Trusted Board Boot Requirements CLIENT (TBBR-CLIENT) Armv8-A

Architecture and Technology Group

Document number:	ARM DEN0006D
Release Quality:	Beta
Release Number:	1
Confidentiality	Non-Confidential
Date of Issue:	20/09/2018

Abstract

Trusted Board Boot Requirements CLIENT (TBBR)

This document describes and defines a Trusted Boot Process for application processors. The document introduces the basic concept of isolation of the Trusted world from the Non-Trusted world when booting as well as methods to handle manufacturing and test in boot. The document also describes a standardized structure for the signing of firmware images, through the use of certificates.

The document describes the set of features required for a boot process to be compliant with GlobalPlatform TEE Protection Profile 1.2 0, and outlines certificate and cryptography standardization and describes how software and hardware partners can use Arm TrustZone® technology or its equivalent to improve on current implementations of Trusted Boot.

Contents

1 About this document

1.1 References

This document refers to the following documents.

Ref	Doc No	Title
	PRD29-GENC-009492C	Arm Security Technology - Building a Secure System using TrustZone Technology
	ARM DDI 0487A-e	Arm Architecture Reference Manual - Armv8-A, for Armv8-A architecture profile
	ARM DEN0016B	Arm Architecture Standard Configurations-System Software on Arm Processors
	ARM DEN 0007C-5	Trusted Base System Architecture CLIENT1-TBSA-CLIENT1
	ARM DEN 0034A	Arm Debug and Trace, Configuration and Usage Models

1.2 Other publications

The following documents list relevant documents published by third parties:

[1.] Unified Extensible Firmware Interface Specification. Version 2.4 (Errata B) http://www.uefi.org/specifications

[2.] Secure Hash Standard (SHS). Federal Information Processing Standards Publication (FIPS PUB 180-4)

[3.] PKCS #1: RSA Cryptography Standard v2.2.RSA Laboratories

[4.] Digital Signature Standard (DSS). Federal Information Processing Standards Publication (FIPS PUB 186-4)

[5.] NSA Suite B Cryptography http://www.nsa.gov/ia/programs/suiteb_cryptography/

[6.] SEC - Recommended Elliptic Curve Domain Parameters http://www.secg.org/sec2-v2.pdf

[7.] Internet X.509 Public Key Infrastructure, Certificate and Certificate Revocation List (CRL) http://tools.ietf.org/html/rfc5280

[8.] Internet X.509 Public Key Infrastructure, additional Algorithms and Identifiers for DSA and ECDSA http://tools.ietf.org/html/rfc5758

[9.] Trusted Platform Module 1.2 Specifications http://www.trustedcomputinggroup.org/resources/tpm_main_specification

[10.] RFC 2119 - Key words for use in RFCs to Indicate Requirement Levels http://www.ietf.org/rfc/rfc2119.txt

[11.] RFC 4122 - A Universally Unique Identifier (UUID) URN Namespace http://tools.ietf.org/html/rfc4122

[12.] GlobalPlatform TEE System Architecture v1.0

[13.] GlobalPlatform TEE Protection Profile Specification v1.2

[14.] GlobalPlatform TEE Client API Specification v1.0

[15.] GlobalPlatform TEE Internal API Specification v1.0

[16.] GlobalPlatform TEE Trusted UI API Specification v1.0

1.3 Terms and abbreviations

This document uses the following terms and abbreviations.

