Encrypted QuadSPI image Implementation

➢ Introduction

The Kinetis family of MCU includes the system security and flash protection features that can be used to protect code and data from unauthorized access or modification. This application note discusses the usage of encrypted boot with KBOOT. To use the encrypted boot feature, you must have basic knowledge of the following terms and abbreviations.

Table 1

Terminology	Description
QCB	QuadSPI Configuration Block, a structure containing configurable parameters needed by the Kinetis bootloader to configure the QuadSPI controller.
KeyBlob	A data structure that contains up to four groups of KeyBlob entries. Each entry consists of the start address, length, decryption key, and counter of an encrypted QuadSPI memory region
KEK	KeyBlob Encryption Key, an AES-128 key used for encrypting plaintext KeyBlob and decrypting encrypted KeyBlob.
SB key	The SB key is an AES-128 key pre-

	programmed into flash's IFR region at word offsets 0x30 to 0x33. Elftosb allows generation of encrypted SB file image using the SB key
QuadSPI	QuadSPI block acts as an interface to one single or two external serial flash devices, each with up to eight bidirectional data lines and supports execute-in-place (XIP) for external SPI flash memory devices
OTFAD	On-the-fly AES Decryption is a powerful IP block in MK81F256 and MK82F256, which supports decryption of the encrypted QuadSPI image on-the-fly using KeyBlob

In general, the QuadSPI image is encrypted using the parameters in the KeyBlob with AES-128 CTR mode, the KeyBlob Block itself encrypted with KEK, and the SB file is encrypted via SB key with AES-128 CBC-MAC. The following figure provides an encrypted SB file containing an encrypted QuadSPI image and the entire content of the SB file is obfuscated. The rest of the application provides the step-by-step instructions on programming keys, generating encrypted QuadSPI image data in the SB file, and encrypting the entire SB file image with the SB key.

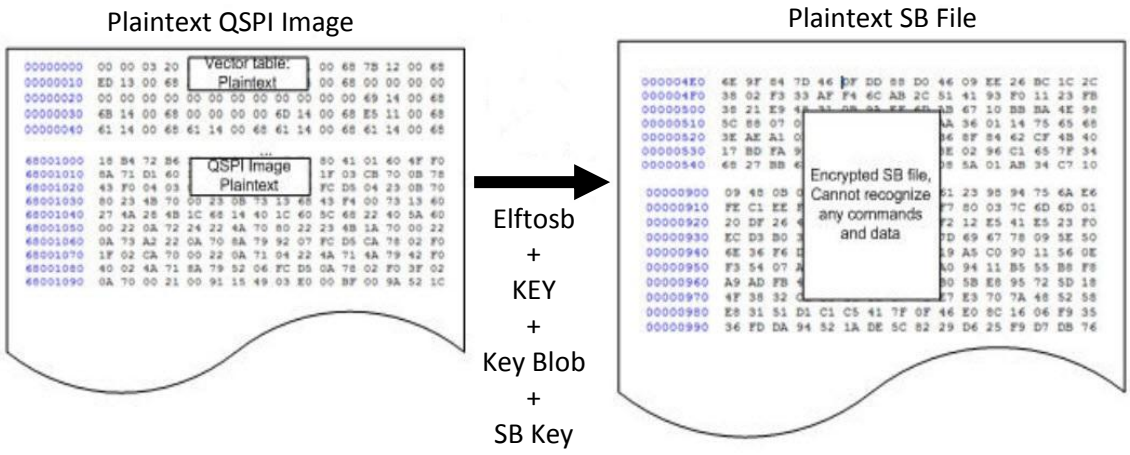

Fig 1 Encrypted SB file with Encrypted QuadSPI image

➢ Generate an SB file with KEK and SB KEY

KEK and SB KEY both are 16 byte array, and they need to be pre-programmed into flash' s IFR region. The generated SB file can be provisioned using Kinetis bootloader to program the keys into IFR region of the device. For instance,

SB KEY:

uint8_t sbKey[16] = {0x00, 0x11, 0x22, 0x33, 0x44, 0x55, 0x66, 0x77, 0x88, 0x99, 0xaa, 0xbb,0xcc, 0xdd, 0xee, 0xff}.

KEK:

uint8_t kek[16] = {0x00, 0x01, 0x02, 0x03, 0x04, 0x05, 0x06, 0x07, 0x08, 0x09, 0x0a, 0x0b,0x0c, 0x0d, 0x0e, 0x0f}.

