Qualcomm
Engineering Human Progress

Secure Boot and Image Authentication

Technical Overview

Joona Kannisto
Qualcomm Finland RFFE Oy

Contents

1. Overview

Secure boot provides a foundation for the security architecture of the device. Secure boot is defined as a boot sequence in which each software image that is loaded and executed on a device is authorized by previously authorized components (see example in Figure 1).

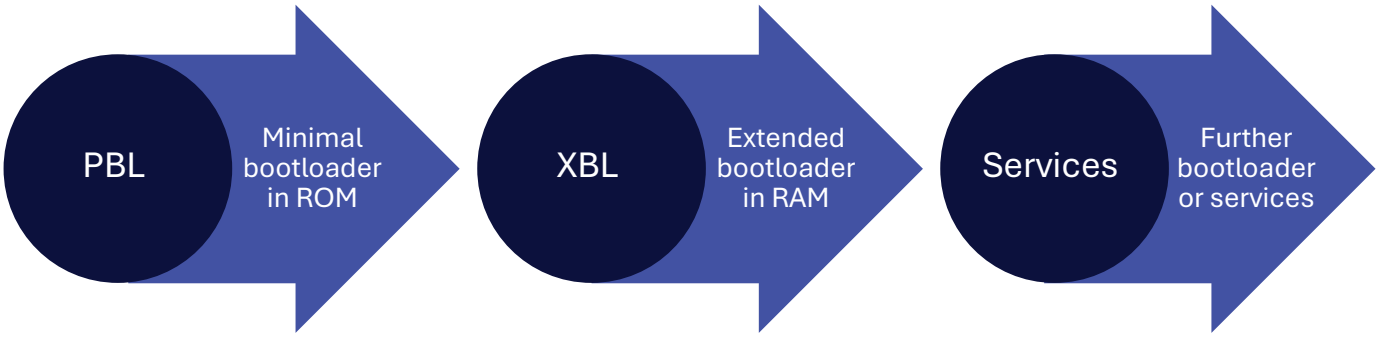

Figure 1: Simplified boot chain

The first component in this "chain of trust" is called the Primary Boot Loader (PBL). The Primary Boot Loader is stored in immutable read-only memory (ROM); it is literally part of the fabric of each chip. The user can have confidence that this component has not been altered because it cannot be physically altered. This initial software cryptographically verifies digital signatures on the images that it loads. Although immutability naturally provides integrity it also imposes practical limits on complexity and code size. Hence, the PBL contains only the minimum functionality to load, verify, and execute further images. Those images form the later boot stages, which verify the digital signatures on the next set of images that they load, and so on.

In recent versions of Qualcomm Technologies' secure boot, we have introduced a new component: the Trust Management Engine (TME). TME acts as the Root of Trust (RoT) for the whole System on Chip (SoC). Firstly, it manages the fuse-based configuration and provides hardware backed key management services. Secondly, it handles the initial assignment of SoC hardware resources, which is necessary to isolate all the pre-configured security domains on the SoC.

During the PBL boot stage execution, TME provides the image authentication services necessary to load the images for the next boot stage. The PBL loads and authenticates two images: one for the TME final runtime and one for the eXtensible Boot Loader System Controller (XBL-SC).

Together these two images constitute the second stage bootloader functionality. The XBL-SC image is tasked to initialize peripheral hardware and to load further images from storage. It calls TME to authenticate the images. TME will, upon successful authentication, allocate proper resources for each image.

The second stage is loaded software, hence it can be modified after device manufacturing. This allows it to have more features than the PBL and it can be configured to support the exact peripherals attached to the SoC by the manufacturer. The second stage will, for instance, set up external DRAM memory, where most of the further images are loaded. Eventually these loaded images will replace the XBL-SC component. TME image, once loaded, stays in SoC internal memory.

The images for TME, XBL-SC, many peripherals, and parts of the application processor boot chain, 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 images that are executed. Furthermore, we aim to safeguard critical assets, such as user-data encryption keys and payment-application authentication keys controlled by TME, Qualcomm® Trusted Execution Environment (TEE), and Qualcomm® Hypervisor Execution Environment (HEE), even in the event of a breach of a less-trusted environment.

