Qualcomm Technologies, Inc.

Secure Boot and Image Authentication

Technical Overview (v2.0) Alexander W. Dent

August 21, 2019

Disclaimer

Contents

1. Overview

Secure boot provides a foundation for the security architecture of the device. Technically, secure boot is defined as a boot sequence in which each software image that is loaded and executed on a device is authorized using software previously authorized by this system. This sequence is designed to prevent unauthorized or modified code from being run by ensuring that all code is checked before it is executed.

The first image in this "chain of trust" is called the *Primary Boot Loader* (PBL). The Primary Boot Loader is stored in immutable read-only memory; it is literally part of the fabric of each chip. The user can have confidence that this image has not been altered because it cannot be physically altered. This initial software image cryptographically verifies digital signatures on the images that it loads. Those images cryptographically verify the digital signatures on the next set of images that they load, and so on. Hence, the user can have confidence that these images have not been altered because, in every case, a piece of trusted software checks the new image before it can be executed.

In the latest version of Qualcomm Technologies' (QTI's) secure boot feature, the PBL loads *two* images as shown in Figure 1:

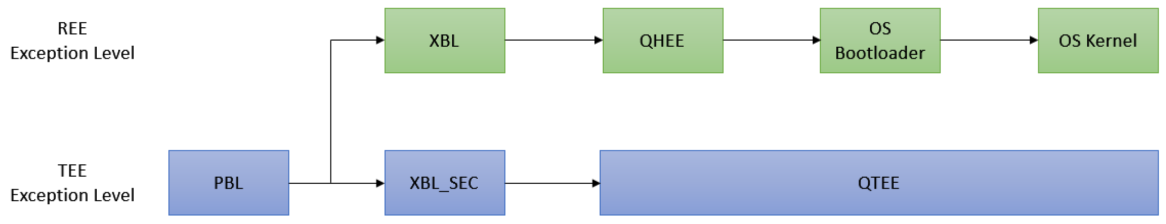


Figure 1: PBL loading the REE and TEE images.

- An eXtensible Boot Loader (XBL) image which coordinates the loading of the rich execution environment (REE). This includes the OS kernel and peripheral firmware images. This chain of software images runs at the same exception level as the OS kernel.

- A highly trusted Secure eXtensible Boot Loader (XBL_SEC) image which coordinates the loading of the trusted execution environment (TEE). This consists of the Qualcomm® Trusted Execution Environment (Qualcomm TEE) image and related images. This chain of software images runs at the same exception level as the Qualcomm TEE.

The isolation of the TEE images during the loading process is designed to improve security by shortening the chain of images that must be loaded, authorized, and executed before the Qualcomm TEE image is operational. Essentially, it reduces the opportunity for the Qualcomm TEE image to be corrupted. The images in the TEE may be digitally signed by both Qualcomm Technologies and the device manufacturer. This "double-signing" is designed to increase the confidence that all reliant parties have in the Qualcomm TEE image that is executed. We aim to safeguard critical assets protected by the Qualcomm TEE, such as user-data encryption keys and payment-application authentication keys, even in the event of a breach of a less-trusted environment.

All images make use of the standard ELF. An ELF image consists of some number of individual ELF segments, which might contain code, data, or metadata about the image. The data that is used to authenticate that image is contained in a special segment within the ELF file, called the *hash segment*.

The hash segment contains a table of cryptographic hash values and a collection of metadata about the image. The cryptographic hash values in the table are computed over the other segments in the ELF image and are used to verify the correctness of those segments when they are loaded into memory. The metadata contains information about the type of image and the type of hardware on which the image is designed to be executed.

The hash segment also contains a chain of X509 certificates and a digital signature which authenticates the hash table and image metadata. The first X509 certificate in the chain (known as the *root certificate*) is authenticated against values burnt into the hardware and each subsequent certificate is authenticated against the prior certificate. This chain of X509 certificates finally authenticates a public key that is used to verify the digital signature, which proves the correctness of the hash table and image metadata.

