How to Enable Secboot Step by Step

Contents

Secure Boot

Enable Secure Boot

Debug

References

Questions?

Secure Boot

Secure boot refers to the bootup sequence that establishes a trusted platform for secure applications.

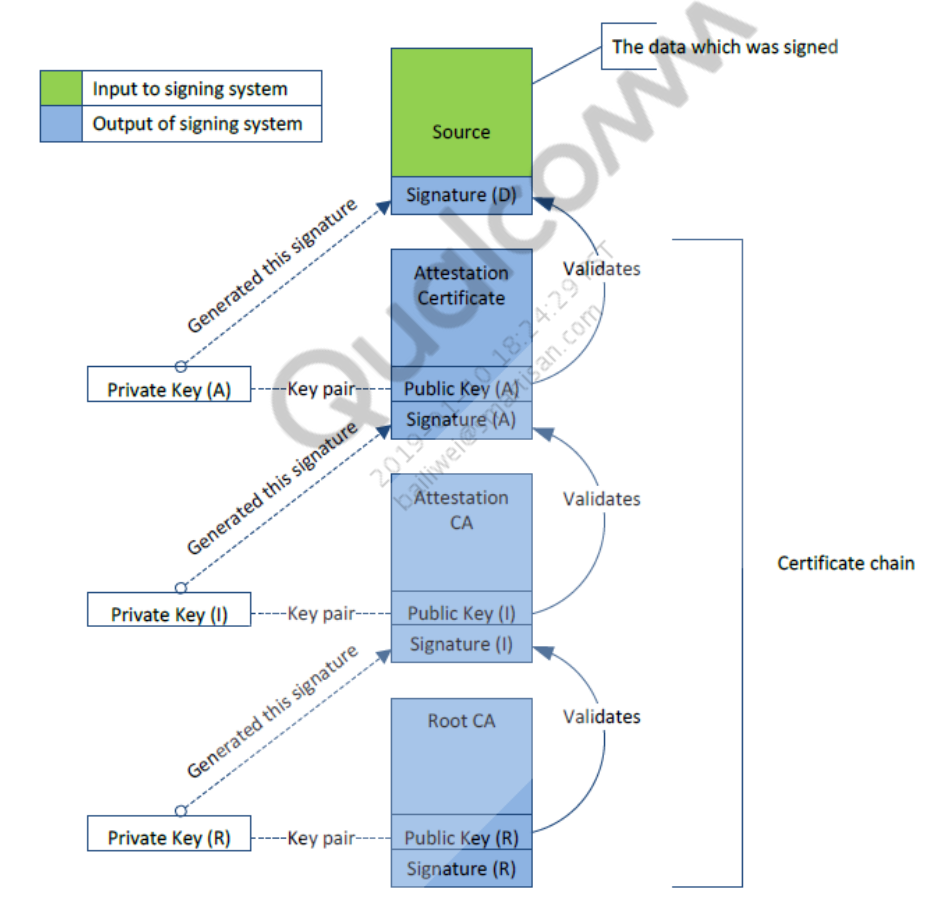
It starts as an immutable sequence that validates the origin of the code using cryptographic authentication so only authorized software can be executed. The bootup sequence places the device in a known security state and protects against binary manipulation of software and reflashing attacks.

A secure boot system adds cryptographic checks to each stage of the boot up process. This process asserts the authenticity of all secure software images that are executed by the device.

