Secure Boot and Image Authentication

Technical Overview

Ryan P Nakamoto
Staff Engineer, Product Security
Qualcomm Technologies, Inc.
October 2016

Disclaimer

Qualcomm Technologies Inc.

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

Contents

1 Overview

Secure boot is defined as a boot sequence in which each software image to be executed is authenticated by software that was previously verified. This sequence is designed to prevent unauthorized or modified code from being run. Our chain of trust is built according to this definition, starting with the first piece of immutable software to be run out of read-only-memory (ROM). This first ROM bootloader cryptographically verifies the signature of the next bootloader in the chain, then that bootloader cryptographically verifies the signature of the next software image or images, and so on.

On the applications processor the first piece of ROM-based software mentioned above, which we call the Primary BootLoader (PBL), typically loads and authenticates a Secondary BootLoader (SBL) or eXtensible BootLoader (XBL) as the next image to be run. This image then loads and authenticates a feature-rich applications bootloader such as Little Kernel (LK) or the Unified Extensible Firmware Interface (UEFI) that is specific to the Operating System (OS) that it will subsequently load. In modern Qualcomm Technologies products, these software images are all standard Executable and Linkable Format (ELF) images.

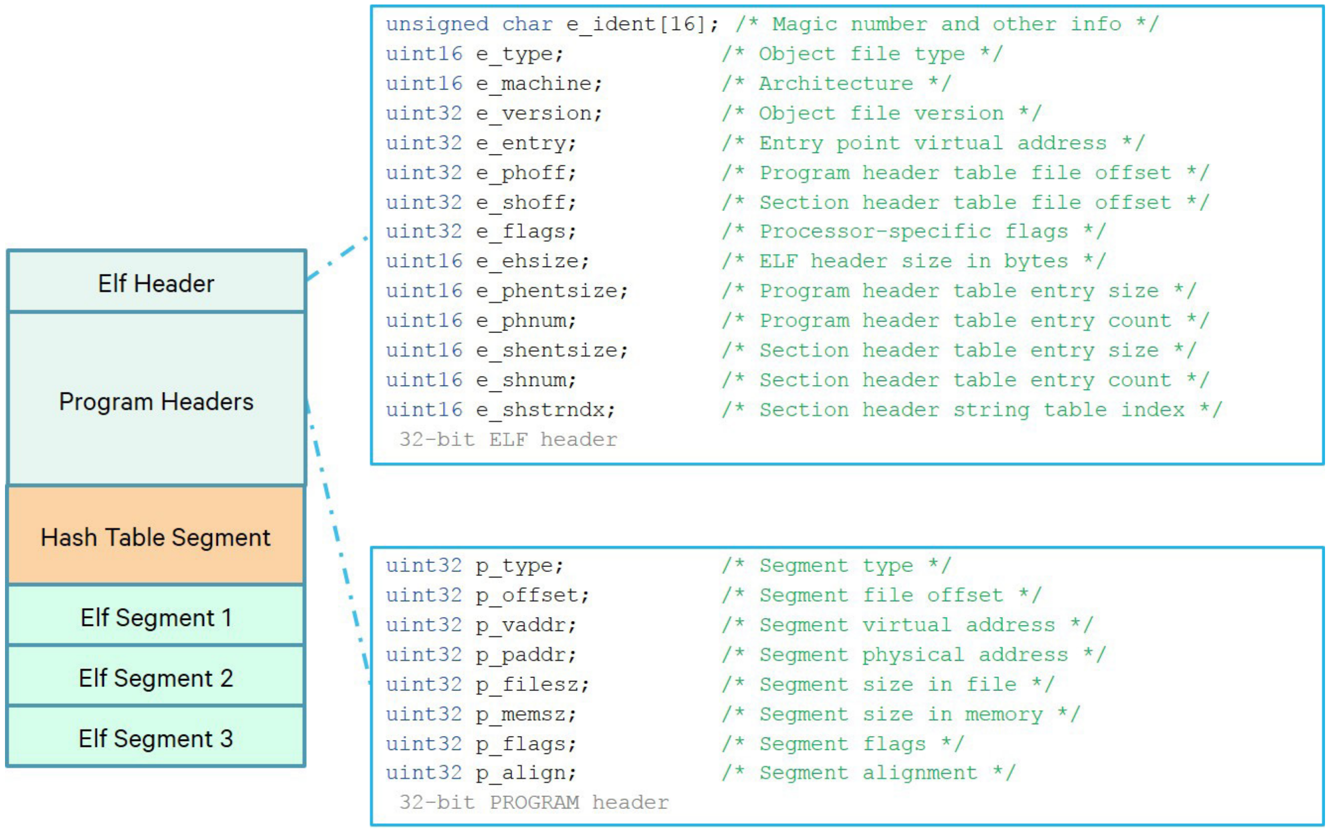
Like most digitally signed software, these image signatures include a certificate chain. The "Attestation certificate" refers to the lowest level certificate authorizing the signature of the software image. This certificate is signed by the "Attestation CA certificate" which is in turn signed by the "Root CA certificate". The Root CA certificate is validated by computing its hash and comparing to a value stored either in eFuse or in ROM (eFuse is a set of hardware embedded one-time programmable bits that once "blown" cannot be reverted). This stored Root CA hash value is provisioned to the device by the OEM, giving them full control of the device's cryptographic root of trust. The certificate chain hierarchy is shown in Figure 3 and later described in more detail. Two-certificate chains are also supported, wherein the Attestation certificate is signed directly by the Root CA certificate.

Unlike other signed software images, the signature for Qualcomm Technologies signed images is only computed over a single segment in the image and not the entire image. The segment containing the signature is called the hash segment. This hash segment is a collection of the hash values of the other ELF segments that are included in the image. In other words we sign the collection of ELF segment hashes, rather than signing the entire ELF image. This representation is designed to relax memory size requirements and increases flexibility during loading. The Attestation certificate used to verify the signature on this hash segment also includes additional fields that can bind restrictions to the signature (preventing "rolling back" to older versions of the software image, restricting the signature to a particular type of software image, a particular model of hardware, a particular OEM, etc.).

The following sections of this document discuss the format of our signed ELF images, the process of loading and authenticating those images, certificate contents, and signature algorithms in greater detail.

