RT1050 HAB Encrypted Image Generation and Analysis

1, Introduction

The NXP RT series can support multiple boot modes, it incluses: unsigned image mode, HAB signed image mode, HAB encryption image mode, and BEE encryption image mode.

In order to understand the specific structure of the HAB encryption app, this article will generate a non-XIP app image, then generate the relevant burning file through the elftosb.exe tool in the flashloader i.MX-RT1050, and use MFGTOOL to enter the serial download mode to download the .sb file.

This article will focus on the download steps of RT1050 HAB encryption related operations, and analyze the structure of the HAB encrypted app image.

2, RT1050 HAB Encypted Operation Procedure

At first, we analyze the steps of MFGtool burning, which files are needed, so as to give specific preparation, open the ucl2.xml file in the following path of the flashloader:

Flashloader_i.MXRT1050_GA\Flashloader_RT1050_1.1\Tools\mfgtools-rel\Profiles\MXRT105X\OS Firmware

Because we need to use the HAB encrypated boot mode, then we will use MXRT105X-SecureBoot, from the ucl2.xml file, we will find the following related code:

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

Fig 1. MXRT1050-SecureBoot structure

As you can see from the above, to implement the secure boot of RT1050, you need to prepare these three files:

- ivt_flashlloader_signed.bin: it is the signed flashloader binary file
- enable_hab.sb: it is used to modify the SRK and HABmode in the fuse map
- boot_image.sb: HAB encrypted app program file

Here is a flow chart of the overall HAB encryption operation step, after checking this figure, then we will follow it step by step.