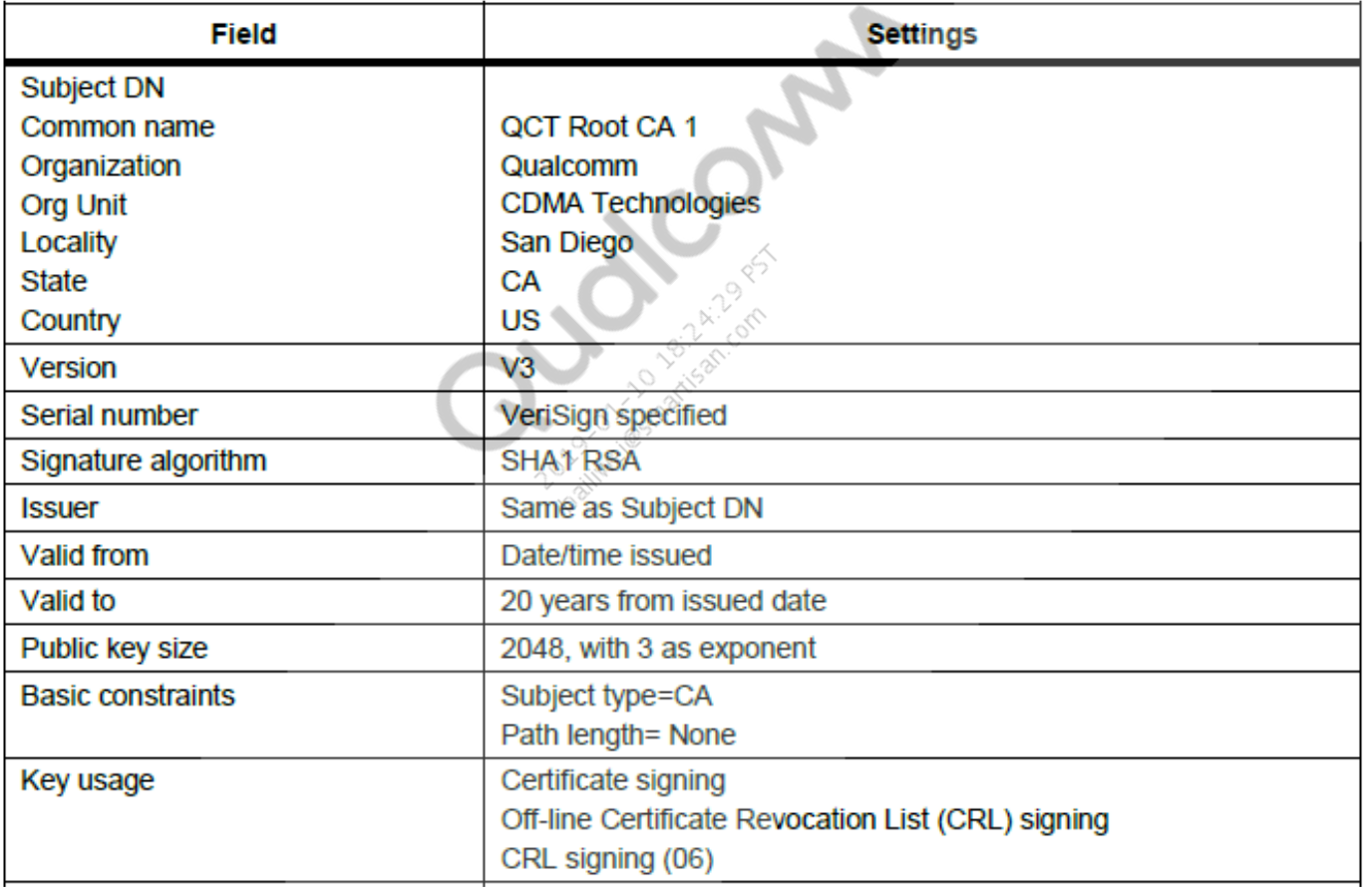
This additional check prevents any unauthorized or maliciously modified software from running on the device. Secure boot is enabled through a set of hardware fuses. For the code to be executed, it must be signed by the trusted entity identified in the hardware fuses.

To sign the images, a trusted vendor uses their private key to generate a signature of the raw code that they want to use, and adds this to the device alongside the software binary. Images (the format in which the code is packaged) can be signed using QTI’s Code signing management system (CSMS), or using the licensee’s own code signing system. Signed images include the code signature and the certificate chain. For the two-certificate chain model, the root of the certificate chain must be self-signed, which is not needed for the three-certificate chain model.