2 Signed Image Format

As previously mentioned, the software image is an ELF file and contains the standard ELF and program headers. Both 32-bit and 64-bit ELF classes are supported. See Figure 1 for an example 32-bit ELF image.



```
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 1: ELF and PROGRAM headers in a 32-bit ELF

The hash table segment, or simply hash segment, typically follows the ELF and Program headers in the image. It contains its own 40-byte header which specifies the size of the entire hash segment, the size of the table of hashes, the size of the attestation signature, and the size of the certificate chain (all in bytes). This hash table contains the SHA-256 digest of each segment in the ELF image, as well as a digest for the ELF and Program headers. It is important to authenticate these headers since they contain important information (addresses, execution entry point, etc.). The signature for the image is computed over this table of digests and appended to it, along with the certificate chain, to form the hash segment. The entry in the hash table corresponding to the hash segment itself is empty, as the entire hash segment is authenticated during signature verification. The signature is accompanied by a certificate chain, as seen in Figure 2.

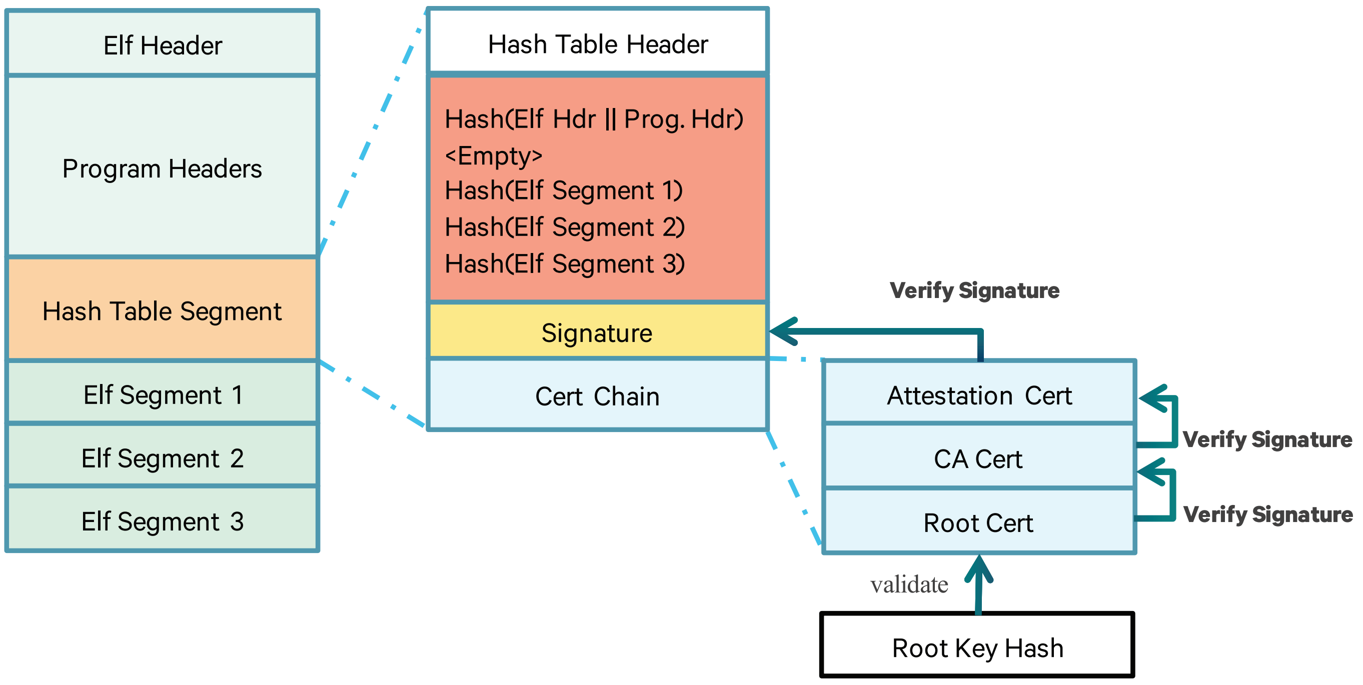


Figure 2: Hash Table Segment and Signature

3 Image Parsing and Loading Process

When an image is to be authenticated and executed, it must first be loaded from persistent storage into internal memory. Internal memory is generally protected and trusted but only available when the device is powered. External flash storage and external RAM are accessible by other entities and thus considered untrusted. In order to ensure the image has not been modified or tampered with, it must be authenticated within the trust boundary in internal memory before being executed.

First the ELF header is loaded from storage, parsed, and validated. The ELF header specifies a number of things, including program and section header sizes and offsets, as well as the virtual address of the image's entry-point. Next the Program headers are loaded, parsed, and validated. See the ELF specification for ARM architecture for more detail on ELF[1].

One and only one of the program headers must represent a hash segment, which is indicated by a segment type value in the program header's `p_flags` field. When that hash segment is found, it is loaded to internal memory for authentication. The hash segment is then authenticated by verifying its signature and accompanying certificate chain (further detailed in the following sections). The ELF headers and program headers are hashed and compared to their corresponding entry in the authenticated hash table segment. Each non-paged LOAD segment in the ELF file is then loaded from storage into internal memory and hashed. Each segment's calculated hash is compared to its corresponding entry in the authenticated hash table in the same order as it appears in the program header (segments therefore cannot be reordered). Since the hash segment is verified, comparing computed hashes of segments and the headers provides equivalent security to verifying a signature of the entire image while also allowing for loading and authenticating on a per-segment basis.

[1]ELF for ARM architecture – http://infocenter.arm.com/help/topic/com.arm.doc.ihi0044f/IHI0044F_aaelf.pdf

When headers or segments are loaded from external storage, their destination address and size in memory is verified to be fully contained in a whitelist to confirm that they are strictly confined to the memory space allotted for them. This is designed to prevent data from being written to unauthorized memory addresses, addresses where other data already resides, or from spilling over into another region. When calculating these offsets and sizes, the arithmetic is checked to ensure the result does not overflow (i.e. wrap around past zero). These checks are also performed on the other headers, the certificates, and other items where offsets and sizes are calculated.

4 Certificates Chain and Format

The certificate chain can consist of two or three certificates, all following the ITU-T X.509 v3 format. The certificate chain includes an Attestation certificate, (optionally) an Attestation CA certificate[2], and a Root CA certificate. Each is signed by the next certificate in the chain, as shown in Figure 3. The SHA-256 hash digest of the Root CA certificate must match the value stored in eFuse or stored in ROM in order to anchor the chain of trust.

A two-certificate chain is designed to reduce the on-device verification time and certificate chain size of each image by one certificate. In exchange, the server-side signer has less flexibility in authorizing entities to generate Attestation certificates (the job of the Attestation CA). This server-side signer also loses the ability to revoke that authorization since the Root CA directly signs Attestation certificates in this model and is anchored to eFuses on-device.

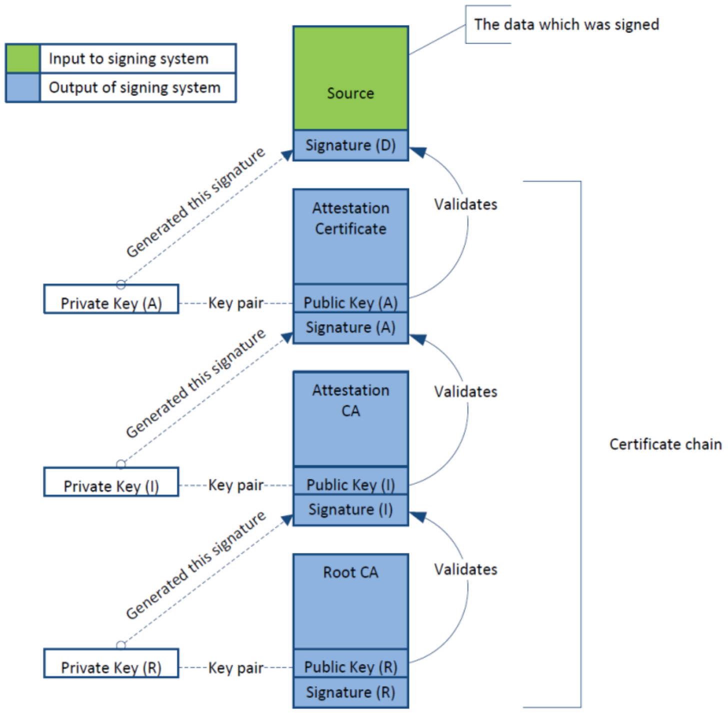


Figure 3: Certificate Chain Structure

The Root CA and Attestation CA certificates are typical X.509 certificates containing the same fields as those you might find securing a web browsing session. The Attestation certificate however, will include additional details in its Org Unit (OU) fields. All certificates are fully conformant to X.509. Profiles 1-3 below display certificate fields taken from the certificate chain of a signed Qualcomm® Snapdragon™ 820 processor eXtensible BootLoader (XBL) image.

[2]In a two-certificate chain, the Attestation certificate appears first and is signed directly by a self-signed Root certificate that follows.

