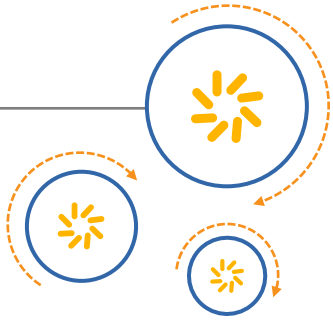
Qualcomm Technologies, Inc.



IPQ40xx Security Design

User Guide

80-Y8950-22 Rev. F

August 30, 2016

Qualcomm Technologies, Inc.
5775 Morehouse Drive
San Diego, CA 92121
U.S.A.

Revision history

Revision	Date	Description
A	September 2015	Initial release
B	November 2015	Updated Section 7
C	November 2015	Added Section 8
D	December 2015	Updated the following sections: ▪ Section 6.3 ▪ Section 6.4
E	January 2016	Added Section 8.1
F	August 2016	Updated Section 8.1.3

Contents

1 Purpose and scope

This document provides an overview about the hardware security features on the IPQ40xx platform.

It also describes the secure boot architecture and provides details about various fuses and steps involved in using this feature.

FCC NOTICE: This kit is designed to allow:

(1) Product developers to evaluate electronic components, circuitry, or software associated with the kit to determine whether to incorporate such items in a finished product and

(2) Software developers to write software applications for use with the end product. This kit is not a finished product and when assembled may not be resold or otherwise marketed unless all required FCC equipment authorizations are first obtained. Operation is subject to the condition that this product not cause harmful interference to licensed radio stations and that this product accept harmful interference. Unless the assembled kit is designed to operate under part 15, part 18 or part 95 of the FCC's rules, the operator of the kit must operate under the authority of an FCC license holder or must secure an experimental authorization under part 5 of the FCC's rules.

2 Pseudo Random Number Generator (PRNG)

The IPQ40xx platform has a hardware PRNG block for random number generation.

It is a hardware implementation of NIST SP800-90 (recommendations for random number generators), and meets the requirements of FIPS 140-3 and NIST SP800-90. It is tested using a statistical test suite from NIST as well (SP800-22).

3 Crypto

The IPQ40xx platform has a single Crypto core. The Crypto core provides hardware acceleration of standard cryptographic algorithms, which frees the CPU to perform other tasks. The Crypto core has a memory-mapped interface through which a processor can configure it. The core also has a data mover interface to move the data in and out from the Crypto core. The Crypto core handles symmetric key encryption, decryption, and authentication operations.

The Crypto core is connected to BAM (DMA engine), which allows the multiple concurrent operations to be queued to the Crypto core. There are four BAM pipes (two read-write pairs). One pair is used by the secure world (QSEE) and the other pair is used by the non-secure world (HLOS).

The following are the encryption operations:

- DES ECB, CBC
- AES ECB, CBC, IDSA, Counter, XTS, CCM

The following are the supported authentication operations:

- SHA
- SHA HMAC
- AES CMAC, CCM

The Crypto block can work using the following keys:

- Hardware based
  - Qualcomm® hardware Key
  - OEM hardware key
- Software based
  - Standard software key
  - Pipe-specific software key

The SoC implements the logic for deriving unique hardware keys for the Crypto engine according to NIST special publication 800-108-recommendation for key derivation using pseudo random functions. The block takes in the primary and secondary hardware derivation keys (256 bits each) stored in QFPROM as the input to a key derivation function. The derivation key is combined with other data and passed through multiple invocations of a pseudo random function to generate the new primary (Qualcomm hardware key), and secondary (OEM hardware key) keys to be used by the Crypto engines.

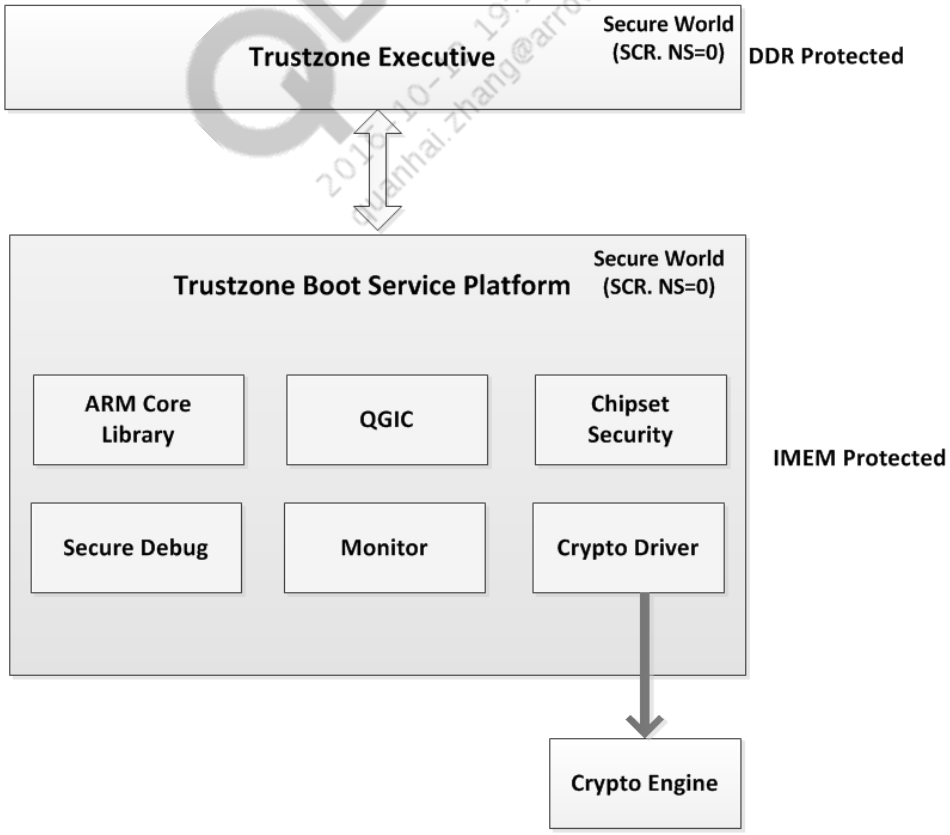
The primary and secondary hardware derivation keys in the QFPROM are write-only.

The pipe-specific software key is used when a cryptographic operation is initiated from a BAM pipe-pair. One key set is associated with each pipe-pair. The standard software key is used when the Crypto core is used directly without the BAM interface.

