

# How to use i.MXRT Security Boot

## 1. Introduction

NXP i.MX RT provide the High Assurance Boot (HAB) feature which can make the hardware have some mechanism to ensure that the software can be trusted. So the OEM can utilize it to make their product reject any system image which is not authorized to run. The image can also be encrypted to protect the important content. The encrypted boot is always a combination of an authenticated boot and an encrypted boot is an advanced usage of an authenticated boot. Apart from this RT also supports eXecute In Place (XIP) on the Serial NOR flash device.

This document describes the authenticate boot, encrypted boot and encrypt XIP on i.MX RT applications.

## Contents

1.	Introduction.....	1
2.	Basic Knowledge of Security Boot .....	2
2.1.	Definitions, Acronyms, and Abbreviations .....	2
2.2.	The authentic boot .....	2
2.3.	The encrypted boot .....	4
2.4.	Encrypt XIP .....	7
2.5.	Encrypt XIP flow .....	10
2.6.	Decrypt XIP flow .....	11
2.7.	Typical use cases .....	11
3.	How to Use .....	12
3.1.	Authenticate using elftosb.....	12
3.2.	Encrypt boot using elftosb .....	19
3.3.	Encrypt XIP using elftosb .....	25
4.	Download Image Using mfgtool.....	28
4.1.	Connect to the device.....	28
4.2.	Program image during development .....	28
4.3.	Program fuse .....	29
4.4.	Program image for production .....	31
5.	Revision History .....	31
6.	Reference Material .....	32

## 2. Basic Knowledge of Security Boot

### 2.1. Definitions, Acronyms, and Abbreviations

- AES: Advanced Encryption Standard.
- 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.
- DCP: Data co-processor, an accelerator for AES encryption and SHA hashing algorithms.
- HAB: High Assurance Boot, a software library executed in internal ROM on the Freescale processor at boot time which, among other things, authenticates software in external memory by verifying digital signatures in accordance with a CSF.
- OTP: One-Time Programmable. OTP hardware includes masked ROM, and electrically programmable fuses (eFuses).
- XIP: Execute-In-Place, refers to a software image that is executed directly from its non-volatile storage location rather than first being copied to volatile memory.

### 2.2. The authentic boot

#### 2.2.1. Brief description

We use the asymmetric encryption to implement the HAB feature. The OEM use a utility provided by NXP to generate private key and corresponding public key pairs. Then the private key is used to encrypt the image which OEM want to release. This encryption generates a unique identifier for the image which is called a certificate. The public key is also attached to the image. Before applying the new system, the public key is used to decrypt the certificate.

A comparison is performed to check whether the certificate and the image match. The image is considered "trusted" only if a match occurs. Otherwise the image is deemed as "unsafe" and will not be permitted to be loaded and ran. This process is called authentication.

A hacker can only have access to the public key. Per the property of asymmetric encryption, the private key, which the OEM uses, cannot be deduced from that. Without the private key, the hacker cannot attach a valid certificate for his malicious system image. The HAB will reject it in a very early stage. The OEM will burn the digest (hash) of the public key to the eFuses of RT chip. Once burned, it cannot be modified. This prevents the possibility that the hacker can use another pair of private and public keys to cheat.

*Figure 1* to *Figure 3* give briefer descriptions.

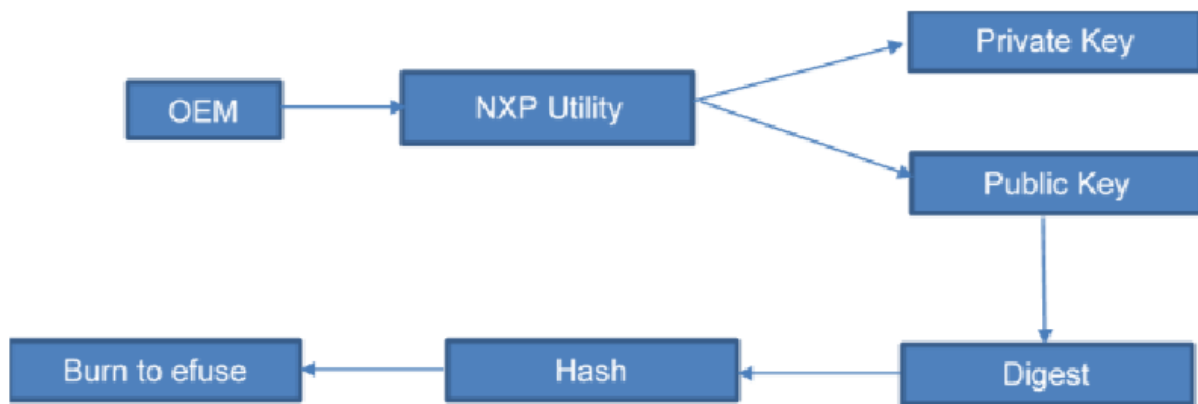


Figure 1. Generation of public/private key

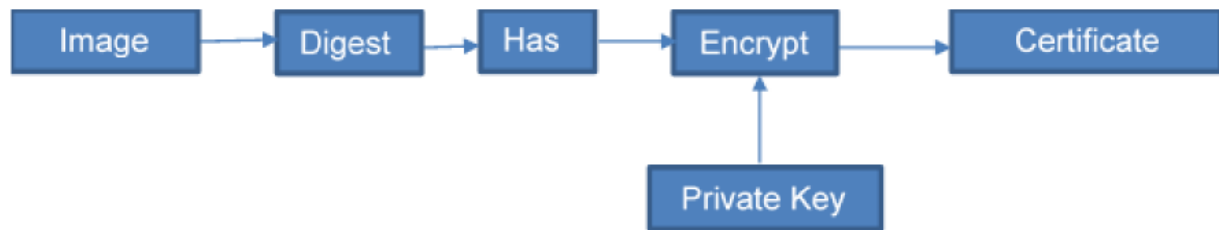


Figure 2. Certificate generation

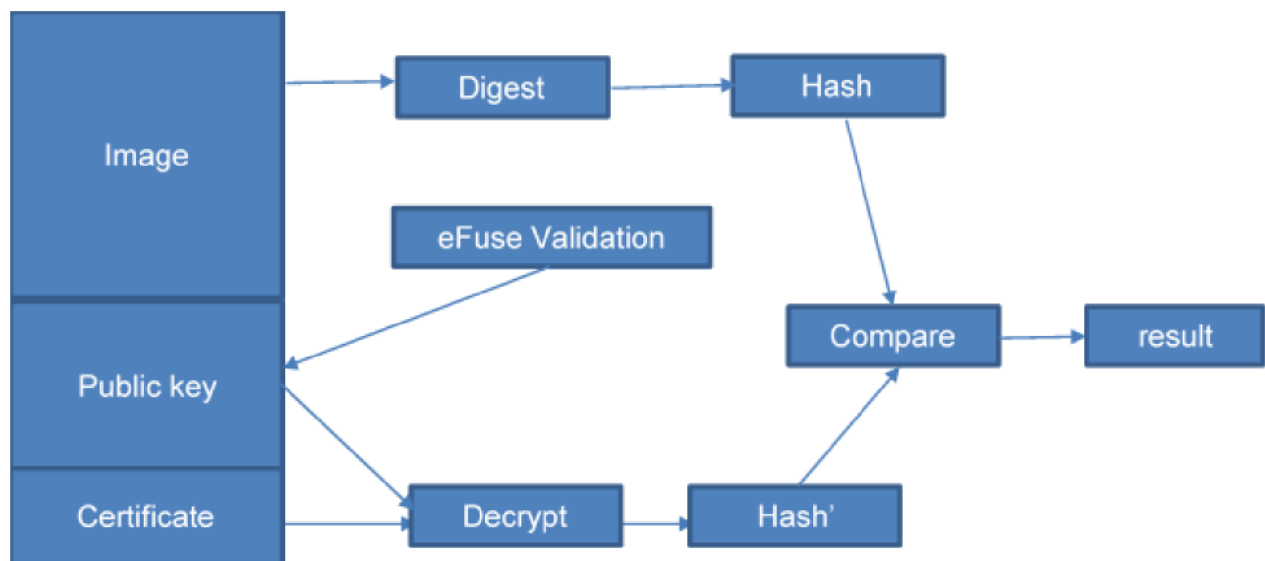


Figure 3. Authenticate process (run time)

## 2.2.2. Image layout

*Figure 4* is the image layout of HAB. A Elftosb tool is used to generate the HAB data, which includes commands that the HAB uses to verify the image, public key and certificate data. The usage of elftosb tool will be introduced in *How to Use*. The following is the detailed information about HAB data.

The HAB data contains:

- CSF commands that are interpreted by the HAB library in the i.MX ROM
- Digital signatures covering the image
- Digital certificates holding the public key the HAB library in ROM uses to verify the image signatures.

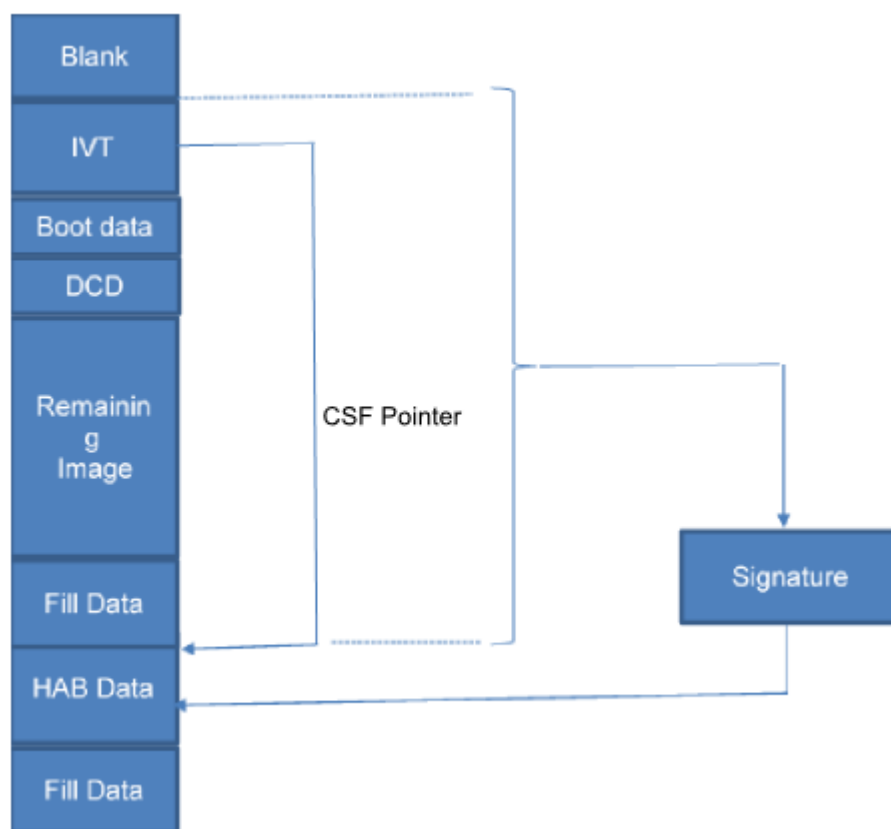


Figure 4. HAB image layout

## 2.3. The encrypted boot

### 2.3.1. Brief description

The encrypted boot is similar to the authenticate boot, with the difference that the boot image is encrypted instead of signed. It is always a combination of digital signature verification with encrypted boot.