The process for authenticating a secure-boot-enabled ELF image is therefore:

1. Verify the root X509 certificate against the value in hardware.
2. Verify each X509 certificate in the certificate chain using the previous certificate in the chain.

3. Verify the digital signature on the image metadata and hash table using the public key in the final X509 certificate in the certificate chain.
4. Verify that the image being loaded has the correct image type and is designed to run on the given hardware using the image metadata.
5. Verify each ELF segment against the corresponding cryptographic hash value in the hash table.

After this process is complete, the image should be trusted by the device, which may involve relying on the data in the image or passing execution control to the entry point provided by that image.

The digital signature is computed over the hash table and image metadata, rather than over the entire ELF image, as this relaxes memory size requirements and increases the flexibility of the loading process.

Unlike earlier versions of QTI's secure boot feature, the image metadata is contained in a separate region of the hash segment, rather than being encoded in the X509 certificate chain. This is engineered to simplify the image metadata parsing, reduce opportunities for mistakes, and make it easier for standard industry tools to analyze the X509 certificate chain.

The following sections of this document discuss the way in which the TEE images are loaded, the structure of the signed ELF image, the structure of the hash segment, and the options that are available to the signer within the image metadata region.

2. TEE Loading

The QTI secure boot architecture is designed to maintain a separation between the trusted execution environment (TEE) that acts as the trusted core of the device and the rich execution environment (REE) which provides the wide range of services required by users.

We achieve this separation by having the initial boot loader—the ROM-based primary boot loader—load two images. The first is the REE boot loader image. It is responsible for authenticating and executing the OS image and ultimately the firmware within the device. It runs at the same exception level as the OS kernel (EL1). The second image is the TEE boot loader image. It is responsible for authenticating and executing the TEE image. This image runs in the ARM Secure Monitor execution environment (EL3).

The secure boot architecture maintains a separation between the REE and TEE environments from the initial execution of software on the chip. The images have been designed to prevent any software running outside of the ARM TrustZone environment from being able to compromise software running inside the ARM TrustZone environment, even during the secure boot process.

The first image that will be executed after the ROM-based primary boot loader is the TrustZone-based XBL_SEC image. This image will configure the access control system in a way which is designed to isolate the memory used by ARM TrustZone from all other execution environments on the chip and then execute the XBL image at a less-privileged exception level. Communication between the XBL image and the XBL_SEC image will use the standard ARM SMC mechanism designed to facilitate communication between an REE and a TEE.

The XBL_SEC image acts as a root of trust for all TrustZone images and the Qualcomm TEE in particular. Since the XBL_SEC image does not have direct access to the storage device, it relies on the XBL image to copy the Qualcomm TEE images from storage into the chip's memory. The XBL_SEC image is engineered to isolate the Qualcomm TEE image so that it can only accessed by XBL_SEC, before authenticating and executing the Qualcomm TEE image.

The XBL image acts as a root of trust for all non-TrustZone images that will run on the chip, including the Qualcomm® Hypervisor Execution Environment, the OS boot loader (e.g., UEFI), the OS kernel (e.g., the Android kernel) and the peripheral images (such as the Bluetooth and WLAN images). The XBL image will directly load those images or will be responsible for loading intermediate software which will load those images (see Figure 2).

The Qualcomm TEE image may be signed by both QTI and the device manufacturer in a process known as double-signing. This is designed to ensure that this security-critical image can only be executed if it has been approved by QTI *and* the device manufacturer.

