AN13994

i.MX Encrypted Boot on AHAB-Enabled Devices

Rev. 1 — 21 August 2023 Application note

Document Information

Information	Content
Keywords	AN13994, AHAB, i.MX, encryption, SECO, secure boot, crypto, encrypted boot, ELE
Abstract	This document provides an overview of the mechanisms and approach behind encrypted boot on AHAB-enabled devices. The encrypted boot provides an extra layer of security to the boot process to protect bootloader data from unauthorized access.

1 Introduction

This document provides an overview of the mechanisms and approach behind encrypted boot on AHAB-enabled devices. Encrypted boot provides an extra layer of security to the boot process to protect bootloader data from unauthorized access. Devices such as i.MX 8, i.MX 8X, and i.MX 9 are AHAB-enabled with NXP security IP and firmware (FW).

This application note describes the encrypted boot feature available in AHAB-supported devices, such as the i.MX family of application processors, which can be tailored to the user requirements.

To deploy secure boot on AHAB-enabled i.MX processors, refer to *Secure Boot on AHAB Supported Devices* (document AN12312).

For MPU step-by-step technical guides, refer to the U-Boot project documentation in Section 5.

1.1 Acronyms

Table 1 lists the acronyms in this document.

Table 1. Acronyms

Acronyms	Meaning	Remarks
AES	Advanced encryption standard	—
AHAB	Advanced high assurance boot	A software (SW) library executed in internal ROM on the NXP processor at boot time, which authenticates software in external memory by verifying digital signatures. This document is strictly limited to processors 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 algorithms, with a random number generator and runtime integrity checker.
CMS	Cryptographic message syntax	A general format for data that can have cryptography applied to it, such as digital signatures and digital envelopes. AHAB uses the CMS as a container holding PKCS#1 signatures.
CSF	Command sequence file	A binary data structure used by CST and interpreted by the 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 an early boot stage.
DEK	Data encryption key	The AES key used to encrypt/decrypt the boot image.
ELE	EdgeLock secure enclave	EdgeLock secure enclave offers a more full-featured security subsystem with enhanced security features. ELE combines platform security and cryptographic services into a centralized subsystem, which allows for an additional, well-defined logical security perimeter. ELE supports access to external security components.
ELE FW	EdgeLock secure enclave firmware	EdgeLock secure enclave firmware provides functionality for ELE.
IVT	Image vector table	A vector table that consists of pointers to the image start address, signature address, and so on.
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.

Table 1. Acronyms...*continued*

Acronyms	Meaning	Remarks
RSA	Rivest–Shamir–Adleman	Public key cryptography algorithm developed by Rivest, Shamir, and Adleman.
S400	S400	Synonymous with EdgeLock enclave
SA	Signature authority	The holder of the private key used to sign software components.
SCFW	SCU firmware	The SCU firmware provides functionality for the SCU.
SCU	System controller unit	The system controller unit runs on the Cortex-M processor, which is the first processor to boot the chip. The SCU is responsible for the following: • Booting the system • Interfacing with external PMIC • Managing power and resource partitioning, among other responsibilities
SDP	Serial download protocol, also called UART/USB serial download mode	SDP allows code provisioning through UART or USB during production and development phases.
SECO	Security controller	The security controller subsystem runs on the Cortex-M processor. This subsystem is responsible for the following: • Booting the system • Providing security services to other cores • Managing ROM and RAM, among other responsibilities
SECO FW	Security controller firmware	The security controller firmware manages the security controller subsystem and provides access to the security feature, including the following: • Hardware security module (HSM) • Secure RAM • Secure storage • Debug permissions • Key access
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. For this document, unless explicitly noted, SRK refers to the public key.

1.2 Purpose

The security suite in the i.MX family of application processors provides adequate features to establish a chain of trust for high-assurance computing while also meeting the trust computing requirements of embedded solutions, such as firmware data assurance.

Securing a hardware platform requires examination of its hardware components and verification of the authenticity and integrity of the critical code that controls the platform. These security checks are executed after the boot code resets in the Advanced High-Assurance Boot (AHAB) component of the on-chip ROM and the firmware running on the chip security enclave, security controller (SECO), or EdgeLock secure enclave (ELE).

The ROM boot process verifies the authenticity of the bootloader script, which resides either in the flash or in the external memory. After the validation, the operating system and data are loaded into the external memory.

Considering that the bootloader script resides in the external memory, the information about the boot process can leak to the adverse parties.

The encrypted boot feature adds an extra security operation to the boot-loading sequence. The encrypted boot uses cryptographic techniques to obscure the bootloader data (it can be extended to the entire image), so that unauthorized users cannot view or use it. This mechanism protects and conceals the bootloader code residing in the flash (or external) memory.

1.3 Audience

This document is intended for the audiences who:

- Need an explanation of the procedure for encrypting a bootloader
- Need help with designing encrypted software images to be used with an AHAB-enabled processor

1.4 Scope

This application note describes the encrypted boot feature added to the AHAB and the Code Signing Tool (CST).

The encrypted boot is featured only on the following application processors from the i.MX family:

- i.MX 8: i.MX 8QuadPlus, i.MX 8QuadMax
- i.MX 8X: i.MX 8SoloXLite, i.MX 8DualX, i.MX 8DualXPlus, i.MX 8DualXLite, i.MX 8QuadXPlus
- i.MX 8ULP
- i.MX 93

Note:

This document only demonstrates the encrypted boot solution on the i.MX 8, i.MX 8X, i.MX 8ULP, and i.MX 93 processors. The internals of the encrypted boot require an extensive knowledge of cryptography and security trust models, which is out of the scope of this document. For more details about the internals, see the appropriate security reference manual.

Secure boot is not included in this document. For further details, refer to Secure Boot on AHAB Supported Devices (document AN12312).