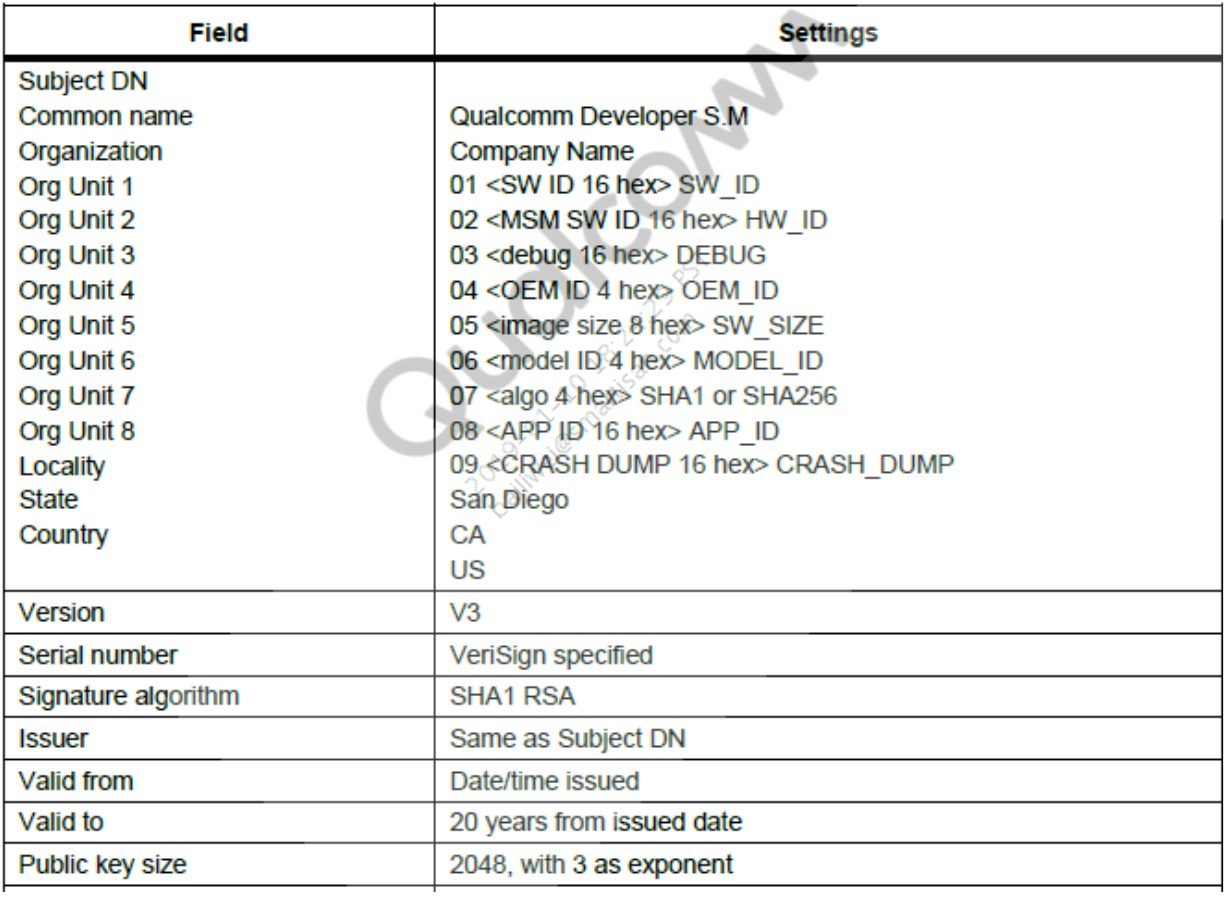
The boot up of a device comprises a multiple-stage process. Each image in the stage performs a specific function, and each image is verified by the previous image. For example, Primary Boot Loader (PBL) → Secondary Boot Loader (SBL) → ARM® TrustZone



Root CA



Attestation Certificate



SW\_ID Definitions

BL1 = 0x0 /\* SBL1 image \*/

MBA = 0x1 /\* MBA image \*/

AMSS\_HASH\_TABLE = 0x2, /\* Modem image hashtable \*/

EHOSTD = 0x3 /\* Emergency downloader image \*/

DSP\_HASH\_TABLE = 0x4, /\* lpass etc running on ADSP\*/

TZ\_KERNEL = 0x7, /\* TZBSP image \*/

APPSBL = 0x9, /\* APPSBL \*/

RPM\_FW = 0xA, /\* RPM firmware \*/

TZ\_EXEC\_HASH\_TABLE = 0xC, /\* TrustZone applications - Playready/TrustZone \*/