```
Signature Algorithm: RSA PKCS#1 v1.5 Signature with SHA256
Valid notBefore: 03-Mar-16 03:24    Valid notAfter: 27-Feb-36 03:24
Serial Number: 0x01

Issuer:
    Organisation Name:      OEM
    Organisation Unit:      General OEM Root CA
    Common Name:            OEM ROOT CA
    Locality:               SANDIEGO
    State:                  CA
    Country:                US

Subject:
    Organisation Name:      OEM
    Organisation Unit:      General OEM Root CA
    Common Name:            OEM ROOT CA
    Locality:               SANDIEGO
    State:                  CA
    Country:                US

Public Key Modulus (hex):
    e4 b3 30 07 25 24 b0 92 b1 22 2d da 3e 87 c9 41 6a ac 7e 53 06 5e 5c f3 4c a6 f0 6a 72 35 52 28        ..0.%$..."-.>..Aj.~S.^\.L..jr5R(
    bb a6 ff 30 0b 36 15 71 07 23 a7 7b e0 ef 21 0c 57 39 13 69 96 a0 ad 5f 26 66 52 03 e2 43 dc 25        ...0.6.q.#.{..!.W9.i..._&fR..C.%
    08 d1 da 3f 46 e1 60 fd b3 21 72 76 d6 03 7d a7 54 86 a8 97 9d 24 5c 54 c4 17 92 3b b9 7e 99 42        ...?F.`..!rv..}.T....$\T...;.~.B
    13 4b f2 cf 06 a3 5d 3c a9 b2 95 e0 5c 3c 3b 93 91 47 4f 5a ad d9 69 3c 0f f4 fc 15 59 79 b7 58        .K....]<....\<;..GOZ..i<....Yy.X
    b5 99 8e ff cc aa c5 db 7a 0b 87 09 2a 06 1c 5f b1 85 38 b8 12 6b 96 11 61 02 82 06 5e b7 1e 44        ........z...*.._..8..k..a...^..D
    3d 8d a3 95 0a 99 eb f2 bf 1f c4 e1 18 21 cf e9 ed cd bb 49 af dc 3d 26 64 32 04 e4 97 1a aa bd        =............!.....I..=&d2......
    b9 af 06 ad 77 4f ad da cd 92 29 ae ec a8 94 2f 5b ec 83 f2 08 e0 61 9d e3 8b 62 17 0a d0 80 e1        ....wO....)..../[.....a...b.....
    83 1e 36 aa 3a 07 3f 36 39 73 ff e4 59 6b b0 ec 1a 51 fc 09 a6 da 5c 05 83 98 f3 f1 33 93 93 c5        ..6.:.?69s..Yk...Q....\.....3...

Public Key Exponent (int): 3 (aka F0 exponent)

Extensions:

    Extension Name            Critical    Value
    ------------------------------------------------------------------------------
    Basic Constraints         FALSE       Subject is a CA
    Key Usage                 FALSE       CRL Signature, Certificate Signature
    Subject Key Identifier    FALSE       9731A10D3E6DB33FA93FC348EFA9D0C4171534D9