Term	Meaning
AArch64 state	The Arm 64-bit execution state that uses 64-bit general purpose registers, and a 64-bit program counter (PC), stack pointer (SP), and exception link registers (ELR). AArch64 execution state provides a single instruction set, A64.
AES	Advanced Encryption Standard, a symmetric-key encryption standard
AFM	Application/Functional Mode – the normal operational state of a device, including boot of the device, through to loading and execution of the operating system. On some devices, it may be possible to boot to an alternative production state for the purpose of executing production tests. To reduce manufacturing time, this may not involve booting the operating system present on the device.
AP	Application Processor.
eFuse	One Time Programmable (OTP) memory, usually only available to Trusted world software and in very limited quantity.
FOTA	Firmware Over-The-Air
NIST	National Institute of Standards and Technology (http://www.nist.gov/)
Non-Trusted world	The code, data and peripherals accessible when executing in the non-secure state of the CPU. This is often referred to as the Normal world.
Non-Trusted RAM	RAM addressable by the Non-Trusted world. This may be on-chip or off-chip
NV	Non-Volatile
NVM	Non-Volatile Memory such as NAND Flash or LPPDR2-NVM
operating point	a recommended operating condition defined by frequency and voltage
PKCS	Public Key Cryptography Standards
RSA	Rivest, Shamir and Adleman. An algorithm for public-key cryptography
RSA-PSS	A shortened form of RSASSA-PSS, RSA Signature Scheme with Appendix - Probabilistic Signature Scheme.
SCP	System Control Processor
SoC	System On Chip
TCG	Trusted Computing Group (http://www.trustedcomputinggroup.org/)
TEE	**Trusted Execution Environment is the combination of Arm TrustZone hardware and the Trusted OS**
Trusted RAM	RAM addressable only in the Trusted world. This may be on-chip SRAM or off-chip, unless otherwise stated
Trusted World	The code, data and peripherals accessible when executing in the secure or trusted state of the CPU. This is sometimes referred to as the Secure World.
UEFI	Unified Extensible Firmware Interface.
XIP	eXecute-In-Place

1.4 Scope of this Document

Only the part of the boot process for secure on-chip firmware that executes *before* the operating system or the operating system boot loader is described.

It should be noted that the implementation of the specification can also enable secure boot of the operating system, *only when combined with a secure bootloader for the operating system*, such as that defined by Unified Extensible Firmware Interface (UEFI) 0.

To provide a trustworthy boot process all mandatory requirements described in this document must be implemented fully.

1.5   Assumptions

This document makes no assumptions about the contents of the firmware running in the Non-Trusted world.

The document assumes the reader is familiar with standard cryptography techniques, such as authentication, hashing, encryption, and Public Key Cryptography Standards (PKCS) 0, NSA suite B, 128-bits security level 0 and X.509 certificates 0. It does not describe these concepts in detail.

2   Overview of the Trusted Boot Process

Platforms that wish to provide secure services have a fundamental requirement to be trustworthy. A key component of their design is *a trustworthy boot process*, which ensures the integrity of the code executed on the SoC from the very first instruction through to the loading of the operating system.

The *trusted boot process* described in this document is a method to ensure that from the point of reset of a SoC, only the correct and intended firmware, operating system, and Trusted Execution Environment, will be authenticated, loaded and initialized onto the Soc.

2.1   Authentication of Code Images by Certificate

The trusted boot process works by authenticating a series of cryptographically signed binary images each containing a different stage or element in the system to be loaded and executed. The signatures for each image are stored in X.509 certificates. Each image is authenticated by a public key, which is stored in a signed certificate and can be traced back to a root key stored on the SoC in OTP memory or ROM.

This process can provide assurance that the SoC will only load the system components intended by the OEM or silicon vendor, as only images which have been signed can be loaded and executed from boot.

Because the images are signed by public key cryptography, the boot process can authenticate the images against the public key stored on the device, and the private key used to generate the signature need never be exposed on the device itself.

Several keys are specified in this document for flexibility and to allow an OEM and silicon vendor to act independently of each other.

When implemented either fully or with the subset of requirements defined in section 3.7, the boot process can enable a secure system capable of payment and financial use cases as defined by Boundary 1 in section 3.8, or alternatively can enable content protection, DRM and enterprise use cases and applications as defined by Boundary 2 in section 3.8.