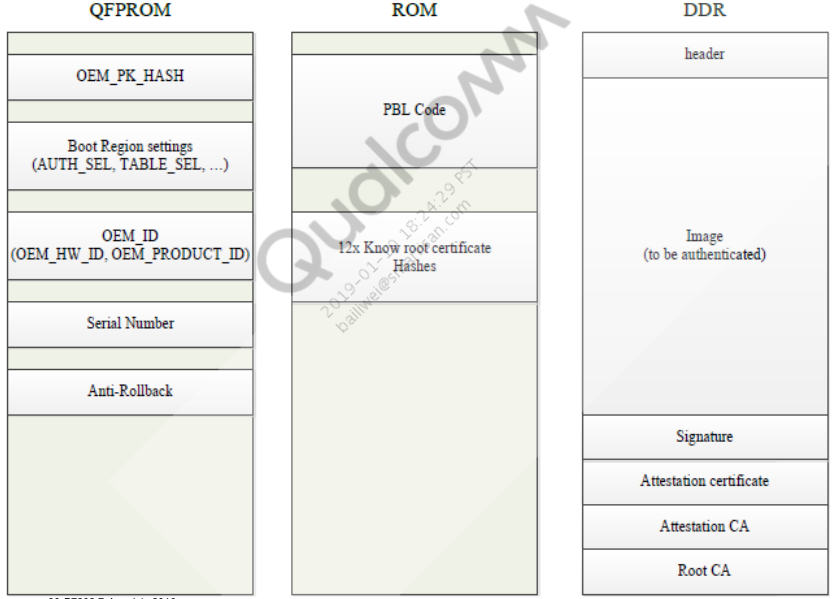
WCNSS\_HASH\_TABLE = 0xD /\* Pronto/WCN image \*/

VIDEO\_HASH\_TABLE = 0xE, /\* Venus image \*/

WATCHDOG = 0x12 /\* System debug image \*/

HYP = 0x15 /\* Hypervisor image \*/

Image Authentication Components



Enable Secure Boot

Signing tools

Generate certificate

FuseBlower configuration

SecImage configuration

Signing tools

Online

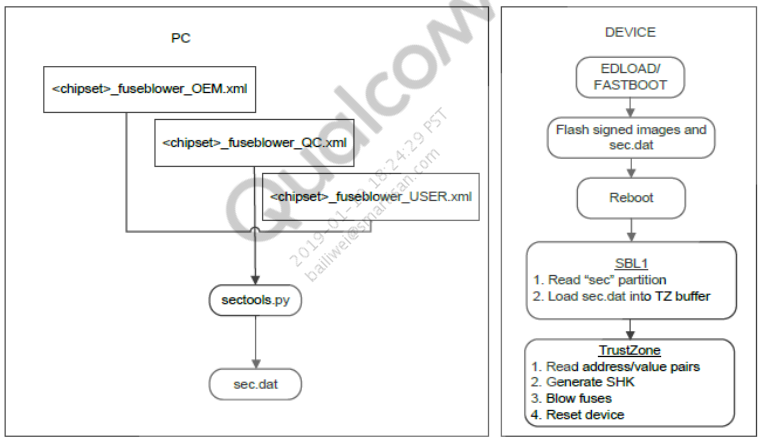
CSMS

CASS

Local

QPSA

Sectools



3. Generate Certificate

3.1 Generate root key and certificate.

3.1.1 Generate root key: Input – None, Output – oem\_rootca.key

OpenSSL Genrsa –out oem\_rootca.key -3 2048( openssl genrsa -out oem\_rootca.key -3 2048)

3.1.2Generate root cert: Input – oem\_rootca.key AND opensslroot.cfg, Output – oem\_rootca.crt

OpenSSL req -new -key oem\_rootca.key -x509 -out oem\_rootca.crt -subj /C="US"/ST="CA"/L="SANDIEGO"/O="OEM"/OU="General OEM rootca"/CN="OEM ROOT CA" -days 7300 -set\_serial 1 -config opensslroot.cfg