4 Qualcomm Secure Execution Environment (QSEE)

TrustZone (TZ) was pioneered by ARM in their v6 architecture (ARM1176 (J) processor) and significantly redesigned for v7. This hardware implementation of the TZ architecture provides a security framework that enables a device to counter many security threats from both software and hardware levels.

By providing a hardware solution, software uses this architecture to design applications or services that runs in a secure environment. This secure environment is a hardware provided, isolated execution unit that establishes separation from other non-secure execution environments. This TZ hardware solution is implemented in the IPQ40xx family of chipsets.



For IPQ40xx, the TZ software solution is referred to as Qualcomm Secure Execution Environment (QSEE).

During the boot, QSEE configures system security hardware, providing the overall system security configuration, such as:

- Configuring the secure monitor mode
- Setting up the XPUs
- Setting up the security configuration for the Crypto engine
- Configuring FIQ as secure interrupts
- Disabling the MMU and invalidating the caches for the non-secure environment
- Initializing the secondary cores for multi-core configuration
- Handling watchdog bite interrupt to perform crash dump collection
- Initiating and handling SGI interrupts, which are used to reset the other cores during a crash
- Handling XPU violation interrupts
- Controlling access to the QFPROM and other secure registers

During boot, the QSEE image is loaded into DDR by SBL and SBL transfers control to QSEE. At this stage, the processor is in a secure world. After setting up the secure configuration, QSEE switches the processor to non-secure mode before transferring controls to U-Boot.

After the secure configuration is enabled, only QSEE can access the security control registers and other secure regions, and any request to make update in these regions must be routed via QSEE. The Linux kernel and U-Boot provide a scm implementation which is used to invoke the ‘smc’ instruction to switch the processor context to secure mode. QSEE also provides an environment to execute secure applications (QsApps).

5 XPU

XPU is a combination of multiple security blocks known as protection units. The XPU configuration is performed by QSEE during the boot up.

The protected resource groups in XPUs include the following:

- Memory protection unit (MPU) – It is a software defined region of the memory selected by the address.

  The region's start and end addresses can be configured from QSEE

- Register protection unit (RPU) – It is a register or set of registers.
- Address protection unit (APU) – It is a pre-decoded address region.

  For RPU and APU, the addresses are defined by the hardware, and QSEE can configure the security parameters associated with them.

The following is the list of XPUs supported for IPQ40xx:

XPU name	XPU type
TCSR	APU
USB20_CFG	APU
USB30_CFG	APU
ESS_CFG	APU
SDCC_CFG	APU
DDR_CFG	APU
CRYPTO_CFG	APU
ADSS_CFG	APU
PCIE_CFG	APU
ANALOG CFG	APU
GCNT_CFG	APU
PRNG_CFG	APU
QPIC_CFG	APU
BLSP_CFG	APU
SEC_CTL_CFG	APU
IMEM_CFG	APU
SPDM_CFG	APU
A7_DDR_RCH	MPU
A7_DDR_WCH	MPU
SNOC_DDR	MPU
OCIMEM	MPU

XPU name	XPU type
TLMM	MPU
WIFI0_CFG	MPU
WIFI1_CFG	MPU
A7SS	MPU
PCIE	MPU
ROM	MPU
SNOC_CFG	MPU
PCNOC_CFG	MPU
QPIC	MPU
GCC	RPU

5.1 VMIDMT

XPU provides the slave side protection using either or both of these options:

- Specifying the list of masters allowed to access the slave
- Specifying the security state of the master to allow the access

On the slave side, the security state of a transaction is determined by the protection attributes on the BUS, and the Virtual Master ID (VMID) value is used by the XPU to identify the master which initiated the transaction.

On the master side, each master is connected to a VMIDMT block, which is used to specify the attributes for the BUS transactions initiated by that master. QSEE configures the VMID values for each master and also programs the permissible VMIDs for each XPU.

The following is the list of VMIDMTs in IPQ40xx:

- SPDM_CFG
- ADSS
- WSS0
- WSS1
- ESS
- PCIE
- QDSS_DAP
- QDSS_ETR
- USB2
- SDCC
- BLSP
- QPIC
- USB3
- CRYPTO_BAM

- CRYPTO_AXI
- A7SS0
- A7SS1

5.2 XPU configuration

The XPU configuration is performed by QSEE during boot up. The configurations are defined by tzbsp_xpu_config.c. The default XPU configuration can only be modified by customizing this file, and rebuilding the QSEE image.

Sample configuration:

```
static const tzbsp_mpu_rg_t g_rg_ddrc0_a7_mpu[] = {
  {0, TZBSP_XPU_NON_SEC | TZBSP_XPU_ENABLE, TZBSP_ALL_VMID_BMSK,
TZBSP_ALL_VMID_BMSK, 0x80000000, 0x87E7FFFF},
  /* RGs 1 - 3 are TZ regions */
  {1, TZBSP_XPU_SEC | TZBSP_XPU_ENABLE, TZBSP_VMID_AP_BIT |
TZBSP_VMID_CRYPTO_BIT, TZBSP_VMID_AP_BIT | TZBSP_VMID_CRYPTO_BIT,
0x87E80000, 0x87FDEFFF},
  /* TZ Diag region */
  {2, TZBSP_XPU_SEC | TZBSP_XPU_ENABLE | TZBSP_ROE, TZBSP_VMID_AP_BIT,
TZBSP_VMID_AP_BIT, 0x87FDF000, 0x87FDFFFF},
  {3, TZBSP_XPU_SEC | TZBSP_XPU_ENABLE, TZBSP_VMID_AP_BIT |
TZBSP_VMID_CRYPTO_BIT, TZBSP_VMID_AP_BIT | TZBSP_VMID_CRYPTO_BIT,
0x87FE0000, 0x87FFFFFF},
  {4, TZBSP_XPU_NON_SEC | TZBSP_XPU_ENABLE, TZBSP_ALL_VMID_BMSK,
TZBSP_ALL_VMID_BMSK, 0x88000000, 0x90000000},
```

In this sample:

- DDR is partitioned into five resource groups, and partition 0 and 4 represent the non-QSEE regions. These regions are marked as non-secure (specified by TZBSP_XPU_NON_SEC) and access is given to all masters (specified by TZBSP_ALL_VMID_MASK).
- The QSEE regions are marked as secure (specified by TZBSP_XPU_SEC) and only specific masters are given access to these regions. Access to these regions is permitted only from the secure-world.

6 Secure boot

6.1 Overview

A secure boot system adds cryptographic checks to each stage of the system boot process. Cryptographic processing verifies the authenticity of all of the secure software images that are executed by the device. This additional check prevents any unauthorized or maliciously modified software from running on the device.

