

Setting up Secure Boot on PBL Based Platforms in Prototype Stage

This document introduces secure boot setting up methods and procedures for QorIQ PBL based PowerPC and ARM architecture platforms. It explains the basic concept of secure boot; describes how to create PBL, CSF headers and ESBC images for secure boot; introduces how to deploy secure boot related images on the target and how to blow OTPMK and write SRK hash keys to the mirror registers in the prototype stage; provides the step by step troubleshooting method when secure boot fails without any message printed out on the UART console.

LS1043 is used as an example in this document, if the user uses different processor, it is need to change registers and images deployment addresses according to the processor reference manual and QorIQ SDK user manual.

1. Basic Concept of Secure Boot on PBL Based Platform

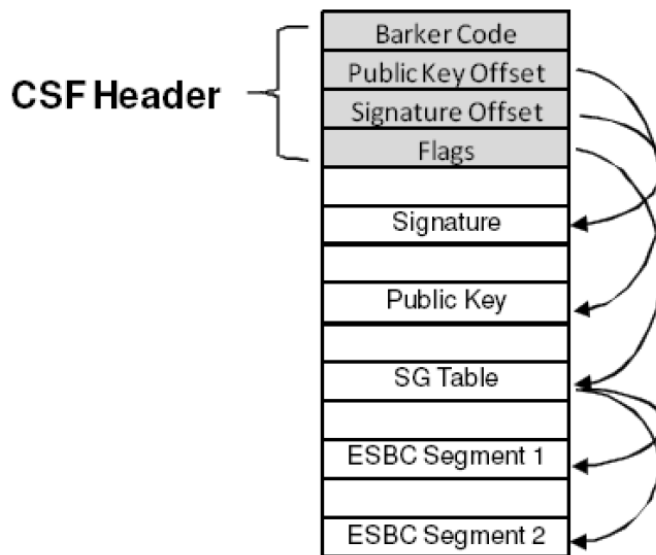
Secure boot process uses a digital signature validation routine already present in INTERNAL BOOT ROM. This routine performs validation using HW bound RSA public key to decrypt the signed hash and compare it to a freshly calculated hash over the same system image. If the comparison passes, the image can be considered as authentic.

With the validated public key, CPU 0 decrypts the digital signature stored with the CSF header. The ISBC then uses the ESBC u-boot lengths and pointer fields in the CSF header to calculate a hash over the code.

There are RSA public and private key pairs, private keys are used to sign the images and public keys are keys are used to validate the image during ISBC and ESBC phase. Public keys are embedded in the header and the hash of SRK table is fused in SRKH register of SFP.

OTPMK(One Time Programmable Master Key) is used to encryption and decryption of additional secret keys(also usable only by the SEC module) that can be used to protect arbitrary data.

The figure below provides an example of and ESBC with CSF(Command Sequence File) header. The CSF header includes lengths and offset which allow the ISBC to locate the operands used in ESBC image validation, as well as describe the size and location of the ESBC image itself.



2. Preparing Images for Secure BOOT

This section describes how to create RCW, OTPMK and SRK keys, u-boot CSF header and u-boot image for secure boot.

2.1 Create RCW Supporting Secure Boot

It is mandatory to use PBL when performing secure boot. The PBL must read a command file from a location determined by the Reset Configuration Word(RCW) and perform a store of a value to the ESBC Pointer Register within the SoC. Unless the ESBC u-boot is stored in NOR flash, the developer is required to create a PBL Image that copies the image to be validated from NVRAM to main memory or internal SRAM prior to writing the SCRATCHRW1 Register and executing the ISBC code.

SCRATCHRW1 is used to store the address used during the secure boot process to access the first location of external secure boot code, point to ESBC Header(Primary Boot Image) . The PBL must initialize this register for proper secure boot operation.

The user needs to edit PBI commands to configure LAWS for ESBC, CPC as SRAM and SCRATCH registers to generate a PBL image to support secure boot. The following is PBI commands for T1/T2/T4/B4 NOR SECURE BOOT.

```
#LAW for ESBC
09000c10 00000000
09000c14 c0000000
09000c18 81f0001b
# LAW for CPC/SRAM
09000d00 00000000
09000d04 bff00000
09000d08 81000013
# Scratch Registers
090e0200 c0b00000
090e0208 c0c00000
# CPC SRAM
09010100 00000000
09010104 bff00009
# CPC Configuration
09010f00 08000000
09010000 80000000
```

In addition, there are two ways to enable secure boot on the target board, programming the ITS fuse, or set SB_EN bit to 1 in RCW, in the prototype stage, we suggest users to use the latter.

