iMX6 SECURE BOOT

Nitrogen 6X SOM module

BY

RISE LAB, Computer Sciences Department,

IIT, Madras

Chennai 600036

Table of Contents

Introduction to secure boot on iMX6 nitrogen board

The Boundary devices made Nitrogen iMX 6x board supports secure boot, and this document explains the steps required to do it. Other iMX6 boards are equipped with booting from SD card, but the nitrogen 6x SOM module comes pre-fused for booting from SPI EEPROM. Hence secure boot using SD card is disabled in the Nitrogen6X board.

The U-Boot enabled with secure boot functionality is not available in new and updated U-Boot 2013.01 Version. For secure boot functionality, the older U-Boot version 2009-08 should be used.

Two important data structures for secure boot are as follows:

- Initial Vector Table (IVT): This contains pointers to the different portions of the image like the start of the actual U-Boot image, the start of the Command Sequence File and related information.
- Command Sequence File (CSF): This contains other information like the signature generated at the time of building the image.

The overall structure of the signed U-Boot image will be typically as follows:

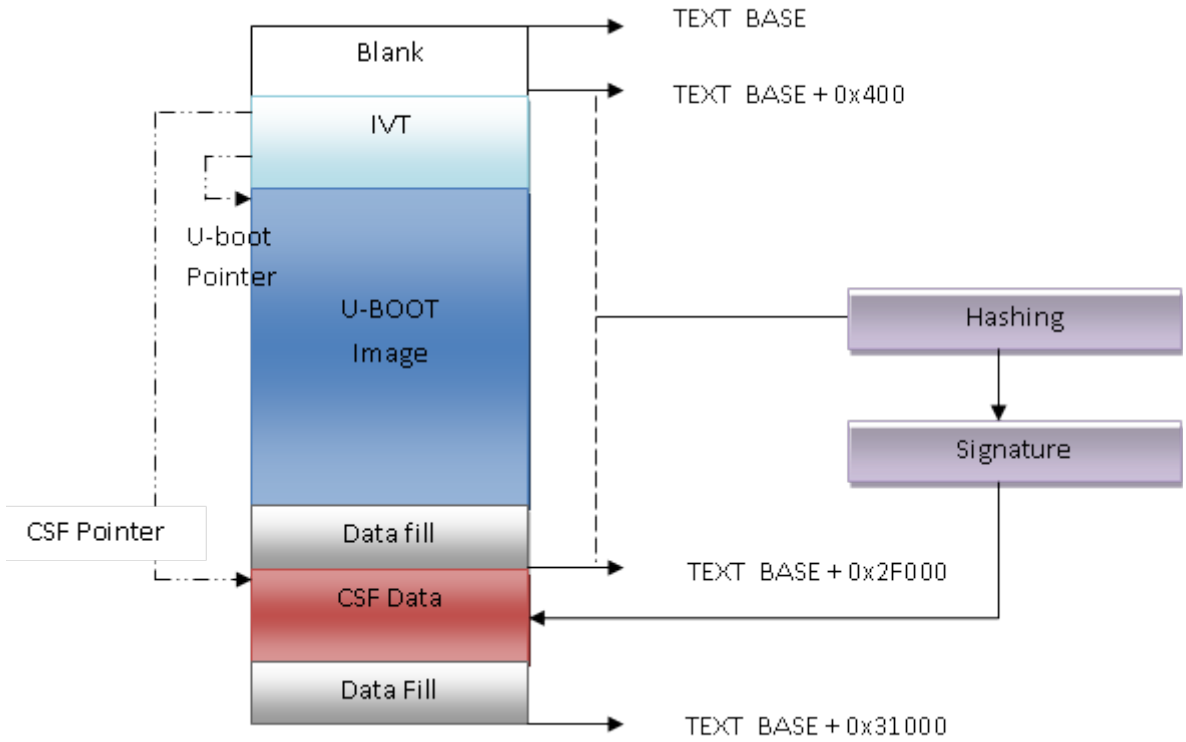

Figure 1: Structural Overview of Secure Boot Image

The secure U-Boot image targeted for Nitrogen 6X and Sabrelite board should contain an empty space till 0x400 offset. An Image Vector Table (IVT) starting at offset 0x400 would be created. This IVT table in turn points to the other portions of the U-Boot image and the Command Sequence File (CSF) portion. The CSF should be included at offset 0x2F000. The full length of the image is padded up to 0x31000.