To sign the images, a trusted vendor uses a private key to generate a signature of the raw code and adds this signature to the device along with the software binary. The device also contains the corresponding public key of the vendor, which can be used to verify the binary modifications and it was provided by the trusted vendor in question.

There is a three-certificate chain in use which include:

- Attestation certificate
- Attestation CA certificate
- Root certificate

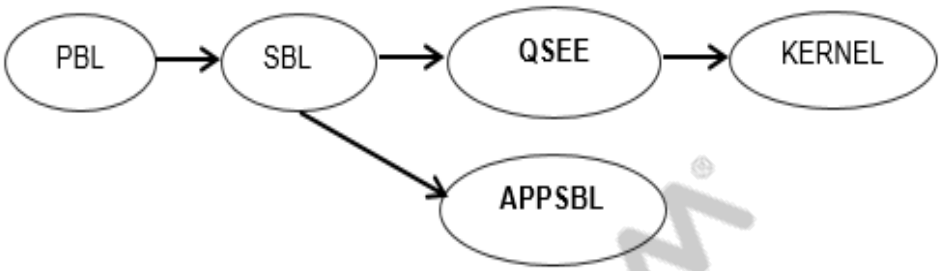
If customers use their own code signing system, then they can use the same three-certificate chain model.

The root certificate is authenticated using the SHA-256 hash of the root certificate. The hashes of 12 known root certificates are already available in the Boot ROM. In case the customer uses a different root certificate the hash can be programmed into the QFPROM region (OEM_PK_HASH).

The list of QFROM regions specific to enabling and using secure boot are mentioned in section 7.

In addition to this, Qualcomm provides Code Signing Management System (CSMS) to enable customers to sign their binaries. Customer can use CSMS along with sectools for code signing and image generation.

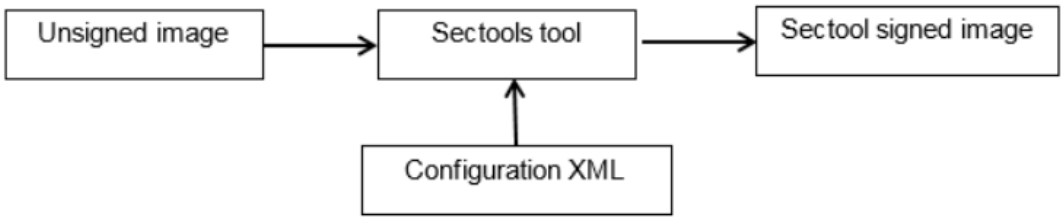
6.2 Authentication flow



- PBL loads, authenticates, and executes the SBL
- SBL loads and authenticates QSEE and APPSBL (U-Boot)
- QSEE code is executed
- QSEE code jumps to APPSBL for execution
- APPSBL authenticates the kernel image using QSEE
- QSEE authenticates the kernel and gives the control back to APPSBL
- APPSBL loads and executes the kernel

6.3 Image signing tools

The images are signed using sectools, which is developed using Python and XML. All the configuration options are specified in the config XML file. In the development phase, the default certificates provided by sectools are used (Qualcomm presigned certificates), which generates the signature and the certificates for the signed image.



The sectools are released to the customers along with the other deliverables.

6.3.1 Configuration XML

The configuration XML contains the parameters required by signing tool to generate the certificate.

For example: This is a sample from the configuration XML file:

```
<SW_ID>0x0000000000000000</SW_ID>
<OEM_ID>0x0000</OEM_ID>
<MODEL_ID>0x0000</MODEL_ID>
<MSM_PART>0x000000E1</MSM_PART>
<PART_NAME>Qualcomm</PART_NAME>
<HASH_ALG>SHA256</HASH_ALG>
<DEBUG>0x0000000000000002</DEBUG>
```

In this sample:

- SW_ID represents software revision and image type which is used for version rollback
- MSM_ID is used while generating the image digest. This is a 64 bits value. The higher order 32-bits is the MSM_PART (or JTAG ID). The lower order 32 bits of the MSM_ID can be generated using two options:
  - Serial number (32 bits)
  - OEM_ID (higher 16 bits), MODEL_ID (lower 16 bits)

For more details about the DEBUG field, see Section 6.7.

6.3.2 Sample command to sign the images

- `python sectools.py secimage -i sbl1_nor.mbn -s -c config/401x/401x_secimage.xml -o ./signed_images/`
- `python sectools.py secimage -i tz.mbn -s -c config/401x/401x_secimage.xml -o ./signed_images/`
- `python sectools.py secimage -i openwrt-ipq40xx-u-boot-stripped.elf -s -c config/401x/401x_secimage.xml -o ./signed_images/`
- `python sectools.py secimage -i openwrt-ipq806x-3.4-uImage.mbn -s -c config/401x/401x_secimage.xml -o ./signed_images`

Instead of specifying a single command as shown in the following example, the config.xml file (401x_secimage.xml) can be updated with all the images, and then sectools signs all images at once.

For Example:

```
<images_list>
   <image sign_id="sbl1_nor" name="sbl1_nor.mbn" image_type="elf_has_ht">
        <general_properties_overrides></general_properties_overrides>
        <meta_build_location>$(BUILD_PATH:boot)/boot_images/build/ms/bin/40
        1x/flashless/sbl1.mbn</meta_build_location>
             <signing_attributes_overrides>
                  <sw_id>0x0000000b00000000</sw_id>
             </signing_attributes_overrides>
   </image>
   <image sign_id="sbl1_nand" name="sbl1_nand.mbn"
   image_type="elf_preamble">
        <general_properties_overrides></general_properties_overrides>
        <meta_build_location>$(BUILD_PATH:boot)/boot_images/build/ms/bin/40
        1x/flashless/sbl1.mbn</meta_build_location>
             <signing_attributes_overrides>
                  <sw_id>0x0000000b00000000</sw_id>
             </signing_attributes_overrides>
   </image>
   <image sign_id="sbl1_emmc" name="sbl1_emmc.mbn" image_type="elf_has_ht">
        <general_properties_overrides></general_properties_overrides>
        <meta_build_location>$(BUILD_PATH:boot)/boot_images/build/ms/bin/40
        1x/flashless/sbl1.mbn</meta_build_location>
             <signing_attributes_overrides>
                  <sw_id>0x0000000b00000000</sw_id>
             </signing_attributes_overrides>
   </image>
   <image sign_id="tz" name="tz.mbn" image_type="elf_has_ht">
        <general_properties_overrides></general_properties_overrides>
        <meta_build_location>$(FILE_TYPE:download_file, ATTR:cmm_file_var,
        VAR:QSEE_BINARY)</meta_build_location>
             <signing_attributes_overrides>
                  <sw_id>0x0000000e00000007</sw_id>
             </signing_attributes_overrides>
   </image>
   <image sign_id="appsbl" name="openwrt-ipq40xx-u-boot-stripped.elf"
   image_type="elf_has_ht">
        <general_properties_overrides></general_properties_overrides>
        <meta_build_location>$(FILE_TYPE:download_file, ATTR:cmm_file_var,
        VAR:APPSBOOT_BINARY)</meta_build_location>
             <signing_attributes_overrides>
                  <sw_id>0x0000000e00000009</sw_id>
             </signing_attributes_overrides>
   </image>
   <image sign_id="openwrt_kernel" name="openwrt-ipq806x-3.4-uImage.mbn"
   image_type="mbn_40b">
```

```
        <general_properties_overrides></general_properties_overrides>
        <meta_build_location>$(BUILD_PATH:common)/common/build/ipq/openwrt-
        ipq806x-3.4-uImage.mbn</meta_build_location>
            <signing_attributes_overrides>
                <sw_id>0x0000000500000017</sw_id>
            </signing_attributes_overrides>
    </image>
</images_list>
```