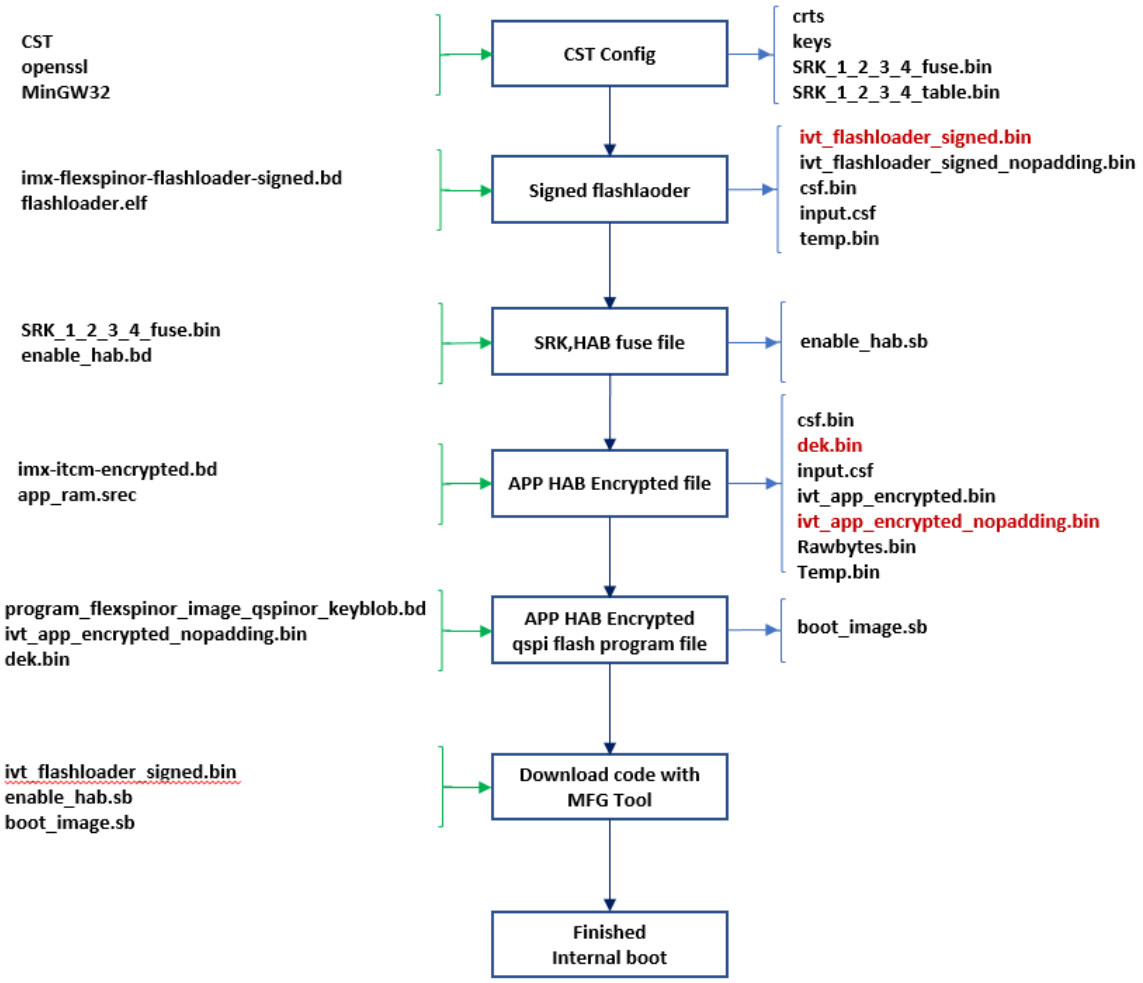

Fig 2. MXRT1050 HAB encrypted image flow chart

The app image we used in this article is the RAM app, so, at first, we need to prepare one RAM based app image. In this document, we are directly use the prepared RAM based app image: evkbimxrt1050_led_softwarereset_0xa000.s19, this app code function is: After download the code to the MIMXRT1050-EVKB(qspi flash) board, the on board led D18 will blinky and printf the information, after pressing the WAKEUP button SW8, the code will implement software reset and printf the related information. The unsigned code test print result are as follows:

```
BOARD RESET start.
 Helloworld.
WAKEUP key pressed, will do software system reset.


 BOARD RESET start.
 Helloworld.
```

2.1 CST tool preparation

Because the contains a lot of steps, then customer can refer to the following document do the related configuration, this document, we won't give the CST configuration detail steps. Please check these documents:
https://www.cnblogs.com/henjay724/p/10219459.html
https://community.nxp.com/docs/DOC-340904
Security Application Note AN12079
After the CST tool configuration, please copy the cst.exe, crts filder, key folder from cst folder to the same folder that holds elftosb executable files:
Flashloader_i.MXRT1050_GA\Flashloader_RT1050_1.1\Tools\elftosb\win
Please also copy SRK_1_2_3_4_fuse.bin and SRK_1_2_3_4_table.bin to the above folder.

2.2 Sign flashloader

Please refer to application note AN12079 chapter 3.3.1, copy flashloader.elf from folder path:
Flashloader_i.MXRT1050_GA\Flashloader_RT1050_1.1\Flashloader
And the imx-flexspinor-normal-signed.bd from folder path:
Flashloader_i.MXRT1050_GA\Flashloader_RT1050_1.1\Tools\bd_file\imx10xx
to the folder:
Flashloader_i.MXRT1050_GA\Flashloader_RT1050_1.1\Tools\elftosb\win
Please open commander window under the elftosb folder, then input this commander:

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

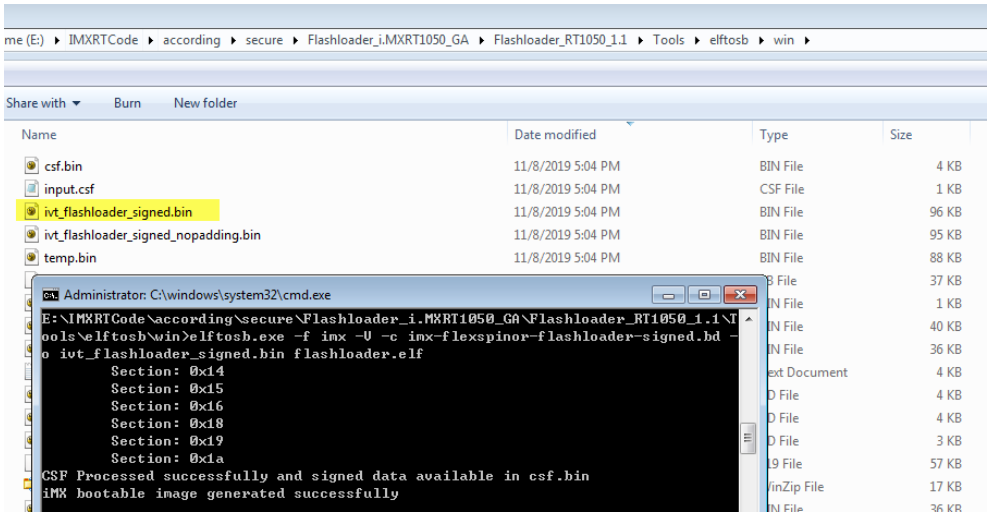

Fig 3. Sign flashloader

This steps will generate the ivt_flashlaoder_signed.bin, which is needed to put under the MFGtool OS Firmware folder, just used for enter the signed flashloader mode.

2.3 SRK and HAB mode fuse modification files

Please refer to AN12079 chapter 4.3, copy the enable_hab.bd file from folder path:
Flashloader_i.MXRT1050_GA\Flashloader_RT1050_1.1\Tools\bd_file\imx10xx

to this folder path:
Flashloader_i.MXRT1050_GA\Flashloader_RT1050_1.1\Tools\elftosb\win
Please refer to the chapter 2.1 generated SRK_1_2_3_4_fuse.bin, modify the enable_hab.bd like the following picture:

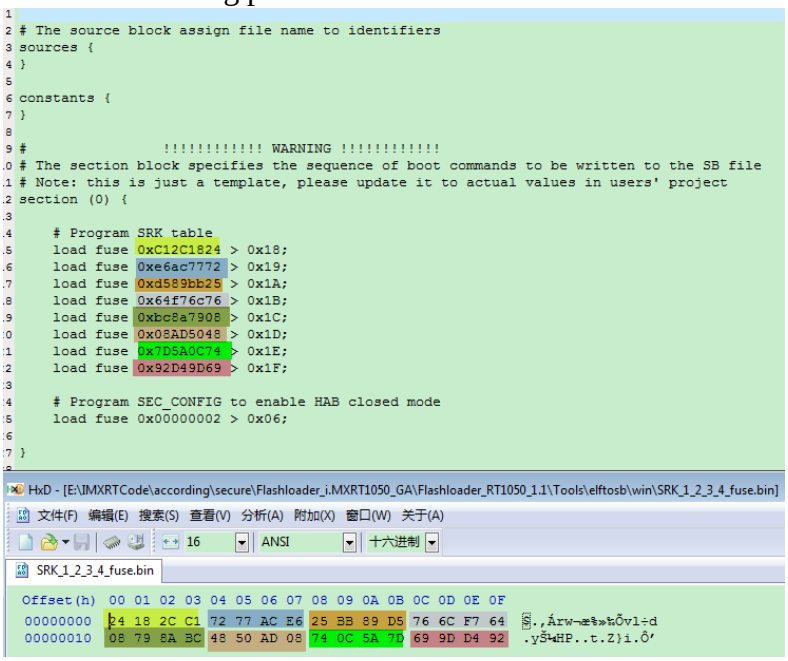

Fig 4. enable_hab.bd SRK and HAB mode fuse modification

Then, in the elftosb window, please input the following command, just used to generate the enable_hab.sb program file:

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

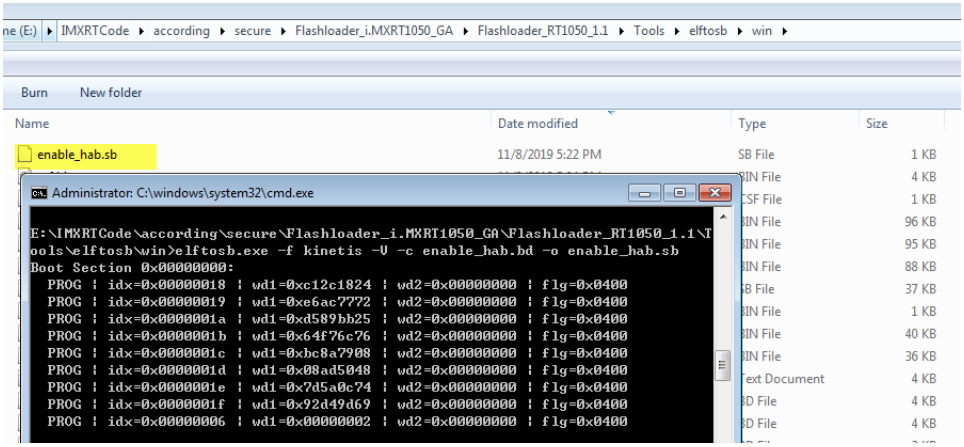

Fig 5. SRK and HAB mode program files generation

2.4 APP Encrypted Image

If you want to do the HAB encrypted image download, you need to prepare one non-XIP app image, here we prepared one RAM based APP srec files.

Because the app file is the RAM files, then we also need the related RAM encrypted .bd files, please copy imx-itcm-encrypted.bd from the folder path:
Flashloader_i.MXRT1050_GA\Flashloader_RT1050_1.1\Tools\bd_file\imx10xx
to this folder path：
Flashloader_i.MXRT1050_GA\Flashloader_RT1050_1.1\Tools\elftosb\win
Open imx-itcm-encrypted.bd, then modify the following content:

```
options {
   flags = 0x0c;
   # Note: This is an example address, it can be any non-zero address in ITCM region
   startAddress = 0x8000;
   ivtOffset = 0x1000;
   initialLoadSize = 0x2000;
   # Note: This is required if the cst and elftsb are not in the same folder
   # Note: This is required if the default entrypoint is not the Reset_Handler
   #       Please set the entryPointAddress to Reset_Handler address
 entryPointAddress = 0x0000a2dd;
}
```

Here, we need to note these two points:

(1) ivtOffset = 0x1000;

If the external flash is flexspi flash, then we need to modify ivtOffset as 0X1000, if it is the nandflash, we need to use the 0X400.

(2) entryPointAddress = 0x0000a2dd;

The entryPointsAddress should be the app code reset handlder, it is the app start address+4 data, the entry address is also OK, but we suggest you to use the app Reset_Handler address.