Padding is simply filling up the image space with a character of no significance to arrive the image at a particular size.

Creating a secure U-Boot

1. Extract the source code of Freescale mainline U-Boot repository,

```
$ tar -xvzf U-Boot-2009-08.tar.gz
$ cd U-Boot-2009-08/
```

2. Export the Compilation variables to support the cross compilation,

```
$ export arch=arm
$ export CROSS_COMPILE=arm-linux-gnueabi-
```

3. Set the base address, from which the image would be loaded in to the memory and processed by doing the following.

```
$ vi board/freescale/mx6q_sabrelite/config.mk
```

Make sure the base address is 0x27800000, in config.mk

```
ifndef TEXT_BASE
    TEXT_BASE = 0x27800000
endif
```

This is the absolute address of the memory where the uboot will be loaded during power on reset.

4. Change the file U-Boot.lds, to include the High Assurance Boot (HAB) related information. The U-Boot.lds file is shown in the 1. U-Boot-2009-08/board/freescale/mx6q_sabrelite/U-Boot.lds, with the addition highlighted. The files would be located in the U-Boot source directory ***board/freescale/mx6q_sabrelite/***

Change the flash_header.S file to enable secure boot details. Edit the CSF start address, and the image length manually.

app_code_CSF: .word 0x2782f000

image_len: .word 0x31000

For u boot Image Sizes that are bigger, change the sizes accordingly, for e.g. in sabresd boards,

app_code_CSF: .word 0x2787f000

image_len: .word 0x80000

5. Add another file to the U Boot source code in order to enable a command, hab_status which would be used to read the HAB events. This would be helpful later in secure boot debugging. The file is given in the appendix.

In order to accommodate this file in to the u boot binary and to use the command, include it in the compilation by making change to Makefile in the same directory structure.

Edit the line

```
COBJS   := $(BOARD).o
```

as

```
COBJS   := $(BOARD).o mx6q_hab.o
```

6. Start the compilation

```
$ make
```

The unsigned u boot binary file would be created as U-Boot.bin

Install the Code Signing Tool (CST) tool

NOTE: Please ensure that the CST 2.0 version alone is used

1. Extract the CST tool,

```
$ tar -xvzf ../<user>/BLN_CST_MAIN_02.00.00.tgz .
$ cd BLN_CST_MAIN_02.00.00/
```

2. Change the permissions of the keys folder for write and read access,

```
$ chmod u+x linux/* keys/*
$ cd keys/
```

3. Run the HAB 4 key generation script,

```
$ ./hab4_pki_tree.sh
```

```
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
```

This script is a part of the Code signing tools for Freescale's

High Assurance Boot. It generates a basic PKI tree. The PKI

tree consists of one or more Super Root Keys (SRK), with each

SRK having two subordinate keys:

+ a Command Sequence File (CSF) key

+ Image key.

Additional keys can be added to the PKI tree but a separate

script is available for this. This this script assumes openssl

is installed on your system and is included in your search

path. Finally, the private keys generated are password

protectedwith the password provided by the file key_pass.txt.

The format of the file is the password repeated twice:

my_password

my_password

All private keys in the PKI tree are in PKCS #8 format will be

protected by the same password.

++

Do you want to use an existing CA key (y/n)?: n

Enter key length in bits for PKI tree: 2048

Enter PKI tree duration (years): 10

How many Super Root Keys should be generated? 4

Do enter the values for the highlighted areas in the above output, to generate new set of keys.

Keys would be generated in the ***keys*** folder.

There are different keys that would be generated by this command. These are

- CA key is the top most key and is only used for signing SRK certificates.
- SRK is the root key for HAB code signing keys. The cryptographic hash of a table of SRK is burned to one-time programmable efuses to establish a root of trust. Only one of the SRKs in the table may be selected for use on the Freescale processor per reset cycle. The selection of which SRK to use is a parameter within the Install Key CSF command. The SRK may only be used for signing certificate data of subordinate keys.
- CSF is a subordinate key of the SRK and is used to verify the signature across CSF commands.
- IMG is a subordinate key of the SRK key and is used to verify signatures across product software.