NOTE: In sw_id, the higher 32-bit represent the version, and lower 32-bit represent the type, e.g., <sw_id>0x0000000500000017</sw_id>, represents version 0x5 for image type 0x17

6.3.3 CSMS/CASS

In production environments, the private key is not stored in sectools since it is only used to generate the image in a hardware-friendly format.

CSMS or customer-specific signing engine is used for image signing and signature generation. CASS is a Qualcomm maintained signing engine. CSMS can be used as the web frontend for CASS.

To support this, the <signing> section in configuration xml file has to be updated as shown in this example.

For example:

```
<signing>
    <!--
        The selected signer for signing. Supported signers are:
        local: use local signer with test keys

        csms:generate tosign file for csms upload (first run)
            package csms output zip to form signed image (second run)

        cass: use cass signer with production key
    -->
 <selected_signer>cass</selected_signer>
```

Update the corresponding signer attributes as well:

```
<cass_signer_attributes>
    …
</cass_signer_attributes>
```

6.4 Image signing format

The following image signing formats are supported:

- Binary format
- ELF format

6.4.1 Binary format

This format is used to sign binary images. The binary image cannot be signed directly using sectools. The MBN header is first added to the binary image using 'mkheader.py'. This generates the unsigned MBN image.

MBN is a Qualcomm proprietary header which contains details on the signed image, such as the image type, certificate offsets etc.

This intermediate image is then signed using the sectools.

```
python mkheader.py 0x84000028 0x17 openwrt-ipq806x-qcom-ipq40xx-fit-
uImage.itb openwrt-ipq806x-qcom-ipq40xx-fit-uImage.mbn
```

The use of mkheader.py is:

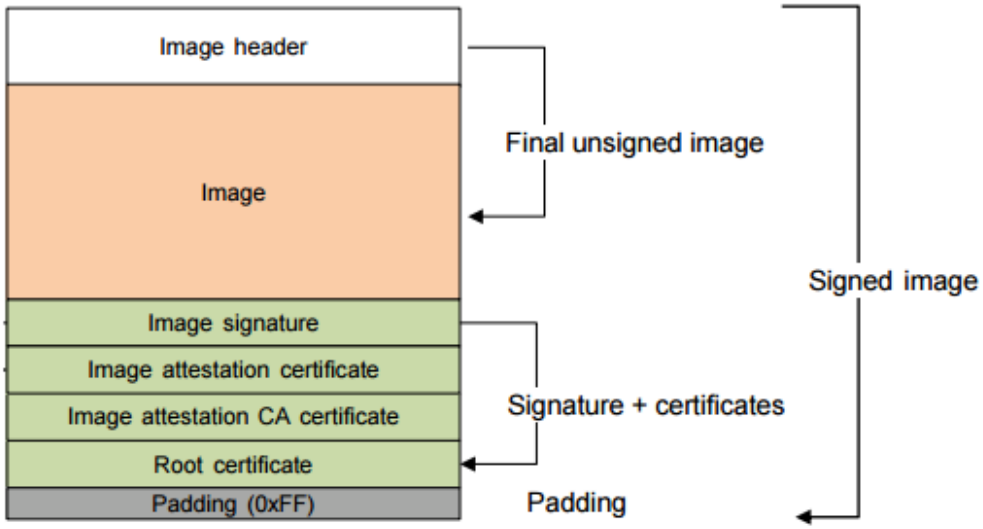
```
mkheader.py <base-addr> <img_type> <input-file> <output-file>
```

In case of NAND flash, the u-image is packed inside the UBI image. So the entire .itb image is signed.

For example:

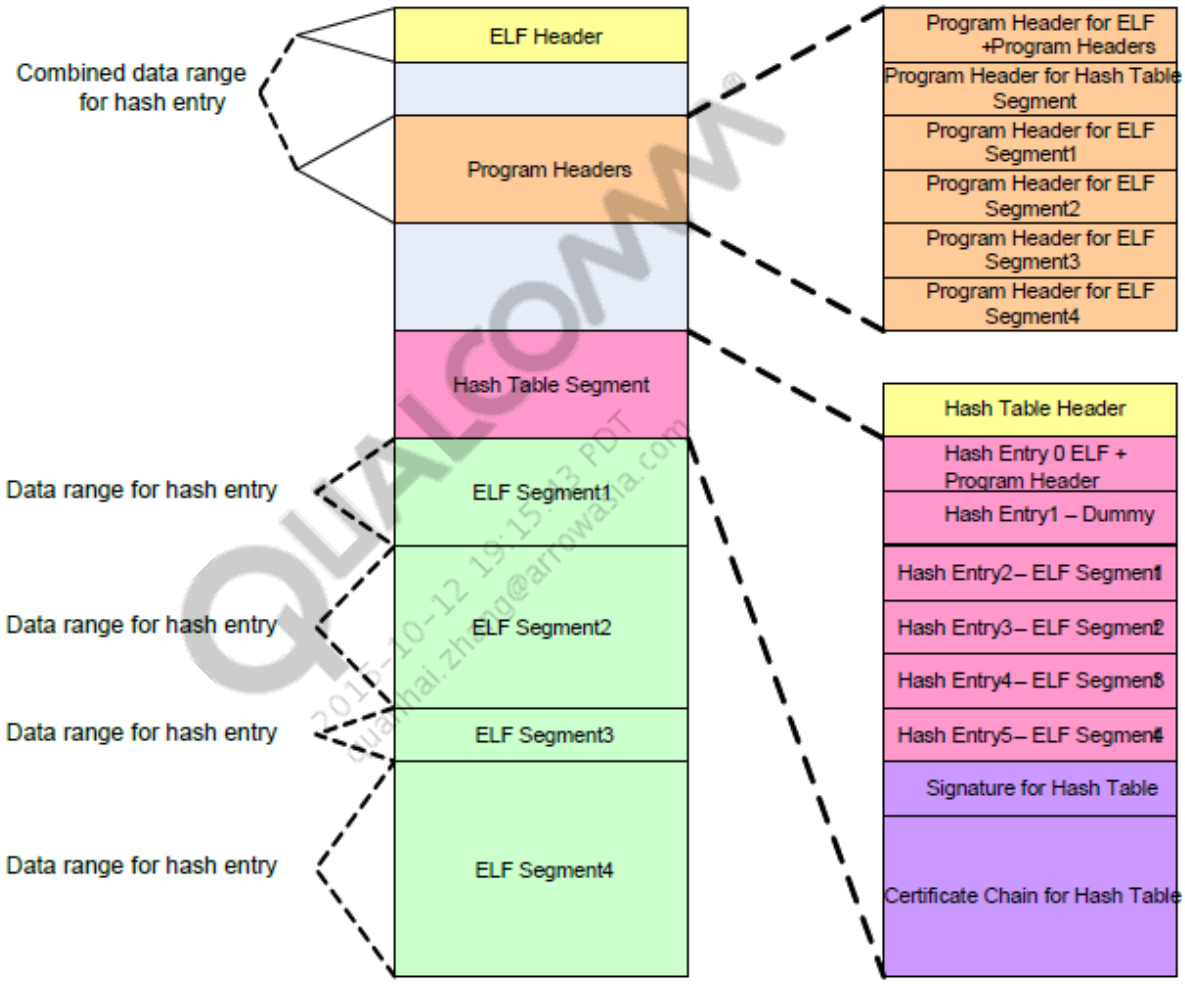
```
python mkheader.py 0x84000028 0x17 openwrt-ipq806x-qcom-ipq40xx-fit-
uImage.itb openwrt-ipq806x-qcom-ipq40xx-fit-uImage.mbn
```

The following illustration shows the final image format after the binary has been signed.



6.4.2 ELF format

The ELF images are signed using this format. Here an additional segment which contains the authentication details for the entire image is inserted into the image by sectools. The format is as follows:

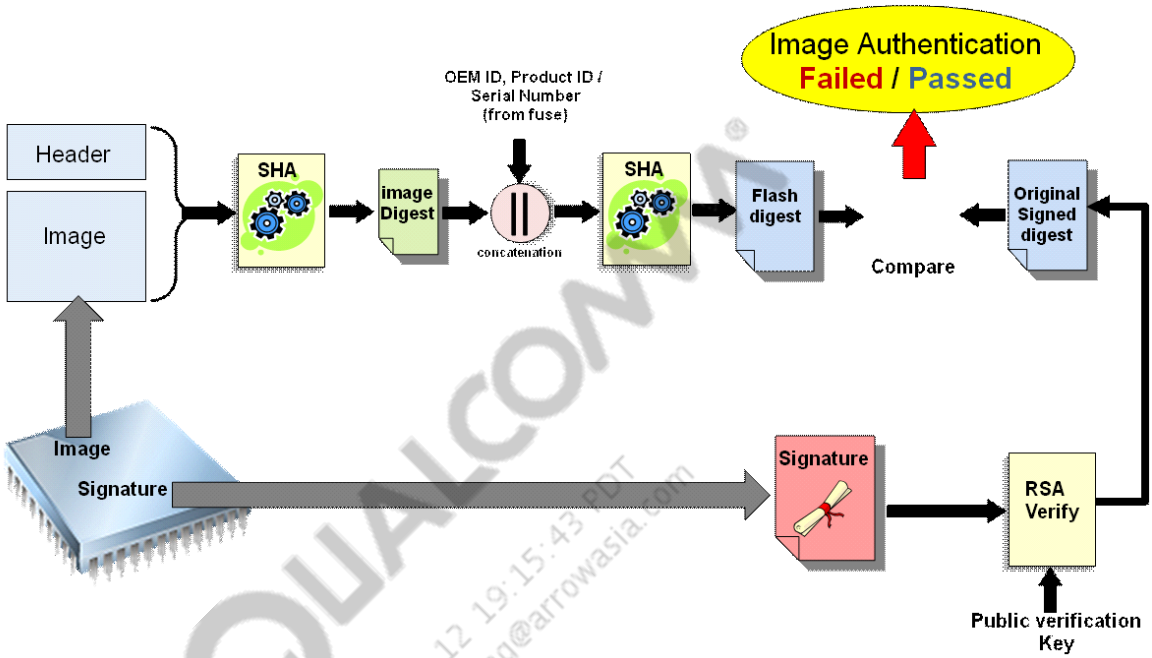


The following table shows the image formats used for the various components.