2.2   Software Architecture

Figure 1 shows the software architecture for an Armv8-A AArch64 system, which is the assumed reference model in this document. However, the same model can be applied to any CPU architecture that supports an equivalent capability. The SoC firmware is shown at the highest Exception level, (although the firmware may also execute at lower exception levels as well). The SoC firmware contains the mechanism, used to switch between the Trusted world and the Non-trusted world, as well as SoC specific firmware services, such as power management.
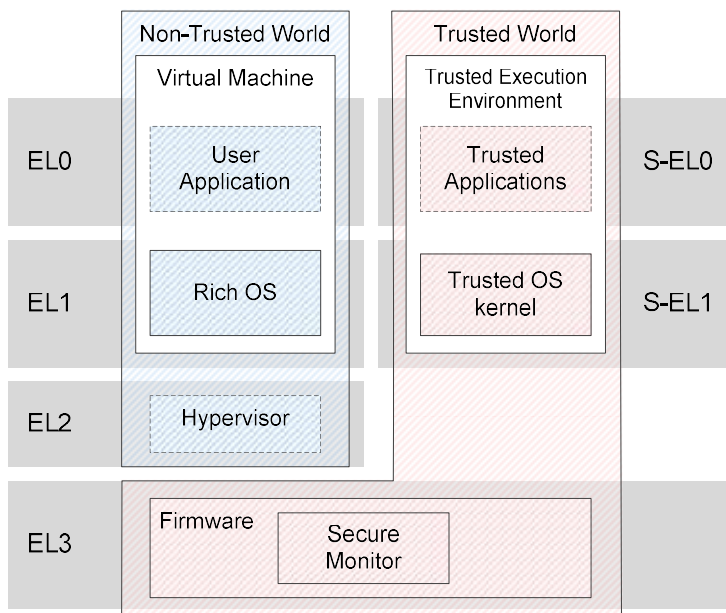
Figure 1: Software Architecture

Non-Confidential

Within the Trusted World, the Trusted Execution Environment exists, containing the Trusted OS kernel at Kernel privilege level and Trusted Applications running at application privilege level.

In the Non-trusted world, at an Hypervisor privilege level may exist, in addition to the OS Kernel & User application privilege level ay be present.

Table 1 – Privilege Level Mapping

Privilege Level	Description	Implementation
EL-0	Application Privilege Level	Supported by CPU architecture
EL-1	Kernel Privilege Level	Supported by CPU architecture
EL2	Virtualization Privilege Level (Optional)	Supported by CPU architecture
EL-3	Secure Privilege Level	Either supported by CPU architecture or a dedicated embedded security processor

2.3 Trusted Base System Architecture

The boot process, assumes the hardware platform topology illustrated that provides the base Trusted Base System Architecture.