To generate SB file, a specified BD file needs to be generated first, assuming the BD file is called "program_keys.bd".

Table 2

```
# No source file needed, keep this block empty
sources {
}

# The section block specifies the sequence of boot commands to be
```

```
written to the SB file.
section (0) {

   # Use the 'load ifr' statement to program the SB key to IFR memory.
   # The SB key occupies IFR index 0x30-0x33.
   # The SB key is 128-bit specifies as 4 little-endian long-word values.
   # SB Key = {0x00, 0x11, 0x22, 0x33, 0x44, 0x55, 0x66, 0x77, 0x88,
0x99, 0xaa, 0xbb, 0xcc, 0xdd, 0xee, 0xff}
   load ifr 0x33221100 > 0x30;
   load ifr 0x77665544 > 0x31;
   load ifr 0xbbaa9988 > 0x32;
   load ifr 0xffeeddcc > 0x33;

   # Use the 'load ifr' statement to program the OTFAD KEK to IFR
memory.
   # The KEK is used to unwrap(decrypt) the keyblob at boot time in
order to correctly set up the OTFAD engine.
   # The key is specified as 4 little-endian values, with the "least
significant" key word going into the lowest IFR index
   # KEK = {0x00, 0x01, 0x02, 0x03, 0x04, 0x05, 0x06, 0x07, 0x08, 0x09,
0x0a, 0x0b, 0x0c, 0x0d, 0x0e, 0x0f}
   load ifr 0x0f0e0d0c > 0x20;
   load ifr 0x0b0a0908 > 0x21;
   load ifr 0x07060504 > 0x22;
   load ifr 0x03020100 > 0x23;

   # Reset target in order to let these keys take effect.
   reset;
}
```

Using elftosb, the desired SB file is generated. The elftosb command line and output is shown in the following figure.

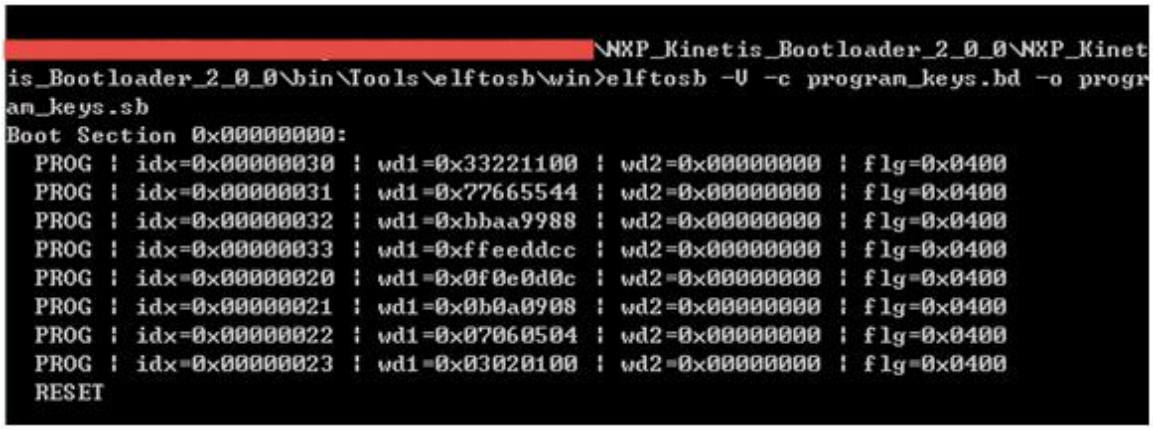

Fig 1 Generate program_keys.sb

The blhost can be used to flash the SB file to the target device.