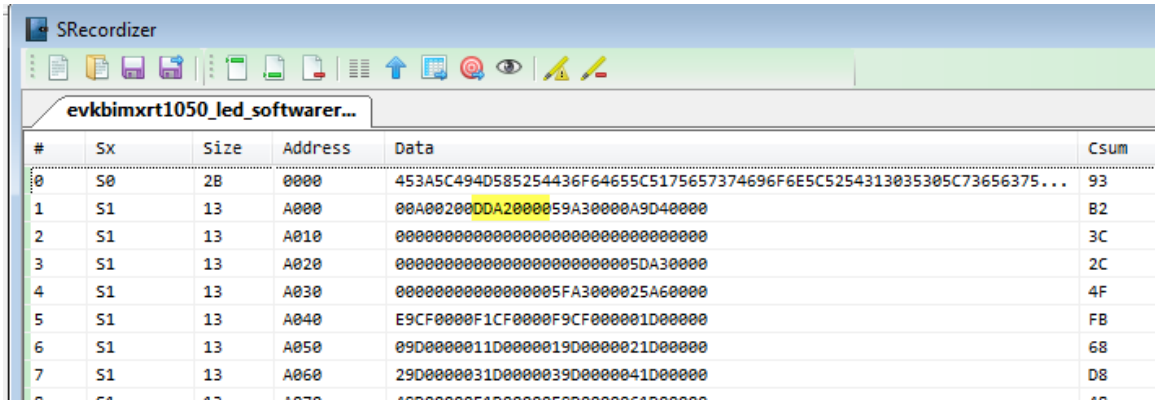

Fig 6. App reset handler address

Then input the following commander in the elftosb windows:

```
elftosb.exe -f imx -V -c imx-itcm-encrypted.bd -o ivt_evkbimxrt1050_led_softwarereset_0xa000_encrypted.bin
evkbimxrt1050_led_softwarereset_0xa000.s19
```

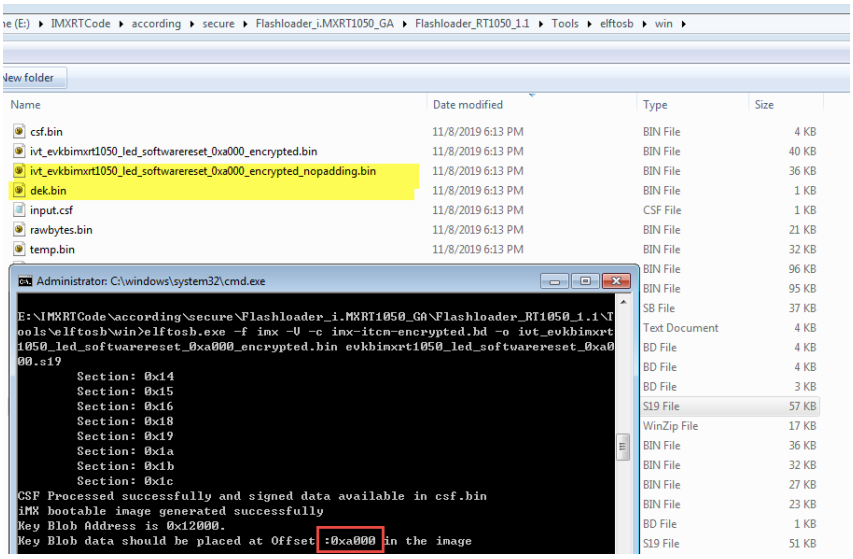

Fig 7. App HAB Encrypted file generation

Please note, we need to record the generated key blob offset address, it is 0XA00, just like the above data in the red frame, this address will be used in the next chapter's .bd file. After this step, it will generate 7 files:

(1) ivt_evkbimxrt1050_led_softwarereset_0xa000_encrypted.bin, this file includes the FDCB which is filled with 0, IVT, BD, DCD, APP HAB encrypted image data, CSF data.

(2) ivt_evkbimxrt1050_led_softwarereset_0xa000_encrypted_nopadding.bin, compare with ivt_evkbimxrt1050_led_softwarereset_0xa000_encrypted.bin, this file deletes the 0s which is above IVT range.

(3) Csf.bin, it is the HAB data area, you can find the data contains the csf data, it is from 0X8000 to 0X8F80 in the generated ivt_evkbimxrt1050_led_softwarereset_0xa000_encrypted.bin.

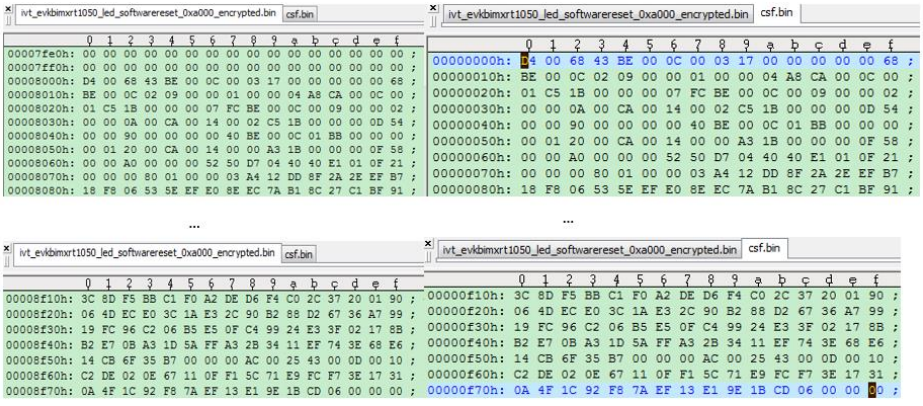

Fig 8. Csf data and the encrypted app relationship

(4) dek.bin，

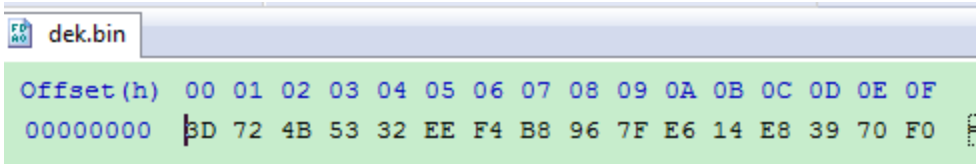

Fig 9. Dek data

DEK data is the AES-128 bits key, it is not defined by the customer, it is random generated automatically by the HAB encrypted tool.