The data structures used for boot are authenticated with a same PKI tree that is used for pure authenticated boot. These structures do not contain any secret, and therefore remain in plain text, allowing the ROM code to simply access the different pointers necessary for the boot process.

Unlike RSA public keys used for digital signatures, encrypted boot makes use of a secret key called Data Encryption Key (DEK). The actual image content is encrypted by that key. The DEK is sensitive and has to be protected when in a product and when manipulated for manufacturing. And we can use the OTP Master Key (OTPMK) to make a blob to protect the DEK. The on-chip unique OTPMK is used to create a cryptographic blob of that key (DEK blob). This step has to be executed with software running on the target, because the OTPMK can only be accessed internally by the DCP or BEE and on a chip set to close configuration. This means that the blob is unique per chip. At boot time the blob is encapsulated in such a way that only the DCP on i.MX RT will have access to the DEK. *Figure 5* shows a brief introduction of the Encrypted boot.

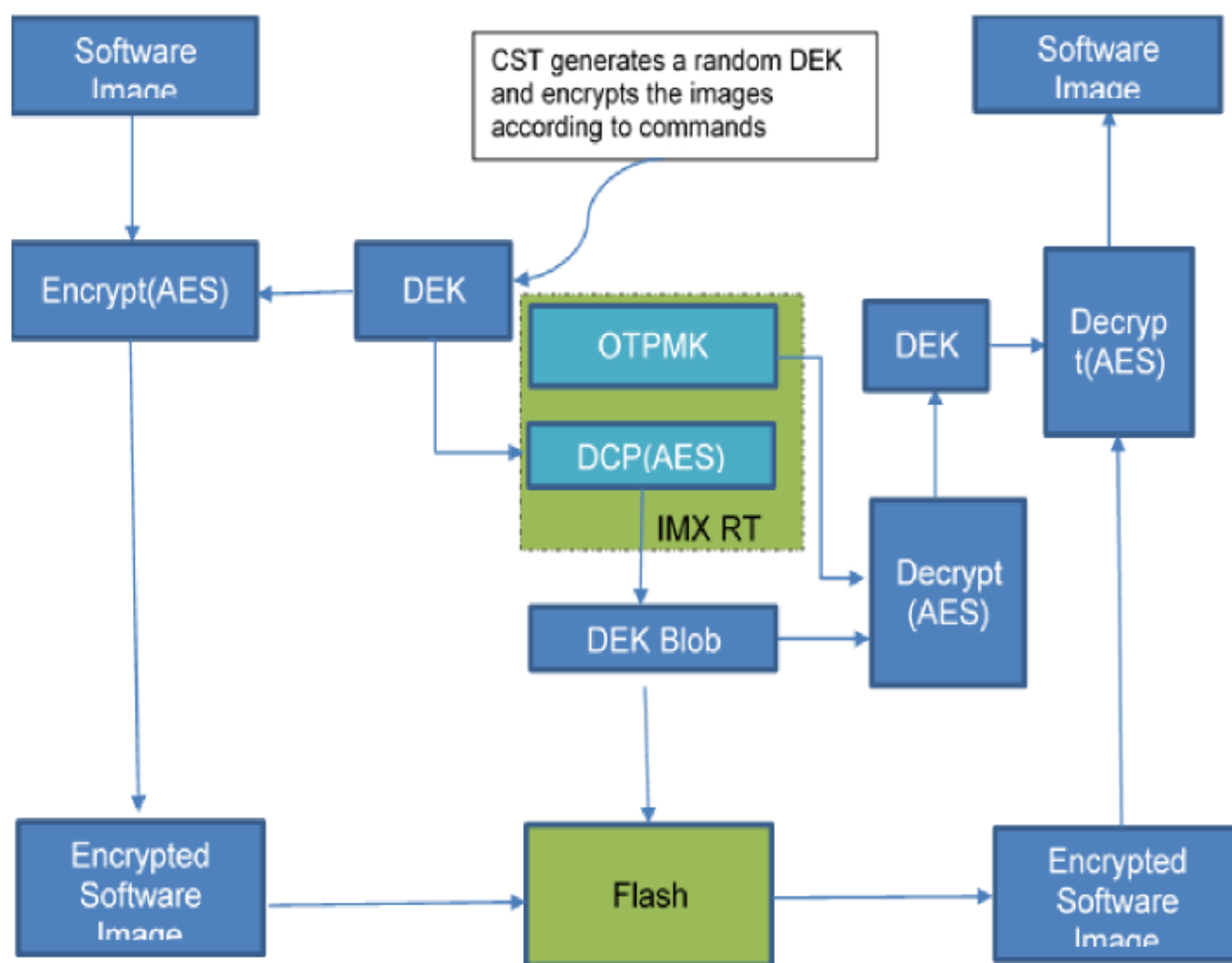


Figure 5. Introduction of the encrypted boot

### 2.3.2. Image layout

A secure boot with encryption requires some data components to be added to an image. This includes keys, certificates, signatures, IC configuration data and CSF data. When performing a secure boot on i.MX Boot ROM and HAB require the following data components to be defined in the image:

- Image Vector Table
- Boot data structure: A simple structure indicating where to load the boot image and specifying the size of the boot image.
- Device Configuration Data (DCD)
- Command Sequence File (CSF) and associated data
- Data Encryption Key blob (DEK blob): A Data Encryption Key that is used to decrypt the boot image, and that is wrapped into an encrypted container thanks to the unique per chip OTP Master Key.

This is an example of encrypted image. The image and padding data is encrypted by the AES key. When the system start to boot, we can decrypt the DEK blob by using the OPT Master key. Then we can get the AES key and decrypt the encrypted image.

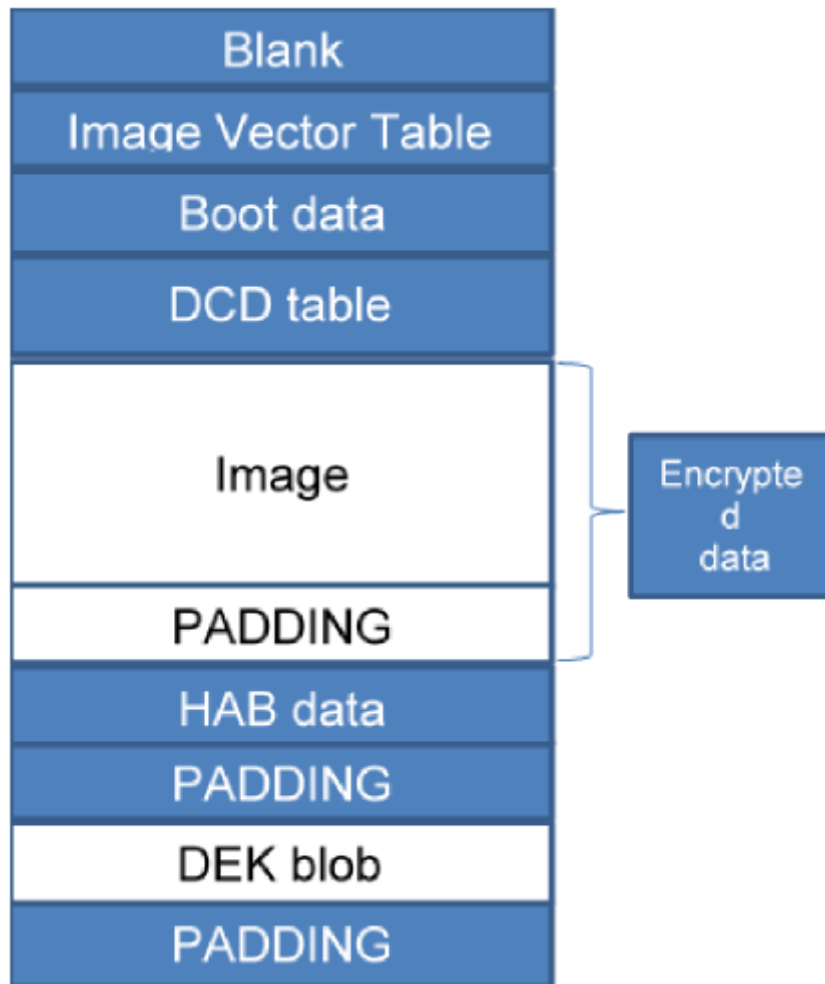


Figure 6. Layout of the encrypted image

## 2.4. Encrypt XIP

### 2.4.1. Brief description

i.MX RT Boot ROM supports XIP on the Serial NOR flash device directly with On-the-fly decryption feature (using AES-CTR-128 or AES-ECB-128) powered by BEE controller.

Before doing Encrypted XIP, the Boot ROM needs to set the BEE controller correctly, the configurable parameters are organized as Protection Region Descriptor Block (PRDB), the entire PRDB is encrypted using AES-CBC-128 mode with the AES KEY and IV in a Key Info Block (KIB). The KIB is encrypted as Encrypted KIB (EKIB) using the AES key provisioned in eFUSE.

Entire or partial SW image is encrypted with customized private secret key (PVK). The secret key is then burned to on chip eFuse block (OCOTP) and limited to be on-the-fly QSPI decryption engine (BEE) access only. During boot after ROM code initialize the BEE block, the encrypted and

unencrypted data stored in QSPI flash can then be accessed on-the-fly. Each chip could use a unique secret key to encrypt the SW image, so each image can only boot on the chip with the right secret key, “image copy” can be prevented.