Signature (Signature Verification Passed):
    A9 48 E1 91 5A E5 25 B9 C4 6A 11 BC 7F 21 BC 38 46 49 2B C0 A0 B9 71 D1 32 38 44 9F F0 C4 65 C6        .H..Z.%..j...!.8FI+...q.28D...e.
    05 5A 95 A6 3F 67 A2 4B DF 10 32 D0 27 BB 8E 15 5B 0B B0 D3 7A 27 C5 54 94 37 5A 86 28 E1 C7 A5        .Z..?g.K..2.'...[...z'.T.7Z.(...
    5F A1 06 C0 74 19 61 64 6A E4 57 DB 6F B8 48 DD F5 85 56 14 4D E7 6B 71 EB 27 38 38 13 AA CD B5        _...t.adj.W.o.H...V.M.kq.'88....
    7D F3 66 24 30 83 65 A1 C3 A5 91 61 D6 9D 50 40 82 BE D0 28 8B 40 2C A6 89 72 0D 7D 99 F3 88 08        }.f$0.e....a..P@...(.@,..r.}....
    3F C2 44 69 98 FB 87 79 14 62 84 60 EE 02 0F 11 60 89 CB D2 C7 60 F5 EB E0 FD FB 60 CA 25 76 E0        ?.Di...y.b.`....`....`.....`.%v.
    FB E2 65 B1 5E E1 23 9E B3 76 49 94 FC DE 79 13 11 ED 0A A1 72 77 D0 FE C4 02 C8 95 27 B4 DE 94        ..e.^.#..vI...y.....rw......'...
    32 E7 93 AC 4D 5C 41 F4 2F 86 67 9D 80 3E 2B 05 58 69 D4 86 29 14 12 56 DC 75 8D 94 DE AC BB 22        2...M\A./.g..>+.Xi..)..V.u....."
    5F 8A 6B 97 4F B9 28 CF CD 24 9C 70 16 86 32 B1 C1 59 F8 EA AF 68 5E 5E 57 20 B3 88 EE F8 E6 79        _.k.O.(..$.p..2..Y...h^^W .....y
```

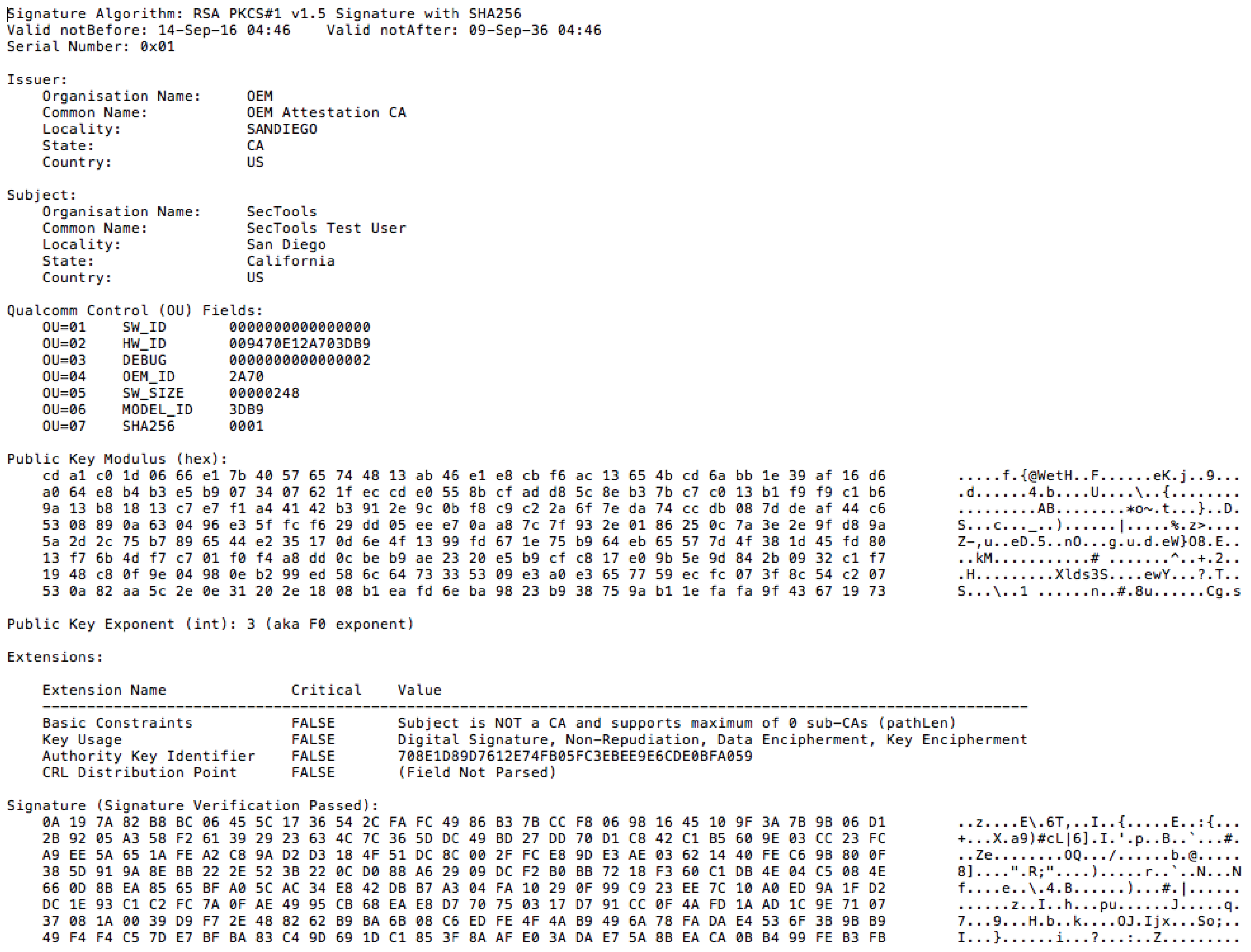
Profile 1: Root Certificate Profile

```
Signature Algorithm: RSA PKCS#1 v1.5 Signature with SHA256
Valid notBefore: 03-Mar-16 03:28    Valid notAfter: 27-Feb-36 03:28
Serial Number: 0x05

Issuer:
    Organisation Name:     OEM
    Organisation Unit:     General OEM Root CA
    Common Name:           OEM ROOT CA
    Locality:              SANDIEGO
    State:                 CA
    Country:               US

Subject:
    Organisation Name:     OEM
    Organisation Unit:     General OEM Attestation CA
    Common Name:           OEM Attestation CA
    Locality:              SANDIEGO
    State:                 CA
    Country:               US