(5) input.csf

Open it, you can find the following content:

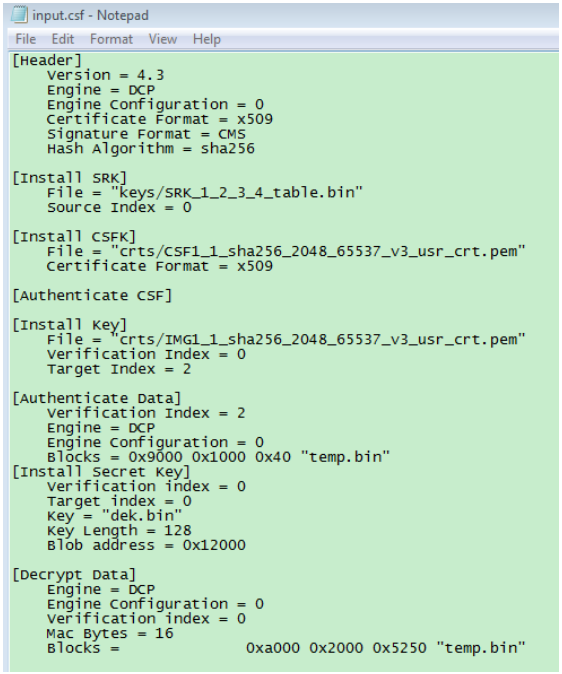

Fig10. Input csf file content

(6) rawbytes.bin, this is the app image plaintext data, it doesn't contains the FDCB,IVT,BOOTDATA, DCD, csf etc.

(7) temp.bin, it is the temperate file, compare with ivt_evkbimxrt1050_led_softwarereset_0xa000_encrypted.bin, no csf files.

2.5 HAB Encrypted QSPI program file

Here we need to prepare one program_flexspinor_image_qspinor_keyblob.bd file, and put it under the same folder as elftosb, this file is used to generate the HAB encrypted program .sb file. Because the flashloader package didn't contains it, then we paste all the related content, and I will also attach it in the attachment.

```
# The source block assign file name to identifiers
sources {
 myBinFile = extern (0);
 dekFile = extern (1);
}
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
   # 0xc0000006 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] Quad Mode Entry Setting
   #          0 - Not Configured, apply to devices:
   #              - With Quad Mode enabled by default or
   #              - Compliant with JESD216A/B or later revision
   #          1 - Set bit 6 in Status Register 1
   #          2 - Set bit 1 in Status Register 2
   #          3 - Set bit 7 in Status Register 2
   #          4 - Set bit 1 in Status Register 2 by 0x31 command
   # bit [07: 04]  Misc. control field
   #          3 - Data Order swapped, used for Macronix OctaFLASH devcies only (except MX25UM51345G)
   #          4 - Second QSPI NOR Pinmux
   # bit [03: 00] Flash Frequency, device specific
   load 0xc0000006 > 0x2000;
   # Configure QSPI NOR FLASH using option a address 0x2000
   enable flexspinor 0x2000;

   #2 Erase flash as needed.
   erase 0x60000000..0x60020000;

   #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_Ivt;

                #6. Generate KeyBlob and program it to flexspinor
    # 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 flexspinor
    # 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 0x0000a000 > 0x10208;
    enable flexspinor 0x10200;

    #7. Program KeyBlob to firmware0 region
    load 0xb1000000 > 0x10300;
    enable flexspinor 0x10300;
}
```

Please note, in the above chapter, fig 7, we mentioned the keyblob offset address, we need to modify it in the following code:

load **0x0000a000** > 0x10208;

Now, combine program_flexspinor_image_qspinor_keyblob.bd, ivt_evkbimxrt1050_led_softwarereset_0xa000_encrypted_nopadding.bin

and dek.bin file together, we use the following commander to generate the boot_image.sb:

elftosb.exe -f kinetis -V -c program_flexspinor_image_qspinor_keyblob.bd -o boot_image.sb ivt_evkbimxrt1050_led_softwarereset_0xa000_encrypted_nopadding.bin dek.bin