All images make use of the standard ELF format. 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 along with a collection of metadata about the image. The 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 where the image is allowed to execute.

The following sections of this document discuss the security architecture, the way in which the SoC 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. Security Architecture

The Qualcomm Technologies' secure boot architecture is designed to maintain a separation between the various security domains, including the TME that is the root of trust of the device as well as others, such as the Qualcomm TEE and the modem, which provide a wide range of services required by users.

As depicted in the Figure 2, the first code that executes is the ROM-based primary boot loader PBL. This code is trusted due to its immutability and runs on the TME and Application Processor subsystems (APSS). TME isolates itself from the rest of the PBL but the other SoC resources are in a common PBL trust domain.

The first image that will be executed after the ROM-based primary boot loader is the TME image (Figure 2). This image will at its startup configure the initial access control rules that are necessary to protect the resources used by various security domains on the chip.

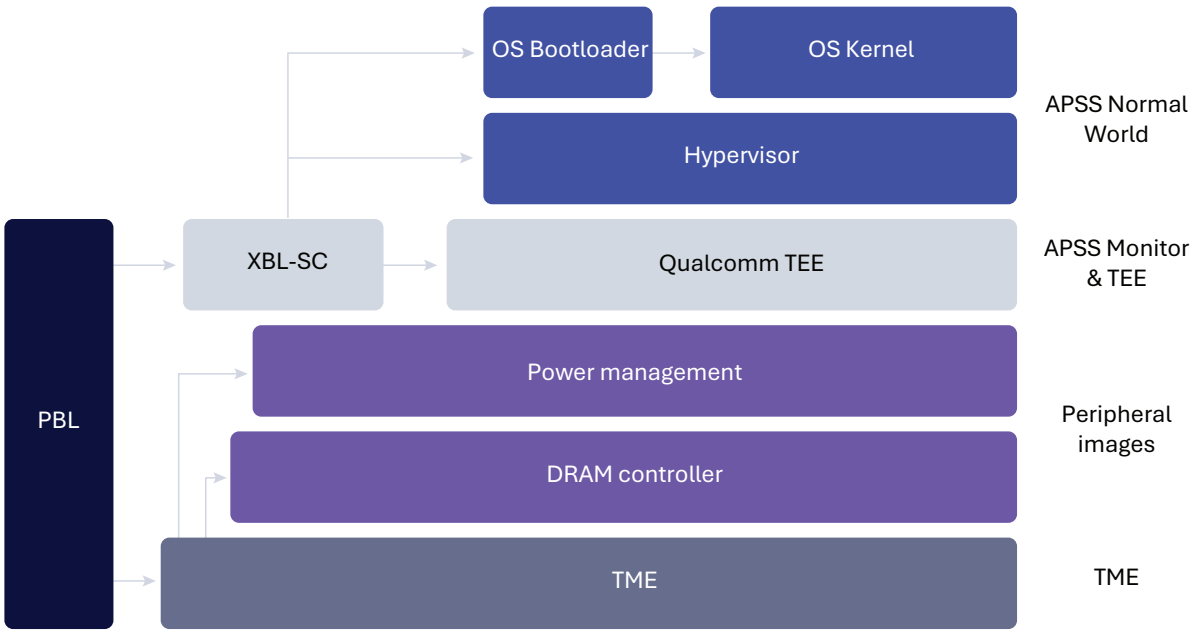

Figure 2: Boot Chain of the system on chip with TME.

The second image is the XBL-SC boot loader image. It is responsible for driving the boot process, which includes loading peripheral and subsequent images from storage into internal memory and DRAM. XBL-SC runs at application processor highest exception level (monitor mode) and is double-signed.