Public Key Modulus (hex):
    9f 52 1d 34 bd cc 5e 25 25 7b c9 69 eb b5 65 6c 03 c3 54 aa 26 59 b2 a8 0a 5c 3f c7 82 f8 88 3d        .R.4..^%%{.i..el..T.&Y...\?....=
    f2 d2 c0 54 37 13 b6 e9 2c 4f a3 76 aa 97 d2 43 52 8d d2 a2 b3 58 c9 8b f7 e9 29 44 6b af 66 65        ...T7...,O.v...CR....X....)Dk.fe
    6a 45 83 2c 53 2f 6d 54 ed 7d c9 17 0e c7 74 67 9e 83 7b 58 1f 7d e1 40 b3 e6 58 1e c1 31 be fe        jE.,S/mT.}....tg..{X.}.@..X..1..
    bc a5 3f 39 0f f7 39 db d9 03 ee e6 1e 29 03 47 30 19 07 21 92 30 f7 f9 f8 07 8b 4e 39 a9 7f ab        ..?9..9......).G0..!.0.....N9...
    8e 71 28 af d3 29 36 64 1f 49 f8 65 21 18 af fb 0c e1 95 a1 cd 45 46 24 b4 c8 ed f1 44 23 c3 d0        .q(..)6d.I.e!........EF$....D#..
    60 ef 7e 77 3c 29 12 7a 7c 32 a0 a4 d6 07 af 74 b1 43 7e 81 25 95 88 2f 7b b9 0a b9 81 78 27 60        `.~w<).z|2.....t.C~.%../{....x'`
    f0 6b dd a8 53 5d ab 6d 82 7f ab 4f 90 7f 5f 98 a5 50 c8 29 86 19 aa 06 16 e8 f6 b1 e1 75 44 0f        .k..S].m...O.._..P.).........uD.
    82 83 f5 eb 96 24 ec 35 33 a9 96 04 81 d4 76 6f 2e 2e 92 21 07 72 e0 01 3f 7f 15 e3 7e 1f c7 31        .....$.53.....vo...!.r..?...~..1

Public Key Exponent (int): 3 (aka F0 exponent)

Extensions:

    Extension Name             Critical    Value
    -------------------------------------------------------------------------------------------------
    Basic Constraints          FALSE       Subject is a CA and supports maximum of 0 sub-CAs (pathLen)
    Key Usage                  FALSE       CRL Signature, Certificate Signature
    Subject Key Identifier     FALSE       708E1D89D7612E74FB05FC3EBEE9E6CDE0BFA059
    Authority Key Identifier   FALSE       9731A10D3E6DB33FA93FC348EFA9D0C4171534D9

Signature (Signature Verification Passed):
    3A CA 1F 31 AC 59 48 E6 C2 0D 7E A9 FF 38 0B 33 E8 29 11 87 CB 0E 77 15 6D 22 13 A9 BC 48 1E E9        :..1.YH...~..8.3.)....w.m"...H..
    D2 1A A6 AC 33 EB 22 3F 7D FE BC 06 96 5F 7C AB BD 16 91 56 04 B6 97 C5 E7 F7 CF F6 CE 7F F3 97        ....3."?}...._|....V............
    7F 44 3F A7 1E 13 BB B1 13 97 9C D9 8B 5F D4 40 28 1F CD 06 E2 32 7D DD E3 B8 A0 2A 5F B1 45 13        .D?.........._.@(....2}....*_.E.
    91 C3 C7 07 6A 72 E5 F2 5E 45 B9 BA 0C 0E 28 13 6F E8 F3 9E 8A 2D 2A 3E 7C 97 17 FD 77 F6 78 A6        ....jr..^E....(.o.....-*>|...w.x.
    DE 5E 35 F5 AB 9D 79 33 74 FE 55 84 5C 34 70 3F 18 DD BF 90 08 1A F8 95 06 21 1D 92 35 66 97 0F        .^5...y3t.U.\4p?.........!..5f..
    C5 EB F9 4A 0A B9 7B 0E 2E B2 23 9B 9E D5 55 4A 1D A2 A5 12 9F 06 F5 7F 28 43 29 B9 17 EB F5 C6        ...J..{...#...UJ........(C).....
    A1 8D FD F2 D6 F0 02 45 A1 06 4B B5 1E EE DF D3 E1 65 D7 E1 C3 DC 38 9F 38 98 20 1C E2 69 39 06        .......E..K......e....8.8. ..i9.
    66 A3 5D F6 1C 9A 7B 93 DA CF 2F 2E DE 90 FA 75 95 CF E6 50 C8 9A 63 97 CC 93 60 93 7B F2 BE 6E        f.]...{.../....u...P..c...`.{..n
```

Profile 2: Attestation CA Certificate Profile

The calculated SHA-256 digest of the Root CA certificate seen in Profile 1 would need to match the value in the device's eFuse or ROM in order for this Root CA certificate to be valid on that device.

The Basic Constraints indicate that this certificate can sign other certificates (`CA=TRUE`). We see that the Attestation CA certificate can also sign certificates, but only a leaf certificate like the Attestation certificate (`CA=TRUE, pathLen=0`). Both certificates' Key Usage indicate they are authorized for certificate signatures. The Subject Key Identifier and Authority Key Identifier fields indicate that the Root CA is the authority for the Attestation CA (and the Attestation CA is the authority for the Attestation certificate below). These two fields are included mostly for tracking rather than for security purposes.



```
Signature Algorithm: RSA PKCS#1 v1.5 Signature with SHA256
Valid notBefore: 14-Sep-16 04:46    Valid notAfter: 09-Sep-36 04:46
Serial Number: 0x01

Issuer:
    Organisation Name:        OEM
    Common Name:              OEM Attestation CA
    Locality:                 SANDIEGO
    State:                    CA
    Country:                  US

Subject:
    Organisation Name:        SecTools
    Common Name:              SecTools Test User
    Locality:                 San Diego
    State:                    California
    Country:                  US

Qualcomm Control (OU) Fields:
    OU=01    SW_ID        0000000000000000
    OU=02    HW_ID        009470E12A703DB9
    OU=03    DEBUG        0000000000000002
    OU=04    OEM_ID       2A70
    OU=05    SW_SIZE      00000248
    OU=06    MODEL_ID     3DB9
    OU=07    SHA256       0001