2 Overview

This section provides the signing procedure of image binaries required to understand the use cases and processes described later in this document.

2.1 AHAB library

The AHAB library is a ROM component and contains security mechanisms, such as authentication, encryption, and decryption operations. The secure boot sequence allows the ROM code to use the AHAB library to enforce cryptographic checks at each booting stage. The AHAB library gains this capability through the SECO firmware and EdgeLock firmware. This process provides the foundations for a secure environment by asserting the integrity of the software images to be executed. These cryptographic checks prevent any unauthorized software from running on the target. The same library calls can be accessed at later boot stages to extend the trust chain past the ROM-booting stage. These cryptographic checks also ensure confidentiality through the encrypted boot sequence.

The AHAB library can use the onboard hardware accelerators, such as CAAM on i.MX 8 devices, to improve the boot performance and access the OTP master keys.

2.2 Encrypted boot sequence

The security solution for an embedded system starts as early as the boot ROM execution. The secure boot extends the ROM execution flow for AHAB-enabled processors, while the encrypted boot extends it even further. Figure 1, Figure 2, and Figure 3 show examples of extended boot flows.

When configured for encrypted boot operation, the boot ROM on these devices does not allow unauthenticated or modified code to execute. Any failure or security violation generated at this stage halts the boot process.

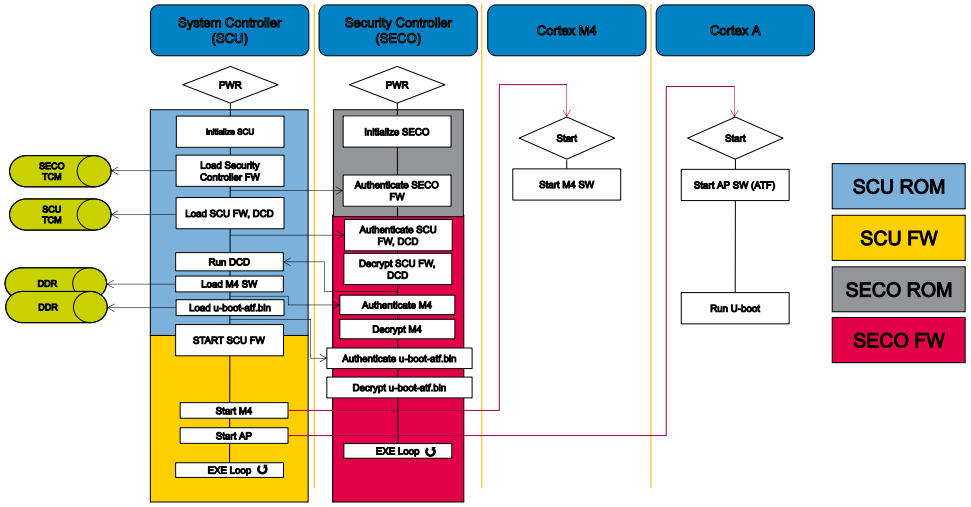

Figure 1. Encrypted boot flow on i.MX 8/8X device

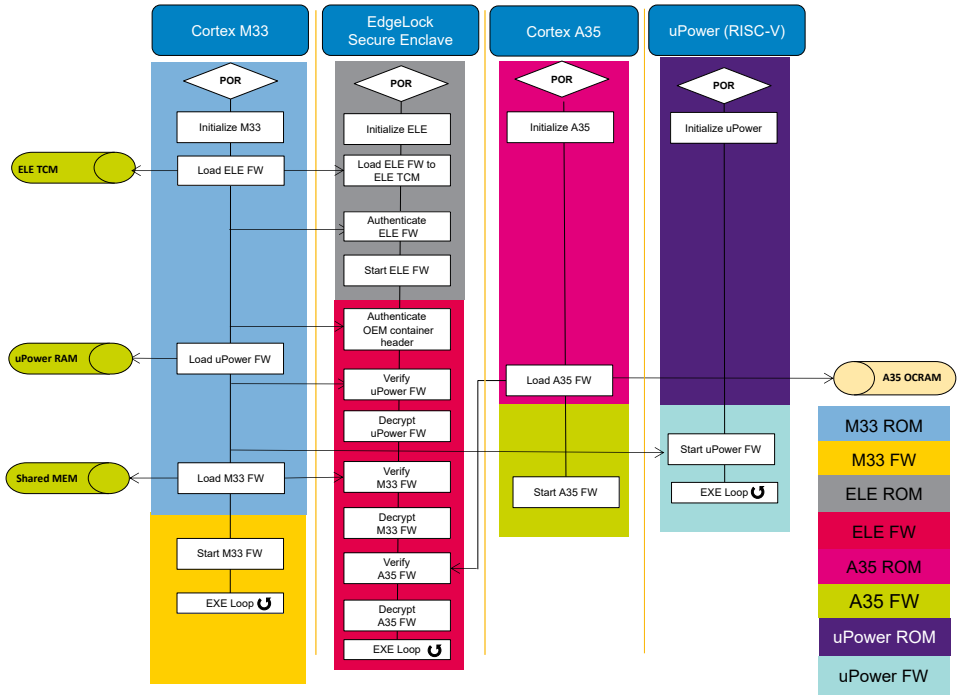

Figure 2. Encrypted boot flow on i.MX 8ULP device

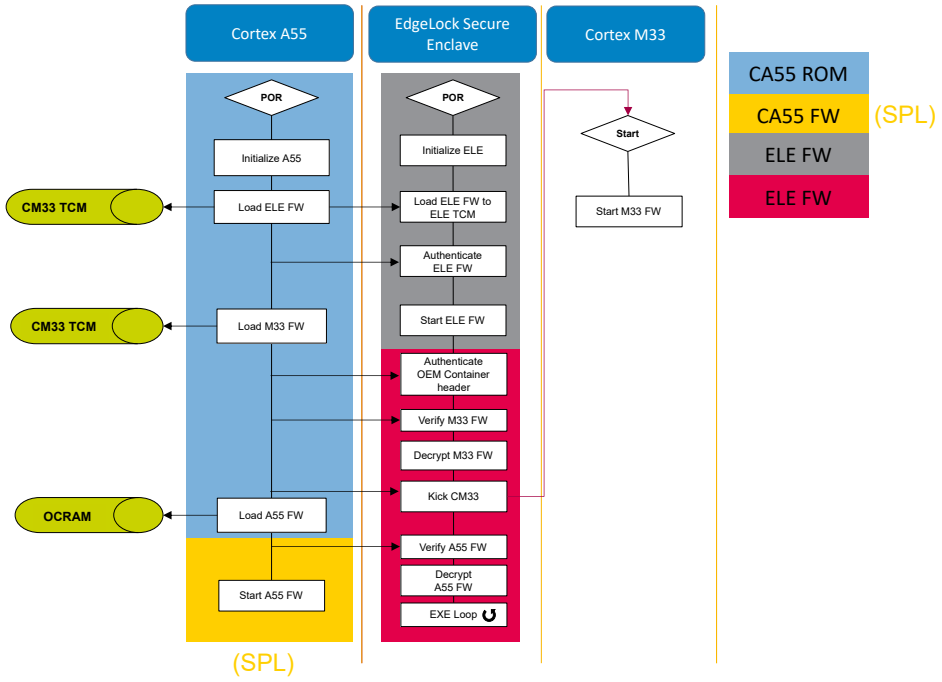