Figure 2 shows a high-level system topology diagram of an example system that incorporates the Trusted Base System Architecture. It is beyond the scope of this specification to describe it all in detail.
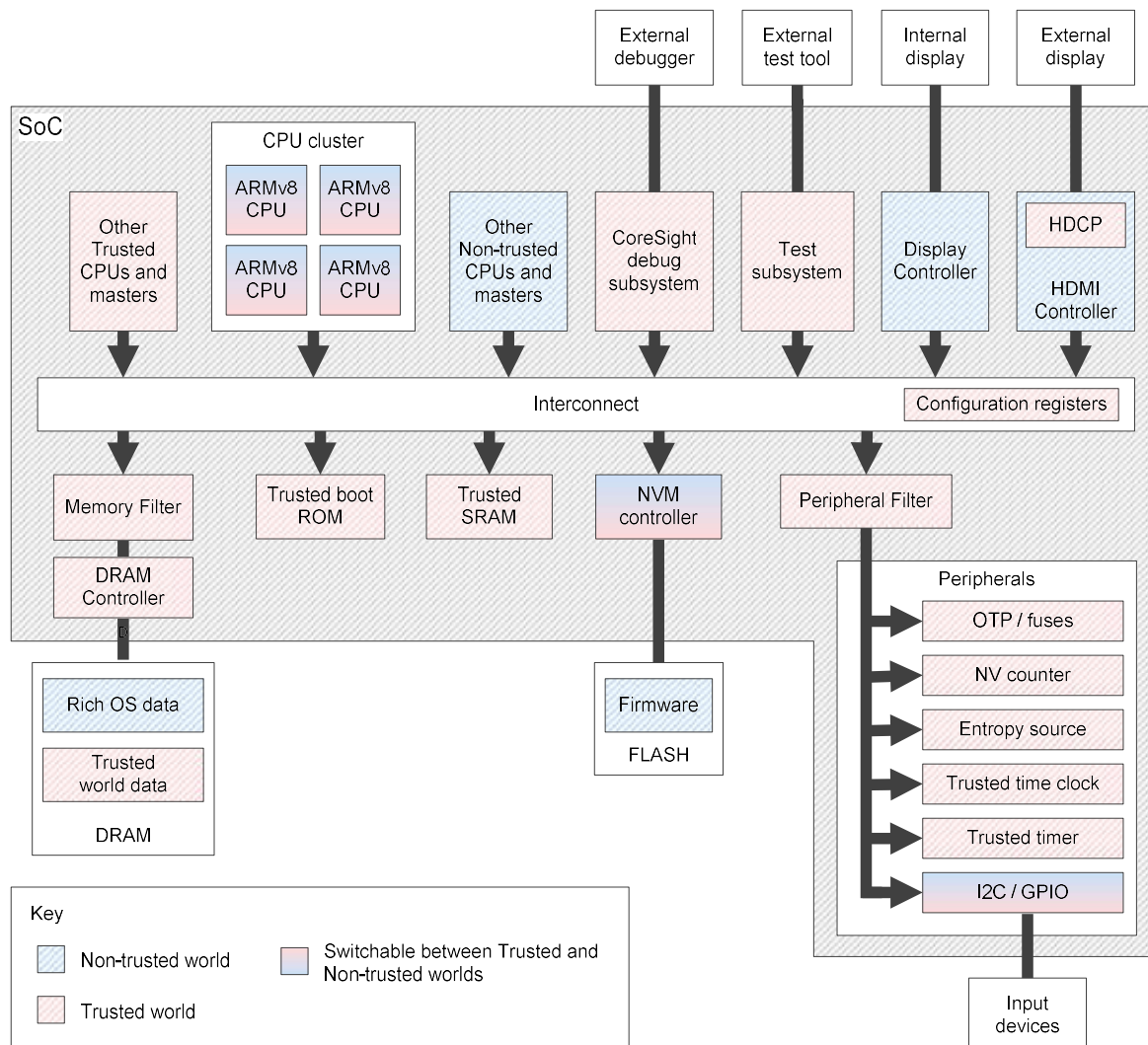
Figure 2: Trusted Base System Architecture

2.3.1 CPU Cluster

Typically, as shown in Figure 2, a SoC will have one or more clusters of processor cores.

Conventionally, modern operating systems perform much of their kernel boot process on one core, called the *primary core*, bringing other *secondary cores* online at a later stage.

In this document, unless otherwise stated the Application Processor refers to the *primary core*, and it is assumed secondary cores will be initialized at a later point by the main operating system.

2.3.2 Trusted RAM

Trusted RAM is memory which is addressable only from the Trusted world when executing in the Secure state. Trusted RAM may be on-chip SRAM or off-chip DRAM.

2.3.3 System Control and Power Management Subsystem (SCP)

The System Control Processor (SCP) is an optional trusted on-chip MCU that performs power management functions and has responsibility for power management of the SoC. The SCP controls clocks and reset of each core and is responsible for power state transitions for the power regions on the SoC.

The SCP may not be present in all designs, and its role in the boot process is described in Appendix A.

2.3.4 Power Control Function

Where an SCP is not present in a SoC, its function may be taken by the Application Processor. In this document, the term 'Power Control Function' is used to refer to the entity that takes responsibility for power management functions, whether it's the SCP or Application Processor.

3 Trusted Boot Requirements

The following section describes Trusted Boot requirements.

3.1 Fundamental Features

The Trusted Boot process has multiple stages and uses multiple firmware images. It is intended to maintain the chain of trust between the different boot stages using standard cryptography.