XBL-SC replaces the XBL component as well as the XBL_SEC from the previous Qualcomm® Secure Boot format. TME and XBL-SC are isolated from each other with physical access control mechanisms and component level isolation. XBL-SC calls into TME to initialize the resources on the SoC via controlled interfaces. For instance, an authentication request of the loaded peripheral image, like DRAM(Figure 2) sets up the resources associated with that image in a state where they can be used by all entities on the SoC in a trusted manner.

The isolation between XBL-SC and the TME during the boot process is designed to allow more flexibility in the SoC configuration, and to minimize the amount of code in the SoC RoT. In the previous version of Qualcomm Secure Boot, we achieved a similar trusted code minimization with application processor exception-level-based isolation. Fully separate Qualcomm Technologies controlled RoT provides a clean isolation boundary between itself and the main application processor and avoids dependencies on architecture-specific technologies.

In Figure 3 we see a depiction of the SoC trust model. At the top we have the PBL code and the hardware it is running on. The behavior of the hardware is controlled by One Time Programmable (OTP) eFuses.

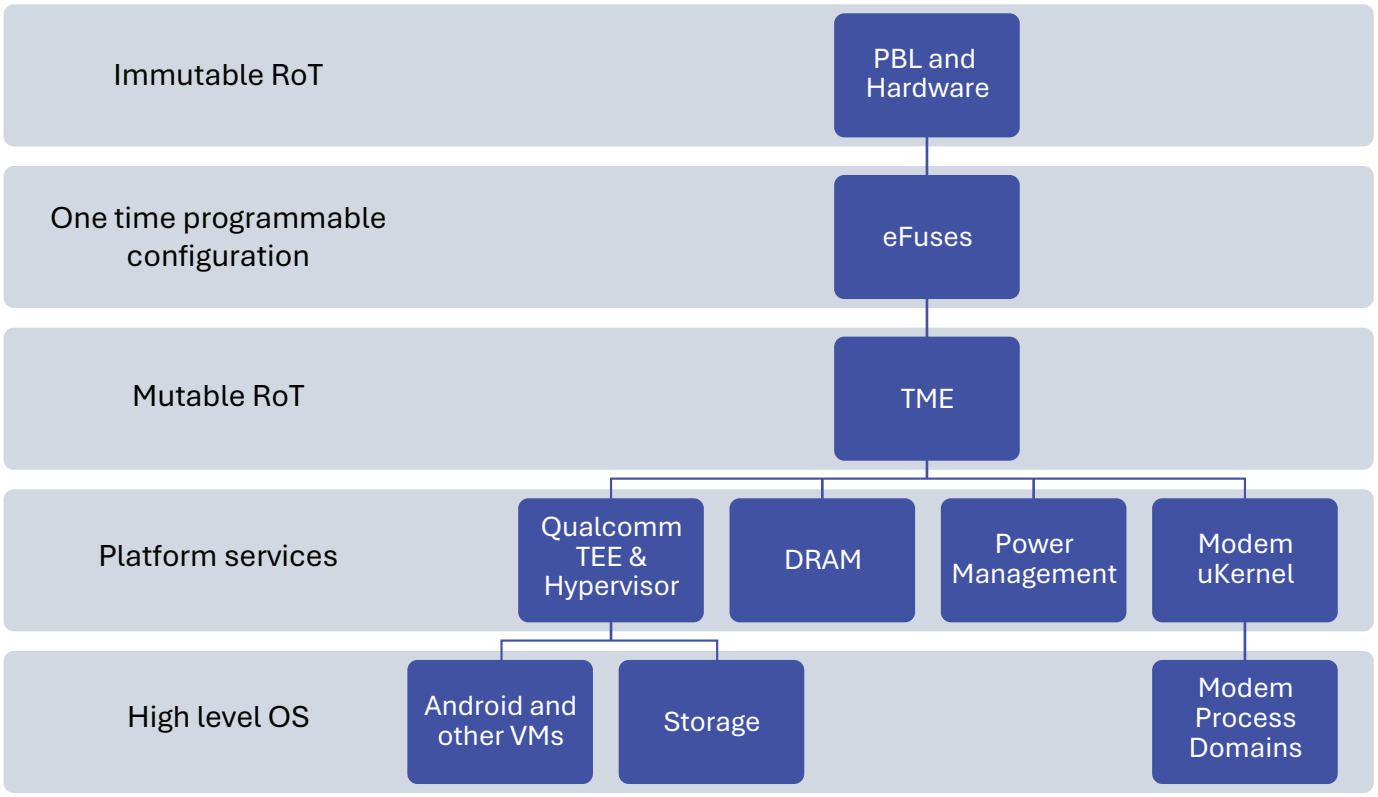

Figure 3: Trust Hierarchy of the system on chip with TME.

The OTP eFuse configuration dictates the device custodian, who is commonly referred as the Original Equipment Manufacturer (OEM). The device OEM takes control of the device mutable software by loading a configuration file with their custom fuse values. The configuration file sets the fuses that enable secure boot and stores a cryptographic hash of the manufacturer public key certificate in the fuses reserved for it.

Moreover, the immutable hardware has the ability to include the OEM fuse configuration in platform key derivations along with hardware secrets provisioned into the chip by Qualcomm Technologies and the device OEM. Secure boot restricts all keys with these bindings to software signed by that OEM and to them alone. One of the guiding principles of Qualcomm Secure boot is that no software runs without an authorization from the OEM.