The hab4_pki_tree script generates a basic tree in which up to a maximum of four SRKs may be generated. For each SRK a single CSF key and IMG key are also generated.

4. Change to crts directory where certificates are stored.

```
$ cd ../crts
```

5. Run the SRK key generation tool, to produce the fuse values as well as table of SRK keys that would be embedded in the CSF data. This would produce the following files:

1. fuse.bin which would go in to the efuses on the Nitrogen SOM module
2. table.bin which contains the SRK values that would be concatenated in to the CSF file.

Run the following command:

```
$ ../linux/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
```

Note: Do not leave a space between comma and the start of another key in the above command

6. The values that needs to be fused in to the efuses of the iMX6 board would be obtained by:

```
$ hexdump -C SRK_1_2_3_4_fuse.bin
```

An example output will be as follows:

```
00000000  8e e3 6b f2 61 82 b3 35  9f 41 fb d6 13 b4 46 b3 |..k.a..5.A....F.|
00000010  12 e6 c1 31 11 57 14 d1  28 83 6a b7 a4 c4 1d 15  |...1.W..(.j.....|
00000020
```

Signing the images

The signing of the Images are done with CST 2.0 tool.

1. Create a directory to store the unsigned U-Boot images, and copy the U-Boot.bin file to this directory for signing.

```
$ mkdir U-Boot
$ cd U-Boot
```

2. Choose the keys for signing CSF and Key. SRK_1_2_3_4_table.bin file should be the table created by the previous command (i.e. ./srktool). This file name including the relative file path should be mentioned in the “Install SRK” section of the CSF file (U-boot.csf file) as below:

```
[Header]
    Version = 4.0
    Security Configuration = Open
    Hash Algorithm = sha256
    Engine Configuration = 0
    Certificate Format = X509
    Signature Format = CMS
[Install SRK]
    File = "../crts/SRK_1_2_3_4_table.bin"
    Source index = 0
[Install CSFK]
    File = "../crts/CSF1_1_sha256_2048_65537_v3_usr_crt.pem"
[Authenticate CSF]
[Install Key]
    Verification index = 0
    Target index = 2
    File = "../crts/IMG1_1_sha256_2048_65537_v3_usr_crt.pem"
# Sign padded U-Boot starting at the IVT through to the end with
```

```
# length = 0x2F000 (padded U-Boot length) - 0x400 (IVT offset) =
0x2EC00

# This covers the essential parts: IVT, boot data and DCD.

# Blocks have the following definition:

#    Image block start address on i.MX, Offset from start of image file,

#    Length of block in bytes, image data file

[Authenticate Data]

   Verification index = 2

   Blocks = 0x27800400 0x400 0x2EC00 "U-Boot-pad.bin"
```

In the above sample CSF file, 0x27800400 is the absolute address of the location in which the IVT table would start from, 0x400 is the offset and 2EC00 is the length of the image that is to be certified with signature. The 0x2EC00 is obtained from subtracting the length of the U-Boot image and the empty space which is to be discarded for signature. i.e 0x2F000 - 0x400 = 0x2EC00.

3. Create and run the script file habimagegen.sh to generate signed padded u boot file.

```
$ ./habimagegen.sh
#!/bin/sh

echo "Extend U-Boot to 0x2f000..."

objcopy -I binary -O binary --pad-to 0x2f000 --gap-fill=0xff U-Boot.bin U-
Boot-pad.bin

echo "generate CSF data..."

../linux/cst --o U-Boot_CSF.bin < U-Boot.CSF

echo "merge image and CSF data..."

cat U-Boot-pad.bin U-Boot_CSF.bin > U-Boot-signed.bin

echo "extend final image to 0x31000..."

objcopy -I binary -O binary --pad-to 0x31000 --gap-fill=0xff U-Boot-
signed.bin U-Boot-signed-pad.bin

echo "U-Boot-signed-pad.bin is ready"
```

A file called U-Boot-signed-pad.bin would be created, which is the signed U-Boot executable file to be uploaded to the board for secure boot. The above script file builds the secure boot image with the structure as shown in the Figure 1: Structural Overview of Secure Boot Image.