3.1.3 Configuration files are in sectools\resources\data\_prov\_assets\General\_Assets\Signing\openssl

3.2 Generate attest CA key, CSR, and certificate.

3.2.1 Generate attestCA key: Input – None, Output – oem\_attestca.key

OpenSSL Genrsa -out oem\_attestca.key -3 2048

3.2.2 Generate attestCA CSR: Input – oem\_attestca.key AND opensslroot.cfg , Output – oem\_attestca.csr

OpenSSL req -new -key oem\_attestca.key -out oem\_attestca.csr -subj /C="US"/ST="CA"/L="SANDIEGO"/O="OEM"/OU="General OEM attestation

CA"/CN="OEM attestation CA" -days 7300 -config opensslroot.cfg

3.2.3 Generate attestCA CERT: Input – oem\_attestca.csr AND oem\_rootca.crt AND oem\_rootca.key AND v3.ext , Output – oem\_attestca.crt

openssl x509 -req -in oem\_attestca.csr -CA oem\_rootca.crt -CAkey oem\_rootca.key -out oem\_attestca.crt -set\_serial 5 -days 7300 -extfile v3.ext

3.3 If you run into error – Unable to write 'random state', in command prompt:

Set RANDFILE=.rnd AND set HOME=[C:\](file:///C:/)

mv oem\_rootca.key qpsa\_rootca.key

mv oem\_attestca.key qpsa\_attestca.key

openssl x509 -in oem\_rootca.crt -inform PEM -out oem\_rootca.cer -outform DER

openssl x509 -in oem\_attestca.crt -inform PEM -out oem\_attestca.cer -outform DER

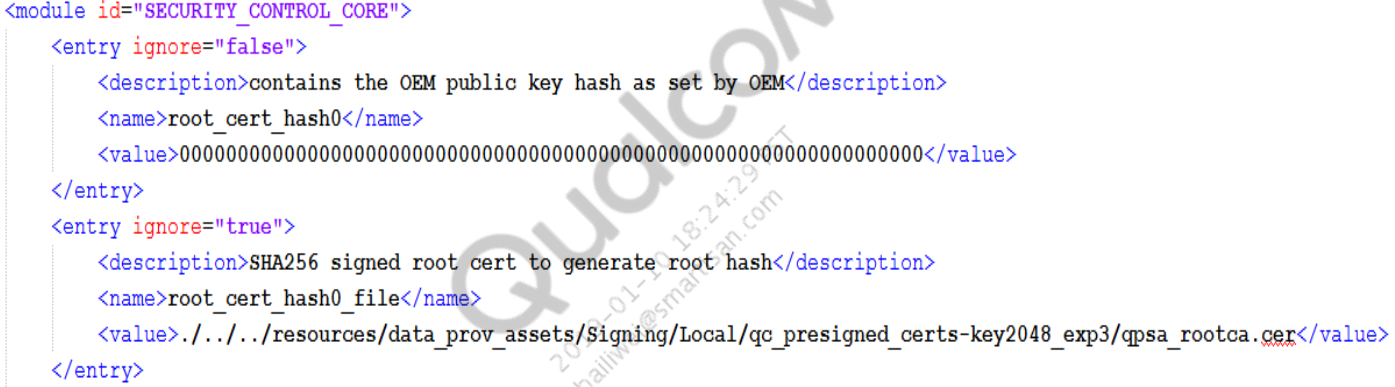
openssl dgst -sha256 qpsa\_rootca.cer

SHA256(qpsa\_rootca.cer)=8ecf3eaa03f772e28479fa2f0bbae2141ccad6f106b384d1c46263edb5b02838

4. FuseBlower Configuration

4.1 Modify cert hash value:

File: oem/common/sectools/config/xxx/xxx\_fuseblower\_USER.xml



<entry ignore="false">

<description>PK Hash is in Fuse for SEC\_BOOT1 : Apps</description>

<name>SEC\_BOOT1\_PK\_Hash\_in\_Fuse</name>

<value>true</value>

</entry>

...

<entry ignore="false">

<description>PK Hash is in Fuse for SEC\_BOOT2 : MBA</description>

<name>SEC\_BOOT2\_PK\_Hash\_in\_Fuse</name>

<value>true</value>

</entry>

...

