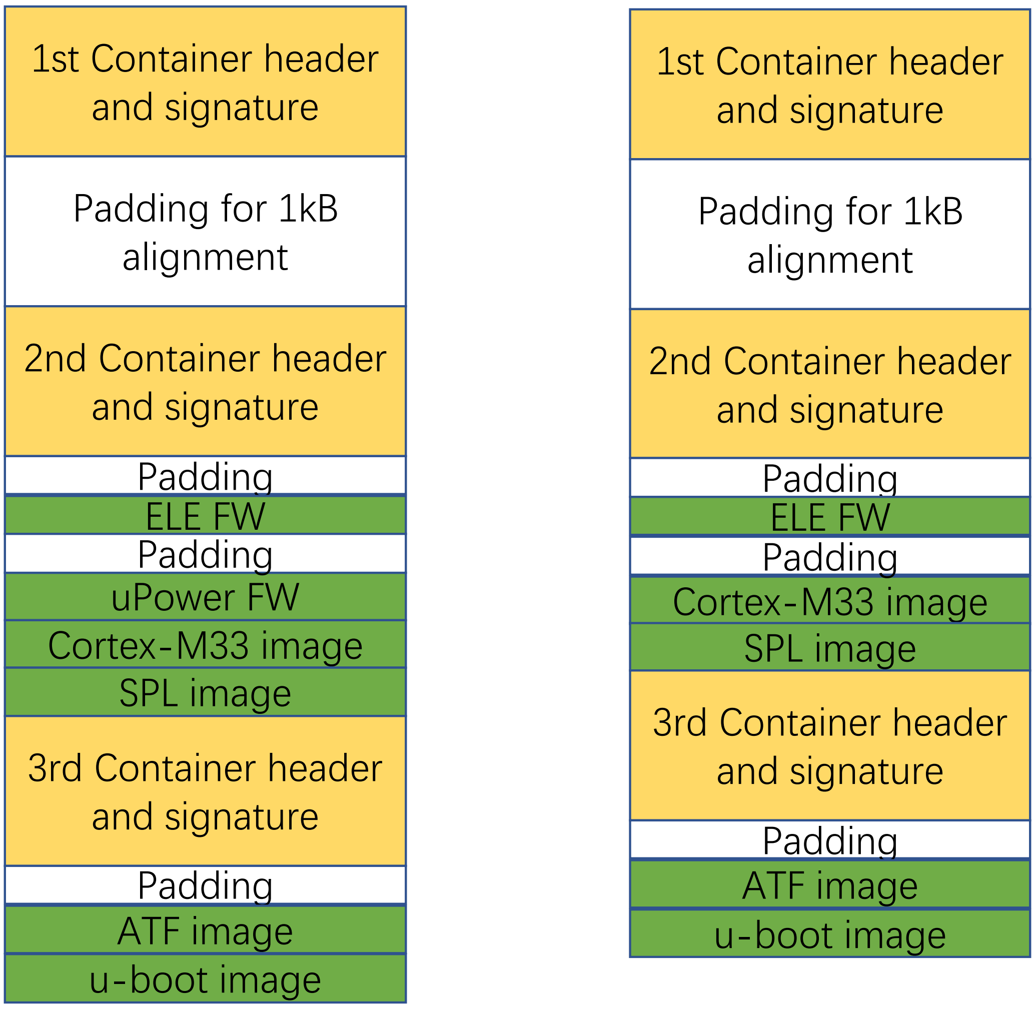
1st Container header and signature
Padding for 1kB alignment
2nd Container header and signature
Padding
ELE FW
Padding
uPower FW
Cortex-M33 image
SPL image
3rd Container header and signature
Padding
ATF image
u-boot image
1st Container header and signature
Padding for 1kB alignment
2nd Container header and signature
Padding
ELE FW
Padding
Cortex-M33 image
SPL image
3rd Container header and signature
Padding
ATF image
u-boot image

**Figure 7. Secure boot layout in single boot mode on i.MX8ULP (left) and i.MX93 (right)**

In contrast with the secure boot process used in the HABv4 architecture, there is no need to sign CSF and insert it into the boot image. Figure 8 shows how CST is responsible to handle the signature block.

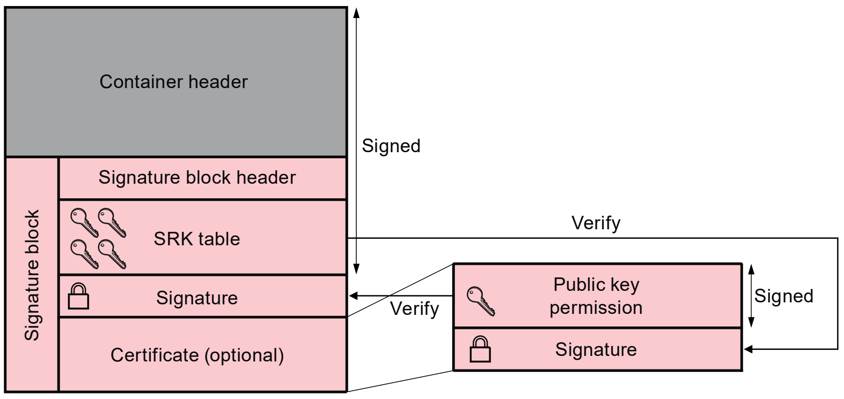

**Figure 8. Signature block**

For the verification process:

1. SRK Table is verified against the NXP/OEM SRK fuses.

2. If SGK is used, SGK is authenticated against the installed NXP/OEM SRK in AHAB keystore.
3. The container signature is verified against the SGK/SRK, depending on whether fast authentication is used.