R010_TBBR_FUNCTION. The key functionalities that the Trusted Boot process **must** support are:

1. Configuration of SoC hardware
2. Application Processor (AP) Trusted Firmware installation
3. AP Trusted OS booting
4. Rich OS or Hypervisor boot handover

R020_TBBR_FUNCTION. The functionalities that the Trusted Boot process **may** support are:

1. ROM patching
2. Manufacturing Test mode and/or Key Provisioning mode
3. System Control Processor (SCP)
4. Trusted Debug control

Support for the System Control Processor is described in Appendix A.

R030_TBBR_FUNCTION. All the certificates and firmware (except the debug certificates) **must** be protected against rollback using an NV counter.

R040_TBBR_FUNCTION. The AP Boot ROM **may** be patchable.

R050_TBBR_FUNCTION. It **must** be possible to install the SoC configuration update independently of the Trusted OS to correct hardware bugs, such as, for example, SoC errata. The SoC configuration update may be part of the SoC AP firmware image or held in a separate image file.

R060_TBBR_FUNCTION. The certificates, Trusted firmware images, and other information loaded from the Non-Trusted world **may** be encrypted for anti-cloning protection.

R070_TBBR_FUNCTION. If the certificates are encrypted, they **may** be authenticated before decryption. The keys **must** be unique to the device or a batch of devices.

3.2 Certificate Structure and Cryptography

The following sections briefly describe the security concepts used in this document.

3.2.1 Cryptography

The cryptography algorithms referred to here are well-understood industry standards. The choice of algorithms and security level does not radically affect the key certificate infrastructure and the SoC Trusted boot sequence, but it gives a useful indication of the size of the certificate structures needed.

R010_TBBR_CRYPTO. The certificate signatures **must** be NSA suite B 128-bit security compliant and consequently use the following cryptographic algorithms:

AES-128

SHA-256

ECC256 (ECDSA) or (legacy support of) RSA2048 (RSA-PSS)

R020_TBBR_CRYPTO. The following standards **must** be followed as applicable:

PKCS #1 – Standard for RSA. See reference 0

FIPS 186-4 – Standard for ECDSA. See reference 0

The choice between using RSA-PSS or EC-DSA may be made for patent related reasons, hardware cost, software certificate size and time required for authentication.

The RSA-PSS signature scheme additionally signs a hash of the message instead of the message directly. This technique is almost always used with RSA because the amount of data that can be directly signed is proportional to the size of the keys, which is usually much smaller than the original message.

A digital signature produced using RSA-PSS is the same length as the RSA key modulus, that is, at least 2048 bits.

The EC-DSA signature scheme signs a hash of the message instead of the message directly. The notable benefit of using EC-DSA over RSA-PSS is that the resulting signature size is much smaller.

This document assumes that certificate signatures have the same length as keys.

> R030_TBBR_CRYPTO. Cryptographic algorithms **may** be implemented using hardware dedicated engines or in software. Both **may** be protected against non-invasive side-channel attacks.

> R040_TBBR_CRYPTO. The software implementation of the cryptographic primitives **must** ensure that their execution have the same processing time and cache foot print for every code path in the cryptographic algorithm.

Note side-channels are listed as a possible threat by the GlobalPlatform Protection Profile0.

3.2.2 Boot certificates

In this specification, fixed format, DER encoded, self-signed X.509 certificates using ecdsa-with-SHA256 [1] have been chosen as an example of implementation. They are described using PKIX profile as specified in RFC 52800. Other solutions exist and can be used to successful apply to GlobalPlatform certification 0.

Moreover, X.509 standard is well documented and free tools and source code are available.

The certificates must comply with the following:

> R010_TBBR_CERTIFICATE. Each certificate used for the Trusted Boot **may** be X.509v3 compliant, the x590v3 extension fields **may** be used for passing security parameters such as the NV counters, firmware hash, public keys and SoC security parameters.