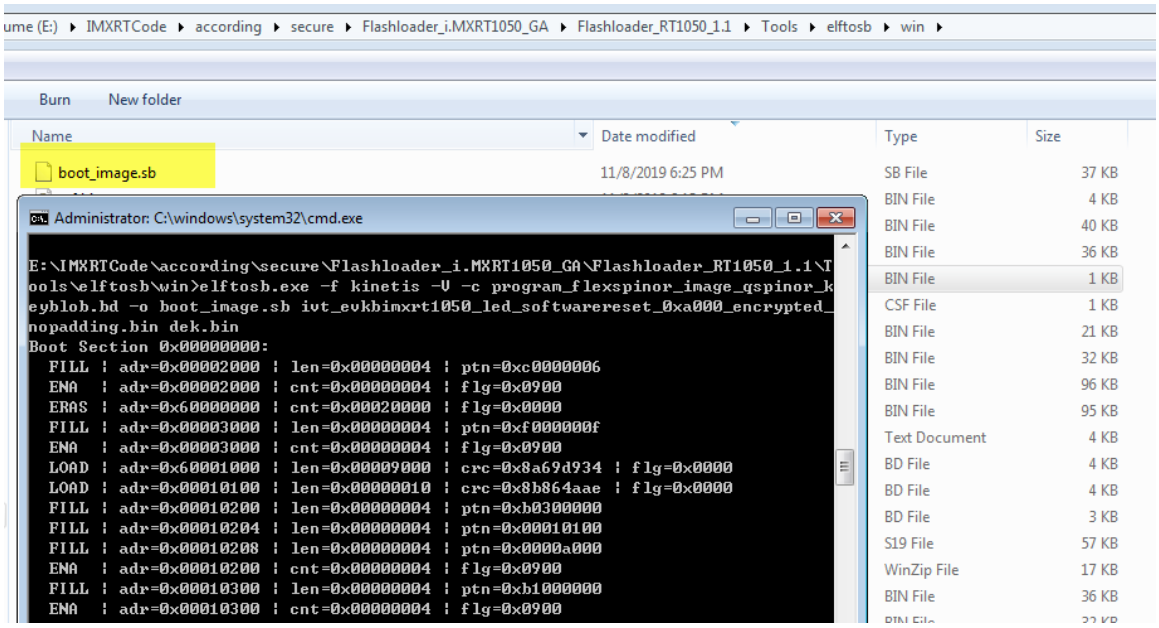

Fig 11. App HAB encrypted program file generation

Until now, we will find, all the related HAB encrypted files is prepared.

2.6 MFG Tool program HAB Encrypted files to RT1050-EVKB

Before we program it, please copy the following 3 files which is in the elftosb folder:

ivt_flashloader_signed.bin

enable_hab.sb

boot_image.sb

to this folder: Flashloader_i.MXRT1050_GA\Flashloader_RT1050_1.1\Tools\mfgtools-rel\Profiles\MXRT105X\OS Firmware

Please modify cfg.ini, the file path is:

Flashloader_i.MXRT1050_GA\Flashloader_RT1050_1.1\Tools\mfgtools-rel

Modify the content as:

[profiles]

chip = MXRT105X

[platform]

board =

[LIST]

name = MXRT105X-SecureBoot

Choose MXRT105X-SecureBoot program mode.

Then open the tool MfgTool2.exe, the board MIMXRT1050-EVKB(need to modify the on board resistor, use the qspi flash) mode should be serial download mode, just modify SW7:1-OFF,2-OFF,3-OFF, 4-ON, connect two usb cable between PC and the board J28 and J9. After the connection, you will find the MfgTool2.exe can detect the HID device:

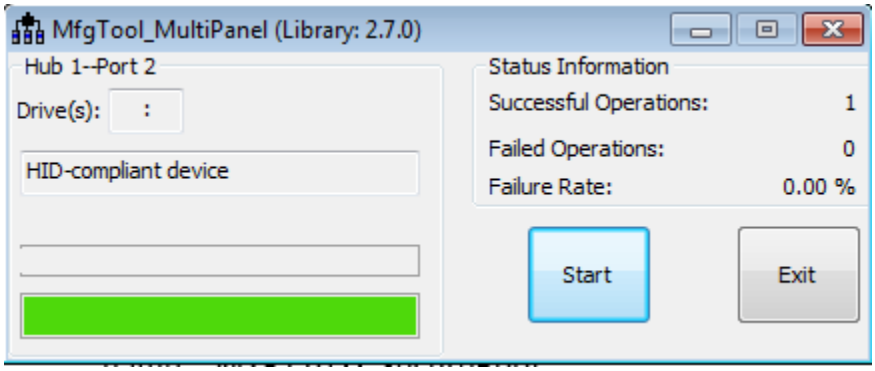

Fig 12. MFG tool program

After the program is finished, power off the board, modify the boot mode to internal boot, it is SW7:1-OFF,2-OFF,3-ON, 4-OFF， connect the COM terminal, power on the EVKB board, after reset, you will find the D18 led is blinking, after you press the SW8, you will find the following printf information:

```
BOARD RESET start.
 Helloworld.
WAKEUP key pressed, will do software system reset.

?
 BOARD RESET start.
 Helloworld.
```

So, the HAB encrypted image works OK now.

3. app HAB encrypted image structure analysis

3.1 MCUBootUtility Configuration to check the RT Encrypted image

Here, we can also use MCUBootUtility tool to check the RT chip encrypted image and the fuse data.

If the cst is your own configured, please do the following configuration at first:

（1）Copy the configured cst folder to folder:

NXP-MCUBootUtility-2.0.0\tools

Delete the original cst folder.

（2）Copy SRK_1_2_3_4_fuse.bin and SRK_1_2_3_4_table.bin to folder:

NXP-MCUBootUtility-2.0.0\gen\hab_cert

Now, you can use the new MCUBootutility to connect your board which already done the HAB encrypted method.