Figure 3. Encrypted boot flow on i.MX 93 device

3 Encrypted boot implementation

This section includes the implementation of the encrypted boot.

3.1 Overview

In simple words, an encrypted boot is a secure-boot version of an encrypted bootloader. Therefore, this protocol can be divided into two protection mechanisms as follows:

- The digital signature, which authenticates the source of the binary image.
- The bootloader code encryption, which bestows confidentiality to the bootloader data.

Both mechanisms are used on separate parts of the bootloader image. The container headers and signature blocks are signed, but must remain in plaintext, therefore, not encrypted.

3.2 Requirements

The ROM requires an image container from the program image. For more information, see the respective i.MX application processor reference manual. The example container in Figure 4 shows the components of the container format as follows:

- Container header: The container header contains container information and image address, signed by the NXP CST. Also, the header contains flags, such as one indicating to the secure enclave (SECO or ELE) that the image data is encrypted and must be decrypted.
- Signature block: The signature block contains security-related information such as the super root key (SRK) hash table and the data encryption key (DEK) blob, of which only the header and SRK table are signed.
- Images: The images are signed and encrypted inside the container. CST and AHAB use the DEK blob to perform the encryption and decryption of the container images. The DEK blob is used as a security layer to wrap and store the device-unique DEK off-chip.

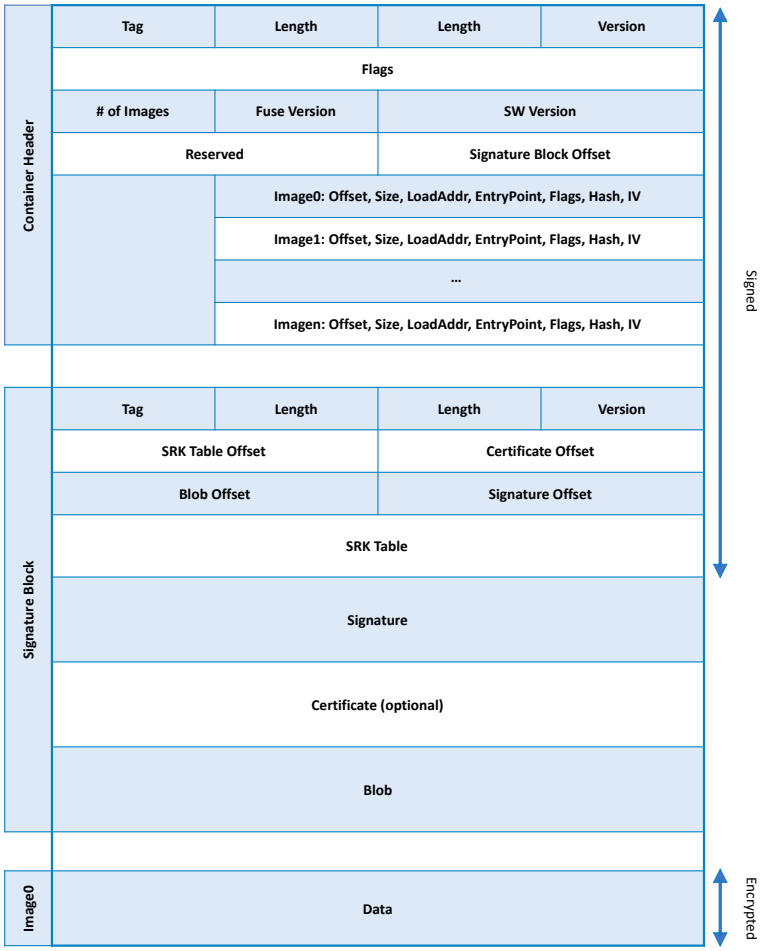

Figure 4. Container format

For more information on the memory layout of the encrypted boot image, see Figure 7 and Figure 8.