2.1.1 Create Secure boot RCW in Linux SDK

Linux SDK user could use "rcw" package to generate RCW for secure boot. For some platforms, secure boot rcw configuration files has already been provided, for example LS1043ARDB, PBI secure boot configuration file ls1043aqds/u-boot_hdr_addr.rcw is provided as the following(SCRATCHRW1 register specifies the ESBC u-boot header address), the secure boot RCW configuration file ls1043ardb/RR_FQPP_1455/rcw_1600_sben.rcw includes u-boot_hdr_addr.rcw and with "SB_EN" field enabled. So the user could build rcw package to generate secure boot RCW directly.

```
/*
* PBI Commands for LS1043ARDB and LS1043AQDS.
*/

.pbi
// Workaround for u-boot address.
write 0xee0200, 0x60080000
.end
```

This section will introduces how edit rcw configuration file and build secure boot RCW for LS1043ARDB platform in Linux SDK 2.0.

```
$ bitbake rcw -c cleansstate
```

```
$ bitbake rcw -c configure
```

Go to the source folder `build_ls1043ardb/tmp/work/ls1043ardb-fsl-linux/rcw/git-r0/git/ls1043ardb/RR_FQPP_1455`, create a file `rcw_1600_sben_ho.rcw` copy from `rcw_1600_sben.rcw`, and modify "BOOT_HO" field as "1" in `rcw_1600_sben_ho.rcw`.

Rebuild rcw

```
$ bitbake rcw
```

Get `rcw_1600_sben.bin` and `rcw_1600_sben_ho.bin` in the image deploy folder

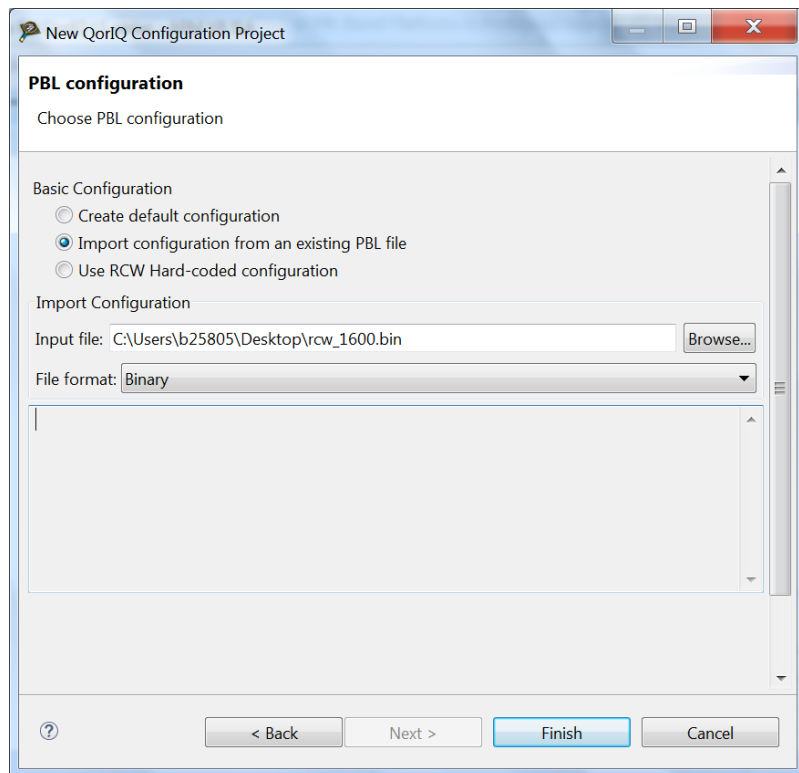
`build_ls1043ardb/tmp/deploy/images/ls1043ardb/rcw/ls1043ardb/RR_FQPP_1455`.

RCW `rcw_1600_sben_ho.bin` is used to put the core in boot hold off mode at startup to wait for using CCS to connect to the target board to write keys in SFP_SRKRH shadow registers.