<entry ignore="false">

<description>PK Hash is in Fuse for SEC\_BOOT3 : MPSS</description>

<name>SEC\_BOOT3\_PK\_Hash\_in\_Fuse</name>

<value>true</value>

</entry>

…

Configure oem id

<entry ignore="false">

<description>The OEM hardware ID</description>

<name>oem\_hw\_id</name>

<value>0x0</value>

</entry>

…

Configure product ID

<entry ignore="false">

<description>The OEM product ID</description>

<name>oem\_product\_id</name>

<value>xxxx</value>

</entry>

…

</entry>

<entry ignore="false">

<description>If PK Hash in Fuse is 0, then this index selects which of 16 keys in ROM to use</description>

<name>SEC\_BOOT1\_rom\_pk\_hash\_index</name>

<value>0</value>

</entry>

…

</entry>

<entry ignore="false">

<description>Use Serial Num for secure boot authentication (0: Use OEM ID (Default), 1: Use Serial Num)</description>

<name>SEC\_BOOT1\_use\_serial\_num</name>

<value>false</value>

</entry>

5. Register SEC\_BOOTx

5.1 OEM fuse settings for secure boot configuration. This can be:

Overridden with the QTI settings if necessary.

Bit 7 – Reserved

Bit 6 – Use serial number for secure boot authentication (0 – Use OEM ID(Default), 1 – Use serial number)

Bit 5 – Authentication enable (0 – No authentication required, 1 – Authentication required)

Bit 4 – PK hash in fuse (0 – SHA-256 hash of root certificate is ROM, 1 SHA-256 hash of root certificate used is in OEM\_PK\_HASH)

Bits 3 to 0 – ROM PK hash index (if PK hash in fuse is 0, then this index selects from the 16 keys in ROM for use)

6. Generate and Verify Sec.Dat

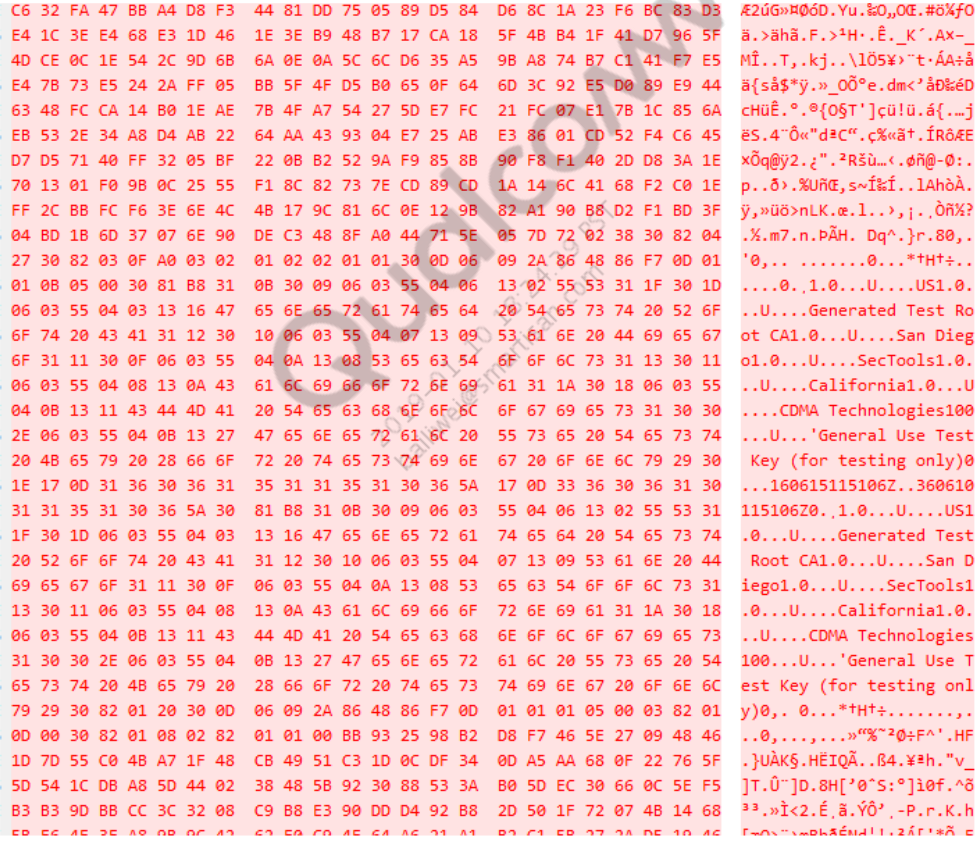
6.1 Sectools.py fuseblower -e config\xxx\xxx\_fuseblower\_OEM.xml -q config\xxx\xxx\_fuseblower\_QC.xml -u config\xxx\xxx\_fuseblower\_USER.xml -g verbose -vvv