3.3 Protocol

The encrypted boot protocol is an advanced use of the secure boot protocol. The encrypted boot is a combination of digital signature verification and decryption of the bootloader image code. These steps are a simplification of the encrypted boot protocol, explained as follows:

1. **Encrypting and signing of a boot image**:
   This step involves signing a binary image. This procedure follows the instructions from *Secure Boot on AHAB Supported Devices* (document AN12312).
   Note: *The authenticated boot image is distinct from the encrypted boot image, even though the digital signature authentication follows the same procedure for both. This is because an encrypted boot image is an extension of the authenticated boot image.*
   CST generates a unique set of keys that are used to create the digital signature. During the process, the SRK table and certificates are also generated. The same set of keys is used for the encrypted boot since the keys and certificates are unique to the target processor.
   Using these SRKs, the boot data is encrypted and a digital signature of the whole boot data image including the boot data structure (or vice versa) is generated. The boot data structures, the container header and the signature block, do not contain any confidential data and are used by the ROM. Therefore, these data

structures remain in plaintext and are included in the digital signature. This allows the boot ROM to access the necessary pointers to initialize the data structures and modules required by AHAB.
The encryption part in this step is straightforward. By adding a new command to the command sequence file (CSF), the CST continues to generate the digital signature and produces a DEK to encrypt the boot image. Figure 5 shows the process of the boot image encryption.

2. **DEK blob generation and encrypted boot image assembling**:
This step generates a secure blob using on-chip private keys. This security measure ensures that this specific chip is the only chip that can encrypt or decrypt the blob. This secure blob is generated through the ELE firmware or SECO firmware, which manages access to the private keys required for the secure DEK blob. NXP provides specialized tools for DEK blob generation through the NXP U-Boot port. The AHAB container format requires a DEK blob to be generated for each non-security firmware container.
Finally, the DEK blob is inserted into the encrypted image.

Step 1 and Step 2 are described in more detail in the further sections of this document. For a step-by-step guide of Step 1, see *Secure Boot on AHAB Supported Devices* (document AN12312) and the U-Boot project documentation in Section 5.

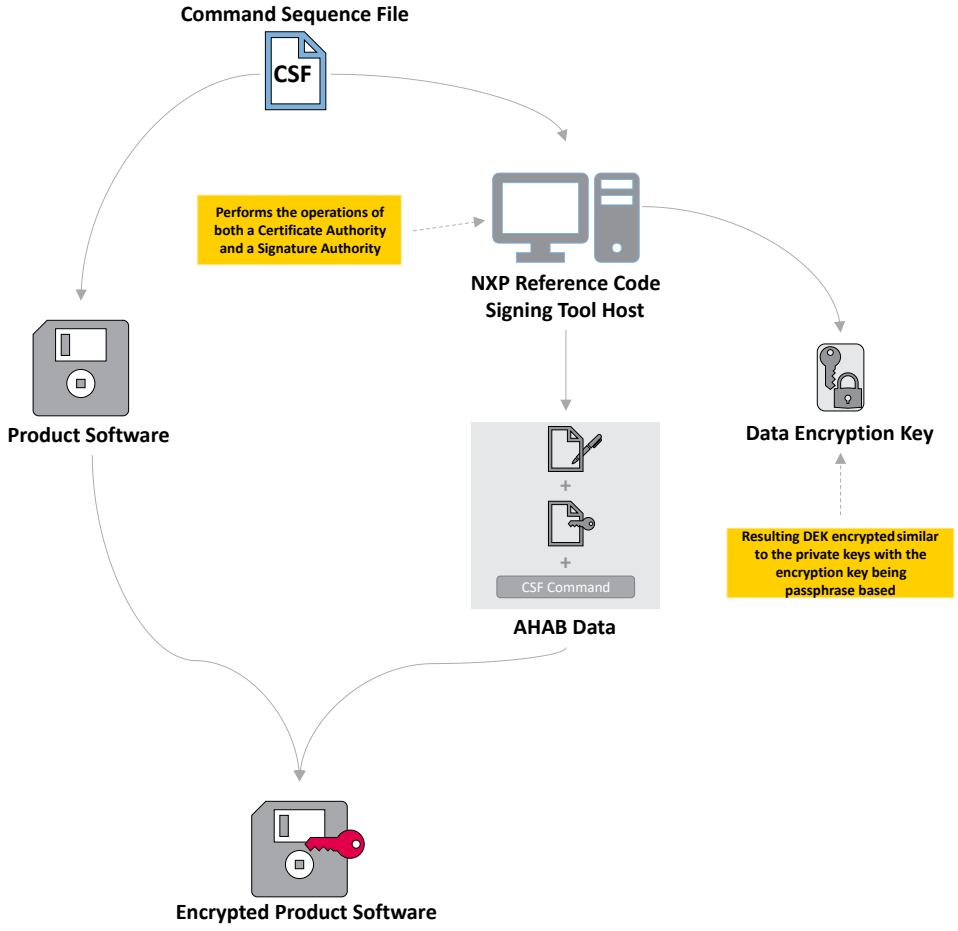

Figure 5. Encrypting image using CST

3.4 Data encryption key

The CST automatically generates a random DEK when the `[Install Secret Key]` command is present in the CSF and the CST is in encryption mode. After the digital signature is authenticated, the ROM decrypts the encrypted bootloader using the DEK. Unlike the RSA public-key system used for authentication, the encrypted

boot uses symmetric-key algorithms, especially multiple key-length variants of the advanced encryption standard (AES) algorithm. Therefore, DEK confidentiality is especially important and the DEK must be protected (including manufacturing) and especially in the end product. The following sections address both cases.

3.4.1 Data encryption key handling

On SECO devices, such as the i.MX 8/8X family, a unique OTP Master Key (OTPMK) is used to encrypt and wrap the DEK in a blob. For ELE devices, the OTPMK is replaced with the device unique key (DUK). These keys are protected by the hardware and only accessed through security firmware. NXP provides a freely available U-Boot port with the capability of using these private keys through SECO firmware and ELE firmware API calls.