In Figure 3 the loaded mutable TME image is on top of all the other loaded images. This implies run-time control over them. However, as the boot progresses past initial resource allocation this interpretation becomes less accurate due to TME distributing the resource ownership to other subsystems and removing its own access. It should be noted that the TME image is also subject to the same hardware-based access control mechanisms as all other execution environments. Indeed, the access control prevents TME accesses to e.g. Qualcomm TEE protected resources, if a permission has not been given. This fault isolation is intended to both block software issues from causing silent corruption to other subsystems as well as to prevent adversarial use of TME as a proxy to access other subsystems' resources. This manifestation of the principle of least privilege is enabled by the access control design of our SoC, which is explained in more detail in our Access Control whitepaper.

Many chip functions such as power management and DRAM management have dedicated microcontrollers with associated firmware. The XBL-SC loads these firmware images into chip memory and calls for TME to authenticate the image signatures and data. The management systems that as part of their duty hold special privileges over shared resources are initialized under TME control, and they will not be allowed to execute without a valid signed image. After successful initialization, these systems become their own access domains and work independently without TME influence.

Besides the OEM there are other parties that have some type of stake in the final product. For instance, independent software vendors, content providers, and certification bodies want to ensure that their requirements are met with the product. Those parties may find it easier to get their assurance from a commonly used platform like the Qualcomm TEE, rather than inspect an individual OEM's product. Hence, images that control assets linked to multiple stakeholders, or control shared resources on the SoC, are signed by both Qualcomm Technologies and the device manufacturer in a process known as double-signing. This is designed to ensure that any security-critical image can only be executed if it has been approved by Qualcomm Technologies and the device manufacturer.

The components that must be double-signed depend on the device image authentication profile, which is pre-configured by Qualcomm Technologies on a product-line basis. The exact configuration is influenced by the properties of the device hardware as well as the needs of the planned use-cases.

The XBL-SC image that precedes Qualcomm TEE enforces the authentication for all images that will run on the application processor, including the Qualcomm TEE, the Qualcomm HEE, the OS boot loader (e.g., UEFI). XBL-SC will call into TME to authenticate images that are in the Qualcomm® Signed Image format.