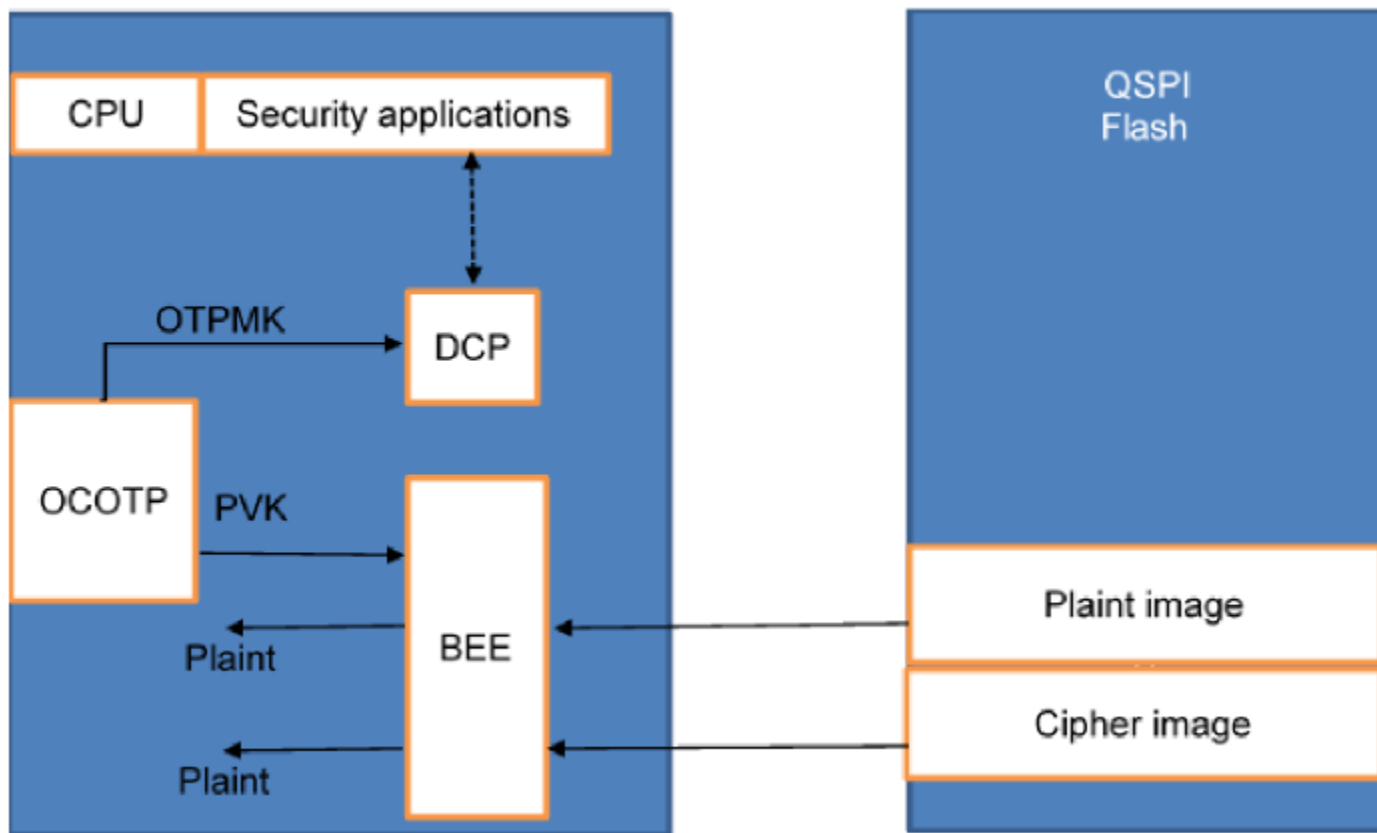


Figure 7. Introduction of the BEE encrypted XIP

### 2.4.2. Image layout & encrypt XIP flow

The FlexSPI NOR image layout is as shown in table 1, Both PRDB0 and PRDB1 are optional, and they are required only if corresponding eFUSE bits are enabled.

PRDB1 takes precedence over PRDB0, this design is for this use case that the Independent Design House (IDH) uses this block to protect their intellectual property before shipping products to end customers.

Table 1. Boot image layout

Offset	Description
0x0000 - 0x01FF	FlexSPI Serial NOR Configuration Block
0x0200 - 0x03FF	Reserved
0x0400 - 0x041F	Encrypted Key Info Block 0 (EKIB0)
0x0420 - 0x047F	Reserved
0x0480 - 0x057F	Encrypted Protection Region Descriptor Block 0(EPRDB0)
0x0580 - 0x07FF	Reserved
0x0800 - 0x081F	Encrypted Key Info Block 1(EKIB1)
0x0820 - 0x087F	Reserved



**Table 1. Boot image layout**

Offset	Description
0x0880 - 0x097F	Encrypted Protection Region Descriptor Block 1(EPRDB1)
0x0980 - 0x0FFF	Reserved

## 2.5. Encrypt XIP flow

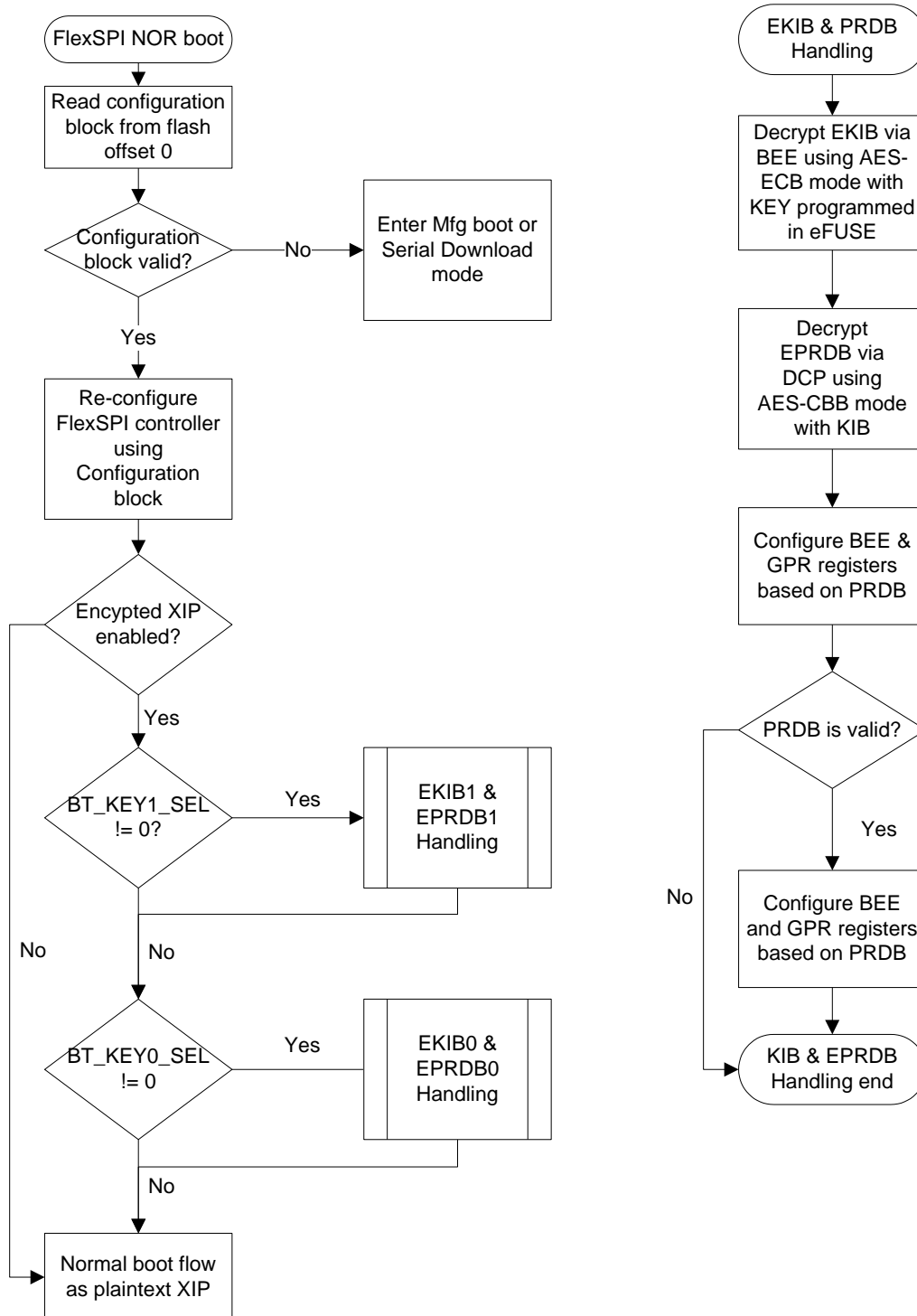


Figure 8. Encrypt XIP Flow

## 2.6. Decrypt XIP flow

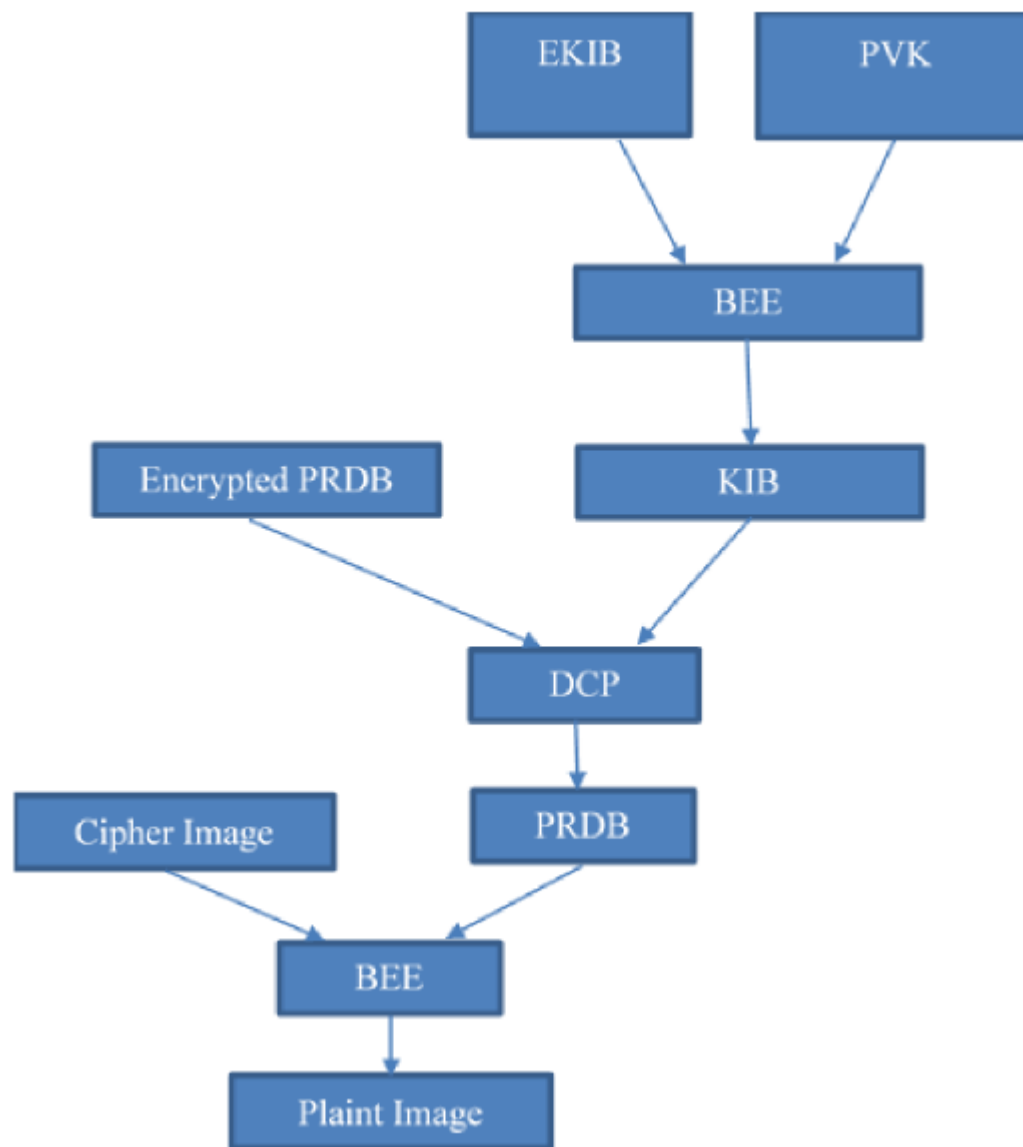


Figure 9. Decrypt XIP flow

## 2.7. Typical use cases

### 1. Encrypted XIP (End customers)

End customers generates encrypted boot image using specified AES key 0, generates PRDB0, program encrypted boot image and PRDB0 to Serial NOR Flash device, programs the BEE\_KEY0\_SEL fuse, program OTPMK or SW\_GP2 based on setting on BEE\_KEY0\_SEL.