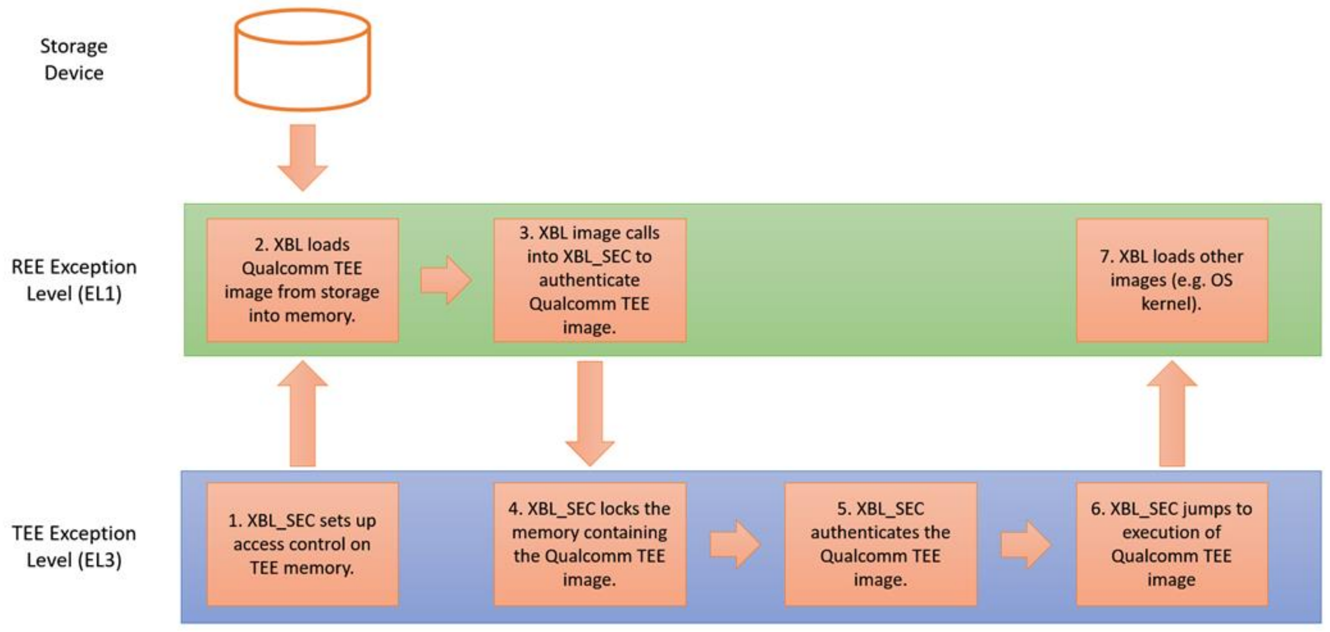
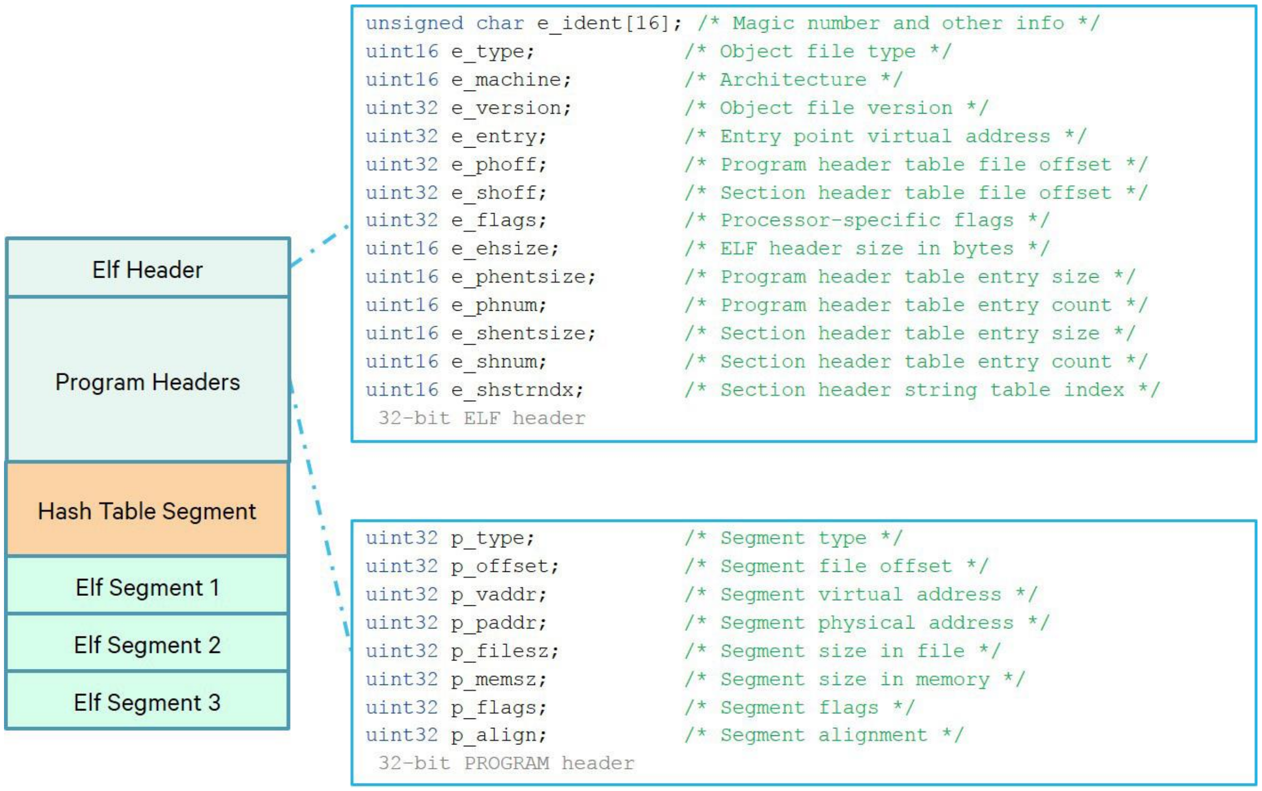