The following are the main activities that are being done as part of the above script:

1. The image is padded before attaching the CSF information.
2. Create a signature by encrypting it
3. Put them all together as CSF file which would also contain the public key that is needed to decrypt the signature.
4. The CSF information would be stored in the U-Boot_CSF.bin according to the script file. This CSF file is finally concatenated along with the original image thus forming a complete secure U-Boot image.

Also padding is done before and after signing as described in the introduction, using the above script function.

Interpretation of the signed u-boot image

The Image Vector Table (IVT) is a mandatory part of the boot image. The structure of IVT is defined as follows:

```
typedef struct
{
        uint32_t      header;
        uint32_t      *entry;
        uint32_t      reserved1;
        uint32_t      *dcd;
        boot_data_t         *boot_data;
        uint32_t      *self;
        uint32_t      *CSF;
        uint32_t      reserved2;
} image_vector_table_t;
```

where:

- uint32_t: A type representing a 32-bit unsigned integer.
- header: Header identifying the type of data structure (0xD1), its size (0x0020), and HAB version
- (such as, 0x40). For example, i.MX53 uses D100 2040h.
- *entry: Absolute address of the first instruction to be executed from the image.
- reserved1: Reserved and should be zero.
- *dcd: Absolute address of the image Device Configuration Table (DCD). The DCD is optional, so this field may be set to NULL, if no DCD is required.
- *boot_data: Absolute address of the Boot Data structure.
- *self: Absolute address of the IVT. Used internally by the ROM.
- *CSF: Absolute address of the Command Sequence File (CSF) used by the HAB library. This field
- must be set to NULL, if not performing a secure boot.
- reserved2: Reserved and should be zero.

The Boot Data is an associated structure that indicates where to load the boot image and that specifies the size of the boot image. It is defined as follows:

```
typedef struct
{
        uint32_t      *start;
        uint32_t      length;
        uint32_t      plugin_flag;
} boot_data_t;
```

Where:

• *start: Absolute address of the boot image. Typically, somewhere in the SDRAM.

• length: Size of the boot image to copy from the boot device to address *start.

• plugin_flag: Reserved and must be zero.

The Hexdump of the binary would be:

$ hexdump -C U-Boot-signed-pad.bin
Note: The -C in the hexdump command is mandatory to get the correct values.

The Figure 2shows the empty space created by the U-Boot compiler from freescale repositories.

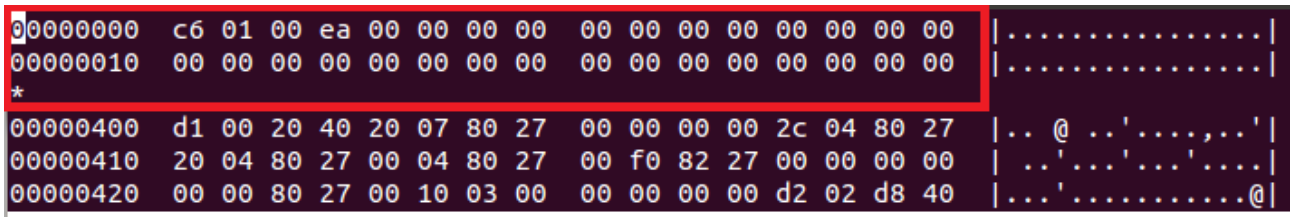

Figure 2Signed boot Binary - Empty space

The Figure 3 with colored boxes represents the absolute address of the IVT table. The address and the values are tabled below,

ADDRESS	VALUE	DESCRIPTION
0X400	402000D1	Header
0x404	27800720	Pointer to absolute address of entry, i.e. U-Boot image
0x408	00000000	Reserved
0x40B	2780042C	Pointer to absolute address of DCD values.
0x410	27800420	Pointer to absolute address of Boot data.
0x414	27804000	Start of the IVT table
0x418	2782F000	Start of the CSF data.