> R020_TBBR_CERTIFICATE. Each certificate **may** have an issuer name, a subject name, a unique serial number, algorithm used and public key information (with the exception that public keys value may not be passed for every certificate for size reduction).

> R030_TBBR_CERTIFICATE. Each certificate **may** have a validity period.

> R040_TBBR_CERTIFICATE. Prior to any authentication of a Trusted Firmware certificate, the certificate **must** be present in Trusted RAM.

> R050_TBBR_CERTIFICATE. Prior to any integrity check of a modifiable Trusted software image involved in the Trusted boot process, the software image **must** be downloaded into the Trusted RAM.

Figure 3 is an overview of the structure of X.509 v3 certificate with an example of filing as explained in the Certificate Section 3.6.

---

[1] NSA suite B 0 recommend transitioning to ECC.

Version of the certificate (v3=2)
Certificate Serial Number (Integer)
Certificate Algorithm Identifier for certificate Issuer's Signature   =1.2.840.10045.4.3.2 (ecdsa-with-SHA256 as defined in RFC 5758 [12.])
Certificate Issuer name (same as Subject Name, follows RFC 5280 [11.])
Validity Period (Start and expiry dates/times, follows RFC 5280 [11.])
Subject Name (e.g. Trusted OS certificate, follows RFC 5280 [11.])
Subject Public Key Information:
AlgorithmIdentifier:algorithm = 1.2.840.10045.2.1 (ecPublicKey)
AlgorithmIdentifier:parameters = 1.2.840.10045.3.1.7 (secp256r1 as defined in SECG [10.]
SubjectPublicKey = 528 bits (First byte is 0x00 for number of unused bits in last octet; secon byte is 0x04 for uncompressed format. Followed by 256bit x coordinate; 256bit y coordinate)

Certificate extension (X.509 v3 only):

Extension Name	Criticality	Value
(e.g. Debug Scenario)	(=1)	(e.g. 0b01= Public Debug)
(e.g. TrustedFirmwareNVcounter)	(=1)	(e.g.5 bits)
(e.g. Key Certificate PK, Trusted OS PK)	(=1)	(e.g.512 bits for EC-DSA)
….	….	….

Signature:

Signature Algorithm Identifier   =1.2.840.10045.4.3.2 (ecdsa-with-SHA256 as defined in RFC 5758 [12.])
Signature value = 256Bits for EC-DSA

Hashing (SHA256) followed by signature generation (EC-DSA256)

Figure 3: Structure of X.509 v3 certificate

3.3   SoC Cryptographic Keys

The following sections summarize the Cryptographic keys that are required by the Trusted Based System Architecture.

R01 0_TBBR_KEY_STORAGE. The SoC **must** implement a minimum OTP/Fuse size of 384 bits, and if the EK is used a total of 640 bits, or if the optional SSK key is used, a total of 768 bits.

R02 0_TBBR_KEY_STORAGE. Once keys are programmed further programming **may** be prevented by an additional OTP/Fuse bit. How this OTP/Fuse bit inhibits further programming is IMPLEMENTATION DEFINED.

Table 2 OTP/Fuse Data Fields

Name	Size (bits)	Description
Root of trust Public Key Hash (ROTPK)	≥256	Message digest of the root of trust public key
Hardware Unique Key (HUK)	≥128	Key used for Secure Storage and binding
Endorsement Key (EK, optional)	≥256	Key used for trustworthiness proving
Secret Symmetric Key (SSK, Optional)	≥128	Key used for firmware image decryption

The manufacturing mode information can be directly encoded using OTP/Efuse or generated from other OTP/Efuse information such as the ROTPK. It is expected that at least the following device life cycle states are available to the Trusted SW developers:

- Un-initialized device
- Test and Development device
- Mass production device
- Device not working (identified at factory)

- Device returned for repair (this state may be the same as a test or production state)

Should a device be returned for repair and enter a repair state, appropriate steps should be taken to protect confidential user data present on the device, and particularly that of stored in the Trusted world.