Storage Device
REE Exception Level (EL1)
TEE Exception Level (EL3)
2. XBL loads Qualcomm TEE image from storage into memory.
3. XBL image calls into XBL_SEC to authenticate Qualcomm TEE image.
7. XBL loads other images (e.g. OS kernel).
1. XBL_SEC sets up access control on TEE memory.
4. XBL_SEC locks the memory containing the Qualcomm TEE image.
5. XBL_SEC authenticates the Qualcomm TEE image.
6. XBL_SEC jumps to execution of Qualcomm TEE image

Figure 2: Storage device, REE Exception Level, and TEE Exception Level.

3. Signed Image Format

As previously mentioned, Qualcomm Technologies firmware images use the standard ELF format and thus each image contains a standard ELF Header and Program Header. Both 32-bit and 64-bit ELF classes are supported. An example of a 32-bit ELF file is shown in Figure 3:



```
unsigned char e_ident[16]; /* Magic number and other info */
uint16 e_type;             /* Object file type */
uint16 e_machine;          /* Architecture */
uint32 e_version;          /* Object file version */
uint32 e_entry;            /* Entry point virtual address */
uint32 e_phoff;            /* Program header table file offset */
uint32 e_shoff;            /* Section header table file offset */
uint32 e_flags;            /* Processor-specific flags */
uint16 e_ehsize;           /* ELF header size in bytes */
uint16 e_phentsize;        /* Program header table entry size */
uint16 e_phnum;            /* Program header table entry count */
uint16 e_shentsize;        /* Section header table entry size */
uint16 e_shnum;            /* Section header table entry count */
uint16 e_shstrndx;         /* Section header string table index */
 32-bit ELF header
```

```
uint32 p_type;             /* Segment type */
uint32 p_offset;           /* Segment file offset */
uint32 p_vaddr;            /* Segment virtual address */
uint32 p_paddr;            /* Segment physical address */
uint32 p_filesz;           /* Segment size in file */
uint32 p_memsz;            /* Segment size in memory */
uint32 p_flags;            /* Segment flags */
uint32 p_align;            /* Segment alignment */
 32-bit PROGRAM header
```

Figure 3: Example of a 32-bit ELF file.

The ELF Header is primarily used to locate the Program Header in the ELF image file. The Program Header contains the location of all the segments in the ELF image file. In particular, it is used to locate the hash segment in the ELF file, which contains the authentication information which allows the ELF image to be verified as an authorized image.

