Secure boot for Non-PBL Platform

1. Secure boot Overview

Non PBL platform supports two boot modes, trusted/secure boot and legacy/non secure boot.Secure boot where boot images are validated against a digital signature before executing, only OEM's signed boot images can be executed on these SOCs. Legacy/non-secure boot where boots happens from various boot targets.

1.1 ITS bit

Intent to secure(ITS) bit, the most important fuse value at power on reset stage. If the user set ITS, the system is operated in a secure and trusted manner, interfaces, memory permissions and MMU configurations will be locked down until trusted software is executed.

After the ITS bit is set, the system jumps to an internal secure ROM fro booting, regardless of value provided in the boot location POR. When execution begins from internal boot ROM(IBR), it checks the cfg_rom_loc value which indicates the source of the external secure boot code location. The IBR is to determine if the external secure boot code(ESBC) is valid before allowing the code to execute. If the user doesn't want to fuse the ITS bit, he can set CFG_SB_DIS to 0 to pass the system control to IBR.

1.2 ESBC with CF header and CSF header

The I2C boot sequencer provides a way to configure the system using a set of address data stored in I2C EEPROM, the boot sequencer works only in the legacy mode. In the secure boot mode, the boot sequencer is disabled, the same functionality is achieved by adding a similar data structure called a CF header, this header is on lines of the SD/MMC and eSPI data structure containing the control and configuration words. IBR running on the core first authenticates the CF header and then executes the configuration words.

ESBC with CF and CSF(Command Sequence File) Header prepended to it. The CF header contains legacy mode fields, configuration words to initialized destination interface like DDR, ESBC header pointer. The CSF header includes lengths and offset which allow the IBR to located the operands used in ESBC image validation, as well as describe the size and location of the ESBC image itself.

1.3 ESBC and boot script

ESBC can be a monolithic image including u-boot, device trees, boot firmware, drivers along with OS and applications.

The standard u-boot reserves a small space for storing environment variables, this space is typically the sector above or below the u-boot. In acase of secure boot, the micro CONFIG_ENV_IS_NOWHERE is used and environment is compiled in u-boot image called default environment, users are not be able to edit this environment, they cannot read to u-boot prompt either in case of secure boot.

ESBC validates a file called boot script, it contains information about the next image, e.g. Linux, HV etc. Boot script is a text file which contains u-boot commands. ESBC would validate this boot script before executing commands in it.

Secure boot flow with all images loaded in NOR Flash.

(1) IBR code would validate the ESBC code.

(2) On successful validation, ESBC code would run, it then validates the boot script.

(3) On successful validation of boot script, commands in boot script would be executed.

(4) The boot script contains commands to validate next level images, i.e rootfs, linux uImage and device tree.

(5) Once all the images are validated, bootm command in boot script would be executed which would pass control to linux

1.4 Steps to generate secure boot images

Build secure boot u-boot

Edit meta-fsl-ppc/conf/machine/<platform>.conf, modify UBOOT_MACHINES = "<platform_uboot_config>" to UBOOT_MACHINES = "<platform_uboot_config>_SECBOOT"

e.g. UBOOT_MACHINES = "BSC9132QDS_NAND_SECURE_BOOT "

Rebuild u-boot

$ bitbake -c clean u-boot

$ bitbake u-boot

2. Signing the images

Users can sign all the images with same public/private key pair or different key pairs to sign the images. This document will describe the same key pair method.

CST tool provided in Yocto Linux SDK is used for signing the images, which is built for the host and can be run from the host machine. In Yocto, run the command “bitbake cst-native”, and the binaries can be found in build_<machine>_release/tmp/sysroot/ x86_64-linux/usr/bin/cst.

CSF header needs to be generated for all the images.

The following describes the process of signing images.

(1) Generate the key pair to be used for signing the image.

./gen_keys 1024

Key pair – public key file – srk.pub and private key in srk.priv would be generated.

(2) Obtain hash string of the key pair generated to be programmed in SFP.
./uni_sign --hash
This would provide a 256bit hash string of the key pair generated in the previous step.

(3) Create CF and CSF headers for u-boot image.
Execute the binaries uni_cfsign and uni_sign to generate signature over CF Header and CSF header.
Example for BSC9132, generate CF header file cf_hdr.out.
./uni_cfsign input_files/uni_cfsign/bsc9132/input_nand_secure
ESBC Header generation, ESBC header file is named esbc_hdr.out.
./uni_sign --file input_files/uni_sign/bsc9132/input_uboot_nand_secure

(4) Create CSF header for Linux uImage
./uni_sign --file input_files/uni_sign/ bsc9132/ input_uimage_secure

(5) Create CSF header for rootfs
./uni_sign --file input_files/uni_sign/ bsc9132/ input_rootfs_secure

(6) Create CSF Header for hardware device tree
./uni_sign --file input_files/uni_sign/<platform>/input_dtb_secure

(7) Write Boot script
Create a text file bootscript.txt with following commands.