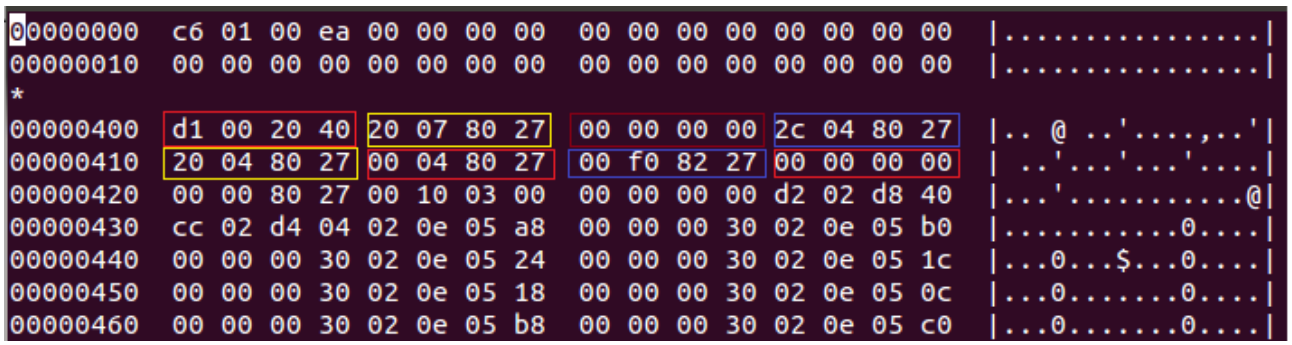

Figure 3Signed Boot Binary - IVT table

The Figure 4, Figure 5 shows the padding that is generated by the habimagegen.sh file.

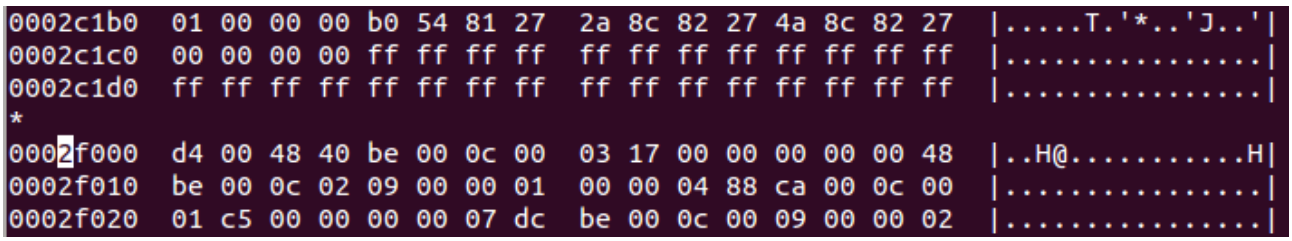

Figure 4Signed Boot Binary - Padding up to CSF

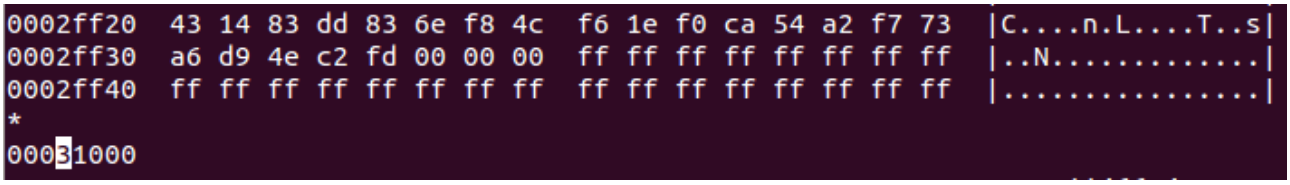

Figure 5Signed Boot Binary - Padding at the end

Loading the image on to the board

To load the secure boot image on iMX6 Nitrogen board, Serial EEPROM has to be used, as it is the default boot location as provided by Boundary devices. As the U-Boot ,image is getting created from the older version (2009.08 version) , there would be empty space until offset of 0x400. Hence unlike U-Boot images generated from later versions, this U-Boot should be flashed right from the start of the EEPROM.

Create a FAT or ext2 (preferable) partition on the SD card and load the signed U-Boot image on to it. On restarting the iMX6 nitrogen board, interrupt the Uboot by pressing a key, for flashing the new image.