Image	Type
SBL	ELF
QSEE	ELF
APPSBL	ELF
KERNEL	Binary

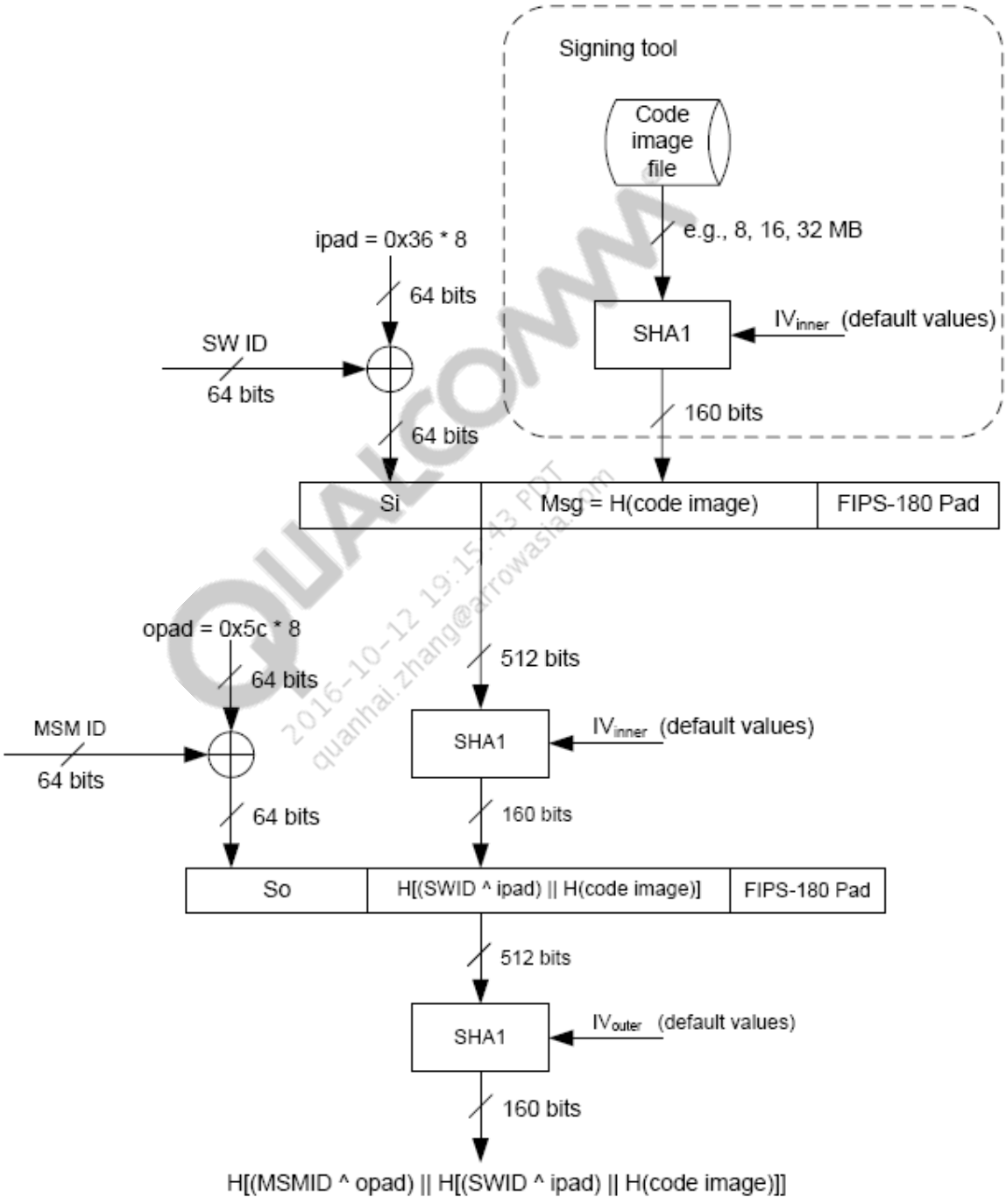
6.5 Signing and hash verification

The signature in the image is decrypted with a public key stored in the certificate, and compared with the local calculated hash digest. If they are same, the image authentication is pass; else it is fail.



6.5.1 SHA calculation

The SHA calculation algorithm is as follows:



NOTE:

- SHA256 is also supported; the output is 160/256 bits for SHA256.
- SHA256 is the recommended algorithm.

6.5.2 MSM™ hardware ID

The OEM_ID or OEM_SERIAL_NUMBER with the MSM hardware revision number (carried in the JTAG_ID register) makes up the 64-bit MSM hardware ID that is used in the hash calculation.

<table>
<tr><th colspan="3">MSM_HW_ID[63:0]</th></tr>
<tr><td>[63:32]</td><td colspan="2">[31:0]</td></tr>
<tr><td rowspan="3">JTAG_ID
masked with
0x0FFFFFFFUL</td><td>[31:16]</td><td>[15:0]</td></tr>
<tr><td>OEM_HW_ID</td><td>OEM_MODEL_ID</td></tr>
<tr><td colspan="2">SERIAL_NUMBER</td></tr>
</table>

The blown e-fuse combination determines whether the OEM_ID or OEM_SERIAL_NUMBER is used.

NOTE: The top four bits of JTAG_ID contain the chip revision, which is not considered while computing the MSM_HW_ID value.

6.6 Version rollback

The version rollback is an optional feature that can be enabled, if secure boot is being used. When enabled, the version of the boot components are checked against their versions in the fuse. The image validation is successful only if all the images have version greater than or equal to the version in their respective fuses.

If the maximum version is reached on the fuse, the boot is allowed until the image version is greater than or equal to the fuse version.

The image version is obtained from the SW_ID, which is inserted into the Attestation certificate. When the software image is upgraded using the QSDK scripts, the corresponding version numbers in the fuse are also updated. During upgrade if the maximum version is reached on the fuse, the upgrade will be permitted if the image authentication is successful, but the version blown into the fuse remains the same.

The following table shows the maximum version for each component. The maximum version is derived from the number of bits available in the fuse for each component.

Image	Maximum support version
SBL	11
QSEE	14
APPSBL	14
KERNEL	32

6.6.1 Configuration

When version rollback is enabled in the QFPROM, the Linux kernel driver for QFPROM creates the following entries.

- These files do not require manual manipulation. The 'sysupgrade' command in QSDK, which performs the firmware upgrade, internally manipulates these files to update the firmware version numbers in the QFPROM.

  If the version number used is higher than the maximum supported value, the Kernel driver prints an error in the kernel log.

  - /sys/devices/system/qfprom/qfprom0/sbl_version
  - /sys/devices/system/qfprom/qfprom0/tz_version
  - /sys/devices/system/qfprom/qfprom0/hlos_version
  - /sys/devices/system/qfprom/qfprom0/appsbl_version

For example:

```
$ cat /sys/devices/system/qfprom/qfprom0/sbl_version
0
$echo 1 > /sys/devices/system/qfprom/qfprom0/sbl_version
$cat /sys/devices/system/qfprom/qfprom0/sbl_version
1
```

Running cat command on these files returns the current version, and writing a version to this file using the echo command, results in that version being blown in to the e-fuse.

If the value exceeds the maximum supported value, then the maximum value is used and all the version bits corresponding to that component blows.

6.7 Debug override

To provide more security, restrict access to device processors through the JTAG ports.

NOTE: When secure boot is enabled, it is recommended that the debug disable fuses are blown to provide additional security.

In case of debugging for field engineers, the provision to re-enable the JTAG is provided. The one-time writable debug override registers can be used to override the debug disable fuse settings and re-enable JTAG if the debug disable fuses are blown. QSEE re-enables JTAG by writing to one to these override registers if the SBL image is signed with debug on.

NOTE: If debug is re-enabled, then Crypto block cannot access the hardware keys in the QFPROM, and uses the dummy hardware keys.