For BSC9132

esbc_validate 0x88040000

esbc_validate 0x88080000

esbc_validate 0x88800000

bootm 88082000 88802000 88042000

(8) Generate header over bootscript.txt which will be consumed by uboot command source
./tmp/sysroots/x86_64-linux/usr/bin/mkimage -A ppc -T script -a 0 -e 0x40 -d bootscript.txt bootscript

(9) Generate CSF header for the boot script
/uni_sign --file input_files/uni_sign/ bsc9132/ input_bootscript_secure

3. Running secure boot on BSC9132QDS board

Configuring the target board in secure boot mode, the following methods could be used.

(a) Programming the ITS fuse.
(b) Use FPGA image to program (CFG_SB_DIS) = 0 in DUT Configuration Register 12.

(1) OTPMK can be blown using CCS, the follow commands are use to blow the OTPMK with CCS.

Consider the OTPMK key obtained is :

ef0f928b52255d2bfc05be5419fd8d724241ecedfbaaaddbf2f246989fb271c1

ccs::write_mem 1 0xff7e705c 4 0 0xef0f928b
ccs::write_mem 1 0xff7e7060 4 0 0x52255d2b

ccs::write_mem 1 0xff7e7064 4 0 0xfc05be54
ccs::write_mem 1 0xff7e7068 4 0 0x19fd8d72
ccs::write_mem 1 0xff7e706c 4 0 0x4241eced
ccs::write_mem 1 0xff7e7070 4 0 0xfbaaaddb
ccs::write_mem 1 0xff7e7074 4 0 0xf2f24698
ccs::write_mem 1 0xff7e7078 4 0 0x9fb271c1
Permanently fuse the values written in shadow register above
ccs::write_mem 1 0xff7e7020 4 0 0x2

(2) ESBC uboot image and its CF and CSF headers are flashed in NAND flash. All the other images are flashed in NOR flash.
Power on the board. You have booted from NOR flash normally. Give following commands on the uboot prompt:

=> nand erase.chip
=> tftp 1000000 cf_hdr.out
=> tftp 1002000 esbc_hdr.out
=> tftp 1004000 u-boot.bin
=> nand write 1000000 0 120000

(3) Give a power on cycle to the board.

For method (a), on power on, IBR code would get control, validate the ESBC image. ESBC image would further validate the signed linux, rootfs and dtb images Linux would come up.
For method(b), on power on cycle, write SRKH, UIDs in the SFP shadow registers. Bring the core out of hold off by writing to EEBPCR register. On doing this, the secure boot flow as mentioned above would execute.
Using I2C commands to put the core in boot hold off, the following register needs to be changed

5.9.12 DUT Configuration Register 11(DUTCFG11)

The DUTCFG11 is used to sample device-specific configuration modes,particularly DUT BOOT configuration

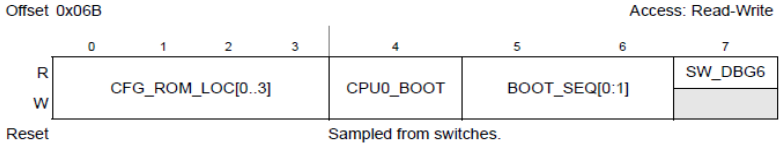

Figure 5-60. DUTCFG11 Register

The values to be programmed in this register are as follows:

Boot Mode	Defualt Value	Value with Boot Hold Off Enabled
NAND	0x9e	0x97
NOR	0xfe	0xf7
SPI	0x6e	0x67
SD	0x7e	0x77

So the I2C commands will be:

CFG_SB_DIS = 0

=>i2c mw.l 0x66 0x6c 0xc7

Core in boot HOLD off

=>i2c mw.l 0x66 0x6b 0x97 <value from table above>

Change LBMAP if ROM_LOC is changed.

=>i2c mw.l 0x66 0x50 0x44

Copy all BRDCFG/DUTCFG registers to their respective shadow registers.

=>i2c mw.l 0x66 0x10 0x60

Lock the register values.

=>i2c mw.l 0x66 0x10 0x30

->reset

Write the keys in SFP_SRKRH shadow registers from CCS.