The hash segment contains the following information in the following order:

1. The hash segment header (metadata) is a 48-byte field which contains information about the sizes of the other parts of the hash segment. This allows the fields to be identified within the hash segment.

2. The QTI and device manufacturer metadata fields are 128-byte fields which contain information about the image, such as the type of image and the hardware on which the image is designed to run. The QTI metadata is only present if the image is double-signed by QTI and the device manufacturer.

3. The hash table field contains hashes of every segment in the ELF file. The first entry is always a hash of the ELF Header and the Program Header. This is designed to ensure that the vital information in the header segments is also validated (including the image entry point address contained in the ELF header).

4. The QTI signature and certificate chain. The digital signature is computed over the hash segment header, image metadata, and hash table. It can be verified using the public key in the leaf certificate. The certificate chain is validated back to a root certificate, which is validated against a QTI-specific value held in the hardware. The QTI signature and certificate chain should only be present if the image is double-signed by QTI and the device manufacturer.

5. The device manufacturer signature and certificate chain. As with the QTI signature and certificate chain, the digital signature is computed over the hash segment header, image metadata, and hash table. It can be verified using a public key in the leaf of the certificate chain, which can be verified back to a root certificate, which is verified against an OEM-specific value in the hardware.

In other words, the format of the hash segment is as shown in Figure 4:

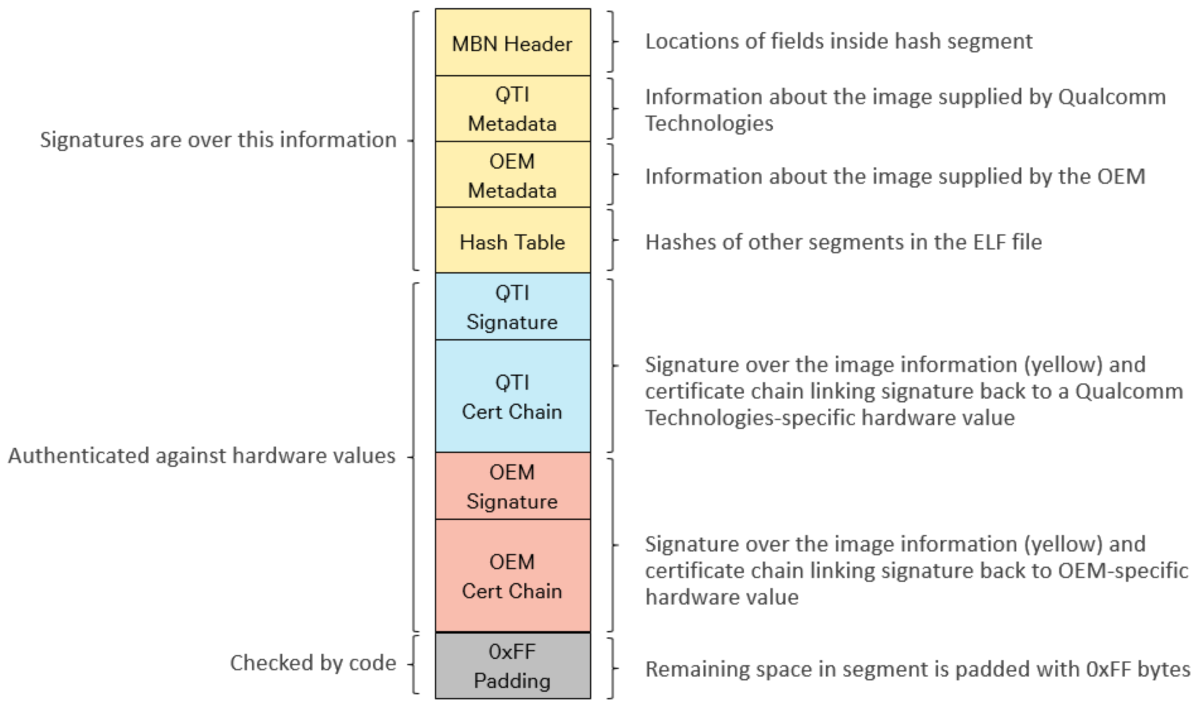

Signatures are over this information
Authenticated against hardware values
Checked by code
MBN Header
QTI Metadata
OEM Metadata
Hash Table
QTI Signature
QTI Cert Chain
OEM Signature
OEM Cert Chain
0xFF Padding
Locations of fields inside hash segment
Information about the image supplied by Qualcomm Technologies
Information about the image supplied by the OEM
Hashes of other segments in the ELF file
Signature over the image information (yellow) and certificate chain linking signature back to a Qualcomm Technologies-specific hardware value
Signature over the image information (yellow) and certificate chain linking signature back to OEM-specific hardware value
Remaining space in segment is padded with 0xFF bytes

Figure 4: The hash segment format.

It is important to note that the QTI and OEM certificate chains/signatures are over the same hash table. Since the time required to verify an image is dominated by checking the values in the hash table, the addition of the QTI certificate chain and signature to an image does not significantly increase the time required to verify the image.

If there are any gaps in the segment, they are filled with 0xFF bytes which are checked by the code. An incorrect padding value will invalidate the image. These padding bytes might occur if:

- The Program Header declares the hash segment to be of a size that is larger than all the data contained in the hash segment.
- The hash segment's MBN Header field declares that the QTI or OEM certificate chain fields are larger than all the data contained in that field.

4. Image Metadata

The image metadata describes the intent of the image. Most importantly, it describes the intent of the image (so that the system doesn't load a WLAN image when a Qualcomm TEE image is meant to be loaded) and the hardware on which the image is meant to be executed (so that the system doesn't load an Qualcomm® Snapdragon™ 835 Modem image on a Snapdragon 845 device).

In previous versions of the QTI secure boot architecture, this information had been encoded into the Organizational Unit (OU) fields of the leaf certificate in the certificate chain. The new version of the secure boot format moves this information into a standalone metadata field. This is designed to improve security by reducing the possibility of parsing or certificate creation errors which may lead to an image not having the properties intended by the device manufacturer.

The metadata field allows the signer to specify information about the image, including the following:

- **The software identity (SW_ID) of the image**. Each different image type has a different software identity. This is designed to ensure that the correct image is loaded at the correct point in the boot process.

- **The hardware identity (HW_ID) of the device**. Each QTI chipset has a different hardware identity. This is designed to ensure that only images designed to execute on that hardware can be executed on that hardware.

- **The device manufacturer identity (OEM_ID)**. This is designed to ensure that software signed by one device manufacturer cannot run on another manufacturer's device even if they share a common root certificate.

- **The debug capability of the device**. Certain images allow the signer to enable debug capabilities on a device; however, a signer can only use these capabilities on specific devices (which must be identified when the image is signed).

- **Whether the image is bound to an individual device or can be used on all devices**. For development and debug purposes, an image can be bound to an individual device by specifying the serial number of the device and setting a flag in the image metadata field. Each device has a different serial number which is burned into the chip during the manufacture process.

- **The anti-rollback version number of the image**. If enabled, the anti-rollback system is designed to prevent on image that is known to have bugs from running on a device. The architecture is designed to prevent a device from accepting an image if it has ever successfully loaded an image with a larger anti-rollback version number.

If the image is signed by QTI and the device manufacturer, then both QTI and the device manufacturer will provide an image metadata field. All conditions in both metadata fields must be valid for the image to load.

5. Certificate Chain and Digital Signature

Devices only support a limited number of signature algorithms. All devices supporting the new secure boot format use the RSA-PSS algorithm for digital signatures. A limited number of devices also support ECDSA signatures using the NIST P384 curve.

Older versions of the QTI secure boot architecture support the RSA PKCS#1 v1.5 signature format and a proprietary variant of the RSA PKCS#1 v1.5 signature format. These signature formats are not supported in the new secure boot architecture.