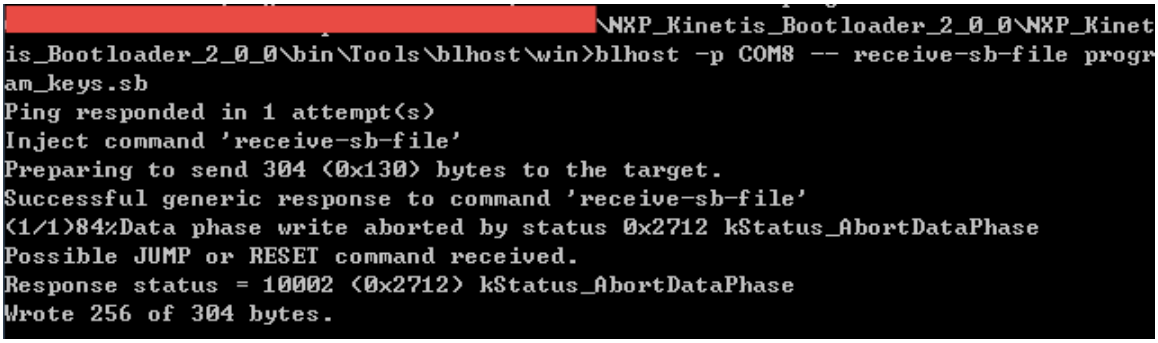

Fig 2 Flash SB file with blhost

➢ Generate an SB file with encrypted QuadSPI image

After the previous operations, another SB file which contains the encrypted QuadSPI image is still needed. Similar as to how the SB file was generated in the previous section, and a BD file is needed to describe all the operations in this SB file which should contains the Key Blob Block, encrypted QuadSPI image and Key Blob encryption wrapper. The following table illustrates the details of the BD file.

Table 3

```
# The sources block assigns file names to identifiers
sources {
   # SREC File path
   mySrecFile = "led_demo_QSPI_0000.srec";
   # QCB file path
   qspiConfigBlock = "qspi_config_block.bin";
}

# The keyblob creates a structure with up to 4 keyblob entries.
# The empty parentheses syntax specifies an entry of all zeros (no
encryption)
# Each entry consists of 4 parameters:
# start     - start address of encrypted block.
# end       - end address of encrypted block.
# key       - AES-CTR mode encryption key for this range.
# counter   - initial counter value for AES-CTR encryption for this
range.
keyblob (0) {
   (
      start=0x68001000,
      end = 0x68001fff,
      key="000102030405060708090a0b0c0d0e0f",
      counter="0123456789abcdef"
   )
   ()
   ()
   ()
}


# The section block specifies the sequence of boot commands to be
```

```
written to
# the SB file
section (0) {

   #1. Erase the vector table and flash config field.
   erase 0..0x800;

   # Step 2 and Step 3 are optional if the QuadSPI is configured at
startup.
   #2. Load the QCB to RAM
   load qspiConfigBlock > 0x20000000;

   #3. Configure QuadSPI with the QCB above
   enable qspi 0x20000000;

   #4. Erase the QuadSPI memory region before programming.
   erase 0x68000000..0x68004000;

   #5.Load the QCB above to the start address of QuadSPI memory
   load qspiConfigBlock > 0x68000000;

   #6,7. The encrypt statement indicates that load commands should
encrypt load from the srec file using AES-CTR mode encrypteion.
   # The encrypt argument (0) specifies the keyblob parameters range
of one of the keyblob entries are left unencrypted.
   encrypt(0) {
      # Load all the RO data from SREC fiel.
      load mySrecFile;
   }

   #8. Load the encrypted keyblob block to specified location.
```

Entire image including encrypted QuadSPI image

```
    # The keywrap statement wraps (encrypts) the keyblob specified in the argument (0) using the specified Key Encryption Key (KEK) and loads the keyblob to internal flas.
    # The load destination (0x1000, which is defined via keyBlobPointer in BCA) must match the defualt location (0x410) or the keyblob pointer in the Bootloader Configuraion Area(BCA) contained in the SREC image.
    # Make sure the sector at 0x1000 has not been written by the srec file load above, otherwise it will need to be erased again.
    keywrap (0) {
        load {{000102030405060708090a0b0c0d0e0f}} > 0x1000;
    }

    #9. Reset target.
    reset;


}
```

Load the encrypted Keyblob to 0x1000

➢ Encrypting SB file with the SB key

To encrypt the SB file with elftosb, a file containing the SB key also needs to be created, as shown in the following table.