Public Key Modulus (hex):
    cd a1 c0 1d 06 66 e1 7b 40 57 65 74 48 13 ab 46 e1 e8 cb f6 ac 13 65 4b cd 6a bb 1e 39 af 16 d6        .....f.{@WetH..F......eK.j..9...
    a0 64 e8 b4 b3 e5 b9 07 34 07 62 1f ec cd e0 55 8b cf ad d8 5c 8e b3 7b c7 c0 13 b1 f9 f9 c1 b6        .d......4.b....U....\..{........
    9a 13 b8 18 13 c7 e7 f1 a4 41 42 b3 91 2e 9c 0b f8 c9 c2 2a 6f 7e da 74 cc db 08 7d de af 44 c6        .........AB........*o~.t...}..D.
    53 08 89 0a 63 04 96 e3 5f fc f6 29 dd 05 ee e7 0a a8 7c 7f 93 2e 01 86 25 0c 7a 3e 2e 9f d8 9a        S...c..._..)......|.....%.z>....
    5a 2d 2c 75 b7 89 65 44 e2 35 17 0d 6e 4f 13 99 fd 67 1e 75 b9 64 eb 65 57 7d 4f 38 1d 45 fd 80        Z-,u..eD.5..nO...g.u.d.eW}O8.E..
    13 f7 6b 4d f7 c7 01 f0 f4 a8 dd 0c be b9 ae 23 20 e5 b9 cf c8 17 e0 9b 5e 9d 84 2b 09 32 c1 f7        ..kM..........# .......^..+.2..
    19 48 c8 0f 9e 04 98 0e b2 99 ed 58 6c 64 73 33 53 09 e3 a0 e3 65 77 59 ec fc 07 3f 8c 54 c2 07        .H.........Xlds3S....ewY...?.T..
    53 0a 82 aa 5c 2e 0e 31 20 2e 18 08 b1 ea fd 6e ba 98 23 b9 38 75 9a b1 1e fa fa 9f 43 67 19 73        S...\..1 ......n..#.8u......Cg.s

Public Key Exponent (int): 3 (aka F0 exponent)

Extensions:

    Extension Name              Critical    Value
    ------------------------------------------------------------------------------------------------------
    Basic Constraints           FALSE       Subject is NOT a CA and supports maximum of 0 sub-CAs (pathLen)
    Key Usage                   FALSE       Digital Signature, Non-Repudiation, Data Encipherment, Key Encipherment
    Authority Key Identifier    FALSE       708E1D89D7612E74FB05FC3EBEE9E6CDE0BFA059
    CRL Distribution Point      FALSE       (Field Not Parsed)

Signature (Signature Verification Passed):
    0A 19 7A 82 B8 BC 06 45 5C 17 36 54 2C FA FC 49 86 B3 7B CC F8 06 98 16 45 10 9F 3A 7B 9B 06 D1        ..z....E\.6T,..I..{.....E..:{...
    2B 92 05 A3 58 F2 61 39 29 23 63 4C 7C 36 5D DC 49 BD 27 DD 70 D1 C8 42 C1 B5 60 9E 03 CC 23 FC        +...X.a9)#cL|6].I.'.p..B..`...#.
    A9 EE 5A 65 1A FE A2 C8 9A D2 D3 18 4F 51 DC 8C 00 2F FC E8 9D E3 AE 03 62 14 40 FE C6 9B 80 0F        ..Ze........OQ.../......b.@.....
    38 5D 91 9A 8E BB 22 2E 52 3B 22 0C D0 88 A6 29 09 DC F2 B0 BB 72 18 F3 60 C1 DB 4E 04 C5 08 4E        8]....".R;"....)......r..`..N...N
    66 0D 8B EA 85 65 BF A0 5C AC 34 E8 42 DB B7 A3 04 FA 10 29 0F 99 C9 23 EE 7C 10 A0 ED 9A 1F D2        f....e..\.4.B......)...#.|......
    DC 1E 93 C1 C2 FC 7A 0F AE 49 95 CB 68 EA E8 D7 70 75 03 17 D7 91 CC 0F 4A FD 1A AD 1C 9E 71 07        ......z..I..h...pu......J.....q.
    37 08 1A 00 39 D9 F7 2E 48 82 62 B9 BA 6B 08 C6 ED FE 4F 4A B9 49 6A 78 FA DA E4 53 6F 3B 9B B9        7...9...H.b..k....OJ.Ijx...So;..
    49 F4 F4 C5 7D E7 BF BA 83 C4 9D 69 1D C1 85 3F 8A AF E0 3A DA E7 5A 8B EA CA 0B B4 99 FE B3 FB        I...}......i...?...:..Z.........
```

Profile 3: Attestation Certificate Profile

The Basic Constraints of the Attestation certificate indicate that this certificate cannot sign other certificates (`CA=FALSE`) but that Key Usage indicates it is authorized for digital signatures (signing the software image). While the same Root CA and Attestation CA may be used for multiple images on multiple devices, a new Attestation certificate is generated for each signed image instance. The Attestation certificate contains OU fields which may provide information or restrictions on that signed image instance. The values found in OU fields may then be validated against those read from eFuse or explicitly expected by the verifying software. We discuss the common OU fields below.

Qualcomm Technologies OU Fields

The OU fields are designed to cryptographically bind various attributes to the signature and to provide information about the image and/or signature.

01 SW_ID

The software ID binds the signature to a particular version of a particular software image. This 64-bit value is a concatenation of those two values:

```
01      SW_ID           VERSION (32-bit) || IMAGE_ID (32-bit)
```

The value "`0x0000000000000000`" indicates version 0 of IMAGE_ID 0 (XBL). If eFuse values indicated that the current version was "`1`", then this image would fail verification. Version enforcement is done in order to prevent loading an older, perhaps vulnerable, version of the image that has a valid signature attached. The IMAGE_ID check is enforced to confirm that we are loading and verifying the software image that is expected to execute next – in this case, XBL Version 0.

02 HW_ID

The hardware ID binds the signature to a particular device family, model, and OEM. In the example shown, the HW_ID is constructed as follows:

```
02      HW_ID           MSM_ID (32-bit) || OEM_ID (16-bit) || MODEL_ID (16-bit)
```

The value "`0x009470E12A703DB9`" indicates that this image was signed for a device with MSM_ID `0x009470E1`, OEM_ID `0x2A70`, and MODEL_ID `0x3DB9`. The MSM_ID is a unique identifier chosen by Qualcomm Technologies to designate a specific chip-type within a given chip-family. The fields contained in HW_ID must match those provisioned in eFuse for the signature to be valid. Therefore this signature would be invalid on any other chip model or chip-type. It would also be invalid on a matching device model and chip-type with a differing OEM_ID value in eFuse; a legitimate signature created by OEM "A" will not be valid on any other OEM's device – even for the exact same software image version and type on a device using the exact same model and chip-type.

03 DEBUG

The Debug OU field indicates whether debug capability should be disabled or not. In the certificate above, the debug value is set to DISABLED ("`0x0000000000000002`").

04 OEM_ID

OU 04 is an easy-to-read copy of the OEM_ID from the HW_ID field, and is for information only. The value of OEM_ID is enforced in the HW_ID validation described above.

05 SW_SIZE

This OU field indicates the size of the data being signed (not the size of the software image). From Figure 2 we see this includes the Hash Table Header (40-bytes) and the Hash Table Entries (32-bytes per entry). The value of `0x248` indicates that there are 17 segments in this image. This OU field is for information only.

06 MODEL_ID

OU 06 is an easy-to-read copy of the MODEL_ID from the HW_ID field, and is for information only. The value of MODEL_ID is enforced in the HW_ID validation described above.

07 SHA256 / SHA1

This OU field indicates which hash algorithm is used to calculate the hash during signature verification. SHA-256 and SHA-1 are the current options. The example specifies SHA-256 (“`0001`”) as the hash algorithm.

5 Signatures

Signature Algorithms

The RSASSA-PKCS#1 v1.5 signature scheme is supported with SHA-256 or SHA-1 as the underlying hash algorithm. Newer chipsets also support RSASSA-PSS with SHA-256 as the underlying hash algorithm. A public exponent of 3 or 65,537 is supported for RSASSA-PKCS#1 v1.5, while only public exponent 65,537 is supported for RSASSA-PSS. Standard ECDSA over P-384 with SHA-384 as the underlying hash algorithm is also supported on a limited number of chipsets.

Certificate Signatures

The certificate signatures are all fully X.509 compliant. The certificate chain can be created, parsed, and verified by an open source X.509 compliant tool (such as OpenSSL). The detailed specification for both the RSASSA-PKCS#1 v1.5 and RSASSA-PSS signature schemes can be found in “*PKCS #1 v2.1: RSA Cryptography Standard*”, published by RSA Laboratories.

Image Hash Segment Signature

RSASSA-PKCS#1 v1.5

After the certificate chain has been verified, the signature on the software image’s hash segment is then verified. (The hash segment signature is the blue “Signature” appended to the green “Source” box shown in Figure 3.) The attestation certificate indicates which signature algorithm is used to verify the image signature. The standard RSASSA-PKCS#1 v1.5 signature scheme message-encoding (sometimes simply referred to as “padding”) is shown in Figure 5. The “Hash” bubble there refers to a standard SHA-256 or SHA-1 hash function, and “T” includes ASN.1 *DigestInfo* concatenated with the result of “Hash”.

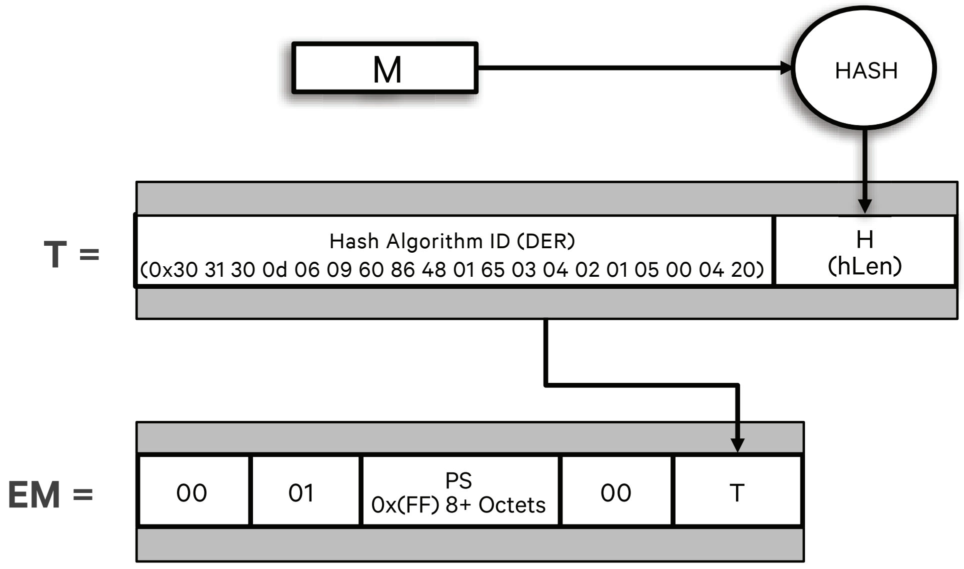


Figure 5: RSASSA-PKCS1 v1.5 Standard Encoding[3]

[3]Drawn from text specification in PKCS #1 v2.1: RSA Cryptography Standard