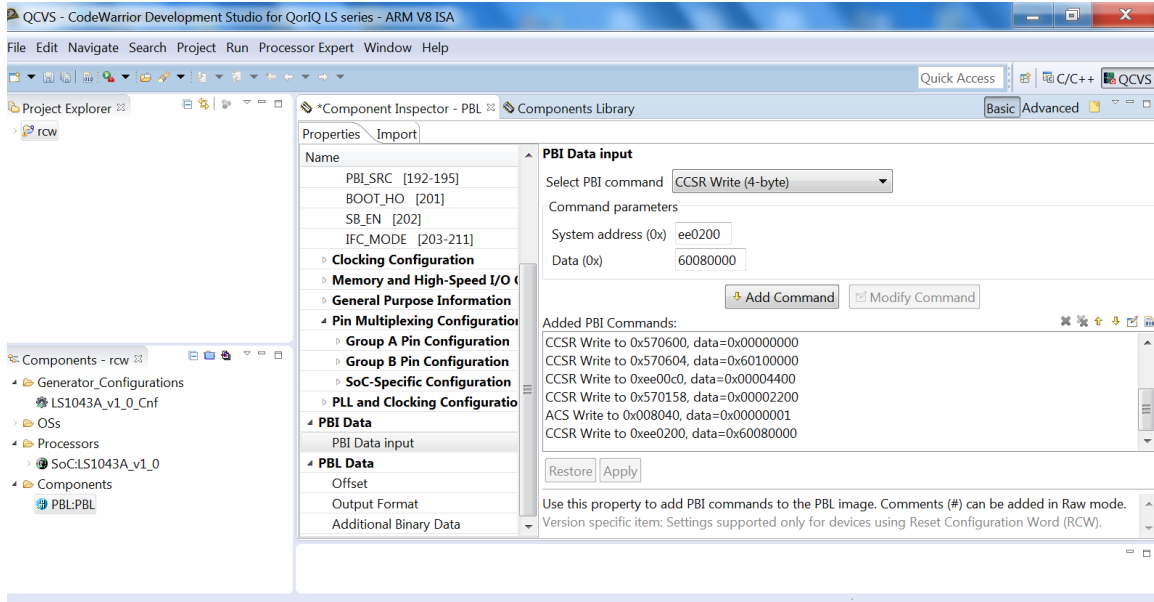
2.1.2 Create Secure Boot RCW with QCVS tool

This section describes how to create required secure boot RCW with QCVS tool, please download the latest QCVS tool from http://www.nxp.com/products/software-and-tools/software-development-tools/codewarrior-development-tools/suite-for-networked-applications/qorIQ-configuration-and-validation-suite:PE_QORIQ_SUITE?code=PE_QORIQ_SUITE&nodeId=015210BAF72726328B&fsp=1&tab=Design_Tools_Tab, please download "QCVS for ARMv7 and ARMv8" for QorIQ LayerScape platforms, download "QCVS for PA" for QorIQ PowerPC platforms, then install QCVS in CodeWarrior IDE.

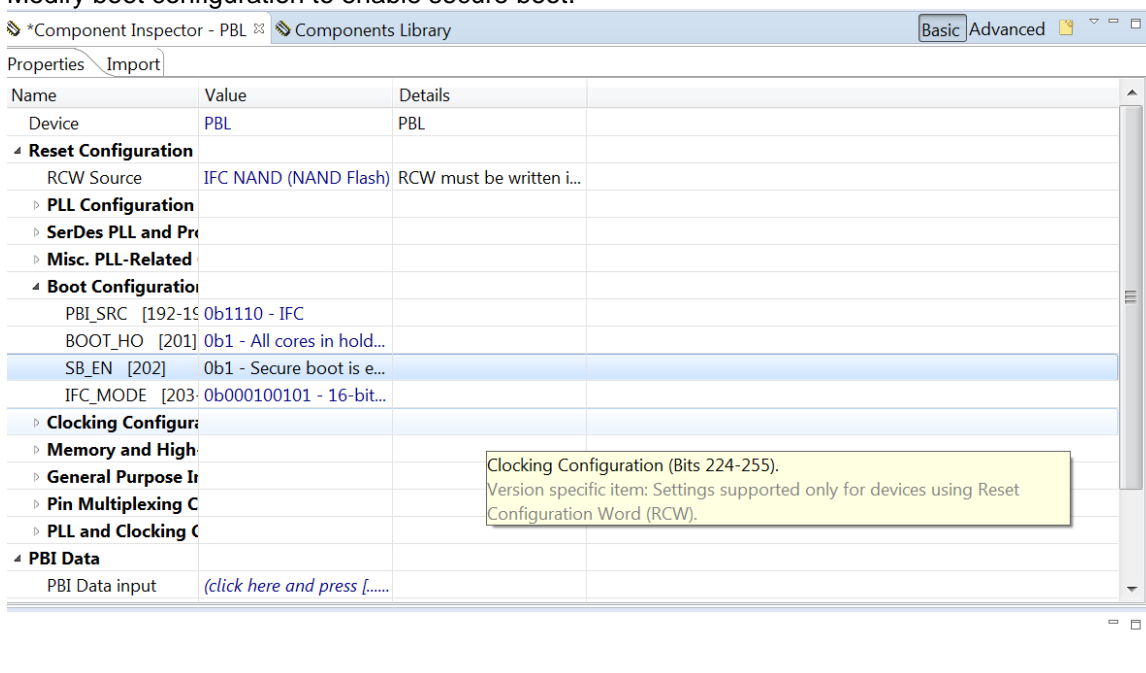
Please create a QCS project from File->New->QorIQ Configuration Project, select PBL project and import the prebuilt RCW image.