Table 4

Key.txt: 00112233445566778899aabbccddeeff

And the following figure shows generation of the encrypted SB file using the BD file drafted in the previous section. The SB key is passed on the command line to elftosb using -k option.

```
[redacted]\NXP_Kinetis_Bootloader_2_0_0\NXP_Kinet
is_Bootloader_2_0_0\bin\Tools\elftosb\win>elftosb -V -c qspi_image_encrypt_0000.
bd -k key.txt -o image.sb
creating encrypted range 0x68001000 len 0x600
creating wrapped keyblob
Boot Section 0x00000000:
  ERAS | adr=0x00000000 | cnt=0x00000800 | flg=0x0000
  LOAD | adr=0x20000000 | len=0x00000200 | crc=0x0719eed2 | flg=0x0000
  ENA  | adr=0x20000000 | cnt=0x00000004 | flg=0x0100
  ERAS | adr=0x68000000 | cnt=0x00004000 | flg=0x0000
  LOAD | adr=0x68000000 | len=0x00000200 | crc=0x0719eed2 | flg=0x0000
  LOAD | adr=0x00000000 | len=0x00000410 | crc=0x92a569af | flg=0x0000
  LOAD | adr=0x68001000 | len=0x00000600 | crc=0x11acb209 | flg=0x0000
  LOAD | adr=0x00001000 | len=0x00000100 | crc=0x446159c2 | flg=0x0000
  RESET
```

Fig3 Generate encrypted SB file with encrypted QuadSPI image

➢ Flash the Encrypted SB file to FRDM-K82F

1. Setup the hardware

According to the reference manual of the K82_150, the LPUART0_RX/TX , LPUART1_RX/TX and LPUART2_RX/TX are supported by the KBOOT, however the OpenSDA interface use the LPUART4 module as the serial terminal. So in this experiment, the J1_2 and J1_4 (LPUART0_RX and LPUART0_TX) are selected to contact with the ROM bootloader(Fig 4).

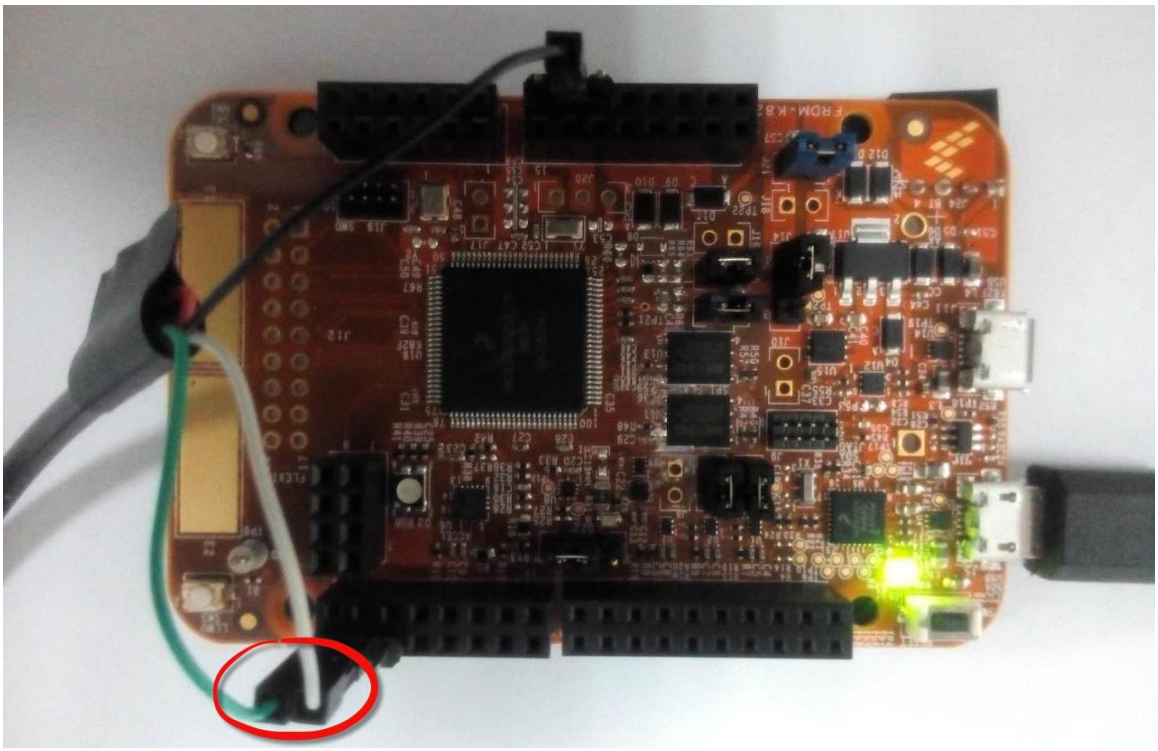

Fig 4 FRDM-K82F board

2. Program the target

The previous generated image.sb can be programmed to the target device by using blhost. Keep in mind, an unencrypted SB file can be flashed to the target only when flash security is disabled. However when flash security is enabled, all memory writes and reads are disabled, with the exception being an encrypted SB file that can be sent to the target.