### 2. Encrypted Execute-only Library (IDH)

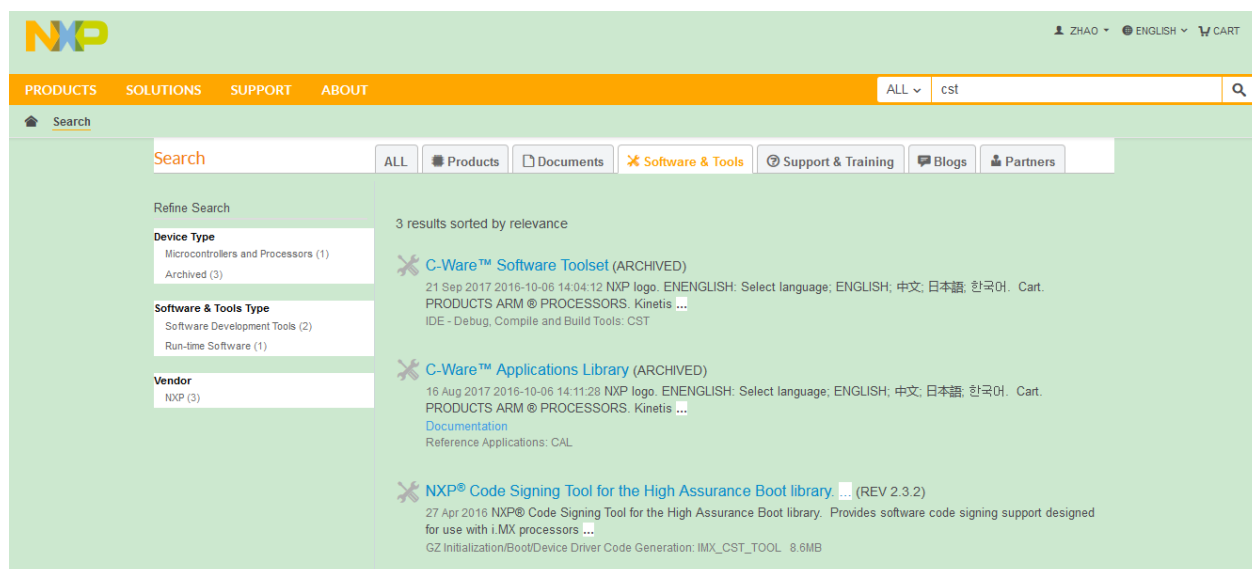
IDH generates encrypted library using specified AES key 1, generates PRDB1 with restricted Flash Access Permission, programs encrypted library binary and PRDB1 to Serial NOR Flash device, programs the BEE\_KEY1\_SEL fuse, program OTPMK or SW\_GP2 based on setting on BEE\_KEY1\_SEL.

## 3. How to Use

### 3.1. Authenticate using elftosb

The following procedures suppose you have a PC running Ubuntu to generate the keys, a PC running win to use elftosb tools and a MIMXRT1050 Evaluation Kit board. For a different environment, the steps may differ. Before the following steps, you must download the Code Signing Tool from the NXP website.

1. Retrieve the CST Tool and Elftosb tool
  - a) Search the CST tool on NXP website and download the tool.



**Figure 10. Example for downloading the Code Signing Tool**

- b) Download the Elftosb tool from the following website:  
Reverse to be added in the future.
2. Unpack the CST package.
3. cd keys
  - Create a text file called serial, which contains 8 digits.
  - Create a text file called key\_pass.txt, which contains two lines of identical text, such as "cst\_test".
4. Make sure you have install the openssl software and run “hab4\_pki\_tree.sh”

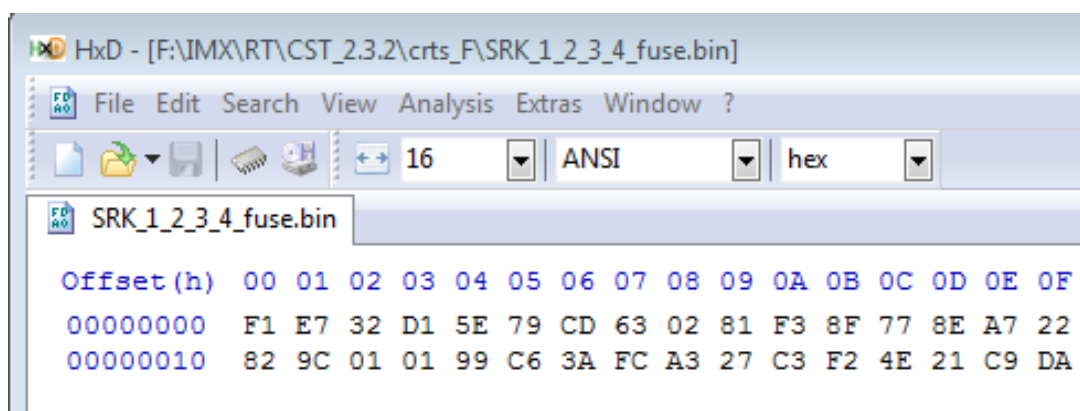
- For question prompt, enter "n", "2048", "10", "4", "y" one by one.
- This script will generate private key and public key pairs in the working directory.

5. `cd ../crts`

```
..\linux64\srktool -h 4 -t SRK_1_2_3_4_table.bin -e SRK_1_2_3_4_fuse.bin -d sha256 -c
./SRK1_sha256_2048_65537_v3_ca.crt.pem,./SRK2_sha256_2048_65537_v3_ca.crt.pem,./
SRK3_sha256_2048_65537_v3_ca.crt.pem,./SRK4_sha256_2048_65537_v3_ca.crt.pem -f 1
```

This command will generate root public key file "SRK\_1\_2\_3\_4\_table.bin" and its corresponding hash "SRK\_1\_2\_3\_4\_fuse.bin". The content of the latter will be later on burned to chip eFuse.

Open the SRK\_1\_2\_3\_4\_fuse.bin by using the HxD software under windows you can get the content as shown in [Figure 11](#).



**Figure 11. Example about the content of SRK\_1\_2\_3\_4\_fuse.bin**

If you want to get more information about how to generate keys and certificates, please refer to the "HABCST\_UG" document under the CST package.

For the step to burn fuse please refer to [Program fuse](#).

### 3.1.1. Sign the flashloader

Unpack the IMX MCU Manufacturing User's Guide Vxx package. To generate signed bootable image using elftosb, the following is an example of flashloader and the steps are as below:

1. Create BD file, the BD file can be blank, like below.

```
/******file start******/
options {
    flags = 0x08;
    startAddress = 0x20000000;
    ivtOffset = 0x400;
    initialLoadSize = 0x2000;
}
sources {
    elfFile = extern(0);
}
constants {
```

```

SEC_CSF_HEADER          = 20;
SEC_CSF_INSTALL_SRK      = 21;
SEC_CSF_INSTALL_CSFK     = 22;
SEC_CSF_AUTHENTICATE_CSF = 24;
SEC_CSF_INSTALL_KEY      = 25;
SEC_CSF_AUTHENTICATE_DATA = 26;
}
section (SEC_CSF_HEADER;
  Header_Version="4.2",
  Header_HashAlgorithm="sha256",
  Header_Engine="DCP",
  Header_EngineConfiguration=0,
  Header_CertificateFormat="x509",
  Header_SignatureFormat="CMS"
)
{
}
section (SEC_CSF_INSTALL_SRK;
  InstallSRK_Table="keys/SRK_1_2_3_4_table.bin", // "valid file path"
  InstallSRK_SourceIndex=0
)
{
}
section (SEC_CSF_INSTALL_CSFK;
  InstallCSFK_File="crts/CSF1_1_sha256_2048_65537_v3_usr.crt.pem", // "valid file path"
  InstallCSFK_CertificateFormat="x509" // "x509"
)
{
}
section (SEC_CSF_AUTHENTICATE_CSF)
{
}
section (SEC_CSF_INSTALL_KEY;
  InstallKey_File="crts/IMG1_1_sha256_2048_65537_v3_usr.crt.pem",
  InstallKey_VerificationIndex=0, // Accepts integer or string
  InstallKey_TargetIndex=2 // Accepts integer or string
)
{
}
section (SEC_CSF_AUTHENTICATE_DATA;
  AuthenticateData_VerificationIndex=2,
  AuthenticateData_Engine="DCP",
  AuthenticateData_EngineConfiguration=0
)
{
}
/*****file end*****/

```

After the blank BD file is created, place it into the same folder that holds elftosb executable.

2. Copy Flashloader.elf provided in release package into the same folder that holds elftosb executable.
3. Copies the “cst” executable, “crts” folder and “keys” folder from “<cst\_installation\_dir>” to the same folder that holds elftosb executable
4. Generate Bootable image using *elftosb*.

```
elftosb.exe -f imx -V -c imx-flexspinor-flashloader-signed.bd -o ivt_flashloader_signed.bin flashloader.elf
```

After above operation, we will see that there are two bootable images generated by elftosb.

- The first one is `ivt_flashloader_signed.bin`, the memory region from 0 till `ivt_offset` are fill with padding bytes (all 0x00s).
- The second one is `ivt_flashloader_signed_nopadding.bin`, which starts from `ivt_offset` directly. The CSF section is generated and appended to the unsigned bootable image successfully.

If you use your own keys, please replace the “`ivt_flashloader_signed.bin`” under the “<Device>/OS Firmware/” folder.

### 3.1.2. Sign the application

#### 3.1.2.1. Generate signed normal i.MX MCU bootable image

To generate signed bootable image using `elftosb`, the steps are as below:

1. Create BD file, the BD file can be blank, like below.

```

/*****file start*****/
options {
    flags = 0x08;
    startAddress = 0x60000000;
    ivtOffset = 0x1000;
    initialLoadSize = 0x2000;
}

sources {
    elfFile = extern(0);
}

constants {
    SEC_CSF_HEADER          = 20;
    SEC_CSF_INSTALL_SRK     = 21;
    SEC_CSF_INSTALL_CSFK    = 22;
    SEC_CSF_AUTHENTICATE_CSF = 24;
    SEC_CSF_INSTALL_KEY     = 25;
    SEC_CSF_AUTHENTICATE_DATA = 26;
}

section (SEC_CSF_HEADER;
    Header_Version="4.2",
    Header_HashAlgorithm="sha256",
    Header_Engine="DCP",
    Header_EngineConfiguration=0,
    Header_CertificateFormat="x509",
    Header_SignatureFormat="CMS"
)
{
}

section (SEC_CSF_INSTALL_SRK;
    InstallSRK_Table="keys/SRK_1_2_3_4_table.bin", // "valid file path"
    InstallSRK_SourceIndex=0
)
{
}

section (SEC_CSF_INSTALL_CSFK;

```

```

InstallCSFK_File="crts/CSF1_1_sha256_2048_65537_v3_usr crt.pem", // "valid file path"
InstallCSFK_CertificateFormat="x509" // "x509"
)
{
}

section (SEC_CSF_AUTHENTICATE_CSF)
{
}

section (SEC_CSF_INSTALL_KEY;
InstallKey_File="crts/IMG1_1_sha256_2048_65537_v3_usr crt.pem",
InstallKey_VerificationIndex=0, // Accepts integer or string
InstallKey_TargetIndex=2) // Accepts integer or string
{
}

section (SEC_CSF_AUTHENTICATE_DATA;
AuthenticateData_VerificationIndex=2,
AuthenticateData_Engine="DCP",
AuthenticateData_EngineConfiguration=0)
{
}
/*****file end*****/

```

After the blank BD file is created, place it into the same folder that holds elftosb executable.