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 4:

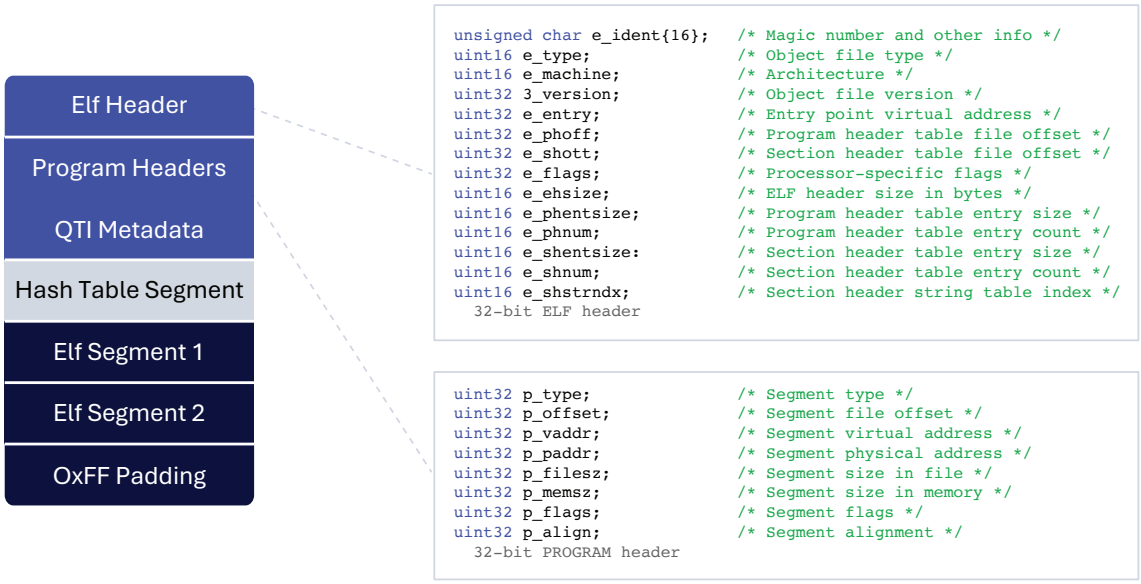

Figure 4: Example of a 32-bit ELF file

The ELF Header is the part that is first accessed in an image. Boot code reads and validates the architecture, the number and size of program header entries, as well as the ELF entry point. There is also an offset (e_phoff) used to locate the Program Header in the image file, which will be used next.

The Program Header contains the location of all the segments in the ELF image file. The program header includes both information for loading and authentication of the image. Loading operation requires the storage file offsets and file sizes. The destination address and memory size are used for loading directly into memory, and later for the authentication as well.

The loader includes special treatment for the hash segment, which contains the authentication information that allows the ELF image, including the headers themselves, to be verified as an authorized image.

The hash segment of Qualcomm Technologies' latest Signed Image Format (MBN version 7) contains the following information in the following order:

1. The hash segment header 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 common metadata contains metadata that is independent of the image signer. For example, the software-id that determines the target execution environment should not change across the signing entities.
3. There are individual metadata fields for both Qualcomm Technologies and the device manufacturer. These fields specify additional restrictions on the hardware which the image is designed to run on.
4. The hash table field contains cryptographic hashes of every segment in the ELF file. The first entry is always a hash of the ELF Header and the Program Header.

5. The Qualcomm Technologies' signature and certificate chain. The digital signature is computed over the hash segment header, all signer owned metadata, and the 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 Qualcomm Technologies specific value held in the hardware.
6. The device manufacturer (OEM) signature and certificate chain. As with the Qualcomm Technologies signature and certificate chain, the digital signature is computed over the hash segment header, signer owned metadata, and hash table. It can be verified using public key from 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 5:

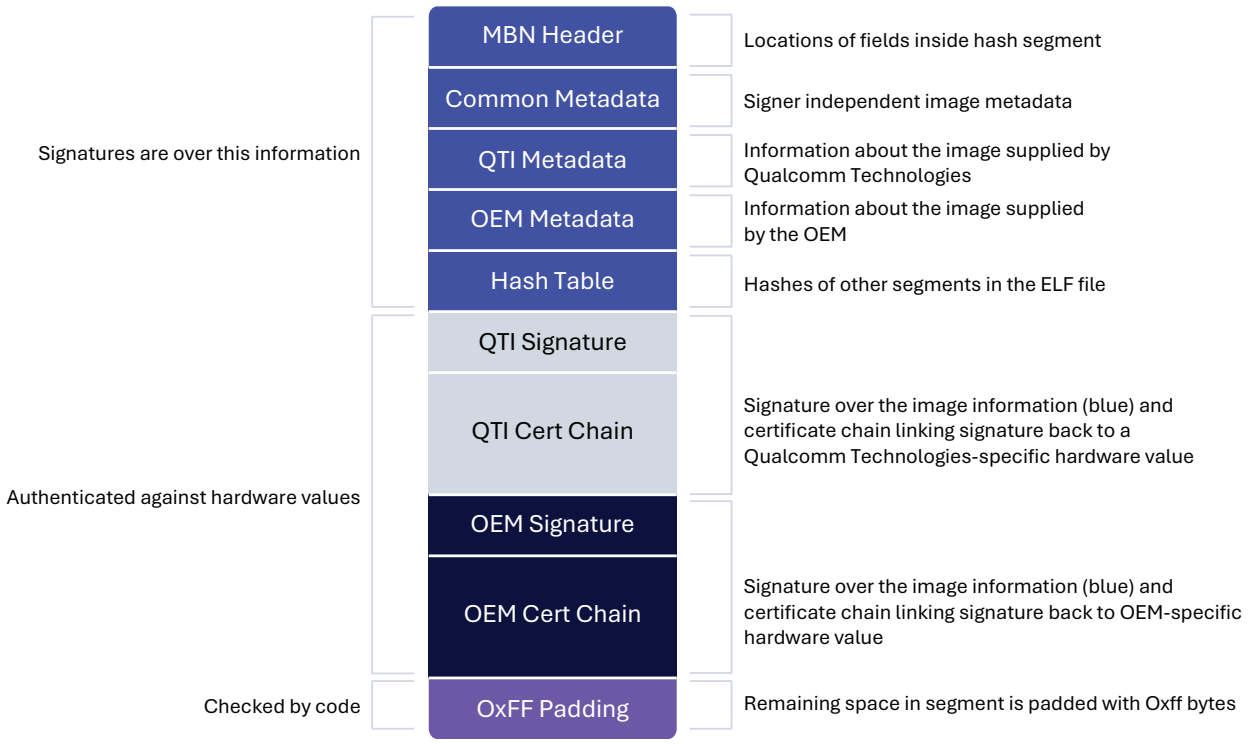

Figure 5: The hash segment format.

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

4. Image Metadata

The image metadata contains information and restrictions on 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 a Snapdragon 835 image on a device with Snapdragon® 8 Gen 1 Platform).

In a previous version (MBNv5) of the Qualcomm Technologies' secure boot architecture, this information had been encoded into the Organizational Unit (OU) fields of the leaf certificate in the certificate chain. The latest two new versions of the Signed Image Format (MBNv6 and MBNv7) have moved this information into a standalone metadata field.

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 SoC hardware version (SoC_HW_VER).** Each Qualcomm Technologies chipset has a hardware identity based on chip revision. This is designed to ensure that only images designed to execute on that hardware can be executed on that hardware. This is a list that includes up to 12 version identifiers to support multiple compatible hardware revisions with the same image.
- **Whether the image is bound to an OEM chosen product identifier.** The OEM may set a product identifier in the product identifier fuse region. This allows to differentiate some or all images between products signed under the same root certificate. For example, a mobile phone and a security camera might have been built on a common SoC by the same manufacturer but should not be allowed to run each other's images.
- **The debug lock.** Debug policies allow the OEM signer to enable debug capabilities on a device; however, an image signer can disable later debug activation via the early boot images.
- **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 the serial number bound flag in the image metadata field. Each device has a different serial number which is burned into the chip during the manufacture process.
- **Whether the device is in development lifecycle phase.** There is an OEM controlled lifecycle fuse to transition device into a development device class, which allows to load images that the OEM intends not to run on commercial devices. This is intended for devices in test farms, as well as for factory use cases, such as calibration or acceptance testing. A development device supports a one-time transition back into the production device class.
- **The anti-rollback version number of the image.** If enabled, the anti-rollback system is designed to prevent an older version of an image, which is, for instance, known to have security issues from running on a device. The architecture prevents the device from accepting an image if it has successfully booted with an image that had a higher anti-rollback version number.
- **The device manufacturer identity (OEM-id).** If multiple manufacturers use the same root certificate, the OEM-id binding is required to ensure that software signed for one device manufacturer cannot run on another manufacturer's device.
- **The device manufacturer root certificate hash.** This restriction together with the OEM-id in Qualcomm Technologies' metadata limit the image to only single OEM's devices.