***Note:*** *For i.MX8ULP and i.MX93, only SRK is supported in the current released firmware.*

## 3.1 Protocols

Following is the list of changes required in the image building process to get a secured boot image:

1. 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.
2. Use CST as described in the *mx8_mx8x_secure_boot.txt/mx8ulp_secure_boot.txt* document which is available in the U-Boot source, updating the offsets of the *doc/imx/ahab/csf_examples/csf_boot_image.txt* file.
3. Generate a PKI tree. For more information, see Generating a PKI tree .
4. Program SRK HASH fuses on the target.
5. Check for the SECO/ELE events. For more information, see Verifying/Decoding SECO events.
6. Close the device in the OEM mode.

## 3.2 Generating a PKI tree

Figure 9 shows an example of the PKI tree.

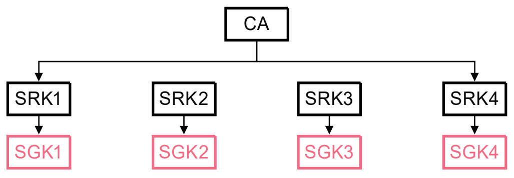

**Figure 9. PKI tree**

CST includes scripts for generating a PKI tree and SRK table.

SRK table includes:

- Super Root Key (SRK): 4 nos of SRKs are used
- SGK: Uses optional image keys (certificate). It can be used to verify the signature, included in the final signed image

To generate the PKI tree, you must run the PKI script and set the CA flags as given below:

1. To generate a P384 ECC PKI tree on CST, 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
```
2. To generate the SGKs for regular authentication, you must set the CA flags while generating the SRK certificates.

Given below is an example to set the CA flag:

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

## 3.3 Examples

Technical step-by-step examples are added to the U-Boot documentation.

The following documents are dedicated to secure boot example for the i.MX 8/8X/8ULP/9x device families:

- mx8_mx8x_secure_boot.txt
- csf_boot_image.txt
- mx8ulp_secure_boot.txt

Additional documents can be found on the following repositories in the form of AHAB introduction and secure boot steps:

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

# 4 SECO/ELE features

This section describes SECO/ELE interfaces useful for secure boot. For more information on functions definitions, refer to *SCU/ELE API Reference Guide*.

***Note:*** *The SCU API document is released along with the SCU firmware porting kit.*

*ELE baseline API Guide* is released as an attachment in SRM. It provides the APIs that the application core can call using the message protocol, which is supported by the ELE message unit.

Refer to the u-boot/Linux kernel ELE related implementation to understand its usage.

## 4.1 Getting chip information

The SCU API document provides an interface to retrieve the SECO chip information. This includes monotonic counter, Lifecycle, and UID.

The messages used to retrieve chip information are given below:

- Monotonic counter, Lifecycle, and UID: `sc_misc_seco_chip_info(…)`.
- Lifecyle and UID of the corresponding EdgeLock secure enclave message: `get info`.

## 4.2 Lifecycle

The Lifecycle defines the security state of the device.

Access to some features is limited by the state. For instance, writing fuses.

For more information about Lifecycle, refer to the *Security Reference Manual*.

The SCU API provides the following interfaces to manage Lifecycle:

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

The corresponding EdgeLock secure enclave message, uses the messages `Forward Lifecycle update` and `Return Lifecycle update`. For changing the Lifecycle to `Partial Field Return`, it requires a message signed by the OEM SRK.

## 4.3 Verifying or decoding SECO events

A new SECO service, which is available to retrieve any singular event that has occurred since the firmware as started is `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 are stored.

About the returned event format, see Figure 10.

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

**Figure 10. Returned event**

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

For information about the indicator field, see Table 2. Indicators .

### 4.3.1 Indicators

Table 1 shows the indicators that are used for verifying the SECO events.

Table 1. Indicators

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

- `0x0087FA00`: This container image is signed with the wrong key.
  Key does not match with the OTP that SRK hashes fused on the target.
- `0x0087EE00`: This container image is not signed.

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

For EdgeLock secure enclave, you can retrieve the AHAB event by using the `Get Events` message.

Table 2 shows the layout of each AHAB event received from edge lock secure enclave.

Table 2. Event layout

| | | | |
|---|---|---|---|
| IPC ID (bits 24 to 31) | Command ID (bits 16 to 23) | Indication (bits 15-8) | Status (bits 7-0) |

Command ID or Indication is same as SECO for the AHAB-related message. Bits 24 to 31 indicates the IPC identifier (1 is for RTD MU, 2 is for APD MU). The status field is updated to 0xD6 for `success` and 0x29 for `failure`.

For example, in the error “0x0287EED6”:

From APD MU:

- Status is fail in the AHAB container authentication command
- Status is success, as the device is in the OEM open lifecycle mode

## 4.4 Containers authentication

You can authenticate more than one container at different levels. For instance, the boot image generated by SPL targets.

Figure 11 shows three containers for authentication.

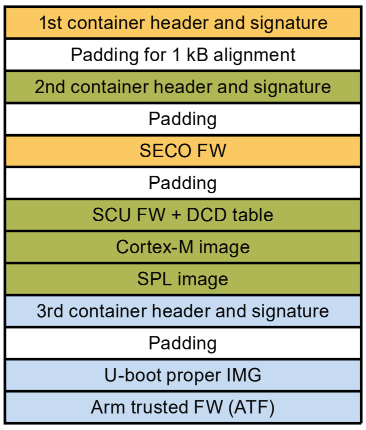
1st container header and signature
Padding for 1 kB alignment
2nd container header and signature
Padding
SECO FW
Padding
SCU FW + DCD table
Cortex-M image
SPL image
3rd container header and signature
Padding
U-boot proper IMG
Arm trusted FW (ATF)

**Figure 11. Containers authentication**

The first and second containers are authenticated at the SCU ROM level. Whereas, the third container is authenticated at the SPL level.

To authenticate the EdgeLock secure enclave device, perform the following steps:

1. Replace the SECO firmware with EdgeLock secure enclave firmware.
2. Remove the SCU firmware.
3. Add a new uPower firmware (i.MX8ULP only), but do not change the container or the image authentication process.

### 4.4.1 Authentication with code signing tool

The first method to authenticate OS containers is similar to the signing method of secure boot which is described in Secure boot implementation.

Following are the steps to sign an OS container:

1. Generate the OS container image.
2. Update the CSF example provided by U-Boot documentation with the offset values returned by the `make` command.
3. Execute the CST with this file to sign the OS image.
   The new generated file is the OS signed image.

For a step-by-step guide to authenticate the OS container, see *sign_os_cntr.txt* document of U-Boot documentation.

### 4.4.2 SECO authentication service using SCU API

You can authenticate an image using 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 data) for some commands.

***Note:*** *It is recommended to load the container header in CAAM Secure Memory (SM) instead of OCRAM. The SECO does not have direct access to OCRAM and OCRAM does not have the same access permission control which CAAM SM has.*

### 4.4.3 SRK revocation

You can revoke an SRK. For instance, if the SRK1 public key must be replaced to SRK2.

If the SRK revoke bit is set, the SECO/ELE firmware sets the fuse.

But, it is possible only after successful authentication of the header that contains the SRK revocation command and the receipt of the `COMMIT` command with the corresponding argument.

Revocation can be performed with the Code Signing Tool and the SCU API/ELE message.

***Note:*** *An SRK used by the current container cannot be revoked.*

As only one SRK can be selected at boot time through an install SRK CSF command, you must ensure that the CSF is updated accordingly.

Also, you can revoke only the first three SRKs by burning the corresponding bit in the `SRK_REVOKE [2:0]` eFuse field.

### 4.4.4 ELE authentication service using ELE message

To authenticate the OEM container header, use `OEM container authenticate` message.

To verify each image in the container, use `Verify image` message.

After completion of the container authentication, you must release the container.

To release the container, call `release container` message.

## 4.5 Known limitations

The following list provides some known limitations:

1. In i.MX 8 QXP B0, the container header size is limited to 4 kB
   The maximum number of images that are supported depends on the cryptographic algorithm. For more information, see Maximum number of images supported.
2. The SECO firmware ignores image integrity failure if detected in the open mode.
3. 1K SPL limitation for the container header size in the release *L4.14.78_1.0.0GA*.

# 5 Supported images

Table 3 lists the maximum number of supported images.

## 5.1 Maximum number of images supported

**Table 3. Supported images**

| Image | 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 |

## 6 References

The following list provides additional documents that you may need to refer to for more information:

- i.MX 8/8x/8ULP/9x Reference Manual and Security Reference Manual
- *AHAB CST* user guide is available in the Code Signing Tool package (IMX_CST_TOOL)
- U-Boot technical guides and examples are available in the location `doc/imx/ahab` of the U-Boot project on the release branch `lf_v2022.04`

## 7 Note about the source code in the document

Example code shown in this document has the following copyright and BSD-3-Clause license:

## 8 Revision history

Table 4 summarizes the revisions to this document.

**Table 4. Revision history**

| Revision number | Release date | Description |
|---|---|---|
| 1 | 14 August 2023 | Added the features which support secure boot on AHAB supported devices in the following sections:<br>• Section 2<br>• Section 4 |
| 0 | 08 May 2019 | Initial public release. |

# 9 Legal information

## 9.1 Definitions

## 9.2 Disclaimers

## 9.3 Trademarks

## Contents

**Date of release: 14 August 2023**
**Document identifier: AN12312**