Qualcomm Technologies hash segment signatures of type RSASSA-PKCS#1 v1.5 vary slightly from the standard in the message encoding, and in the hash construction. The hash computation and encoded message padding are shown pictographically in Figure 6. The values “ipad” and “opad” are taken from the HMAC specification.

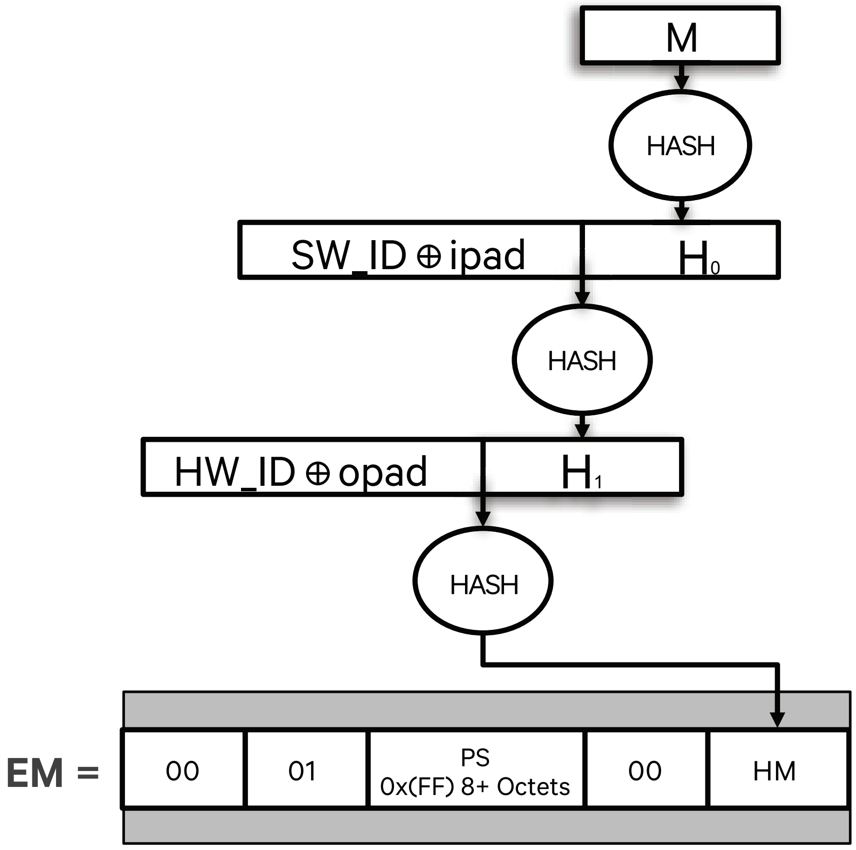


Figure 6: Qualcomm Technologies RSASSA-PKCS1 v1.5 Image Signature Encoding

Our hash segment signature excludes the DigestInfo from the encoded message (EM), and uses an HMAC-like hash keyed with SW_ID, HW_ID rather than a single key “K”. This is designed to provide a strong binding of the two most important identifiers (SW_ID and HW_ID) to the image hash (HM) used in signing, rather than being relegated to software checks. The hash algorithm ID from Figure 5 is not needed here since OU 07 is used to specify the hash algorithm.

RSASSA-PSS

The hash segment signature format for PSS is standards-compliant, as are the certificate signatures. Only SHA-256 is supported for “Hash” and “MGF” in the PSS signature scheme. For reference, the message-encoding scheme for PSS is given in Figure 7.

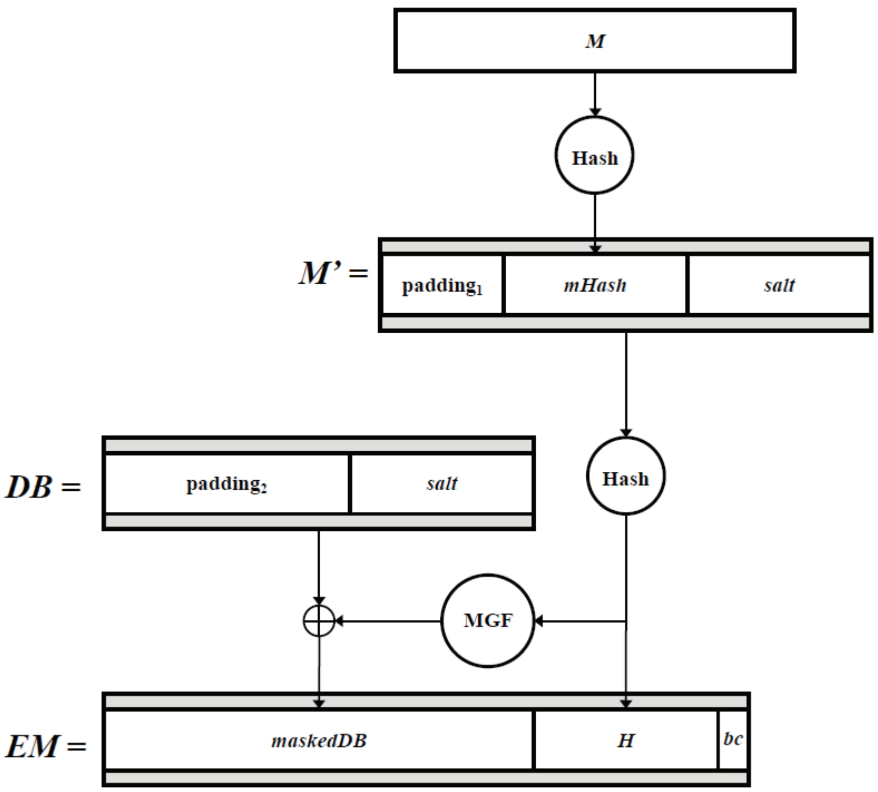


Figure 7: RSASSA-PSS Encoding[4]

6 Summary

Software images are loaded from untrusted storage to internal trusted memory and parsed. The loading and parsing phase includes address and size validation against whitelisted address ranges and integer overflow checks when calculating offsets and end addresses. First the image's ELF and Program headers are loaded and parsed, then the hash segment is loaded and parsed.

The hash segment is signed with a two or three certificate chain. Each certificate's signature is verified with the public key authorized by the certificate above it. The Attestation certificate is verified by the Attestation CA certificate in the three-certificate chain (but is verified by the Root CA certificate in the two-certificate chain). If present, the Attestation CA certificate is verified by the Root CA certificate. The Root CA certificate is validated by comparing its hash to the OEM-provisioned value in eFuse or in ROM. The software image's hash segment signature is verified by the public key authorized by the Attestation certificate. This hash segment signature verification includes enforcement of bindings between Qualcomm Technologies OU fields present in the Attestation certificate and the corresponding values on-device. A new Attestation certificate is generated each time an image is signed, making them unique per software image and instance.

[4]From PKCS #1 v2.1: RSA Cryptography Standard

The certificate chain signature algorithms support standard RSASSA-PKCS#1 v1.5 with SHA-1 or SHA-256 and RSASSA-PSS with SHA-256. The image hash segment signature supports a standard-variant RSA PKCS#1 v1.5 with SHA-1 or SHA-256 and standard RSASSA-PSS with SHA-256. ECDSA over P-384 with SHA-384 is supported for both certificate chain signatures and image hash segment signatures on a limited number of chipsets.

After the certificate chain and image signatures have been verified, each NON-PAGED LOAD segment in the ELF image is hashed and compared against its hash segment entry. If the comparison fails, or an entry does not exist for that segment, then the verification sequence fails. Otherwise, all loaded segments of the software image have passed verification, leading back through the chain of trust to the root of trust.

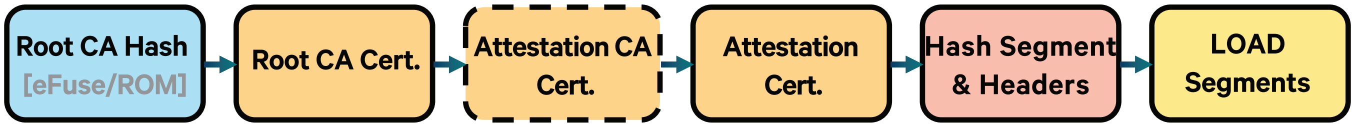


Figure 8: The Chain of Trust

Execution is now transferred to the entry point of the image that has just been successfully verified. When the next software image is to be loaded, authenticated, and executed, the same process is repeated.