6.7.1 Configuration XML

Modify the configuration field 'DEBUG' to achieve the functionality mentioned in section 6.7.

This is a sample from the configuration XML file:

```
<SW_ID>0x0000000000000000</SW_ID>
<OEM_ID>0x0000</OEM_ID>
<MODEL_ID>0x0000</MODEL_ID>
<MSM_PART>0x000000E1</MSM_PART>
<PART_NAME>Qualcomm</PART_NAME>
<HASH_ALG>SHA256</HASH_ALG>
<DEBUG>0x0000000000000003</DEBUG>
```

In this configuration the DEBUG value is set to 0x3. The SBL image must be signed with this configuration. When this SBL image is used to boot the board, QSEE can detect this as debug image, and configures the override registers to re-enable JTAG. During the subsequent boot, if this image is not used, the JTAG remains disabled for the session.

7 QFPROM regions

The following list shows the QFPROM regions relevant to this document:

Name	Address	Bits	Name	Description
QFPROM_RAW_JTAG_ID	0x00058000	19:0	JTAG ID	These bits map to 27:12 of the JTAG ID.
QFPROM_RAW_SERIAL_NUM	0x00058008	31:0	SERIAL_NUM	This is a unique 32-bit serial number blown into the chip during wafer sort to track information like wafer number of coordinates of the chip.
QFPROM_RAW_ANTI_ROLLBACK_1_LSB	0x00058018	11:1	SBL1	This field determines the lowest version of the secondary boot loader software that can run.
		25:12	TZ	This field determines the lowest version of the TrustZone software that can run.
QFPROM_RAW_ANTI_ROLLBACK_1_MSB	0x0005801C	31:18	APPSBL	This field contains the least significant bits of the lowest version of the Apps boot loader software that can run.
QFPROM_RAW_ANTI_ROLLBACK_2_LSB	0x00058020	31:0	KERNEL	This field determines the lowest version of the HLOS software that can run.
QFPROM_RAW_OEM_CONFIG_ROW0_MSB	0x00058034	19	ANTI_ROLLBACK_FEATURE_EN	Bit 0 - BOOT_ANTI_ROLLBACK_EN
		18:14	JTAG Disable	Fuse for disabling JTAG port.
QFPROM_RAW_OEM_CONFIG_ROW1_LSB	0x00058038	31:16	OEM_PRODUCT_ID	The OEM product ID. Bits 15:0
		15:0	OEM_HW_ID	The OEM hardware ID. Bits 15:0
QFPROM_RAW_PRI_KEY_DERIVATION_KEY_ROWn_LSB, n=[0..3]	n=[0..3] : 0x00058058+0x8*n	31:0	KEY_DATA0	32 bits of the key data in this row.
QFPROM_RAW_PRI_KEY_DERIVATION_KEY_ROWn_MSB	n=[0..3] : 0x0005805C+0x8*n	31:0	KEY_DATA1	32 bits of the key data in this row.
QFPROM_RAW_SEC_KEY_DERIVATION_KEY_ROWn_LSB	n=[0..3] : 0x00058078+0x8*n	31:0	KEY_DATA0	32 bits of the key data in this row.
QFPROM_RAW_SEC_KEY_DERIVATION_KEY_ROWn_MSB	n=[0..3] : 0x0005807C+0x8*n	31:0	KEY_DATA1	32 bits of the key data in this row.

Name	Address	Bits	Name	Description
QFPROM_RAW_OEM_SEC_BOOT_ROW0_LSB	0x00058098	7:0	SEC_BOOT1	▫ Bit 7: Reserved ▫ Bit 6: Use Serial Num for secure boot authentication (0: Use OEM ID (Default), 1: Use Serial Num) ▫ Bit 5: Authentication Enable (0: no auth, 1: auth required) ▫ Bit 4: PK Hash in Fuse (0: SHA-256 hash of root cert is ROM, 1: SHA-256 hash of root cert to use is in OEM_PK_HASH) ▫ Bits 3-0: ROM PK Hash Index (If PK hash in fuse is 0, then this index selects which of 16 keys in ROM to use)
QFPROM_RAW_PK_HASH_ROWn_LSB	n=[0..3] : 0x000580A8+0x8*n	31:0	HASH_DATA0	32 bits of the hash data in this row.
QFPROM_RAW_PK_HASH_ROWn_MSB	n=[0..3] : 0x000580AC+0x8*n	31	PK_HASH_ROW_FEC_EN	This fuse is used to enable the Forward error correction for this row of PK Hash.
		30:24	FEC_VALUE	Forward error correction value that matches the value blown in the bits 23:0 below and 32 bits in the LSB register. These are needed only if FEC is enabled on this row of the QFPROM.
		23:0	HASH_DATA1	23 bits of the hash data in this row.
QFPROM_RAW_PK_HASH_ROW4_LSB	0x000580C8	31:0	HASH_DATA0	32-bits of the hash data in this row.
QFPROM_RAW_PK_HASH_ROW4_MSB	0x000580CC	31	PK_HASH_ROW_FEC_EN	This fuse is used to enable the Forward error correction for this row of PK Hash.
		30:24	FEC_VALUE	Forward error correction value that matches the value blown in the 32 bits in the LSB register. These are needed only if FEC is enabled on this row of the QFPROM.

8 Blowing secure boot fuse

8.1 Fuse blower tool

The 401x_fuseblower_USER.xml and 401x_fuseblower_QC.xml files allow users to enable/disable the specific features, and will automatically set the appropriate QFPROM bits in the other XML files. There are also fields to enter any specific values, such as OEM_PK_HASH, OEM HW and product IDs, etc., such that all changes can be made in the fuse blower tool.

8.1.1 QFRPOM write values

Write values contains the correct values to be written to the QFPROM locations. FEC-enables and read/write-disables are at the bottom, as these must be the last two QFPROM locations written. The order of writing the other QFPROM locations is not significant.

8.1.2 Notes

Once QFPROM is programmed, all possible regions should be write-disabled to prevent any further changes, whether specific regions are used/programmed or not. Only anti-rollback version regions should not be write disabled and these regions must be changed with each anti-rollback feature update.

Do not delete or modify any portion of the xml except the fuseblower_USER.xml and fuseblower_QC.xml. Bit auto generation and addressing might be corrupted if any other changes are made.

8.1.3 Steps

This section describes the ways of modifying the fuse blower and generating the sec.dat file.