Add PBI commands from PBI Data->PBI Data input.



Modify boot configuration to enable secure boot.



Generate PBL image under Generted_Code from Project->Generate Processor Expert Code.

2.2 Generate RSA keys and Signing Images to Create CSF Header with CST tool

Linux SDK provides Code Signing Tool to assist with the Secure boot process. CST tool can be used to generate RSA public and private keys, OTPMK and DRV keys with hamming code inserted, CSF header for an image and to sign code using RSA private keys.

This section describes how to use CST tool step by step.

a. In the Linux SDK build cst for the host machine.

```
$ bitbake cst-native
```

Get cst tool working environment in build_ls1043ardb/tmp/sysroots/x86_64-linux/usr/bin/cst.

b. Generate private key public key pair

```
./gen_keys 1024
```

c. Build Secure u-boot image

Modify the recipe sources/meta-freescale/conf/machine/ls1043ardb.conf, add secure u-boot in the build configuration.

```
UBOOT_CONFIG ??= "nand sdc card nor secure-boot"
```

Execute "\$bitbake u-boot" and get secure u-boot image

build_ls1043ardb/tmp/deploy/images/ls1043ardb/u-boot-secure-boot.bin, copy it to folder build_ls1043ardb/tmp/sysroots/x86_64-linux/usr/bin/cst and rename it as u-boot.bin.

d. Generate CSF header for u-boot image

```
./uni_sign input_files/uni_sign/ls1043/input_uboot_nor_secure
```

The successful log is as the following.

```
#-----#  
#-----#  
#----- CST (Code Signing Tool) Version 2.0 -----#  
#-----#  
#-----#
```

This tool includes software developed by OpenSSL Project
for use in the OpenSSL Toolkit (<http://www.openssl.org/>)
This product includes cryptographic software written by
Eric Young (eay@cryptsoft.com)

Input File is input_files/uni_sign/ls1043/input_uboot_nor_secure

```
-----  
- Dumping the Header Fields  
-----
```

- SRK Information

- SRK Offset : 200

- SRK Flag = 0

- Single Key: srk.pub(100)

- UID Information

- UID Flags = 00

- FSL UID = 00000000_00000000

- OEM UID = 00000000_00000000

- FLAGS Information

- Secondary Image = 0

- Manufacturing Protection = 1

- Image Information

- SG Table Offset : 400

- Number of entries : 1

- Entry Point : 60100000

- Entry 1 : u-boot.bin (Size = 000986c3 SRC = 60100000 DST = ffffffff)

- RSA Signature Information

- RSA Offset : 600

- RSA Size : 80

Image Hash:

5a32001653c75ec7f8770603a0232bbf38c5d674c5eb00f150db56365ad73087

```
*****
* Header File is with Signature appended
*****
```

Header File Created: *hdr_uboot.out*

SRK (Public Key) Hash:

83bba1f03e1ce1d336490b5e4b1071f6c8021c72976408e5084e988ce4c1d93a

```
SFP SRKHR0 = 83bba1f0
SFP SRKHR1 = 3e1ce1d3
SFP SRKHR2 = 36490b5e
SFP SRKHR3 = 4b1071f6
SFP SRKHR4 = c8021c72
SFP SRKHR5 = 976408e5
SFP SRKHR6 = 084e988c
SFP SRKHR7 = e4c1d93a
```

e. Generate CSF header for kernel.itb.

Copy kernel.itb image to the folder *build_ls1043ardb/tmp/sysroots/x86_64-linux/usr/bin/cst* and execute the following command to generate the file *hdr_kernel.out*.

```
$ ./uni_sign input_files/uni_sign/ls1043/input_kernel_secure
```

f. Generate CSF header for PPA file.