3.1 RT1050 fuse map comparation

Before do the HAB encrypted image program, I have read out the whole fuse map as follows:

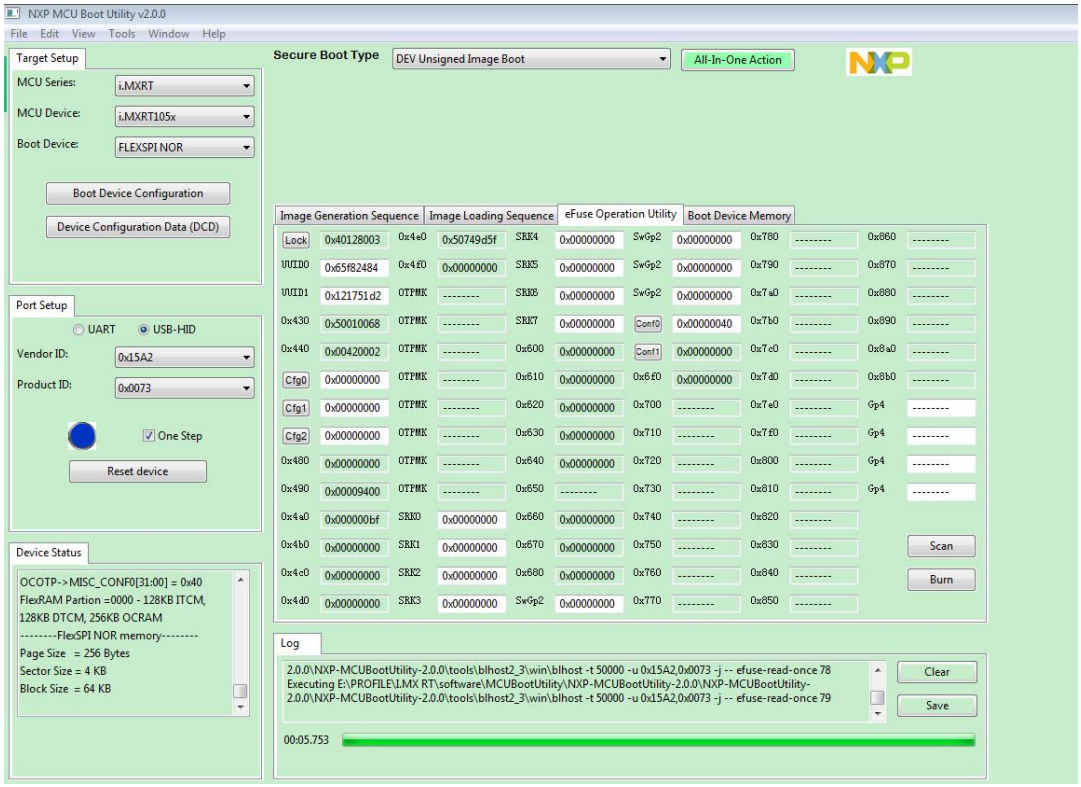

Fig 13. MIMXRT1050-EVKB fuse map before HAB encrypted image

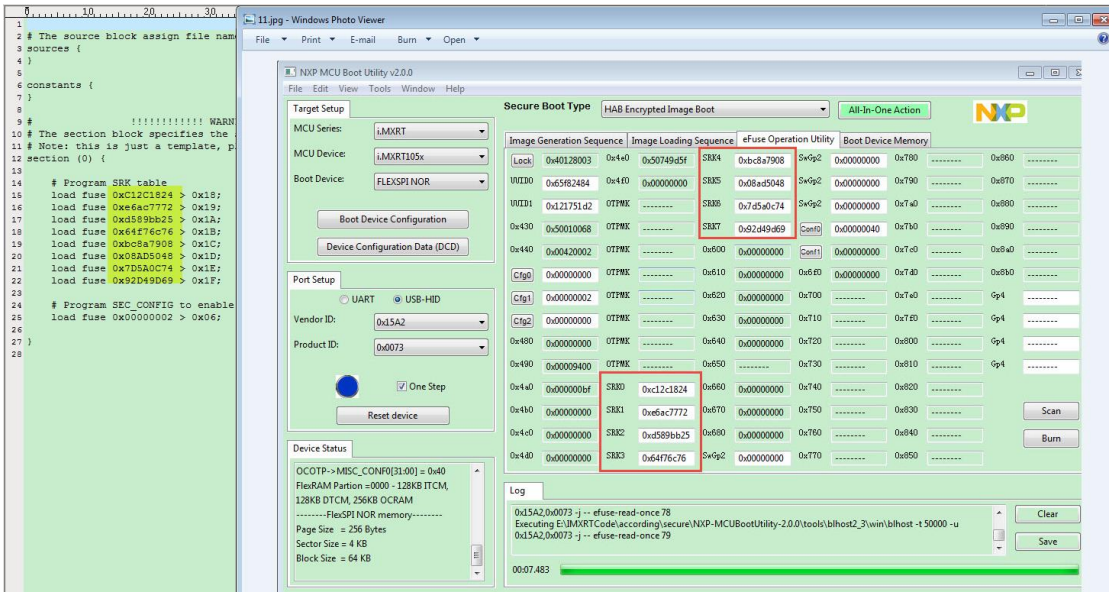

Fig 14. MIMXRT1050-EVKB fuse map after HAB encrypted image

Compare the fuse map between do the HAB encrypted image and no HAB encrypted image, we can find two difference:

- HAB mode, 0X460 bit1：0 open， 1 close
- SRK area

We can find, after program the HAB encrypted image, the SRK fuse data is the same as the SRK data which is defined in the enable_hab.bd.