1. Probe the flash memory:

```
=> sf probe 1
```

2. Erase the flash,

```
=> sf erase 0 0x80000
```

3. Depending on whether FAT or ext2 partition change the command below. Also change the name of the U-Boot file.

```
=> ext2load mmc 0:1 12000000 U-Boot.bin
or
=> fatload mmc 0:1 12000000 U-Boot.bin
```

4. Write the contents from the RAM to the SPI EEPROM. Depending on the U-Boot version, the offset address will vary as mentioned above.

```
=> sf write 0x12000000 0x0 $filesize
```

5. In case of a repetitive flashing of secure boot images for validation purposes a script as below would be more helpful.

```
=> setenv nor_write 'setenv offset 0x0; sf probe 1; mmc dev; ext2load mmc 0:1 12000000 test.bin; sf erase 0 0x80000; sf write 0x12000000 $offset $filesize'
```

Which can be executed by,

```
=>  run nor_write
```

Fusing the values

1. The fuse values to be blown on the efuses on the board are the values obtained from the fuse file created by srktool.

```
$ hexdump -C SRK_1_2_3_4_fuse.bin
```

```
00000000  17 f4 82 c3 cd 7b 13 ed  39 5e 3c 39 d4 b6 f0 d9  |.....{..9^<9....|
00000010  cc 08 8a e1 23 0d 14 f5  8c 44 31 88 20 48 7e 04  |....#....D1. H~.|
00000020
```

Note: The hexdump -C is mandatory for correct values.

U-boot can be used to fuse values in to the efuses permanently. We need to address the index with the U-Boot using imxotp command. The conversion of address to index is done as follows:

```
index = (addr - otp_base) / 0x10
```
e.g. addr is 0x021bc580, otp_base is 0x021bc400, the index = 0x18

The address and opt_base can the obtained from the processor iMX6 reference manual.

NOTE: Here the last values are given for example. SRK hash values generated by your CST 2.0 tool should be fused on the respective boards. The values should be written in reverse order with respect to the hexdump -C command. The highlighted portion in the hexdump above represents the value for index 0x18, and it should be written 0xc382f417.

For the above example fuse file, the order of fuse values should be

```
=> imxotp blow --force 0x18 0xc382f417
=> imxotp blow --force 0x19 0xed137bcd
```

```
=> imxotp blow --force 0x1A 0x393c5e39
=> imxotp blow --force 0x1B 0xd9f0b6d4
=> imxotp blow --force 0x1C 0xe18a08cc
=> imxotp blow --force 0x1D 0xf5140d23
=> imxotp blow --force 0x1E 0x8831448c
=> imxotp blow --force 0x1F 0x047e4820
```

Enabling secure boot

There are two modes of secure boot operations in iMX6 processors. The default one is open configuration, in which even if the authentication and High assurance boot process fails, the processor still loads the images from respective locations. This can be used to validate the secure boot process, and the images before moving in to closed configuration.

When the command hab_status is run, it should report that there were no HAB events (Events denotes failure in authentication of the image). The expected output will be as shown below:

```
MX6Q SABRELITE U-Boot > hab_status
iMX6 HAB status Information :
================================
Checking HAB_status
HAB Configuration: 0xf0  HAB State: 0x66
No HAB Events Found!
```

Only when such an output arrives in open configuration, closing the High assurance boot should be proceeded. Please refer the document HAB4_API.pdf given along with CST 2.0, for events and their meanings.

A test case for invalid image is done with changing one bit in the image. A failure in authentication of the U-Boot will lead to occurrence of HAB events like the one below, with highlighted value 0x18 signifying the invalid signature.

```
MX6Q SABRELITE U-Boot > hab_status
IMX6 HAB status Information :
================================
Checking HAB_status

HAB Configuration: 0xf0  HAB State: 0x66

--------- HAB Event 1 -----------------
```

```
event data:
    0xdb 0x00 0x1c 0x41 0x33 0x18 0xc0 0x00
    0xca 0x00 0x14 0x00 0x02 0xc5 0x00 0x00
    0x00 0x00 0x0d 0x34 0x27 0x80 0x04 0x00
    0x00 0x02 0xec 0x00

--------- HAB Event 2 -----------------
event data:
    0xdb 0x00 0x14 0x41 0x33 0x0c 0xa0 0x00
    0x00 0x00 0x00 0x00 0x27 0x80 0x04 0x00
    0x00 0x00 0x00 0x20

--------- HAB Event 3 -----------------
event data:
    0xdb 0x00 0x14 0x41 0x33 0x0c 0xa0 0x00
    0x00 0x00 0x00 0x00 0x27 0x80 0x04 0x2c
    0x00 0x00 0x02 0xd8

--------- HAB Event 4 -----------------
event data:
    0xdb 0x00 0x14 0x41 0x33 0x0c 0xa0 0x00
    0x00 0x00 0x00 0x00 0x27 0x80 0x04 0x20
    0x00 0x00 0x00 0x01

--------- HAB Event 5 -----------------
event data:
    0xdb 0x00 0x14 0x41 0x33 0x0c 0xa0 0x00
    0x00 0x00 0x00 0x00 0x27 0x80 0x07 0x20
    0x00 0x00 0x00 0x04
```