- To enable the security:

The following picture illustrates the procedure of enabling the security of the MCU

Fig 5 Enable the security

- Flash the image.sb

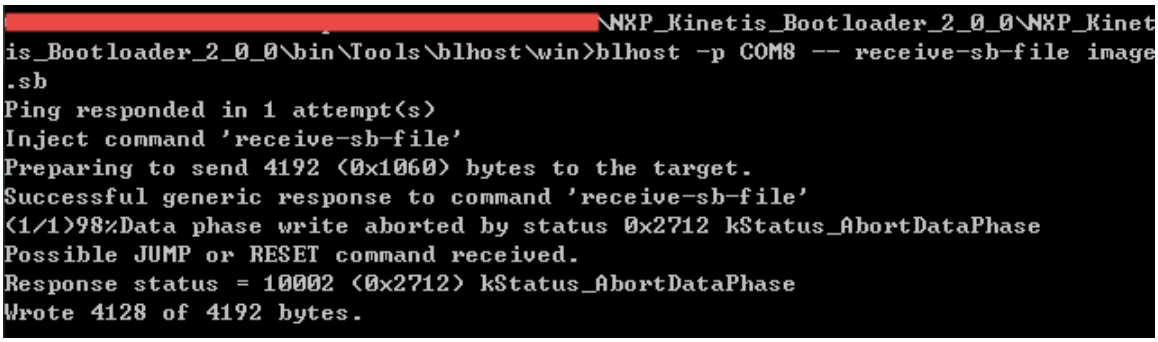

Fig 6 F lash the image.sb

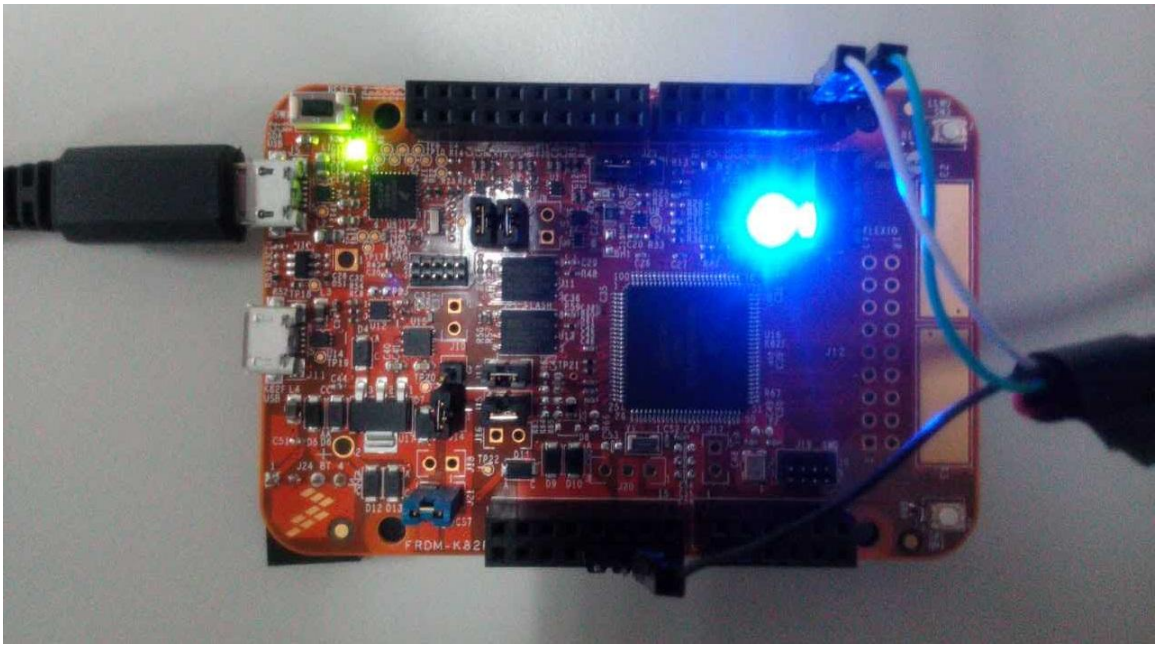

Fig 7 Demo runs