Copy ppa.itb in the folder *build_ls1043ardb/tmp/sysroots/x86_64-linux/usr/bin/cst* and execute the following command to generate *hdr_ppa.out*.

```
$ ./uni_sign input_files/uni_sign/ls1043/input_ppa_secure
```

g. Create bootscript file and generate CSF header *hdr_bs.out* for bootscript.

```
$cat bootscript.txt
```

```
cp.b 0x600A0000 0x81000000 0x3500000
```

```
esbc_validate 0x63F40000
```

```
setenv bootargs "console=ttyS0,115200 root=/dev/ram0 earlycon=uart8250,0x21c0500";
```

```
setenv fdt_high "0xffffffffffff";
```

```
setenv initrd_high "0xffffffffffff";
```

```
bootm $img_addr
```

Note: Set *fdt_high* to "0xffffffffffff" then the fdt will not be copied at all on boot.

```
$ bitbake u-boot-mkimage-native
```

```
$ <installation_folder>/build_ls1043ardb/tmp/sysroots/x86_64-linux/usr/bin/mkimage -A arm
```

```
-T script -a 0 -e 0x40 -d bootscript.txt bootscript
```

```
$ ./uni_sign input_files/uni_sign/ls1043/input_bootscript_secure
```

h. Generate OTPMK keys.

```
$ ./gen_otpmk_drbg 2
```

```
#-----#
#-----#
#----- CST (Code Signing Tool) Version 2.0 -----#
#-----#
#-----#
```

Input string not provided

Generating a random string

** Hash_DRBG library invoked*

* Seed being taken from /dev/random

OTPMKR[255:0] is:

1a4721b1d5371cf735e6975844932d9ce2f460b7aa7816a774e2aba90adca9a2

NAME	BITS	VALUE
OTPMKR 0	255-224	1a4721b1
OTPMKR 1	223-192	d5371cf7
OTPMKR 2	191-160	35e69758
OTPMKR 3	159-128	44932d9c
OTPMKR 4	127- 96	e2f460b7
OTPMKR 5	95- 64	aa7816a7
OTPMKR 6	63- 32	74e2aba9
OTPMKR 7	31- 0	0adca9a2

CST Tool signing input file and explanation is described as the following.

Specify the platform. [Mandatory]
Choose Platform - 1010/1040/2041/3041/4080/5020/5040/9131/9132/9164/4240/C290/LS1
PLATFORM=LS1043
ESBC Flag. Specify ESBC=0 to sign u-boot and ESBC=1 to sign other ESBC images.
ESBC=0

Entry Point/Image start address field in the header.[Mandatory]
(default=ADDRESS of the signed image)
ENTRY_POINT=60100000

Specify the file name of the keys seperated by comma.
The number of files and key select should lie between 1 and 4 for 1040 and C290.
For rest of the platforms only one key is required and key select should not be provided.

USAGE (for 4080/5020/5040/3041/2041/1010/913x): PRI_KEY = <key1.pri>
USAGE (for 1040/C290/9164/4240/LS1): PRI_KEY = <key1.pri>, <key2.pri>, <key3.pri>,
<key4.pri>

PRI_KEY (Default private key :srk.pri) - [Optional]
PRI_KEY=srk.pri
PUB_KEY (Default public key :srk.pub) - [Optional]
PUB_KEY=srk.pub
Please provide KEY_SELECT(between 1 to 4) (Required for 1040/C290/9164/4240/LS1 only) -
[Optional]
KEY_SELECT=

Specify SG table address, only for (2041/3041/4080/5020/5040) with ESBC=0 - [Optional]
SG_TABLE_ADDR=