The certificate chains are used to validate the public key that is used to verify the signature on the hash table and the image metadata fields. The certificate chain may consist of two or three certificates. In prior versions of the secure boot architecture, we recommended the use of a three-certificate chain as the final (leaf) certificate in the chain contained the image metadata and so had to be re-generated during the image signing process. Since image metadata has now been shifted to its own field within the hash segment, we currently do not see an advantage in using a three-certificate chain.

The verification flow of the certificate chain is shown in Figure 5:

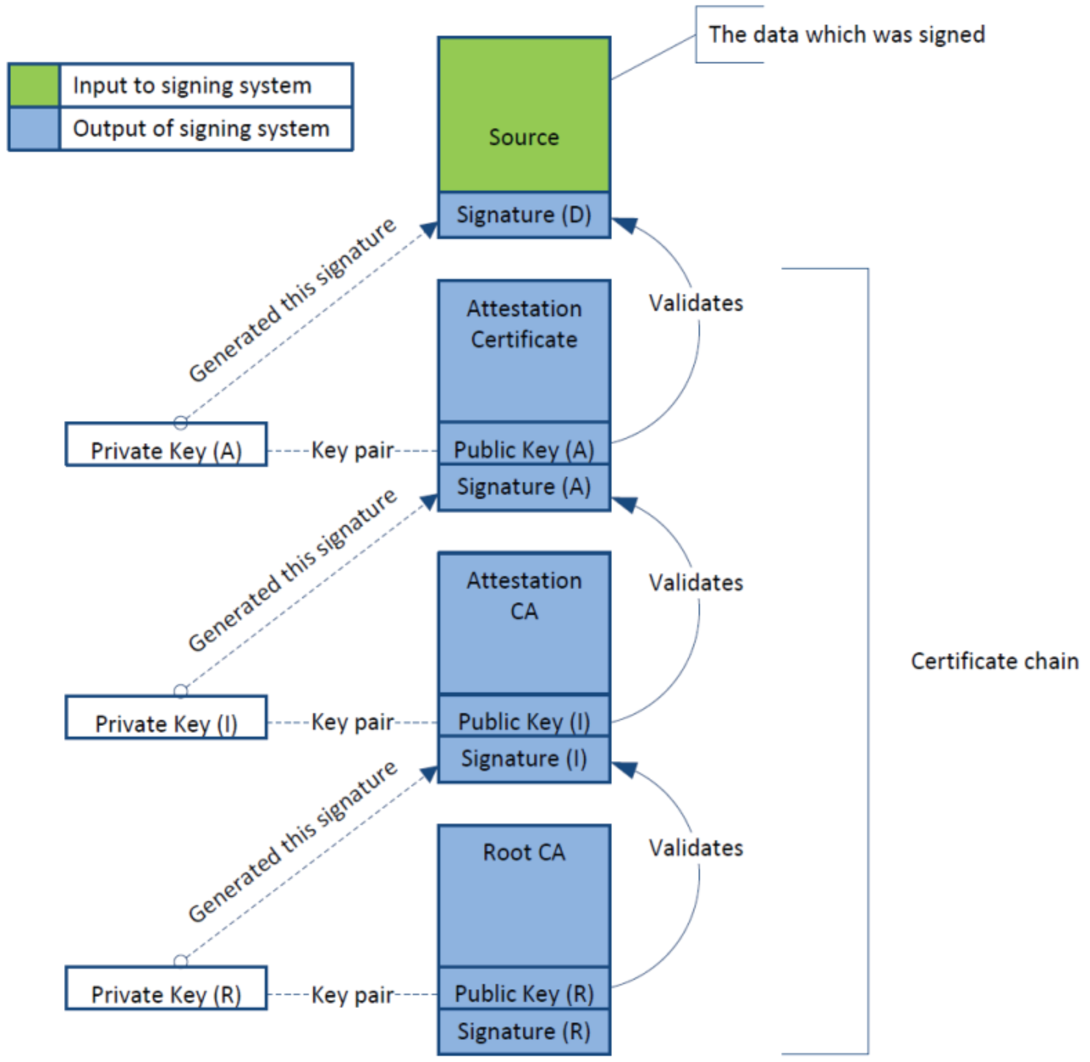


Figure 5: The verification flow of the certificate chain.