2. Copy led\_demo\_evb\_flexspi\_nor.elf provided in release package into the same folder that holds elftosb executable.
3. Copies the “cst” executable, “crts” folder and “keys” folder from “<cst\_installation\_dir>” to the same folder that holds elftosb executable
4. Generate Bootable image using elftosb.

```

./elftosb.exe -f imx -V -c imx-flexspinor-normal-signed.bd -o ivt_application_signed.bin
led_demo_evk_flexspi_nor_0x60002000.srec

```

After above operation, we will see that there are two bootable images generated by elftosb.

- The first one is ivt\_application\_signed.bin.
- The second one is ivt\_flashloader\_signed\_nopadding.bin.

### 3.1.2.2. Generate SB file for FlexSPI NOR image provisioning

Usually, a BD file for FlexSPI NOR boot consists of 7 parts.

1. The bootable image file path is provided in “sources” block
2. The FlexSPI NOR Configuration Option block is provided in “section” block
3. To enable FlexSPI NOR access, the “enable” command must be provided following above option block.
4. In case the Flash device are not erased, an “erase” command is required before programming data to Flash device. Be aware, the erase operation is pretty time consuming, it may be unnecessary for a blank Flash device (factory setting) during Manufacturing.



5. The FlexSPI NOR Configuration Block (FNORCB) is required for FlexSPI NOR boot, to program the FNORCB generated by above option block, a special magic number '0xF000000F' must be load into RAM first
6. To notify the Flashloader to program the FNORCB, an “enable” command must be provided following the magic number loading
7. After above operation, Flashloader can program the bootable image binary into Serial NOR Flash through FlexSPI module using load command.

A simple example containing above steps is shown below.

```

/*****file start*****/

# The source block assign file name to identifiers
sources {
  myBinFile = extern (0);
}

constants {
  kAbsAddr_Start= 0x60000000;
  kAbsAddr_Ivt = 0x60001000;
  kAbsAddr_App = 0x60002000;
}

# The section block specifies the sequence of boot commands to be written to the SB file
section (0) {

  #1. Prepare Flash option
  # 0xc0233007 is the tag for Serial NOR parameter selection
  # bit [31:28] Tag fixed to 0x0C
  # bit [27:24] Option size fixed to 0
  # bit [23:20] Flash type option
  #      0 - QuadSPI SDR NOR
  #      1 - QUadSPI DDR NOR
  #      2 - HyperFLASH 1V8
  #      3 - HyperFLASH 3V
  #      4 - Macronix Octal DDR
  #      6 - Micron Octal DDR
  #      8 - Adesto EcoXIP DDR
  # bit [19:16] Query pads (Pads used for query Flash Parameters)
  #      0 - 1
  #      2 - 4
  #      3 - 8
  # bit [15:12] CMD pads (Pads used for query Flash Parameters)
  #      0 - 1
  #      2 - 4
  #      3 - 8
  # bit [11: 08] fixed to 0
  # bit [07: 04] fixed to 0
  # bit [03: 00] Flash Frequency, device specific
  #
  #
  # In this example, the 0xc0233007 represents:
  #   HyperFLASH 1V8, Query pads: 8 pads, Cmd pads: 8 pads, Frequency: 133MHz
  load 0xc0233007 > 0x2000;
  # Configure HyperFLASH using option a address 0x2000
  enable flexspinor 0x2000;
}

```

```
#2 Erase flash as needed.
erase 0x60000000..0x60010000;

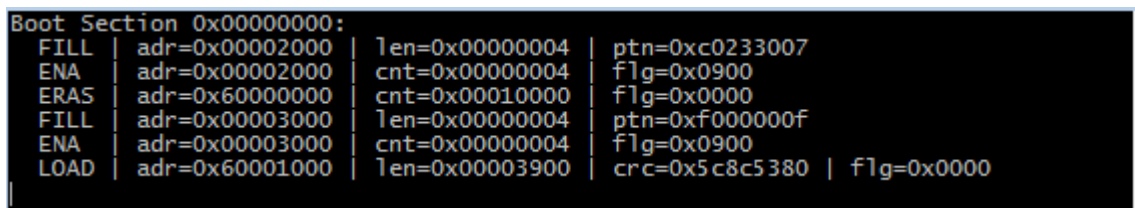
#3. Program config block
# 0xf000000f is the tag to notify Flashloader to program FlexSPI NOR config block to the start of device
load 0xf000000f > 0x3000;
# Notify Flashloader to response the option at address 0x3000
enable flexspinor 0x3000;

#5. Program image
load myBinFile > kAbsAddr_lvt;
}

/*****file end*****/
```

Here is an example to generate SB file using elftosb, ivt\_application\_signed\_nopadding.bin and above BD file.

```
elftosb.exe -f kinetis -V -c program_flexspinor_image_hyperflash.bd -o boot_image.sb ivt_application_signed_nopadding.bin
```



Boot Section 0x00000000:				
FILL	adr=0x00002000	len=0x00000004	ptn=0xc0233007	
ENA	adr=0x00002000	cnt=0x00000004	flg=0x0900	
ERAS	adr=0x60000000	cnt=0x00010000	flg=0x0000	
FILL	adr=0x00003000	len=0x00000004	ptn=0xf000000f	
ENA	adr=0x00003000	cnt=0x00000004	flg=0x0900	
LOAD	adr=0x60001000	len=0x00003900	crc=0x5c8c5380	flg=0x0000

**Figure 12. Example command to generate SB file for flexspi nor provisioning**

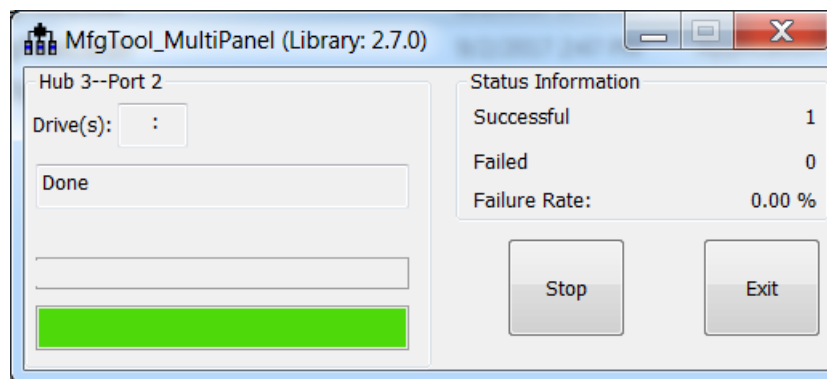
After above command, a file named boot\_image.sb will be created under the same folder that holds elftosb executable.

### 3.1.2.3. Download the image

If the “boot\_image.sb” has been created, we can use the mfgtool to download it. Please refer to [Download Image Using mfgtool](#).

You must make sure that the platform is connected under Serial downloader mode.

Since our image is signed, so we should configure the “name” field in cfg.ini file as <Device>-SecureBoot. Then replace the “boot\_image.sb” under the “<Device>/OS Firmware/” folder. After all of these have been done run the mfgtool and download your image.



**Figure 13. Successful result of downloading boot\_image.sb**

Remember to reset the SW pin selection to change the boot mode, then you can reset and run.

## 3.2. Encrypt boot using elftosb

### 3.2.1. Generate encrypted normal i.MX MCU bootable image

To generate encrypted image, the steps are as below.

1. Create BD file

```

/*****file start *****/
options {
    flags = 0x0c;
    startAddress = 0x8000;
    ivtOffset = 0x400;
    initialLoadSize = 0x2000;
}

sources {
    elfFile = extern(0);
}

constants {
    SEC_CSF_HEADER          = 20;
    SEC_CSF_INSTALL_SRK     = 21;
    SEC_CSF_INSTALL_CSFK    = 22;
    SEC_CSF_AUTHENTICATE_CSF = 24;
    SEC_CSF_INSTALL_KEY     = 25;
    SEC_CSF_AUTHENTICATE_DATA = 26;
    SEC_CSF_INSTALL_SECRET_KEY = 27;
    SEC_CSF_DECRYPT_DATA     = 28;
}

section (SEC_CSF_HEADER;
    Header_Version="4.3",
    Header_HashAlgorithm="sha256",
    Header_Engine="DCP",
    Header_EngineConfiguration=0,
    Header_CertificateFormat="x509",
    Header_SignatureFormat="CMS"

```

**How to use i.MXRT Security Boot, Application Notes, Rev. 0, 10/2017**

```

    )
    {
    }

section (SEC_CSF_INSTALL_SRK;
    InstallSRK_Table="keys/SRK_1_2_3_4_table.bin", // "valid file path"
    InstallSRK_SourceIndex=0
)
{
}

section (SEC_CSF_INSTALL_CSFK;
    InstallCSFK_File="crts/CSF1_1_sha256_2048_65537_v3_usr crt.pem", // "valid file path"
    InstallCSFK_CertificateFormat="x509" // "x509"
)
{
}

section (SEC_CSF_AUTHENTICATE_CSF)
{
}

section (SEC_CSF_INSTALL_KEY;
    InstallKey_File="crts/IMG1_1_sha256_2048_65537_v3_usr crt.pem",
    InstallKey_VerificationIndex=0, // Accepts integer or string
    InstallKey_TargetIndex=2) // Accepts integer or string
{
}

section (SEC_CSF_AUTHENTICATE_DATA;
    AuthenticateData_VerificationIndex=2,
    AuthenticateData_Engine="DCP",
    AuthenticateData_EngineConfiguration=0)
{
}

section (SEC_CSF_INSTALL_SECRET_KEY;
    SecretKey_Name="dek.bin",
    SecretKey_Length=128,
    SecretKey_VerifyIndex=0,
    SecretKey_TargetIndex=0)
{
}

section (SEC_CSF_DECRYPT_DATA;
    Decrypt_Engine="DCP",
    Decrypt_EngineConfiguration="0", // "valid engine configuration values"
    Decrypt_VerifyIndex=0,
    Decrypt_MacBytes=16)
{
}

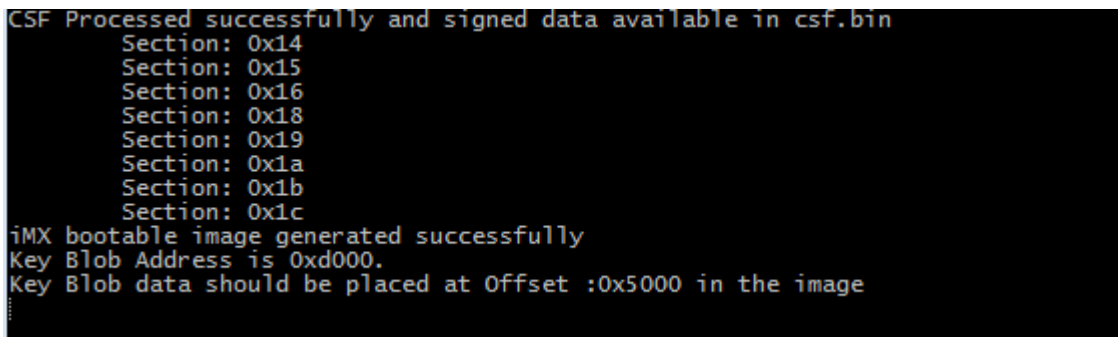
/*****file end *****/

```

2. Copy led\_demo\_evk\_ram\_0xa000.elf provided in release package into the same folder that holds elftosb executable
3. Copy the “cst” executable, “crts” folder and “keys” folder from “<cst\_installation\_dir>” to the same folder that holds elftosb executable

#### 4. Step 4: Generate Encrypted Bootable image using elftosb

```
elftosb.exe -f imx -V -c imx-itcm-encrypted.bd -o ivt_spinand_application_encrypted.bin
led_demo_evk_ram_0xa000.srec
```



```
CSF Processed successfully and signed data available in csf.bin
Section: 0x14
Section: 0x15
Section: 0x16
Section: 0x18
Section: 0x19
Section: 0x1a
Section: 0x1b
Section: 0x1c
iMX bootable image generated successfully
Key Blob Address is 0xd000.
Key Blob data should be placed at Offset :0x5000 in the image
```

**Figure 14. Example command result to generate encrypt image**

After above operation, we will see that there are two bootable images generated by elftosb.