To decrypt the DEK blob, use the same processor that has previously encrypted it. To add further to the security of the DEK, the blob is decapsulated and decrypted inside a secure memory partition. For SECO devices, it is also highly recommended to increment the PRIBLOB setting in CAAM to make this blob undecipherable by any software that runs after the encrypted boot. This way the DEK blob remains private to the instance of the encrypted boot and is secure from any attacks that attempt the extraction of the DEK. For more information about the PRIBLOB, see the security reference manual (SRM) of the respective NXP chipset.

3.4.2 Protection layer for manufacturing

The CST asks for a public key to encrypt the resulting DEK and provides a protection layer for the DEK in the host machine. This layer of security prevents any mishandling of the DEK off-chip, for example, when transferring it from one provider to another. To generate the key pair using the OpenSSL command-line utility, follow the steps given below. The generated self-signed certificate is then used by the CST to encrypt the DEK, which is then safely transferred through a non-secure channel of communication.

1. Generate a 2048-bit RSA key pair using the command as follows:

```
openssl genrsa -out ./dek_rsa_key.pem 2048
```

2. Generate a certificate-signing request with the key pair using the command as follows:

```
openssl req -new -key ./dek_rsa_key.pem -out ./dek_rsa_key.csr
```

3. Generate a self-signed certificate using the command as follows:

```
openssl x509 -req -days 365 -in ./dek_rsa_key.csr -signkey ./dek_rsa_key.pem -out ./
dek_rsa_key_crt.pem
```

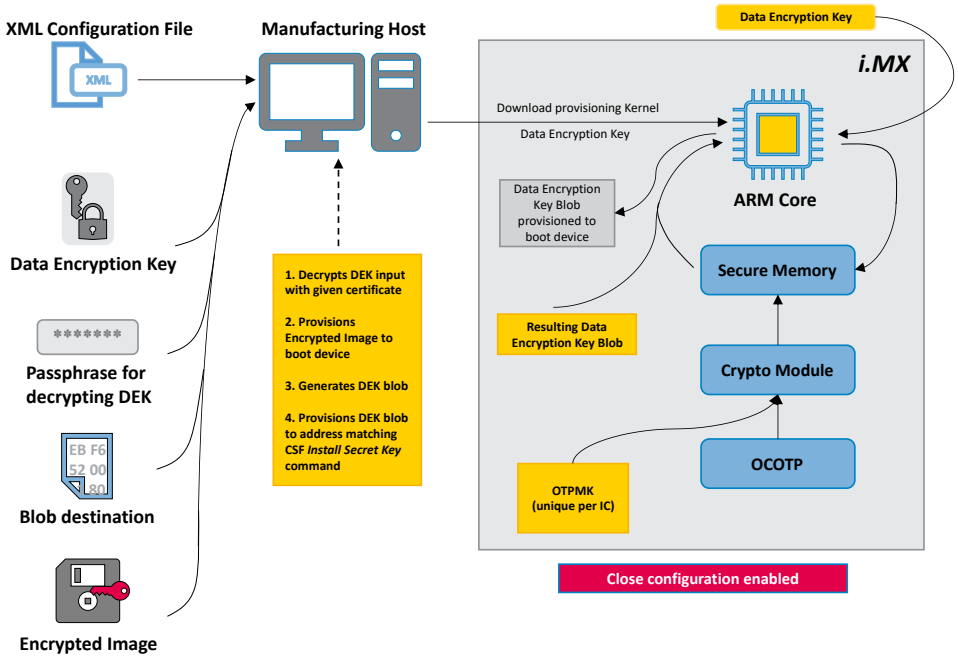

Figure 6. DEK provisioning for manufacturing

The previous security layer anticipates the OEM using a contract manufacturer. A good security practice is never to share passwords. Therefore, the contract manufacturer intends to provide the OEM with a public key. The OEM encrypts the DEK and sends it to the manufacturer, which can decrypt the DEK with the corresponding private key. If a contract manufacturer is not used, the CA key (generated for the authenticating the boot by the CST) can be used for this purpose.

Since the DEK generated by the CST is in plaintext, the OEM can operate on it. If the public key generated above encrypts the DEK, then the DEK can be obtained using the following OpenSSL command:

```
openssl cms -decrypt -in dek_in.bin -inform DER -out dek_out.dec -binary -inkey private_key.pem
```

Where:

- `dek_in.bin` is the DEK protected with the public key `dek_rsa_key.pem`.
- `dek_out.bin` is the plaintext DEK.
- `private_key.pem` is the private key corresponding to the public key given to the CST.

3.5 DEK blob

The AHAB stores DEK blobs in a secret key blob data structure, as shown in Figure 7. The secret key blob data structure is inherited, but slightly modified, from *High Assurance Boot version 4 (HABv4)*.

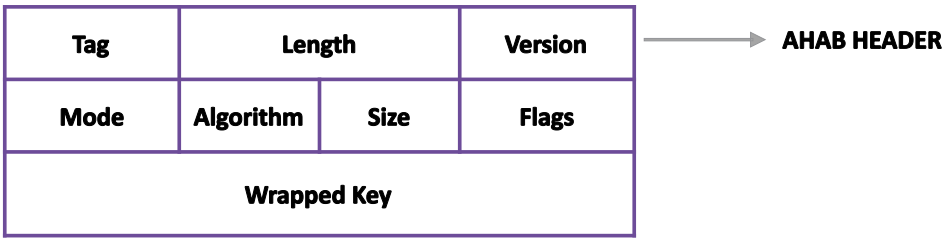

Figure 7. Key blob data structure

The description of the parameters in Figure 7 is as follows:

- *Tag*
  - Constant value specifying the AHAB data structure
  - For the required *Wrapped Key data structure*, this field is equal to `(0x81)`
- *Length*
  - Length of the structure is aligned to an 8-bit value
- *Version*
  - For *AHAB*, this field is equal to `0x81`
- *Mode*
  - Constant value specifying the key cipher or Hash mode
  - For *Counter with CBC-MAC* mode, this field is equal to `(0x66)`
- *Algorithm*
  - Constant value specifying the key cipher algorithm used
  - For the *AES algorithm*, this field is equal to `(0x55)`
- *Size*
  - Unwrapped size of key in bytes
- *Flags*
  - Secret key flags
  - For *KEK Flag*, this field is equal to `(0x80)`
- *Wrapped Key*
  - Encrypted blob key

The DEK blob utility builds this structure from a given DEK. The AHAB supports a set of encryption algorithms, but the encrypted boot protocol expects AES. The key length is a variable and possible lengths are 128-bit, 192-bit, or 256-bit.

For more information about the HAB secret key blob data structure, see the *High-Assurance Boot Version 4 Application Programming Interface Reference Manual*. For specifics on the i.MX 8ULP device key blob, see the *ELE Architecture document*.

4 Encrypted U-Boot example

The following sections detail the process to achieve the encrypted boot on an AHAB-compatible device. For MPU devices, step-by-step technical guides are available in the U-Boot project documentation in Section 5.

The default memory layout of the NXP U-Boot port can be modified to meet the encrypted boot requirements. This layout modification is shown in Figure 8 for i.MX 8/8X devices, Figure 9 for i.MX 8ULP devices, and Figure 10 for i.MX 93 devices. This layout is similar to any other U-Boot port, with the addition of security-related data in the signature block. For memory layout of the container header and signature block, see Figure 4.

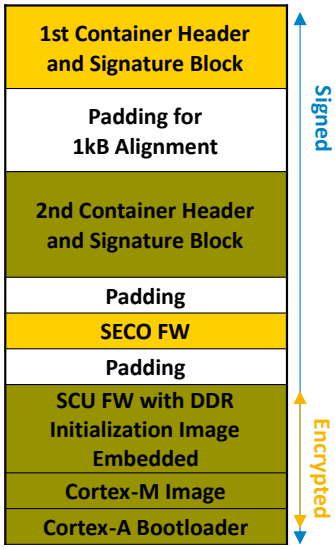

Figure 8. Memory layout of the encrypted boot container for the i.MX 8/8X devices

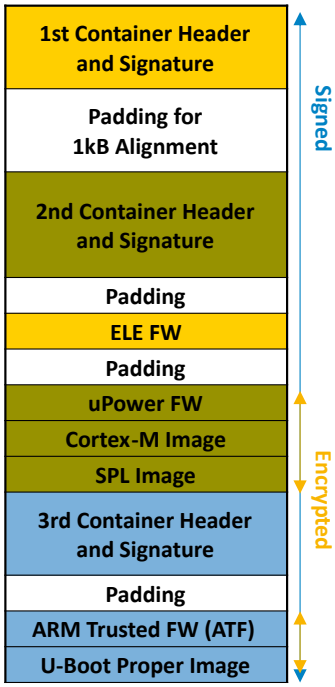

Figure 9. Memory layout of the encrypted boot container for the i.MX 8ULP device

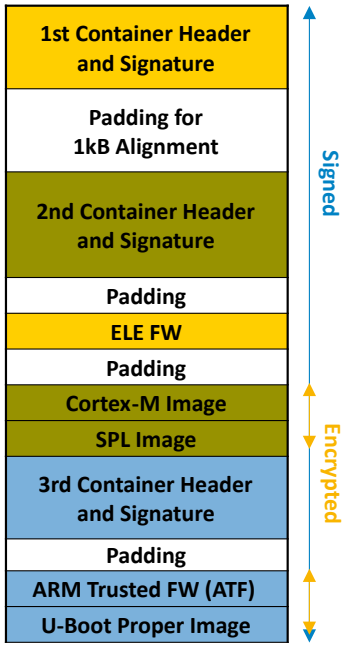

Figure 10. Memory layout of the encrypted boot container for the i.MX 93 device

Note: *Data structures read by the boot ROM (container header and signature block) must remain in plaintext but must be covered in a digital signature.*

4.1 Assumptions

When designing the U-Boot image as an encrypted boot solution, there are three assumptions, which accelerate and simplify the construction process. These three assumptions are as follows:

- The U-Boot image can be built for multiple board configurations. This image is build using the associated U-Boot configuration file and consulting the reference manual of the particular chip to write the fuses.
- The user is familiar with the secure configuration for the U-Boot and can properly sign and boot a U-Boot image.
- An individual party constructs the encrypted image and there is no need to worry about provisioning the DEK.

4.2 Requirements

The requirements are as follows:

- The OpenSSL library (≥ 1.1.1t) is installed on the host machine:

```
sudo apt-get install libssl-dev
```

- The CST is in the encryption mode.
  Note: *CST versions < 3.0.0 are not in the encryption mode by default. This feature must be enabled before encrypting the bootloader image. The performance of the CST can be affected due to its dependency on the host entropy. For more details, see the CST User Guide and the U-Boot project documentation in Section 5.*
- The target device is in a secure mode.
- The U-Boot image with the secure boot and encrypted boot support (with the blob generation tool) is enabled.
- Signed U-Boot image.
- U-Boot image with digital signature attached.

4.3 Implementation

To construct an encrypted U-Boot image, many different implementations are possible. The correct implementation depends on the requirements of the solution. The example considered in this document provides the foundation principles. With the help of the step-by-step guide in the U-Boot project documentation, these principles can be tailored for different needs.