Specify the target where image will be loaded. (Default is NOR_16B) - [Optional]
Only required for Non-PBL Devices (1010/1040/9131/9132i/C290)
Select from -
NOR_8B/NOR_16B/NAND_8B_512/NAND_8B_2K/NAND_8B_4K/NAND_16B_512/NAND_16B_2K/NAND_16B_4K/SD/MMC/SPI
IMAGE_TARGET=

```

-----
# Specify IMAGE, Max 8 images are possible. DST_ADDR is required only for Non-PBL Platform.
[Mandatory]
# Create Entries for SG Table in the format {IMAGE_NAME, SRC_ADDR, DST_ADDR}
IMAGE_1={u-boot.bin,60100000,ffffff}
IMAGE_2={,,}
IMAGE_3={,,}
IMAGE_4={,,}
IMAGE_5={,,}
IMAGE_6={,,}
IMAGE_7={,,}
IMAGE_8={,,}
-----

# Specify OEM AND FSL ID to be populated in header. [Optional]
# e.g FSL_UID=11111111
FSL_UID=
FSL_UID_1=
OEM_UID=
OEM_UID_1=
-----

# Specify the file names of csf header and sg table. (Default :hdr.out) [Optional]
OUTPUT_HDR_FILENAME=hdr_uboot.out

# Specify the file names of hash file and sign file.
HASH_FILENAME=img_hash.out
INPUT_SIGN_FILENAME=sign.out

# Specify the signature size.It is mandatory when neither public key nor private key is specified.
# Signature size would be [0x80 for 1k key, 0x100 for 2k key, and 0x200 for 4k key].
SIGN_SIZE=
-----

# Specify the output file name of sg table. (Default :sg_table.out). [Optional]
# Please note that OUTPUT SG BIN is only required for 2041/3041/4080/5020/5040 when ESBC
flag is not set.
OUTPUT_SG_BIN=
-----

# Following fields are Required for 4240/9164/1040/C290 only
# Specify House keeping Area
# Required for 4240/9164/1040/C290 only when ESBC flag is not set. [Mandatory]
HK_AREA_POINTER=
HK_AREA_SIZE=
-----

# Following field Required for 4240/9164/1040/C290 only
# Specify Secondary Image Flag. (0 or 1) - [Optional]
# (Default is 0)
SEC_IMAGE=0
# Specify Manufacturing Protection Flag. (0 or 1) - [Optional]
# Required only for LS1(Default is 0)
MP_FLAG=1
-----

VERBOSE=1

```

3. Deploy Secure Boot Images to the Target and Write SRKH Mirror Register

This section introduces how to do secure boot validation in the prototype stage. First blow OTPMK keys to fuse array from CCS, then setup u-boot in Bank0, deploy secure boot images to Bank4, swith to Bank4, use CCS to connect to the target board to write SRKH.

a. Blow OTPMK with CCS

Please Enable POVDD on the board.

Write OTPMK registers to fuse array under CCS.

```
% config cc cwtap:10.81.116.21
% ccs::config_server 0 10000
CodeWarrior TAP executable differs from local file.
CodeWarrior TAP Boot Loader version 1.0.1
CodeWarrior TAP OS version 1.0.2
Sending code to CodeWarrior TAP.....done
Running package script
% ccs::config_chain {ls1043a dap sap2}
% display ccs::get_config_chain
Chain Position 0: LS1043A
Chain Position 1: CoreSight ATB Funnel
Chain Position 2: CoreSight ATB Funnel
Chain Position 3: CoreSight TMC
Chain Position 4: CoreSight TMC
Chain Position 5: CoreSight ATB Funnel
Chain Position 6: CoreSight STM
Chain Position 7: CoreSight TMC
Chain Position 8: CoreSight ATB Funnel
Chain Position 9: CoreSight ATB Funnel
Chain Position 10: CoreSight TMC
Chain Position 11: CoreSight TMC
Chain Position 12: CoreSight TMC
Chain Position 13: CoreSight CTI
Chain Position 14: CoreSight CTI
Chain Position 15: CoreSight CTI
Chain Position 16: Cortex-A53
Chain Position 17: CoreSight CTI
Chain Position 18: Cortex-A53 PMU
Chain Position 19: Cortex-A53 ETM
Chain Position 20: Cortex-A53
Chain Position 21: CoreSight CTI
Chain Position 22: Cortex-A53 PMU
Chain Position 23: Cortex-A53 ETM
Chain Position 24: Cortex-A53
Chain Position 25: CoreSight CTI
Chain Position 26: Cortex-A53 PMU
Chain Position 27: Cortex-A53 ETM
Chain Position 28: Cortex-A53
Chain Position 29: CoreSight CTI
Chain Position 30: Cortex-A53 PMU
Chain Position 31: Cortex-A53 ETM
Chain Position 32: DAP
Chain Position 33: SAP2
```

Write OTPMK to mirror registers.

```
ccs::write_mem 32 0x1e80234 4 0 0x1a4721b1
ccs::write_mem 32 0x1e80238 4 0 0xd5371cf7
```

```

ccs::write_mem 32 0x1e8023c 4 0 0x35e69758
ccs::write_mem 32 0x1e80240 4 0 0x44932d9c
ccs::write_mem 32 0x1e80244 4 0 0xe2f460b7
ccs::write_mem 32 0x1e80248 4 0 0xaa7816a7
ccs::write_mem 32 0x1e8024c 4 0 0x74e2aba9
ccs::write_mem 32 0x1e80250 4 0 0x0adca9a2

```

Check OTPMK_ZERO and OTPMK_SYNDROME as 0 in SecMon_HP Status Register

```
% ccs::display_mem 32 0x1e90014 4 0 4
```

Check SFP_SVHESR no parity error.

```
% ccs::display_mem 32 0x1e80024 4 0 4
```

Permanently write OTPMK from the mirror registers into the fuse array

```
ccs::write_mem 32 0x1e80020 4 0 0x00000002
```

b. Deploy image on Bank4 from Bank0

Please refer to the following secure boot images memory map for LS1043ARDB on NOR Flash.