097fb4eaea7aa38eec7c724bf3288a3b68ebfeebe30bd00176d386d905b650ed

ccs::write_mem 1 0xff7e707c 4 0 0x097fb4ea

ccs::write_mem 1 0xff7e7080 4 0 0xea7aa38e

ccs::write_mem 1 0xff7e7084 4 0 0xec7c724b

ccs::write_mem 1 0xff7e7088 4 0 0xf3288a3b

ccs::write_mem 1 0xff7e708c 4 0 0x68ebfeeb

ccs::write_mem 1 0xff7e7090 4 0 0xe30bd001

ccs::write_mem 1 0xff7e7094 4 0 0x76d386d9

ccs::write_mem 1 0xff7e7098 4 0 0x05b650ed

Write the OEMID and FSLID in shadow registers

ccs::write_mem 1 0xff7e709c 4 0 0x99999999 (OEM UID)

ccs::write_mem 1 0xff7e70b0 4 0 0x11111111 (FSL UID)

Get the core out of boot hold off.

ccs::write_mem 1 0xff701010 4 0 0x01000000

(4) The boot log for NAND secure boot

U-Boot 2013.01-00096-gc8c8ae0 (Nov 29 2013 - 20:11:18)

CPU0: BSC9132E, Version: 1.0, (0x86180010)

Core: E500, Version: 5.1, (0x80211151)

Clock Configuration:

CPU0:1000 MHz, CPU1:1000 MHz,

CCB:500 MHz,

DDR:400 MHz (800 MT/s data rate) (Asynchronous), IFC:125 MHz

L1:   D-cache 32 kB enabled

I-cache 32 kB enabled

Board: BSC9132QDS

Sys ID: 0x1f, Sys Ver: 0x31, FPGA Ver: 0x78,

IFC chip select:NAND

I2C: ready

SPI: ready

DRAM: 1 GiB (DDR3, 32-bit, CL=6, ECC off)

Flash: 128 MiB

L2:   512 KB enabled

NAND: 128 MiB

MMC: FSL_SDHC: 0

Using default environment

EEPROM: NXID v1

PCIe1: Root Complex of PCIe Slot, x1, regs @ 0xff70a000

01:00.0   - 8086:10b9 - Network controller

PCIe1: Bus 00 - 01

In: serial

Out: serial

Err: serial

Net: e1000: 00:15:17:1e:22:9e

eTSEC1 [PRIME], eTSEC2, e1000#0

Warning: e1000#0 using MAC address from net device

Hit any key to stop autoboot: 0

esbc_validate command successful

Executing script at 88022000

esbc_validate command successful

Job Queue Output status 40000516

ERROR :: 4000000 :: Error in status of the job submitted to SEC

SNVS state transitioning to Soft Fail.

SNVS state transitioning to Non Secure.

esbc_validate command successful

WARNING: adjusting available memory to 30000000

Booting kernel from Legacy Image at 88082000 ...

Image Name: Linux-3.8.13-rt9

Created: 2013-11-22 8:10:07 UTC

Image Type: PowerPC Linux Kernel Image (gzip compressed)

Data Size: 4103863 Bytes = 3.9 MiB

Load Address: 00000000

Entry Point: 00000000

Verifying Checksum ... OK

Loading init Ramdisk from Legacy Image at 88802000 ...

Image Name: fsl-image-flash-bsc9132qds-20131

Created: 2013-11-10 1:40:54 UTC

Image Type: PowerPC Linux RAMDisk Image (gzip compressed)

Data Size: 6679841 Bytes = 6.4 MiB

Load Address: 00000000

Entry Point: 00000000

Verifying Checksum ... OK

Flattened Device Tree blob at 88042000

Booting using the fdt blob at 0x88042000

Uncompressing Kernel Image ... OK

Loading Ramdisk to 2f9a1000, end 2ffffd21 ... OK

Loading Device Tree to 03ff7000, end 03fff6b5 ... OK

WARNING: Missing crypto node

Using BSC9132 QDS machine description

Memory CAM mapping: 256/256/256 Mb, residual: 256Mb

Linux version 3.8.13-rt9 (xiaodong@sbuilder136.ap.freescale.net) (gcc version 4.7.3 (GCC)) #29 SMP Fri Nov 22 16:10:03 CST 2013

Found initrd at 0xef9a1000:0xeffffd21

CPU maps initialized for 1 thread per core

bootconsole [udbg0] enabled

setup_arch: bootmem

bsc913x_qds_setup_arch()

bsc913x board from Freescale Semiconductor

arch: exit

Zone ranges:

DMA [mem 0x00000000-0x2fffffff]

Normal empty

HighMem [mem 0x30000000-0x3fffffff]