4.3.1 CSF for encryption

The CSF contains several commands, refer IMX_CST_TOOL_NEW. The CSF for encrypted boot is similar to the secure-boot CSF, with the addition of the following command:

`[Install Secret Key]`

CSF examples can be found in the U-Boot project documentation in Section 5.

4.3.1.1 Install Secret Key command

The DEK must be installed in the key storage for the CAAM to use it. The `[Install Secret Key]` command specifies the DEK properties and where to install it in the key storage.

Note: *The key storage used for the DEK is independent from the SRK storage. Therefore, there is no conflict with duplicated key indexes. Observe the indexes for the secret key storage after they are overwritten.*

CAAM uses the AES-CCM algorithm for the encrypted boot. Therefore, the key length specified in this section defines which variant to use. An example of using AES-192 is as follows:

```
[Install Secret Key]
Key = "dek.bin"
Key Length = 192
#Key Identifier = 0x1234CAFE
#Image Indexes = 0xFFFFFFE
```

The `[Install Secret Key]` command contains the parameters detailed in Table 2.

Table 2. Install Secret Key command parameters

Parameter	Definitions
Key	Path to the DEK file generated by the CST
Key length	The length of the key for the AES algorithm. Possible lengths are 128-bit, 192-bit, and 256-bit.
Key identifier	32-bit identifier that must match the value provided during the blob generation (Optional: default value is 0)
Image indexes	List of images that are encrypted (Optional: all images are encrypted by default)

4.3.1.2 Encrypting and signing the image

The `[Install Secret Key]` command is added to encrypt the boot image and create a DEK.

For step-by-step technical guides and CSF examples, see the U-Boot project documentation in Section 5.

4.3.1.2.1 CSF to sign, encrypt, and create DEK

A digital signature must contain the whole boot data image, which consists of the boot data structures and the encrypted boot image. As a result, the encrypted image cannot be swapped with a malicious image even if the

DEK is compromised as the attacker must first match the digital signature generated from the signed encrypted image.

As mentioned earlier, each non-security container requires its own DEK for encryption. In this encrypted U-Boot example, a different number of containers exist for both i.MX 8/8X devices and the i.MX 8ULP.

For i.MX 8/8X images, only the single CSF is required for signing, encryption, and DEK generation. For i.MX 8ULP and i.MX 93 images, two CSFs are required: one for the second container and one for the third container.

The CSFs for this step are shown in the U-Boot project documentation in Section 5.

After DEK generation, the DEK blob must be generated with an encrypted memory U-Boot.

4.3.2 U-Boot encrypted memory layout

To generate the layout for the encrypted and signed U-Boot example, follow the steps from the U-Boot project documentation in Section 5.

This encrypted memory layout is required for DEK blob generation. After using U-Boot secure memory to generate the blob, the blob must be inserted into the container signature blocks of the respective containers. This process is also documented in the U-Boot project documentation in Section 5.

4.4 Protecting the DEK blob after encrypted boot

DEK blobs are a part of the encrypted boot process used to derive the DEK required to decrypt and boot the image. When designing the encrypted boot using the DEK blob, it is necessary to inhibit any modification or replacement of the DEK blob with a counterfeit to prevent the execution of malicious code. For SECO devices, the PRIBLOB setting in the CAAM allows the secure boot software to have its own private blobs that any other user code cannot decapsulate or encapsulate, including any software running in the trusted mode.

When deploying the encrypted boot environment, the PRIBLOB setting must be advanced in the CAAM security configuration register (SCFGR). Generate the DEK blob initially with the default setting PRIBLOB=01 and set the runtime software PRIBLOB=11. With PRIBLOB=11, the newly created blobs are not compatible with the blobs required to decrypt an encrypted boot image. When the HAB later executes the command to decrypt the DEK, an incompatible DEK blob is detected and causes an error. A substitute encrypted boot image is not decrypted or executed. This ensures cryptographic separation of private blob types during the boot process and thereafter, avoiding any modification or replacement of DEK blobs.

5 References

Table 3 lists the documents and resources that can be referred for more information.

Table 3. References

Documents/resources	Link/how to access
U-Boot technical guides	uboot-imx
Code Signing Tool	IMX_CST_TOOL_NEW
Secure Boot on AHAB Supported Devices	AN12312
i.MX 8, i.MX 8X, i.MX 8ULP, and i.MX93 family processors reference and security reference manuals	Contact NXP field application engineer or sales representative.
High-Assurance Boot Version 4 Application Programming Interface Reference Manual	Contact NXP field application engineer or sales representative.
ELE Architecture Document	Contact NXP field application engineer or sales representative.

6 Note about the source code in the document

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

Copyright 2023 NXP Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

1. Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.
2. Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials must be provided with the distribution.
3. Neither the name of the copyright holder nor the names of its contributors may be used to endorse or promote products derived from this software without specific prior written permission.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT HOLDER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE

7 Revision history

Table 4 summarizes revisions to this document.

Table 4. Revision history

Revision number	Release date	Description
1	21 August 2023	Initial public release

8 Legal information

8.1 Definitions

Draft — A draft status on a document indicates that the content is still under internal review and subject to formal approval, which may result in modifications or additions. NXP Semiconductors does not give any representations or warranties as to the accuracy or completeness of information included in a draft version of a document and shall have no liability for the consequences of use of such information.

8.2 Disclaimers

Limited warranty and liability — Information in this document is believed to be accurate and reliable. However, NXP Semiconductors does not give any representations or warranties, expressed or implied, as to the accuracy or completeness of such information and shall have no liability for the consequences of use of such information. NXP Semiconductors takes no responsibility for the content in this document if provided by an information source outside of NXP Semiconductors.

In no event shall NXP Semiconductors be liable for any indirect, incidental, punitive, special or consequential damages (including - without limitation - lost profits, lost savings, business interruption, costs related to the removal or replacement of any products or rework charges) whether or not such damages are based on tort (including negligence), warranty, breach of contract or any other legal theory.