We can close the secure fuse configuration with the below mentioned command. After this the board would work only on secure mode, and any failure of the secure Image would lead to board getting into an unusable state.

```
=> imxotp blow --force 0x08 0x00040000

=>  imxotp blow --force 0x6 0x2
```

Appendix

1. U-Boot-2009-08/board/freescale/mx6q_sabrelite/U-Boot.lds

```
/*
 * January 2004 - Changed to support H4 device
 * Copyright (c) 2004 Texas Instruments
 *
 * (C) Copyright 2002
 * Gary Jennejohn, DENX Software Engineering, <gj@denx.de>
 *
 * (C) Copyright 2011 Freescale Semiconductor, Inc.
 *
 * See file CREDITS for list of people who contributed to this
 * project.
 *
 * This program is free software; you can redistribute it and/or
 * modify it under the terms of the GNU General Public License as
 * published by the Free Software Foundation; either version 2 of
 * the License, or (at your option) any later version.
 *
 * This program is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.      See the
 * GNU General Public License for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with this program; if not, write to the Free Software
```

```
 * Foundation, Inc., 59 Temple Place, Suite 330, Boston,
 * MA 02111-1307 USA
 */

OUTPUT_FORMAT("elf32-littlearm", "elf32-littlearm", "elf32-littlearm")
OUTPUT_ARCH(arm)
ENTRY(_start)
SECTIONS
{
        . = 0x00000000;
        . = ALIGN(4);
        .text      :
        {
          /* WARNING - the following is hand-optimized to fit within  */
          /* the sector layout of our flash chips!     XXX FIXME XXX    */
          board/freescale/mx6q_sabrelite/flash_header.o        (.text.flasheader)
          cpu/arm_cortexa8/start.o
          board/freescale/mx6q_sabrelite/libmx6q_sabrelite.a (.text)
          lib_arm/libarm.a          (.text)
          net/libnet.a                      (.text)
          drivers/mtd/libmtd.a            (.text)
          drivers/mmc/libmmc.a         (.text)

          . = DEFINED(env_offset) ? env_offset : .;
          common/env_embedded.o(.text)
          *(.text)
        }
        . = ALIGN(4);
```

```
.rodata : { *(SORT_BY_ALIGNMENT(SORT_BY_NAME(.rodata*))) }

. = ALIGN(4);
.data : { *(.data) }
. = ALIGN(4);
.got : { *(.got) }
. = .;
__u_boot_cmd_start = .;
.u_boot_cmd : { *(.u_boot_cmd) }
__u_boot_cmd_end = .;
/* reserve this area to store HAB related data such
* CSF commands, certificates, and signatures.
* The offset from TEXT_BASE is chosen to leave some space
* for U-Boot to grow if necessary. It should anyway be bigger
* than the size of the "non-secure" U-Boot binary.
*/
. = TEXT_BASE + 0x2C000;
__hab_data_start = .;
/* leave 8kB for the CSF */
__CSF_data = .;
. = . + 0x2000;
__hab_data_end = .;
/* place this __hab_data memory region before the .bss
* region to avoid being over written at runtime by the
* zero initialized region below
*/
. = ALIGN(4);
_end_of_copy = .; /* end_of ROM copy code here */
```

```
        __bss_start = .;
        .bss : { *(.bss) }
        _end = .;
}
```