Table 4. Memory map for LS1043 platforms

Address NOR(vBank 0)	Address (NOR Alternate Bank) ⁴	Definition (Chain of Trust)	Size Reserved (KB)
60000000	64000000	RCW	128
60060000	64060000	Bootscript	128
60080000	64080000	ESBC U-Boot HEADER	128
600A0000	640A0000	Bootscript Header	128
600C0000	640C0000	PPA Header	128
60100000	64100000	ESBC U-Boot	1024
60500000	64500000	PPA FIT Image	2048
60A00000*	64A00000*	Kernel FIT Image	54272
63F40000	64F40000	kernel Header	128

Program images to bank4 at bank0 on NOR Flash.

```

=> tftp 0xa0000000 ls1043ardb/rcw_1600_sben_ho.bin
=> erase 0x64000000 +$filesize
=> cp.b 0xa0000000 0x64000000 $filesize
=> tftp 0xa0000000 ls1043ardb/hdr_uboot.out
=> erase 0x64080000 +$filesize
=> cp.b 0xa0000000 0x64080000 $filesize
=> tftp 0xa0000000 ls1043ardb/u-boot.bin
=> erase 0x64100000 +$filesize
=> cp.b 0xa0000000 0x64100000 $filesize
=> tftp 0xa0000000 ls1043ardb/kernel.itb
=> erase 0x64A00000 +$filesize
=> cp.b 0xa0000000 0x64A00000 $filesize
=> tftp 0xa0000000 ls1043ardb/hdr_kernel.out
=> erase 0x64F40000 +$filesize
=> cp.b 0xa0000000 0x64F40000 $filesize
=> tftp 0xa0000000 ls1043ardb/ppa.itb
=> erase 64500000 +$filesize

```

```
=> cp.b 0xa0000000 0x64500000 $filesize
=> tftp 0xa0000000 ls1043ardb/hdr_ppa.out
=> erase 0x640C0000 +$filesize
=> cp.b 0xa0000000 0x640C0000 $filesize
=> tftp 0xa0000000 ls1043ardb/bootscript
=> erase 0x64060000 +$filesize
=> cp.b 0xa0000000 0x64060000 $filesize
=> tftp 0xa0000000 ls1043ardb/hdr_bs.out
=> erase 640A0000 +$filesize
=> cp.b 0xa0000000 640A0000 $filesize

=> cpld reset altbank
```

c. Write SRKH keys from CCS

Write SRKH registers under u-boot.

```
ccs::write_mem 32 0x1e80254 4 0 0x83bba1f0
ccs::write_mem 32 0x1e80258 4 0 0x3e1ce1d3
ccs::write_mem 32 0x1e8025c 4 0 0x36490b5e
ccs::write_mem 32 0x1e80260 4 0 0x4b1071f6
ccs::write_mem 32 0x1e80264 4 0 0xc8021c72
ccs::write_mem 32 0x1e80268 4 0 0x976408e5
ccs::write_mem 32 0x1e8026c 4 0 0x084e988c
ccs::write_mem 32 0x1e80270 4 0 0xe4c1d93a
```

Release core 0 from boot hold off mode.

```
ccs::write_mem 32 0x1ee00e4 4 0 0x00000001
```

d. Secure boot console log is as the following.

U-Boot 2016.012.0+ga791812 (May 11 2016 - 11:37:52 +0800)

SoC: LS1043E (0x87920010)

Clock Configuration:

CPU0(A53):1600 MHz CPU1(A53):1600 MHz CPU2(A53):1600 MHz

CPU3(A53):1600 MHz

Bus: 400 MHz DDR: 1600 MT/s FMAN: 533.333 MHz

Reset Configuration Word (RCW):

00000000: 08100010 0a000000 00000000 00000000

00000010: 14550002 80004012 e0625000 61002000

00000020: 00000000 00000000 00000000 00038800

00000030: 00000000 00001100 00000096 00000001

I2C: ready

Model: LS1043A RDB Board

Board: LS1043ARDB, boot from vBank 4

CPLD: V1.4

PCBA: V1.0

SERDES Reference Clocks:

SD1_CLK1 = 156.25MHZ, SD1_CLK2 = 100.00MHZ

DRAM: Initializing DDR....