If the image is signed by Qualcomm Technologies and the device manufacturer, then both Qualcomm Technologies 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

Qualcomm Secure Boot uses X.509 certificate chains for public key encoding and management. The certificate chain validates a public key that is used to verify the signature on the hash table and the image metadata fields. The first link in the chain, the Root CA certificate is hashed and verified against a value stored either in One Time Programmable fuses or in the hardware ROM code.

The verification flow of the certificate chain is shown in Figure 6:

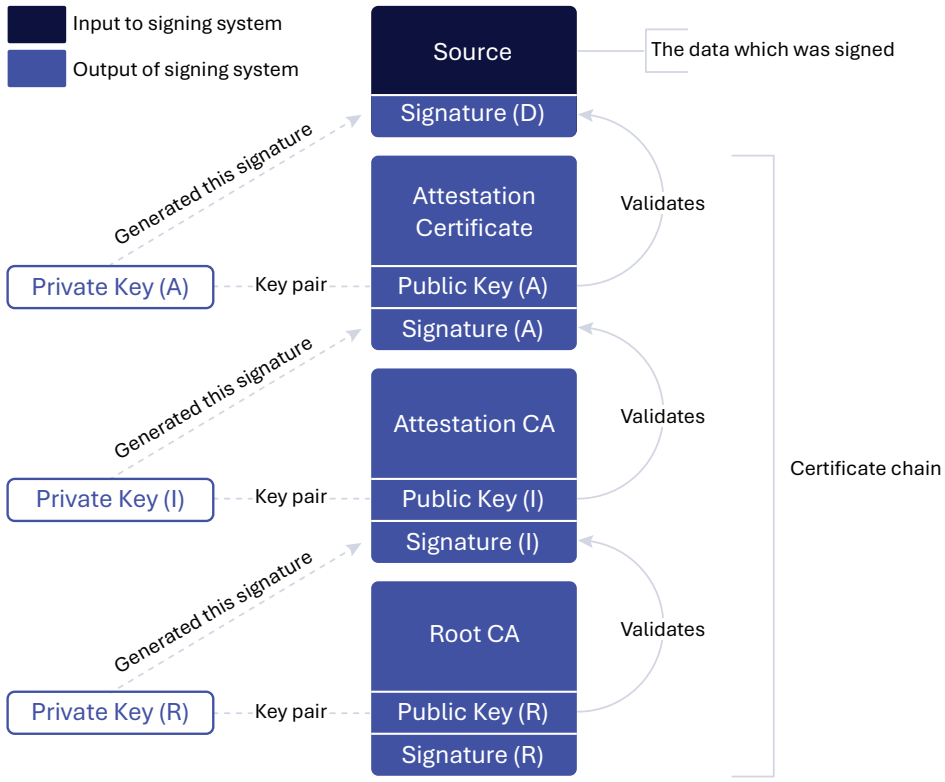

Figure 6: The verification flow of the certificate chain.

The certificate chain may consist of two or three certificates. The two-certificate minimum is a design choice to encourage the use of a completely offline root CA. In such a hierarchical system the keys in the frequently used signing infrastructure do not have to be safeguarded against data loss. This makes it possible, for instance, to use non-exportable keys created inside a Hardware Security Module (HSM).