- The first one is ivt\_spinand\_application\_encrypted.bin. We will use this file in the next step.
- The second one is ivt\_spinand\_application\_encrypted\_nopadding.bin.

#### 5. Generate KeyBlob section using Flashloader, this step will be done in [3.2.2](#).

### 3.2.2. Generate SB file for encrypted FlexSPI NAND image and KeyBlob provisioning

In general, a BD file for FlexSPI NAND image provisioning with KeyBlob consists of 7 parts.

1. The bootable image file path is provided in “sources” block
2. Enable FlexSPI NAND access using FlexSPI NAND Configuration Option block
3. Erase SPI NAND device as needed.
4. Program boot image binary into Serial NAND via FlexSPI module
5. Update KeyBlob information using KeyBlob Option block
6. Program KeyBlob block into SPI NAND for firmware 0
7. Program KeyBlob block into SPI NAND for firmware 1

An example BD file is shown as below.

```
/******file start *****/

# The source block assign file name to identifiers
sources {
  myBootImageFile = extern (0);
  dekFile = extern(1);
}

# The section block specifies the sequence of boot commands to be written to the SB file
section (0) {
```

```

#1. Prepare FlexSPI NAND FCB option block
#-----
# option0
# bit [31:28] Tag fixed to 0x0C
# bit [27:24] Search Count: 1 - 4
# bit [23:20] Search stride: 0 - 64 pages, 1 - 128 pages, 2 - 256 pages, 3 - 32 pages
# bit [19:12] Reserved, fixed to 0
# bit [11: 08] Address Type for Read/Program/Erase in Flashloader, 0 - bypte address, 1 - block address
# bit [07: 04] Reserved, fixed to 0
# bit [03: 00] Option block size in long words, Min size is 3, Max size is 10.
#
# nandOptionAddr
# The address that NandOption stays.For example, 0x10020
#
# imageinfo, up to 8 entries, each entry consists of
# bit [31:16] start block id of firmware
# bit [15:00] maximum allowed block count of firmware
#
#-----
# Preapre FlexSPI NAND Configuration option block
# option0 (mandatory)
# bit [31:28] tag, fixed to 0x0c
# bit [27:24] Option size in long words, actual option block size = (option0[27:24] + 1) * 4
# bit [23:20] Reserved, fixed to 0
# bit [19:16] Flash size: 0 - 512Mbit, 1-1Gbit, 2-2Gbit 4-4Gbit
# bit [15:12] Has multi-planes, 0 - 1 Plane, 1- 2Planes
# bit [11:08] Pages Per block, 0 - 64 pages, 1 - 128 pages, 2 - 256 pages, 3 - 32 pages
# bit [07:04] Page size (Kbytes), 2 - 2KB, 4 - 4KB
# bit [03:00] Max Frequency supported by Serial NAND, see System Boot chapter in SoC RM for more details
#           Currently, the defintion is: 1 - 30MHz, 2 - 50MHz, 3 - 60MHz, 4 - 72MHz, 5 - 80MHz, 6 - 100MHz
#           7 - 133MHz, 8 - 166MHz
#
# option1(optional)
# bit [31:24] Reserved, fixed to 0
# bit [23:16] Ecc Check Mask
# bit [15:08] Ecc Failure Mask
# bit [07:00] Manufacturer ID, for example: 0x2c (Micron)
#-----
# Here is an example show how to create FCB using FlexSPI NAND FCB option block
# option0: tag=0x0c, searchCount=2, searchStride = 64 pages, Address type: Block Address, Option Block size: 4 long
words
load 0xc2000104 > 0x10000;
# nandOptionAddr: 0x10000
load 0x00010020 > 0x10004;
# image info for firmware 0: start from block 4, max block count 4
load 0x00040004 > 0x10008;
# image info for firmware 1: start from block 8, max block count 4
load 0x00080004 > 0x1000c;
# FlexSPI NAND Cofniguration Option Block
# tag = 0x0c, Option size 2 long words, Flash size: 1Gbit, 1 Plane, Pages Per Block: 64, Page Size: 2KBytes, Max
Freq:60MHz
load 0xC1010023 > 0x10020; #(Address must be equal to the value in nandOptionAddr)
# Manufacturer Id = 0 (ignored), ECC Check Mask = 0x70, ECC Failure Mask = 0x20
load 0x00702000 > 0x10024;
# Configure Serial NAND using option block at address 0x10000
enable spinand 0x10000;

#2 Erase flash as needed. (Unit: Block)
# Erase firmware0 region
erase spinand 0x4..0x8;
# Erase firmware1 region
erase spinand 0x8..0xc;

```

```

#3. Program image
# Load to firmware0 region
load spinand myBootImageFile > 0x4;

# Load to firmware1 region
load spinand myBootImageFile > 0x8;

#4. Generate KeyBlob and program it to SPI NAND
# Load DEK to RAM
load dekFile > 0x10100;
# Construct KeyBlob Option
#-----
# bit [31:28] tag, fixed to 0x0b
# bit [27:24] type, 0 - Update KeyBlob context, 1 Program Keyblob to SPI NAND
# bit [23:20] keyblob option block size, must equal to 3 if type =0,
#         reserved if type = 1
# bit [19:08] Reserved
# bit [07:04] DEK size, 0-128bit 1-192bit 2-256 bit, only applicable if type=0
# bit [03:00] Firmware Index, only applicable if type = 1
# if type = 0, next words indicate the address that holds dek
#         the 3rd word
#-----
# tag = 0x0b, type=0, block size=3, DEK size=128bit
load 0xb0300000 > 0x10200;
# dek address = 0x10100
load 0x00010100 > 0x10204;
# keyblob offset in boot image
# Note: this is only an example bd file, the value must be replaced with actual
#       value in users project
load 0x00005000 > 0x10208;
enable spinand 0x10200;

#5. Program KeyBlob to firmware0 region
load 0xb1000000 > 0x10300;
enable spinand 0x10300;

#6. Program KeyBlob to firmware1 region
load 0xb1000001 > 0x10400;
enable spinand 0x10400;

#7. Program Fuse (optional), remove '#' to make it effective
# Please refer to Fusemap for more details
# For example
# Boot device: FlexSPI NAND, Column address width:12 bits,
# CS_Interval: 100ns, Safe Freq: 50MHz, Search Stride: 64 pages, Search Count:2
#load fuse 0x00000040 > 0x05;
}
/*****file end *****/

```

Please pay attention that in step 4, the command “load 0x00005000 > 0x10208”, The address must in accordance with the offset of the key Blob show in [Figure 14](#).

Run the following command:

```

elftosb.exe -f kinetis -V -c program_flexspinand_encrypt_image_keyblob.bd -o boot_image.sb
ivt_spinand_application_encrypted.bin dek.bin

```

```

Boot Section 0x00000000:
FILL | adr=0x00010000 | len=0x00000004 | ptn=0xc2000104
FILL | adr=0x00010004 | len=0x00000004 | ptn=0x00010020
FILL | adr=0x00010008 | len=0x00000004 | ptn=0x00040004
FILL | adr=0x0001000c | len=0x00000004 | ptn=0x00080004
FILL | adr=0x00010020 | len=0x00000004 | ptn=0xc1010023
FILL | adr=0x00010024 | len=0x00000004 | ptn=0x00702000
ENA | adr=0x00010000 | cnt=0x00000004 | flg=0x0110
ERAS | adr=0x00000004 | cnt=0x00000004 | flg=0x0110
ERAS | adr=0x00000008 | cnt=0x00000004 | flg=0x0110
LOAD | adr=0x00000004 | len=0x00005000 | crc=0x45600dda | flg=0x0110
LOAD | adr=0x00000008 | len=0x00005000 | crc=0x45600dda | flg=0x0110
LOAD | adr=0x00010100 | len=0x00000010 | crc=0x7827a8fc | flg=0x0000
FILL | adr=0x00010200 | len=0x00000004 | ptn=0xb0300000
FILL | adr=0x00010204 | len=0x00000004 | ptn=0x00010100
FILL | adr=0x00010208 | len=0x00000004 | ptn=0x00005000
ENA | adr=0x00010200 | cnt=0x00000004 | flg=0x0110
FILL | adr=0x00010300 | len=0x00000004 | ptn=0xb1000000
ENA | adr=0x00010300 | cnt=0x00000004 | flg=0x0110
FILL | adr=0x00010400 | len=0x00000004 | ptn=0xb1000001
ENA | adr=0x00010400 | cnt=0x00000004 | flg=0x0110

```

Figure 15. Example command result to generate SB file

### 3.2.3. Download the image

If the “boot\_image.sb” has been created, we can use the mfgtool to download it. Please refer to [Download Image Using mfgtool](#). If you use your own keys, please replace the “ivt\_flashloader\_signed.bin” under the “<Device>/OS Firmware/” folder.

You must make sure that the platform is connected under Serial downloader mode. Since our image is signed, so we should configure the “name” field in cfg.ini file as <Device>- SecureBoot. Then replace the “boot\_image.sb” under the “<Device>/OS Firmware/” folder. After all of these have been done run the mfgtool and download your image.