8.1.3.1 Modify 401x_fuseblower_QC.xml

```
<field id="SEC_BOOT1">
 <description></description>
 <owner>QC</owner>
 <value>0x20</value>  // enable authenticate
 <bits>7:0</bits>
 </field>
<field id="SEC_BOOT2">
```

8.1.3.2 Modify 401x_fuseblower_USER.xml

```
 <module id="SECURITY_CONTROL_CORE">
 <entry ignore="true">
 <description>contains the OEM public key hash as set by OEM</description>
 </entry>
 <entry ignore="false">
 <description>SHA256 signed root cert to generate root hash</description>
 <name>root_cert_file</name>

<value>./../../resources/data_prov_assets/Signing/Local/qc_presigned_certs-
key2048_exp65537/qpsa_rootca.cer</value>
 </entry>
 <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>If PK Hash in Fuse is 0, then this index selects which of 16
keys in ROM to use</description>

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

<entry ignore="false">
 <description>The OEM hardware ID</description>
 <name>oem_hw_id</name>
-<value>0x0000</value>
+<value>0x0001</value>
```

8.1.3.3 Modify 401x_secimage.xml

```
<!--IPQ4018: 0x009780E1
 IPQ4019: 0x009790E1-->
-<msm_part>0x009790E1</msm_part>
-<oem_id>0x0000</oem_id>
+<msm_part>0x009780E1</msm_part>
+<oem_id>0x0001</oem_id>
 <model_id>0x0000</model_id>
 <debug>0x0000000000000002</debug>

<meta_build_location>$(FILE_TYPE:file_ref, ATTR:cmn_file_var,
VAR:ENORPRG40xx_mbn)</meta_build_location>
 </image>
 <image sign_id="tz" name="tz.mbn" image_type="elf_has_ht">
 <general_properties_overrides>
 <sw_id>0x0000000000000007</sw_id>
 </general_properties_overrides>

</general_properties_overrides>
 <meta_build_location>$(FILE_TYPE:file_ref, ATTR:cmn_file_var, VAR:openwrt-
ipq40xx-u-boot-stripped_elf)</meta_build_loca
 </image>
    <image sign_id="openwrt_kernel" name="openwrt-ipq-ipq40xx-fit-
uImage.mbn" image_type="mbn_40b">
 <general_properties_overrides></general_properties_overrides>
 <general_properties_overrides>
 <sw_id>0x0000000000000017</sw_id>
 </general_properties_overrides>
   <meta_build_location>$(BUILD_PATH:common)/common/build/ipq/openwrt-ipq-
ipq40xx-fit-uImage.mbn</meta_build_location>
 </image>
```

8.1.3.4 Generate efuse data

```
python sectools.py fuseblower -p 401x –ga
```

The sec.dat is generated in the following path:

```
./common_output/v1/sec.dat
```

8.1.3.5 Burn sec.dat to effuse

1. In Uboot, transfer the sec.dat file in a DDR space.
   ```
   tftpb 0x88000000 sec.dat
   ```
2. Use the following command to invoke the scm call to QSEE which in turn blows the fuses from the information in sec.dat.
   ```
   fuseipq 0x88000000
   ```

On success, an on-screen print is displayed and the user may perform other appsbl-related activities and the board has to be rebooted for the fuses to take effect.

On failure cases, an on-screen print is displaced and the root cause for the failure can be analyzed by dumping the TZ diag region. For more information about the fuse blower tool, refer R6.

8.2 Debugger script

The following is an example of debugger script that is used to program the OTP for secure boot.

```
; Blow fuse for secure boot (using OEM PK Hash)
AREA.VIEW
AREA.RESET

;=========================================================;
;  main
;=========================================================;
gosub fuseInit
PRINT "Fuse Init Complete"

gosub fuseBlowBit
PRINT "Fuse Blow Complete"

gosub fuseBlowDone
PRINT "FUSE BLOW FOR OEM PK HASH COMPLETE!!! Ensure JTAG ID is also
blown!!!"
ENDDO

;=========================================================;
;  fuseInit
;=========================================================;
; QFPROM 12MHz clock config
fuseInit:

local &addr &data

; Enable LDO Voltage
&addr=0x1948000
&data=Data.Long(ASD:&addr)
&data=(&data)|0x80000000
```

```
d.s &addr %LE %LONG &data
PRINT "LDO Voltage Enabled"

; Set Blow timer
d.s ASD:0x5a03c %LE %LONG 0x30
PRINT "Blow Timer Set"

; Set the PPI timer value
d.s ASD:0x5a058 %LE %LONG 0xc
PRINT "PPI Timer Set"

RETURN

;=========================================================;
;  fuseBlowBit
;=========================================================;
fuseBlowBit:

PRINT "Blowing Fuse for OEM PK Hash"

d.s ASD:0x000580A8 %LE %LONG 0xaa3ecf8e
d.s ASD:0x000580AC %LE %LONG 0xe072f703
d.s ASD:0x000580B0 %LE %LONG 0xfa7984e2
d.s ASD:0x000580B4 %LE %LONG 0xe5ba0b2f
d.s ASD:0x000580B8 %LE %LONG 0xca1c14e2
d.s ASD:0x000580BC %LE %LONG 0xf006f1d6
d.s ASD:0x000580C0 %LE %LONG 0xc4d184b3
d.s ASD:0x000580C4 %LE %LONG 0xd7ed6362
d.s ASD:0x000580C8 %LE %LONG 0x3828b0b5
d.s ASD:0x000580CC %LE %LONG 0xca000000
d.s ASD:0x00058098 %LE %LONG 0x00303030

RETURN

;=========================================================;
;  fuseBlowDone
;=========================================================;
fuseBlowDone:

local &addr

&addr=0x1948000
&data=Data.Long(ASD:&addr)
&data=(&data)&0x7FFFFFFF
```

```
d.s &addr %LE %LONG &data
PRINT "LDO Voltage disabled"
RETURN
```

9 Limitations

- The system crash dump feature is disabled when secure boot is enabled. This feature can be enabled when debug is re-enabled using the details mentioned in section 6.7
- When debug override is enabled, the Crypto engine does not have access to the hardware keys. Hence dummy Crypto keys are used.

10 References

	Document name	Document number
R1	*Crypto Core Hardware Programming Guide for OEMs*	80-VK772-5HX
R2	*Code Signing Management System User Guide*	80-V8000-1
R3	*Sectools Secimage User Guide*	80-NM248-1
R4	*Sectools Csms Proxy User Guide*	80-NM248-7
R5	*Sectools Cass Overview*	80-P1166-1
R6	*Sectools Key provision Tool User Guide*	80-NM248-5