In prior versions of the secure boot architecture, the leaf certificate in the chain contained the image metadata. This meant that the leaf certificate was typically re-generated during the image signing process. Therefore, we previously recommended the use of a three-certificate chain to have two long-term certificates in the chain. Since the image metadata has now been moved into the hash segment, all certificates are long-term. Hence, we do not see an advantage in using a three-certificate chain but support it for backwards compatibility.

Furthermore, there are no image metadata bindings in the certificate chain, which means that all leaf certificates can sign any image. Any desired restrictions on authorized signers need to be built into the signing infrastructure and operations.

If the image is signed by Qualcomm Technologies and the device manufacturer, then both certificate chains are verified independently. Both must be verified successfully for the image to be authenticated and executed. While verifying the Qualcomm Technologies chain, the Root CA certificate is hashed and verified against a Qualcomm Technologies' stored value in ROM. Whereas during the device manufacturer chain validation the Root certificate is hashed and verified against a device manufacturer value in fuses. The system is designed to ensure that, where double-signed images are required, both the device manufacturer and Qualcomm Technologies attest to the image.

Devices only support a limited number of signature algorithms. All devices supporting the new secure boot format use ECDSA signatures using the NIST P384 curve. Older versions of the Qualcomm Technologies secure boot architecture support RSA PSS and RSA PKCS#1 v1.5 signature formats.

6. Image Loading

Software images in early boot are typically loaded straight into the destination memory. All early boot image loading follows the same general process. In this section, we will call the software that is loading the image the "loader." Similarly, the software that verifies the image will be called "verifier". We use the term verify to mean an authenticity verification and validation to mean checks against allowed parameter ranges. The flow is as follows:

1. The loader allocates a safe area of memory in which to load the ELF Header. 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. The loader validates the ELF header parameters which it will be using during the following steps.
2. The Program Header size and offset in the storage is determined by the ELF header. The loader allocates a safe area of memory in which to load the Program Header. The loader copies the Program Header from (untrusted) storage into memory. If the Program Header is too large to fit into available memory region, the image is rejected. The loader validates that the program header contains a hash segment. If there is no hash segment the image is rejected.
3. The loader allocates a safe area of memory in which to load the hash segment. The size and storage offset of the hash segment are located from the program header. The loader copies the hash segment from (untrusted) storage into the reserved memory. If the hash segment is too large to fit into this memory region, the image is rejected.
4. The loader calls to the verifier to validate and verify the headers and the hash segment. The verifier maps and access controls the memory from other execution environments. The verifier validates the header parameters and hash segment contents, such as sizes for certificate chain, hash table, and image metadata. Then the image metadata is validated. If constraints set in the metadata, such as, chip serial number, anti-rollback version, chip revision are not satisfied the verifier will reject the image.
5. The verifier verifies the root certificate by hashing it and comparing to value in hardware. The verifier parses the root certificate and uses the parsed public key to verify the next certificate in the chain. This is repeated until the final leaf certificate has been verified. The verifier verifies a signature over the hash table and image metadata using the public key from the leaf certificate.

6. The verifier verifies 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.
7. 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 (allowlisted) by the loader as safe and appropriate for that image.
8. The verifier verifies each of the loaded ELF segments by hashing them and comparing the hash value with the corresponding entry in the hash table. Again, the loader calls the verifier to do this step. If any of the computed hash values differ from the value in the hash table, the image is rejected.
9. If appropriate, the verifier passes execution to the image using the entry point defined in the previously loaded and verified 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 where they are parsed. The loading and parsing phase includes address and size validation against allowlisted 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 Qualcomm Technologies 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 file. If the hash of a segment does not match the corresponding value in the hash table, then the image is rejected.

Qualcomm Technologies' new secure boot architecture introduces the Trust Management Engine (TME), which manages isolation between the various execution environments on the chip. TME also provides low-level platform services, such as image authentication, key management, and OTP fuse provisioning. This architecture improves security by reducing the complexity of the root of trust and by de-privileging the SoC management tasks.

Qualcomm

Follow us on:

For more information, visit us at: qualcomm.com

Learn more and get the latest updates:

Sign up for our wireless technology newsletter ↗

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