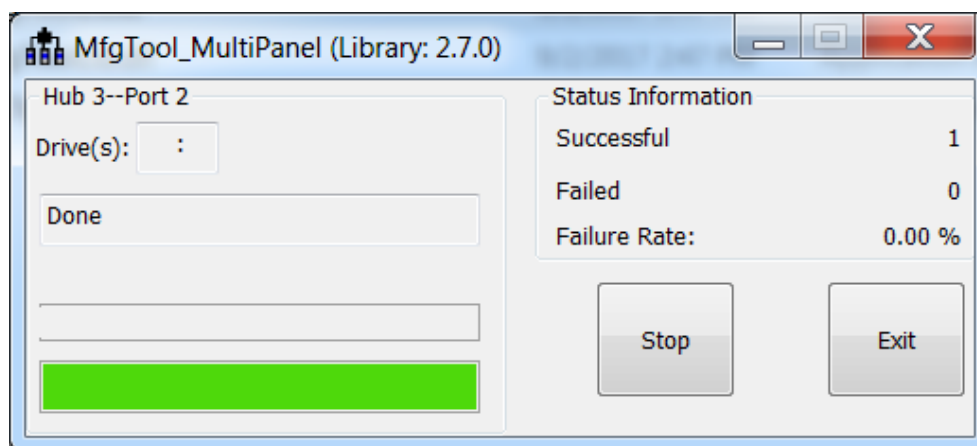


Figure 16. Successful result of downloading boot\_image.sb

Remember to reset the SW pin selection to change the boot mode, then you can reset and run.



### 3.3. Encrypt XIP using elftosb

#### 3.3.1. Generate Unsigned Normal I.MX MCU bootable image

Typically, the unsigned bootable image will be generated and programmed to destination memory during development phase.

Elftosb supports unsigned bootable image generation using options, BD file and ELF/SREC file generated by toolchain. Below are the steps to create bootable image .

1. Create BD file. For unsigned image creation, the “constants” block is optional, for example:

```

/*****file start*****/
options {
    flags = 0x00;
    startAddress = 0x60000000;
    ivtOffset = 0x1000;
    initialLoadSize = 0x2000;
}

sources {
    elfFile = extern(0);
}

section (0)
{
}

/*****file end *****/

```

After the blank BD file is created, place it into the same folder that holds elftosb executable.

2. Copy “led\_demo\_evb\_flexspi\_nor.elf” provided in release package into the same folder that holds elftosb executable.
3. Generate Bootable image using elftosb

```

elftosb.exe -f imx -V -c imx-flexspinor-normal-unsigned.bd -o ivt_application_unsigned.bin
led_demo_evb_flexspi_nor_0x60002000.srec

```

After above operation, we will see that there are two bootable images generated by elftosb.

- The first one is ivt\_application\_unsigned.bin.
- The second one is ivt\_application\_unsigned\_nopadding.bin.

#### 3.3.2. Generate SB file for FlexSPI NOR image encryption and programming

Usually, a BD file for FlexSPI NOR image encryption and programming consists of 7 parts.

1. The bootable image file path is provided in “sources” block
2. Enable FlexSPI NOR access using FlexSPI NOR Configuration Option block

**How to use i.MXRT Security Boot, Application Notes, Rev. 0, 10/2017**

3. Erase the Flash device as needed in case it is not blank. Be aware the erase operation is pretty time consuming, it may be unnecessary for a blank Flash device during manufacturing process.
4. Enable Image Encryption using PRDB option block
5. Program FNORCB using magic number
6. Program boot image binary into Serial NOR via FlexSPI module
7. Enable Encrypted XIP fuse bits.

A simple example containing above steps is shown below.

```

/*****file start*****/

# The source block assign file name to identifiers
sources {
  myBinFile = extern (0);
}

constants {
  kAbsAddr_Start= 0x60000000;
  kAbsAddr_lvt = 0x60001000;
  kAbsAddr_App = 0x60002000;
}

# The section block specifies the sequence of boot commands to be written to the SB file
section (0) {

  #1. Prepare Flash option
  # In this example, the 0xc0233007 represents:
  #   HyperFLASH 1V8, Query pads: 8 pads, Cmd pads: 8 pads, Frequency: 133MHz
  load 0xc0233007 > 0x2000;
  # Configure HyperFLASH using option a address 0x2000
  enable flexspinor 0x2000;

  #2 Erase flash as needed.
  erase 0x60000000..0x60010000;

  #3 Prepare PRDB options
  # 0xe0120000 is an option for PRDB contruction and image encryption
  # bit[31:28] tag, fixed to 0x0E
  # bit[27:24] Key source, fixed to 0 for A0 silicon
  # bit[23:20] AES mode: 1 - CTR mode
  # bit[19:16] Encrypted region count
  # bit[15:00] reserved in A0
  load 0xe0120000 > 0x4000;
  # Region 0 start
  load 0x60001000 > 0x4004;
  # Region 0 length
  load 0x00001000 > 0x4008;
  # Region 1 start
  load 0x60002000 > 0x400c;
  # Region 1 length
  load 0x0000e000 > 0x4010;
  # Program PRDB0 based on option
  enable flexspinor 0x4000;

  #4. Program config block

```

```
# 0xf000000f is the tag to notify Flashloader to program FlexSPI NOR config block to the start of device
load 0xf000000f > 0x3000;
# Notify Flashloader to response the option at address 0x3000
enable flexspinor 0x3000;

#5. Program image
load myBinFile > kAbsAddr_lvt;

#6. Program BEE_KEY0_SEL and BEE_KEY1_SEL
load fuse 0x0000e000 > 0x06;
}
/*****file end*****/
```

The steps to generate SB file is the same as above section.

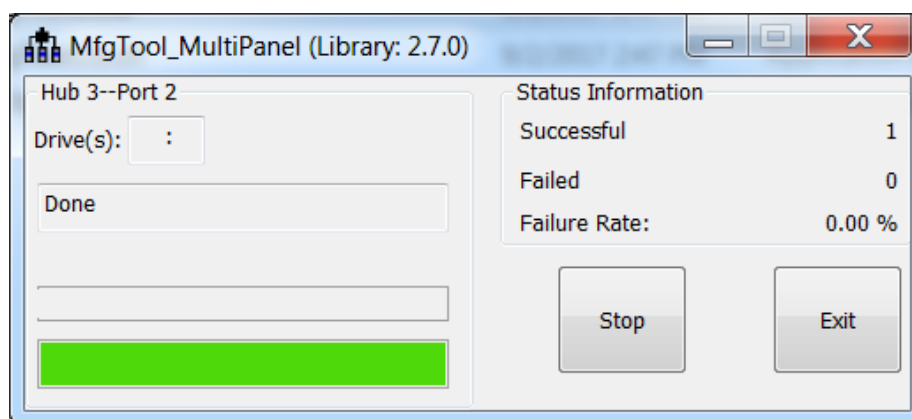
If you have create the bd file, then run the following command:

```
elftosb.exe -f kinetis -V -c program_flexspinor_image_hyperflash_encrypt.bd -o boot_image.sb
ivt_application_unsigned_nopadding.bin
```

### 3.3.3. Download the image

If the “boot\_image.sb” has been created, we can use the mfgtool to download it. Please refer to [Download Image Using mfgtool](#).

You must make sure that the platform is connected under Serial downloader mode. Since our image is unsigned, so we should configure the “name” field in cfg.ini file as <Device>-DevBoot. Then replace the “boot\_image.sb” under the “<Device>/OS Firmware/” folder. After all of these have been done run the mfgtool and download your image. At last set the SW pin on your board to enable XIP and run the application.



**Figure 17. Successful result of downloading boot\_image.sb**

Remember to reset the SW pin selection to change the boot mode and enable XIP, then you can reset and run.

## 4. Download Image Using mfgtool

### 4.1. Connect to the device

The NXP i.MX MCU platform can be connected to a host computer to interface with the i.MX MCU BootROM application. After the platform is connected under Serial downloader mode, use the Mfgtool to program bootable image into the target Flash memory, if the connection establishes successfully and the cfg.ini, UICfg.ini are configured properly, the device will be recognized by MfgTool like below.

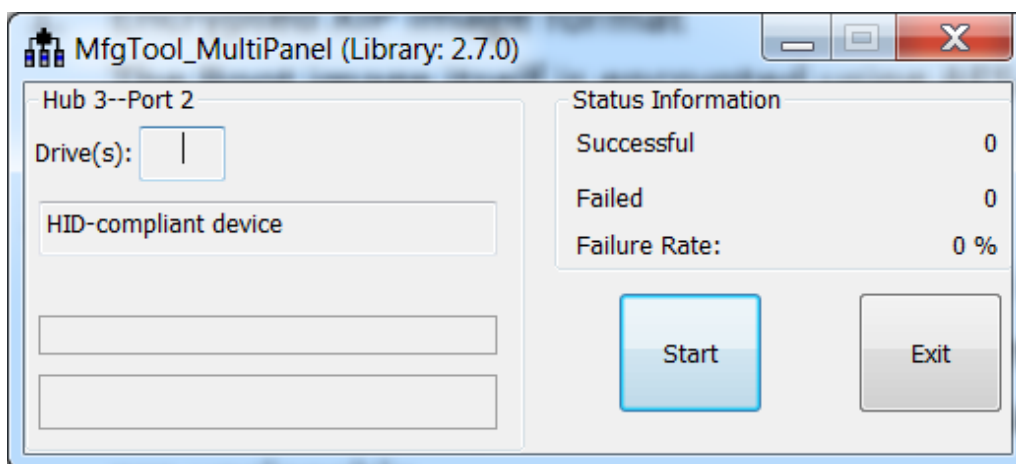


Figure 18. MfgTool GUI with device connected

### 4.2. Program image during development

In Development phase, the device may be under HAB open mode for most use cases, users can just configure the “name” field in cfg.ini file as <Device>-DevBoot, then prepare the boot\_image.sb file using elftosb utility, after the “boot\_image.sb” is generated, place it into “<Device>/OS Firmware/” folder, then switch device into serial downloader, connect it to host PC, finally, Open MfgTool.exe and click “Start” to trigger a provisioning sequence, after the provisioning completes, you will see below window. To exit MfgTool, click “Stop” and “Exit” in turn.

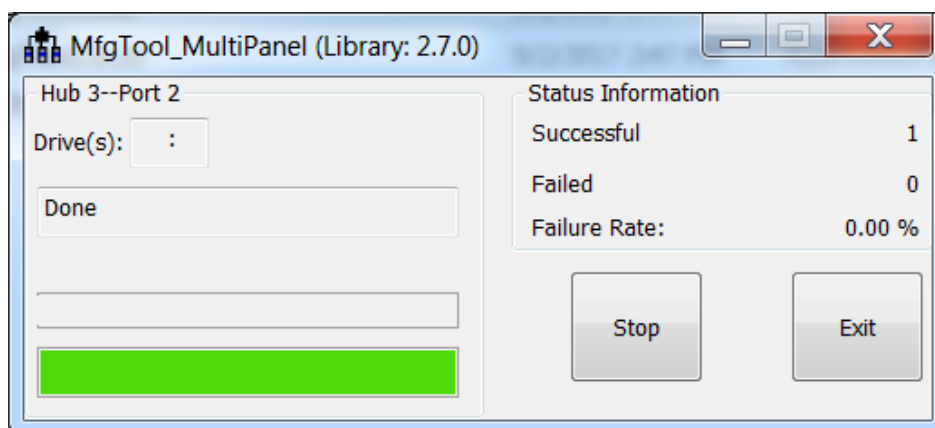


Figure 19. Successful result for provisioning with MfgTool for DevBoot

### 4.3. Program fuse

The elftosb can support programming Fuse using built-in supported “load fuse” command, below is an example to program SRK table and enable HAB closed mode.