Detected UDIMM Fixed DDR on board

2 GiB (DDR4, 32-bit, CL=11, ECC off)

SEC0: RNG instantiated

PPA validation startedPPA validation Successful
PPA Firmware: Version 0.2
Not a microcode
PPA validation startedPPA validation Successful
Using SERDES1 Protocol: 5205 (0x1455)
Flash: 128 MiB
NAND: 512 MiB
MMC: FSL_SDHC: 0
Using default environment

EEPROM: NXID v1
PCIe1: disabled
PCIe2: Root Complex no link, regs @ 0x3500000
PCIe3: Root Complex x1 gen1, regs @ 0x3600000
PCI:
01:00.0 - 8086:10d3 - Network controller
PCIe3: Bus 00 - 01
In: serial
Out: serial
Err: serial
SCSI: Error: SCSI Controller(s) 1B4B:9170 not found
Net: Fman1: Uploading microcode version 106.4.18
e1000: 68:05:ca:12:91:34
FM1@DTSEC1, FM1@DTSEC2, FM1@DTSEC3 [PRIME]
Error: FM1@DTSEC3 address not set.
, FM1@DTSEC4
Error: FM1@DTSEC4 address not set.
, FM1@DTSEC5
Error: FM1@DTSEC5 address not set.
, FM1@DTSEC6
Error: FM1@DTSEC6 address not set.
, FM1@TGEC1
Error: FM1@TGEC1 address not set.
, e1000#0
Hit any key to stop autoboot: 0
esbc_validate command successful
Executing script at 60060000
esbc_validate command successful
Loading kernel from FIT Image at 81000000 ...
Using 'config@1' configuration
Trying 'kernel@1' kernel subimage
Description: ARM64 Linux kernel
Type: Kernel Image
Compression: gzip compressed
Data Start: 0x810000d4
Data Size: 5381294 Bytes = 5.1 MiB
Architecture: AArch64
OS: Linux
Load Address: 0x80080000
Entry Point: 0x80080000
Verifying Hash Integrity ... OK
Loading ramdisk from FIT Image at 81000000 ...
Using 'config@1' configuration
Trying 'ramdisk@1' ramdisk subimage
Description: Ramdisk
Type: RAMDisk Image

```
Compression: uncompressed
Data Start: 0x815285e8
Data Size: 26342779 Bytes = 25.1 MiB
Architecture: AArch64
OS: Linux
Load Address: unavailable
Entry Point: unavailable
Verifying Hash Integrity ... OK
## Loading fdt from FIT Image at 81000000 ...
Using 'config@1' configuration
Trying 'fdt@1' fdt subimage
Description: Flattened Device Tree blob
Type: Flat Device Tree
Compression: uncompressed
Data Start: 0x81521e38
Data Size: 26414 Bytes = 25.8 KiB
Architecture: AArch64
Verifying Hash Integrity ... OK
Loading fdt from 0x81521e38 to 0x90000000
Booting using the fdt blob at 0x90000000
Uncompressing Kernel Image ... OK
Using Device Tree in place at 0000000090000000, end 000000009001972d
PPA validation startedPPA validation Successful
PPA validation startedPPA validation Successful

Starting kernel ...

[ 0.000000] Booting Linux on physical CPU 0x0
```

4. Secure boot Trouble Shooting.

If no message printed in the UART console, please use the following steps to do trouble shooting.

- a. Check the SecMon_HP Status Register (location 0x1e90014), Bits OTPMK_ZERO, OTPMK_SYNDROME and PE should be 0 otherwise there is some error in the OTPMK fuse blown by you.
- b. If OTPMK fuse is correct (see Step 1), check the SCRATCHRW2(0x1ee0204) register for errors.
- c. If Error code of step b is 0 then check the System Security Monitor State filed of HPSR.

System Security Monitor State (0x9)

If ITS fuse = 1, then it means ISBC code has reset the board. This may be due to the following reasons:

Hash of the public key used to sign the ESBC u-boot doesn't match with the value in SRK Hash Fuse

Or

Signature verification of the image failed.

SSM_STATE (0xd) or Non Secure State (0xb)

Check the entry point field in the ESBC header. It should be 60100000 for the demo.

If entry point is correct, ensure that u-boot image has been compiled with the required secure boot configuration.