Notwithstanding any damages that customer might incur for any reason whatsoever, NXP Semiconductors' aggregate and cumulative liability towards customer for the products described herein shall be limited in accordance with the Terms and conditions of commercial sale of NXP Semiconductors.

Right to make changes — NXP Semiconductors reserves the right to make changes to information published in this document, including without limitation specifications and product descriptions, at any time and without notice. This document supersedes and replaces all information supplied prior to the publication hereof.

Suitability for use — NXP Semiconductors products are not designed, authorized or warranted to be suitable for use in life support, life-critical or safety-critical systems or equipment, nor in applications where failure or malfunction of an NXP Semiconductors product can reasonably be expected to result in personal injury, death or severe property or environmental damage. NXP Semiconductors and its suppliers accept no liability for inclusion and/or use of NXP Semiconductors products in such equipment or applications and therefore such inclusion and/or use is at the customer's own risk.

Applications — Applications that are described herein for any of these products are for illustrative purposes only. NXP Semiconductors makes no representation or warranty that such applications will be suitable for the specified use without further testing or modification.

Customers are responsible for the design and operation of their applications and products using NXP Semiconductors products, and NXP Semiconductors accepts no liability for any assistance with applications or customer product design. It is customer's sole responsibility to determine whether the NXP Semiconductors product is suitable and fit for the customer's applications and products planned, as well as for the planned application and use of customer's third party customer(s). Customers should provide appropriate design and operating safeguards to minimize the risks associated with their applications and products.

NXP Semiconductors does not accept any liability related to any default, damage, costs or problem which is based on any weakness or default in the customer's applications or products, or the application or use by customer's third party customer(s). Customer is responsible for doing all necessary testing for the customer's applications and products using NXP Semiconductors products in order to avoid a default of the applications and the products or of the application or use by customer's third party customer(s). NXP does not accept any liability in this respect.

Terms and conditions of commercial sale — NXP Semiconductors products are sold subject to the general terms and conditions of commercial sale, as published at http://www.nxp.com/profile/terms, unless otherwise agreed in a valid written individual agreement. In case an individual agreement is concluded only the terms and conditions of the respective agreement shall apply. NXP Semiconductors hereby expressly objects to applying the customer's general terms and conditions with regard to the purchase of NXP Semiconductors products by customer.

Export control — This document as well as the item(s) described herein may be subject to export control regulations. Export might require a prior authorization from competent authorities.

Suitability for use in non-automotive qualified products — Unless this data sheet expressly states that this specific NXP Semiconductors product is automotive qualified, the product is not suitable for automotive use. It is neither qualified nor tested in accordance with automotive testing or application requirements. NXP Semiconductors accepts no liability for inclusion and/or use of non-automotive qualified products in automotive equipment or applications.

In the event that customer uses the product for design-in and use in automotive applications to automotive specifications and standards, customer (a) shall use the product without NXP Semiconductors' warranty of the product for such automotive applications, use and specifications, and (b) whenever customer uses the product for automotive applications beyond NXP Semiconductors' specifications such use shall be solely at customer's own risk, and (c) customer fully indemnifies NXP Semiconductors for any liability, damages or failed product claims resulting from customer design and use of the product for automotive applications beyond NXP Semiconductors' standard warranty and NXP Semiconductors' product specifications.

Translations — A non-English (translated) version of a document, including the legal information in that document, is for reference only. The English version shall prevail in case of any discrepancy between the translated and English versions.

Security — Customer understands that all NXP products may be subject to unidentified vulnerabilities or may support established security standards or specifications with known limitations. Customer is responsible for the design and operation of its applications and products throughout their lifecycles to reduce the effect of these vulnerabilities on customer's applications and products. Customer's responsibility also extends to other open and/or proprietary technologies supported by NXP products for use in customer's applications. NXP accepts no liability for any vulnerability. Customer should regularly check security updates from NXP and follow up appropriately.

Customer shall select products with security features that best meet rules, regulations, and standards of the intended application and make the ultimate design decisions regarding its products and is solely responsible for compliance with all legal, regulatory, and security related requirements concerning its products, regardless of any information or support that may be provided by NXP.

NXP has a Product Security Incident Response Team (PSIRT) (reachable at PSIRT@nxp.com) that manages the investigation, reporting, and solution release to security vulnerabilities of NXP products.

NXP B.V. - NXP B.V. is not an operating company and it does not distribute or sell products.

8.3 Trademarks

Notice: All referenced brands, product names, service names, and trademarks are the property of their respective owners.

NXP — wordmark and logo are trademarks of NXP B.V.

EdgeLock — is a trademark of NXP B.V.

i.MX — is a trademark of NXP B.V.

Contents

1 Introduction 2
1.1 Acronyms 2
1.2 Purpose 3
1.3 Audience 4
1.4 Scope 4
2 Overview 4
2.1 AHAB library 4
2.2 Encrypted boot sequence 4
3 Encrypted boot implementation 6
3.1 Overview 6
3.2 Requirements 6
3.3 Protocol 7
3.4 Data encryption key 8
3.4.1 Data encryption key handling 9
3.4.2 Protection layer for manufacturing 9
3.5 DEK blob 10
4 Encrypted U-Boot example 11
4.1 Assumptions 13
4.2 Requirements 13
4.3 Implementation 14
4.3.1 CSF for encryption 14
4.3.1.1 Install Secret Key command 14
4.3.1.2 Encrypting and signing the image 14
4.3.2 U-Boot encrypted memory layout 15
4.4 Protecting the DEK blob after encrypted boot 15
5 References 15
6 Note about the source code in the document 16
7 Revision history 16
8 Legal information 17

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

Date of release: 21 August 2023
Document identifier: AN13994