6.2 sectools.py fuseblower

--oem\_config\_path=config\<platform>\<platform>\_fuseblower\_OEM.xml

--qc\_config\_path=config\<platform>\<platform>\_fuseblower\_QC.xml --user\_config\_path=config\<platform>\<platform>\_fuseblower\_USER.xml --secdat=<sec.dat file path> --validate

7. Verify Signed Images



8. Secure Image Configuration



9. Debug

Failure in XBL loading image

Failure in PIL loading

Failure during download through QFIL

Dump and QSEE log

Re-enable JTAG access

9.1 Failure in XBL Loading Image

B – 404674 – clock\_init, Start

D – 183 – clock\_init, Delta

B – 404948 – Image Load, Start

D – 105256 – QSEE Image Loaded, Delta – (1430940 Bytes)

B – 510234 – Image Load, Start

D – 335 – SEC Image Loaded, Delta – (2048 Bytes)

B – 517585 – sbl1\_efs\_handle\_cookies, Start

D – 549 – sbl1\_efs\_handle\_cookies, Delta

B – 525301 – Image Load, Start

D – 36478 – DEVCFG Image Loaded, Delta – (28052 Bytes)

B – 562328 – Image Load, Start

D – 44469 – RPM Image Loaded, Delta – (167672 Bytes)

B – 606828 – Image Load, Start

B – 630465 – Error code 3700000b at boot\_config.c Line 227

B – 679235 – usb:init

9.2 Failure in PIL Loading

9.2.1 Normal image loaded example:

Line 1234: [ 10.045745] subsys-pil-tz c200000.qcom,lpass: adsp: loading from 0x000000008c200000 to 0x000000008d300000

Line 1235: [ 10.240098] subsys-pil-tz c200000.qcom,lpass: adsp: Brought out of reset

Line 1236: [ 10.285683] subsys-pil-tz c200000.qcom,lpass: Subsystem error monitoring/handling services are up

Line 1237: [ 10.293538] subsys-pil-tz c200000.qcom,lpass: adsp: Power/Clock ready interrupt received

9.2.2 Abnormal image loaded example:

[ 16.558111] subsys-pil-tz soc:qcom,kgsl-hyp: a506\_zap: loading from 0x000000008f000000 to 0x000000008f002000

[ 16.565044] scm\_call failed: func id 0x42000201, arginfo: 0x82, args: 0xd, 0xf2cba000, 0x0, 0x0, ret: -2, syscall returns: 0xffffffffffffffff, 0x0, 0x0

[ 16.568231] subsys-pil-tz soc:qcom,kgsl-hyp: a506\_zap: Invalid firmware metadata

9.3 Download Failure Through QFIL

9.3.1 QFIL log:

Start download

Program

Path:D:\8953\factory\_verison\factory\_verison\HLMLAAL10M101020\firehorseimage\pr

og\_emmc\_firehose.mbn

COM port number:28

Sahara connecting

Sahara version:2

Start sending programmer

Sending programmer finished

Switch To FireHose

Wait for 3 seconds...

Max payload size to target: 49152 bytes

Device type: eMMC

Platform: 8x26

Disable ack raw data every N packets

Skip write: False

Always validate:False

Use verbose: False

COM port number:28

Sending NOP

Write file last error: 0

NOP: fail code: 9

Unable to send FireHose NOP, device is not in Firehose mode

Download fail: FireHose Fail FireHose Fail: Failed to send Firehose NOP to the phone.