3.2 Readout HAB encrypted QSPI APP image structure analysis

From MCUBootUtility tool, we can find the HAB Encypted image structure should be like this:

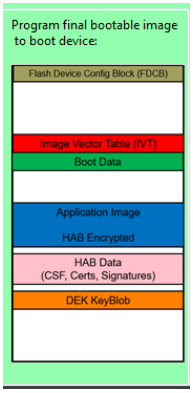

Fig 15. HAB Encrypted image structure

What about the real example image case? Now, we use the MCUbootUtility tool to read out our HAB encrypted image, from address 0X60000000, the readout size is 0XB000. The detail image structure is like following:

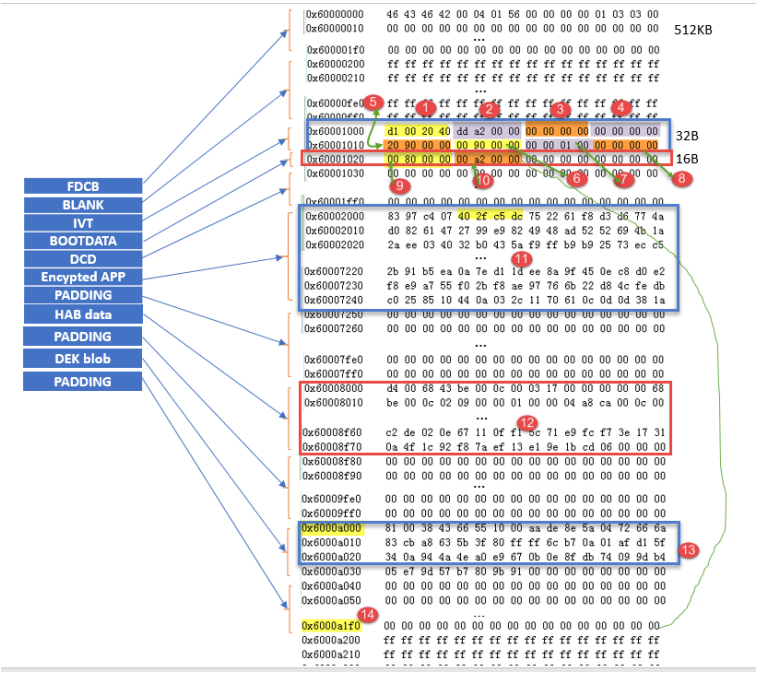

Fig 16. HAB Encypted image example structure

1): IVT: hdr, IVT header, more details, check hab_hdr

2): IVT: entry, the app entrypointAddress, it should be the reset_handler address, in this document example, it is the address 0xa004 data, the plaintext is 0X00A2DD, but after the HAB encrypted, we can find the address -x60002004 data is the encrypted data

3): IVT: reserved
4): IVT: DCD, it is used for the DRAM SEMC configuration, in this example, we didn't use the SDDRAM, so the data is 0.
5): IVT: BOOT_DATA, used to indicate the BOOT_DATA RAM start address 0X9020.
6): IVT: self, ivt self RAM start address is 0X9000
7): IVT:CSF, it is used to indicate the CST start address, this example csf ram address is 0X00010000.
8): IVT:reserved
9): BOOT_DATA: RAM image start, the whole image RAM start address, this RAM example BOOT_DATA is 0X8000,0XA000-0X2000=0X8000
10): BOOT_DATA: size, APP while size, it is 0X0000A200, after checking the while generated HAB encrypted app image size, you can find the image end size is really 0XA200, just lke the fig 16.
11): HAB Encypted app data, please check ivt_evkbimxrt1050_led_softwarereset_0xa000_encrypted.bin file, the address 0X2000-0X7250 data, you will find it is the same.

12): HAB data, it incluses the csf, certificate etc data, you can compare the file ivt_evkbimxrt1050_led_softwarereset_0xa000_encrypted.bin address 0X8000-0x8f70 data, it is the same.
13):DEK blob, it is the DEK key blob related data, the offset address is 0XA000, the same as fig 7.
FDCB,IVT,BOOT DATA are all plaintext, but app image area is the HAB encrypted data, HAB and the DEK blocb is the generated data put in the related memory.

4. Conclusion

This document we mainly use the elftosb and the MFGTool to generate the HAB encrypted image, and download it to the RT1050 EVKB board, document give the whole detail steps, and us ethe MCUBootutility tool to read out the HAB encrypted image, and analysis the HAB encrypted image structure with the examples. After compare with the generated mid files, we can find all the data is consist, and all the encrypted data range is the same. The test result also demonstrate the HAB encrypted code function works, the HAB encrypted boot has no problems. All the related files is in the attachment.