The Root CA certificate is verified against a hash value, which is either stored in QTI's QFPROM eFuses or in the hardware ROM code.

If the image is signed by QTI and the device manufacturer, then each certificate chain must be verified independently. Both certificate chain verifications must be verified successfully for the image to be authenticated and executed. The QTI Root CA certificate is verified against a QTI hash value whereas the device manufacturer Root certificate is verified against a device manufacturer hash value. This is engineered to prevent QTI from signing on behalf of a device manufacturer and vice versa. The system is designed to ensure that, where double-signed images are required, both the device manufacturer and QTI attest to the image.

6. Image Loading

All image loading follows the same general process. In this section, we will call the software that is loading the image the "loader." The flow is as follows:

1. The loader allocates a safe area of memory in which to load the ELF Header. This memory is mapped to prevent it from being tampered with by other execution environments in the device. The loader copies the ELF Header from (untrusted) storage into this memory. If the ELF Header is too large to fit into this memory region, the image is rejected.

2. The loader allocates a safe area of memory in which to load the Program Header. This memory is mapped to prevent it from being tampered with by other execution environments in the device. The loader copies the Program Header from (untrusted) storage into memory. If the Program Header is too large to fit into this memory region, the image is rejected.

3. The loader allocates a safe area of memory in which to load the hash segment. This memory is mapped to prevent it from being tampered with by other execution environments. The loader copies the hash segment from (untrusted) storage into this memory. If the hash segment is too large to fit into this memory region, the image is rejected.

4. The loader validates the hash segment by validating the root certificate, certificate chain, image metadata, and hash table.

5. The loader validates the (already loaded) ELF Header and Program Header by hashing them and comparing the hash value with the first entry in the hash table. If the hashes do not match, the image is rejected.

6. The loader will then attempt to load each of the other ELF segments in the image. For each segment, the loader checks that the entire segment can be loaded into an area of memory that has been approved (whitelisted) by the loader as safe and appropriate for that image. If a segment cannot be loaded into a whitelisted area of memory, the image is rejected.

7. The loader verifies each of the loaded ELF segments by hashing them and comparing the hash value with the corresponding entry in the hash table. If any of the computed hash values differ from the value in the hash table, the image is rejected.

8. If appropriate, the loader passes execution to the image using the entry point defined in the (already loaded and validated) ELF Header.

This process is designed to ensure that the loader will never accidentally overwrite important data in memory (including the loader’s own code and data) with image data being loaded from untrusted storage.

7. Summary

Software images are loaded from untrusted storage to internal memory and parsed. The loading and parsing phase includes address and size validation against whitelisted memory ranges, and integer overflow checks when performing arithmetic/pointer calculations. First, the image's ELF Header and Program Headers are loaded and parsed, then the hash segment is loaded and parsed.

The hash segment may contain authentication information from QTI and the device manufacturer. Each signer provides image metadata, a digital signature over the metadata and hash table, and a certificate chain. The certificate chain may contain two or three certificates. The root certificate is validated against a value held in hardware and each certificate in the chain is used to verify the next certificate in that chain. The public key in the leaf certificate is used to verify the digital signature, which authenticates the image metadata and the hash table.

The image metadata is used to verify that the image is appropriate for loading at this time in the boot process and is designed for this hardware. It may constrain the image in other ways too, such as ensuring that an image may only be used on one specific device through serial-number binding.

The hash table is used to verify the other segments in the ELF field. If the hash of a segment does not match the corresponding value in the hash table, then the image is rejected.

QTI's new secure boot architecture is engineered to improve security by allowing for TEE isolation from the start of the boot chain, simplifying the metadata format, reducing the possibility of unintended errors and decreasing the use of proprietary cryptography which may be difficult to parse using standard tools.