2. U-Boot-2009-08/board/freescale/mx6q_sabrelite/mx6q_hab.c

```
#include <common.h>
#include <config.h>
#include <command.h>

/* The following defines and structures are taken from HAB4 SIS */
/* Status definitions */
typedef enum hab_status {
        HAB_STS_ANY = 0x00,
        HAB_FAILURE = 0x33,
        HAB_WARNING = 0x69,
        HAB_SUCCESS = 0xf0
  }     hab_status_t;

/* Security Configuration definitions */
typedef enum hab_config {
    HAB_CFG_RETURN = 0x33,      /**< Field Return IC */
    HAB_CFG_OPEN = 0xf0,        /**< Non-secure IC */
    HAB_CFG_CLOSED = 0xcc       /**< Secure IC */
} hab_config_t;

/* State definitions */
```

```
typedef enum hab_state {
    HAB_STATE_INITIAL = 0x33,   /**< Initialising state (transitory) */
    HAB_STATE_CHECK = 0x55,     /**< Check state (non-secure) */
    HAB_STATE_NONSECURE = 0x66, /**< Non-secure state */
    HAB_STATE_TRUSTED = 0x99,   /**< Trusted state */
    HAB_STATE_SECURE = 0xaa,    /**< Secure state */
    HAB_STATE_FAIL_SOFT = 0xcc, /**< Soft fail state */
    HAB_STATE_FAIL_HARD = 0xff, /**< Hard fail state (terminal) */
    HAB_STATE_NONE = 0xf0,      /**< No security state machine */
    HAB_STATE_MAX
} hab_state_t;


typedef hab_status_t hab_rvt_report_event_t(hab_status_t, uint32_t, uint8_t* ,
size_t*);
typedef hab_status_t hab_rvt_report_status_t(hab_config_t *, hab_state_t *);
#define HAB_RVT_REPORT_EVENT  (*(uint32_t *) 0x000000B4)
#define hab_rvt_report_event
((hab_rvt_report_event_t*)HAB_RVT_REPORT_EVENT)
#define HAB_RVT_REPORT_STATUS (*(uint32_t *) 0x000000B8)
#define hab_rvt_report_status
((hab_rvt_report_status_t*)HAB_RVT_REPORT_STATUS)
void display_event(uint8_t *event_data, size_t bytes)
{
        uint32_t i;
        if ((event_data) && (bytes > 0))
        {
                for (i = 0; i < bytes; i++)
                {
                        if (i == 0)
```

```
                {
                        printf("    0x%02x", event_data[i]);
                }
                else if ((i % 8) == 0)
                {
                        printf("\n    0x%02x", event_data[i]);
                }
                else
                {
                        printf(" 0x%02x", event_data[i]);
                }
            }
        }

}

int get_hab_status(void)
{
        uint32_t     index = 0;                 /* Loop index */
        uint8_t      event_data[128];           /* Event data buffer */
        size_t       bytes = sizeof(event_data); /* Event size in bytes */
        hab_config_t config = 0;
        hab_state_t  state = 0;

        printf("Checking HAB_status\n");

        /* Check HAB status */
        if (hab_rvt_report_status(&config, &state) != HAB_SUCCESS)
```

```
        {
        printf("\nHAB Configuration: 0x%02x  HAB State: 0x%02x\n", config,
state);

        /* Display HAB Error events */
        while (hab_rvt_report_event(HAB_FAILURE, index, event_data, &bytes)
== HAB_SUCCESS)
                {
                printf("\n");
                printf("--------- HAB Event %d -----------------\n", index + 1);
                printf("event data:\n");
                display_event(event_data, bytes);
                printf("\n");
                bytes = sizeof(event_data);
                index++;
        }
        }
        /* Display message if no HAB events are found */
        else
        {
        printf("\nHAB Configuration: 0x%02x  HAB State: 0x%02x\n", config,
state);
        printf("No HAB Events Found!\n\n");
        }
}

int do_hab_status ( cmd_tbl_t *cmdtp, int flag, int argc, char *argv[] )
{
        /* Validate arguments */
```

```
        if ((argc != 1)){
        cmd_usage(cmdtp);
        return 1;
        }

        printf("iMX6 HAB status Information :\n");
        printf("================================\n");
        get_hab_status();
        return 0;

}
U_BOOT_CMD(
        hab_status, 1, 0, do_hab_status,
        "Displays the HAB status information",
        ""
);
```