1. Create BD file under the same folder that holds elftosb executable. Here is an example named “enable\_hab.bd”.

```

/*****file start*****/

# The source block assign file name to identifiers
sources {
}

constants {
}

# The section block specifies the sequence of boot commands to be written to the SB file
# Note: this is just a template, please update it to actual values in users' project
section (0) {

    # Program SRK table
    load fuse 0xD132E7F1 > 0x18;
    load fuse 0x63CD795E > 0x19;
    load fuse 0x8FF38102 > 0x1A;
    load fuse 0x22A78E77 > 0x1B;
    load fuse 0x01019c82 > 0x1C;
    load fuse 0xFC3AC699 > 0x1D;
    load fuse 0xF2C327A3 > 0x1E;
    load fuse 0xDAC9214E > 0x1F;

    # Program SEC_CONFIG to enable HAB closed mode
    load fuse 0x00000002 > 0x06;

}
/*****file end*****/

```

2. Generate SB file using enable\_hab.bd

Command:

```
elftosb.exe -f kinetis -V -c enable_hab.bd -o enable_hab.sb
```

3. Modify the ucl2.xml file

Open the ucl2.xml under the “<Device>/OS Firmware/” folder and add the following code.

```

/*****code start*****/
<UCL>
.....
<LIST name="MXRT105x-Burnfuse" desc="Boot Flashloader">
<!-- Stage 1, load and execute Flashloader -->
  <CMD state="BootStrap" type="boot" body="BootStrap" file="ivt_flashloader.bin" > Loading Flashloader. </CMD>
  <CMD state="BootStrap" type="jump" onError = "ignore"> Jumping to Flashloader. </CMD>

<!-- Stage 2, Program boot image into external memory using Flashloader -->
  <CMD state="Blhost" type="blhost" body="get-property 1" > Get Property 1. </CMD> <!--Used to test if flashloader runs
successfully-->
  <CMD state="Blhost" type="blhost" timeout="15000" body="receive-sb-file \"Profiles\\MXRT105X\\OS
Firmware\\enable_hab.sb\" > Program Boot Image. </CMD>

```

**How to use i.MXRT Security Boot, Application Notes, Rev. 0, 10/2017**

```
<CMD state="Blhost" type="blhost" body="Update Completed!">Done</CMD>
```

```
</LIST>
```

```
</UCL>
```

```
/******code end *****/
```

Figure 20 is the example.

```
* "boot" - Finds configured device, forces it to "body" device and downloads "file".
* "find" - Waits for "timeout" seconds for the "body" device to connect.
* "blhost" - Executes a blhost command.
-->

<UCL>

<CFG>
  <STATE name="BootStrap" dev="MXRT105X" vid="1FC9" pid="0130"/> <!-- I.MX SDF USB-HID -->
  <STATE name="Blhost" dev="KBL-HID" vid="15A2" pid="0073"/> <!-- KIBBLE USB-HID -->
</CFG>

<LIST name="MXRT105X-DevBoot" desc="Boot Flashloader">
  <!-- Stage 1, load and execute Flashloader -->
  <CMD state="BootStrap" type="boot" body="BootStrap" file="ivt_flashloader.bin" > Loading Flashloader. </CMD>
  <CMD state="BootStrap" type="jump" onError = "ignore"> Jumping to Flashloader. </CMD>

  <!-- Stage 2, Program boot image into external memory using Flashloader -->
  <CMD state="Blhost" type="blhost" body="get-property 1" > Get Property 1. </CMD> <!--Used to test if flashloader runs successfully-->
  <CMD state="Blhost" type="blhost" timeout="15000" body="receive-sb-file \"Profiles\\MXRT105X\\OS Firmware\\boot_image.sb\"\" > Program Boot Image. </CMD>
  <CMD state="Blhost" type="blhost" body="Update Completed!">Done</CMD>

</LIST>

<LIST name="MXRT105X-Burnfuse" desc="Boot Flashloader">
  <!-- Stage 1, load and execute Flashloader -->
  <CMD state="BootStrap" type="boot" body="BootStrap" file="ivt_flashloader.bin" > Loading Flashloader. </CMD>
  <CMD state="BootStrap" type="jump" onError = "ignore"> Jumping to Flashloader. </CMD>

  <!-- Stage 2, Program boot image into external memory using Flashloader -->
  <CMD state="Blhost" type="blhost" body="get-property 1" > Get Property 1. </CMD> <!--Used to test if flashloader runs successfully-->
  <CMD state="Blhost" type="blhost" timeout="15000" body="receive-sb-file \"Profiles\\MXRT105X\\OS Firmware\\enable_hab.sb\"\" > Program Boot Image. </CMD>
  <CMD state="Blhost" type="blhost" body="Update Completed!">Done</CMD>

</LIST>

<LIST name="MXRT105X-SecureBoot" desc="Boot Signed Flashloader">
  <!-- Stage 1, load and execute Flashloader -->
  <CMD state="BootStrap" type="boot" body="BootStrap" file="ivt_flashloader_signed.bin" > Loading Flashloader. </CMD>
  <CMD state="BootStrap" type="jump" onError="ignore"> Jumping to Flashloader. </CMD>

  <!-- Stage 2, Enable HAB closed mode using Flashloader -->
  <CMD state="Blhost" type="blhost" body="get-property 1" ifhab="Open" > Get Property 1. </CMD> <!--Used to test if flashloader runs successfully-->
  <CMD state="Blhost" type="blhost" body="receive-sb-file \"Profiles\\MXRT105X\\OS Firmware\\enable_hab.sb\"\" ifhab="Open" > Program Boot Image. </CMD>
  <CMD state="Blhost" type="blhost" body="reset" ifhab="Open"> Reset. </CMD> <!--Reset device to enable HAB Close Mode-->

  <!-- Stage 3, Program signed image into external memory using Flashloader -->
  <CMD state="Blhost" type="blhost" body="get-property 1" ifhab="Close" > Get Property 1. </CMD> <!--Used to test if flashloader runs successfully-->
  <CMD state="Blhost" type="blhost" timeout="15000" body="receive-sb-file \"Profiles\\MXRT105X\\OS Firmware\\boot_image.sb\"\" ifhab="Close" > Program Boot Image. </CMD>
  <CMD state="Blhost" type="blhost" body="Update Completed!" ifhab="Close" >Done</CMD>

</LIST>

</UCL>
```

Figure 20. The example of adding the burning fuse function

Then you can just configure the “name” field in cfg.ini file as <Device>-Burnfuse and run the mfgtool.

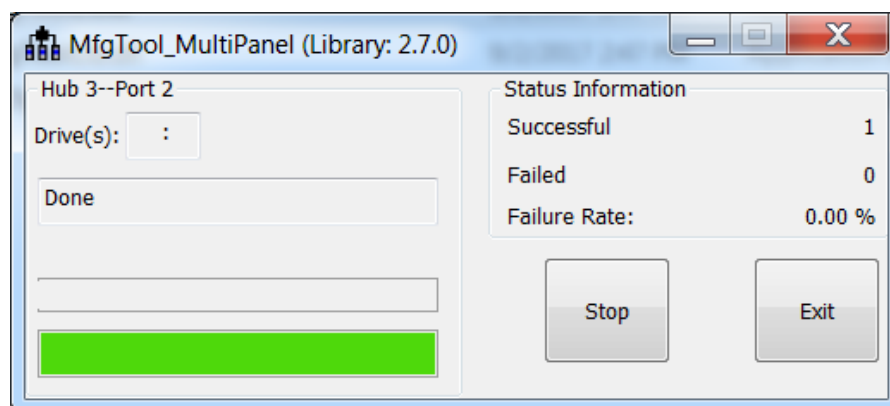


Figure 21. Successful result for Burn fuse p

## 4.4. Program image for production

In Production phase, the device may be under HAB closed mode for most use cases, users can just configure the “name” field in cfg.ini file as <Device>-SecureBoot, then prepare the boot\_image.sb file, enable\_hab.sb and ivt\_flashloader\_signed.bin using elftosb utility, after all are generated, place them into “<Device>/OS Firmware/” folder, then switch device into serial downloader, connect it to host PC, finally, Open MfgTool.exe and click “Start” to trigger a provisioning sequence, after the provisioning completes, you will see below window. To exit MfgTool, click “Stop” and “Exit” in turn.

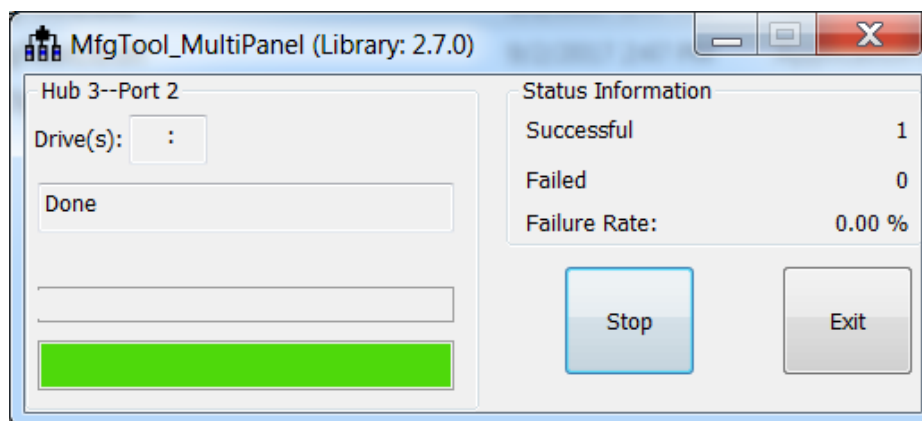


Figure 22. Successful result for provisioning with MfgTool for SecureBoot

## 5. Revision History

[Table 2](#) is the revision history table.

Table 2. Sample revision history

Revision number	Date	Substantive changes
0	10/2017	Initial release

### NOTE

For more detail and further information about HAB please to read [Reference Material](#).

How to use i.MXRT Security Boot, Application Notes, Rev. 0, 10/2017

## 6. Reference Material

- i.MX 6 Linux® High Assurance Boot (HAB) User's Guide [2] FRDM-KL26Z\_QSG, Freescale. Rev. L3.14.28\_1.0.0-ga, 04/2015
- HAB Code-Signing Tool User's Guide. Rev. 2.3.2 3/2016
- MX6UL HAB (High Assurance Boot), <https://community.nxp.com/docs/DOC-275249>
- i.MX6UL Encrypted Boot Supported by Mfgtools. <https://community.nxp.com/docs/DOC-333174>
- i.MX MCU Manufacturing User's Guide
- Kinetis blhost User's Guide
- Manufacturing Tool v.2.0.0 (MfgTool2) for Kinetis Bootloader User's Guide
- MCUX Flashloader Reference Manual





---

**How to Reach Us:**

**Home Page:**

[nxp.com](http://nxp.com)

**Web Support:**

[nxp.com/support](http://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](http://nxp.com/SalesTermsandConditions).

NXP, the NXP logo, Freescale, the Freescale logo, and Kinetis, are trademarks of NXP B.V. All other product or service names are the property of their respective owners. All rights reserved.

© 2017 NXP B.V.

Document Number: AN12079

Rev. 0

10/2017