Finish download

9.3.2 For QFIL flash issue, following changes are required for the MSM8953 chipset in:

boot\_images\core\storage\tools\deviceprogrammer\_ddr\src\firehose\deviceprogrammer\_initialize.c

in function void deviceprogrammer\_init\_hw()

+/\* comment out - start

#ifndef SKIP\_SECBOOT\_CHECK\_NOT\_RECOMMENDED\_BY\_QUALCOMM

// This check below is to ensure that only VIP programmer is run on

secure boot devices

// In otherwords, signing the non VIP programmer is highly not

recommended

if (FALSE == isValidationMode() && TRUE ==

isAuthenticationEnabled()) { strlcat(err\_log, "Secure boot detected.

VIP not enabled:fail ", sizeof(err\_log)); }

#endif

+ comment out - end \*/

9.4 Dump and QSEE Log

For TZ 3.0, modified code is needed to enable dump and QSEE log

trustzone\_images/core/securemsm/trustzone/qsee/kernel/src/tzbsp\_dload\_mode.c

boolean tzbsp\_allow\_memory\_dump()

{

+ if(tzbsp\_is\_dload\_mode\_set())

+

return TRUE;

+

int tzbsp\_security\_allows\_mem\_dump(uint32 \*rsp, uint32 rsplen)

{

+ \*rsp = (uint32)TRUE;

Modify tzbsp\_allow\_unlock\_xpu() to return TRUE.

trustzone\_images\core\securemsm\trustzone\qsee\oem\chipsetxxx\src\tzbsp\_oem\_lo

g.c

boolean tzbsp\_oem\_allow\_logging(void)

{

+ return TRUE;

}

For TZ 4.0, debug policy is needed to enable dump and QSEE log Common\sectools\config\8996\8996\_debugpolicy.xml

Flags

<flag> <bit\_pos>0</bit\_pos> <value>0</value> </flag> <!--DP\_ENABLE\_ONLINE\_CRASH\_DUMPS-->

<flag> <bit\_pos>1</bit\_pos> <value>0</value> </flag> <!--DP\_ENABLE\_OFFLINE\_CRASH\_DUMPS-->

<flag> <bit\_pos>2</bit\_pos> <value>0</value> </flag> <!--DP\_ENABLE\_JTAG-->

<flag> <bit\_pos>3</bit\_pos> <value>0</value> </flag> <!--DP\_ENABLE\_LOGS-->

Root hash

Image bit

<swid> <bit\_pos>0</bit\_pos> <value>0</value> </swid> <!--XBL-->

<swid> <bit\_pos>1</bit\_pos> <value>0</value> </swid> <!--MBA-->

Serial number

9.5 Re-enable JTAG Access

9.5.1 Debug OU

9.5.2 The lower 32 bits denote the action to be taken by the PBL in writing to the one-time writable OVERRIDE\_2 and OVERRIDE\_3 (APPS PBL) and OVERRIDE\_4 (modem PBL) registers. These

registers allow override of the OEM debug disable fuses:

Setting them to 1 maintains the OEM debug disabling the fuse values.

Setting them to 0 overrides the OEM debug disabling the fuse values, with the

QTI debug disabling the fuse values.

9.5.3 If the OEM wants to disable the ability to re-enable JTAG access through possible use of override registers, all the images (including SBL1 and MBA) must be signed with DEBUG OU = 0x0000000000000002. This results in APPS\_PBL and MODEM\_PBL writing 0x0 to the override

registers. Since the override registers are only one-time writable in a device reset cycle, further access to enable JTAG is prohibited by the hardware.

9.5.4 If the OEM wants to re-enable JTAG access using the override registers, the SBL1 and MBA images must be signed with DEBUG OU = (SERIAL\_NUM « 32) | 0x00000003. Other software images need not be resigned and can retain the old default value of DEBUG OU = 0x0000000000000002. This also allows the MBA and TrustZone to enable the RAM dump feature for the various subsystems. The DEBUG OU field is 64-bit, because it must change to (SERIAL\_NUM « 32) | 0x00000003. Re-enabling JTAG on MSM8916 device is done in a per-device manner.

10. References