3.3.1 NV Counters

If a firmware image is updated to fix security vulnerabilities, and the device permits the firmware image to be "rolled-back" to a previous, unsecure version, then a security risk exists. Therefore, firmware must use Non-Volatile (NV) counters to protect against rollback.

The Trusted Based System Architecture requires two trusted non-volatile counters in *R020_NV_COUNTER* specifically for the Trusted Boot process:

- The TrustedFirmwareNVcounter which have a minimum of 31 states plus an "overflow" state.

- The NonTrustedFirmwareNVcounter must have a minimum of 223 states plus an "overflow" state.

After a firmware image is validated, the image revision number taken from the certificate extension field, for example, TrustedFirmwareNvCounter is compared with the corresponding NV counter stored in hardware. If the value is:

- Less than the associated NV counter then the authentication fails.

- Identical to the NV counter then the authentication is successful.

- More than the NV counter then the authentication is successful, and the NV counter is updated.

Optionally, the Trusted Computing Group (TCG) recommends the addition of four 32-bit counters capable of counting 2^{32} states for synchronizing data stores against rollback attacks in 0. These counters are optionally required since they can be handled by a Trusted OS service using the secure storage at boot time or using eMMC v4.4x Replay Protected Memory Block (RPMB).

3.3.2 Key security robustness property

The first link in the chain of trust is the management of the OEM root of trust private key, which signs the subsidiary keys used to sign all subsequent certificates.

R010_TBBR_KEY_REVOCATION. In order to provide key revocation capability in the Trusted boot process, all subsidiary certificates of the Trusted Debug, SoC firmware, AP Trusted OS, SCP Trusted OS (if implemented), Rich OS (if implemented) **must** be signed by different private/public keys pairs and **must** contain the TrustedFirmwareNvCounter for the Trusted world certificates and the NonTrustedFirmwareNvCounter for the Non-Trusted world certificates.

The root of trust private key is the property of the OEM, which is signing the subsidiary public keys contained in the Trusted key certificate. The corresponding private keys are owned by the OEM or the third parties and they are used to sign pieces of the firmware.

R020_TBBR_KEY_REVOCATION. The OEM **must** sign a key certificate containing subsidiary public keys for verifying the Trusted Debug, SoC firmware, AP Trusted OS, SCP Trusted OS (if implemented), Rich OS (if implemented) certificates and containing the TrustedFirmwareNvCounter.

R030_TBBR_KEY_REVOCATION. If one of these subsidiary private keys is compromised, they **must** be revoked as long as the root of trust private is not compromised, by signing a new key certificate with an incremented TrustedFirmwareNvCounter.

If a single version of the root of trust private key exists for an entire class of devices, then the security of all those devices is reliant on this key not being compromised. Although it might be difficult for an attacker to obtain this key, the consequences of a successful attack are severe. This is known as a "class break".

To prevent attacks found on the web to be portable on a class of device, the *Trusted Execution Environment* TEE shall be made in such a way that any key used, for example, for secure storage, is derived from a key that is unique per device, the *Hardware Unique Key* (HUK).

R040_TBBR_KEY_REVOCATION. The HUK **must** be generated randomly per device and be used only for key derivation to permit the binding of secrets uniquely to the platform (consequently encrypted secrets cannot be exported to other devices).

R050_TBBR_KEY_REVOCATION. Compromising the HUK of one device **must** only affect this device.

R060_TBBR_KEY_REVOCATION. Compromising one device **must** not affect the "class" of this device.

If the *Secret Symmetric Key* (SSK) used for firmware decryption before binding with HUK derivative is compromised, this does not lead to the chain of trust being compromised but to the anti-cloning and anti-copying protection to be defeated.

3.4   Trusted Boot Process

This section describes the Trusted Boot Process requirements.

3.4.1   Chain of Trust

Each step in the Trusted Boot process is only as Trusted as the step that precedes it. Therefore, a chain of trust is extended by each step in the Trusted Boot Process.

3.4.2   Overview

The following diagram shows an overview of the Trusted Boot process and the sequence of events.
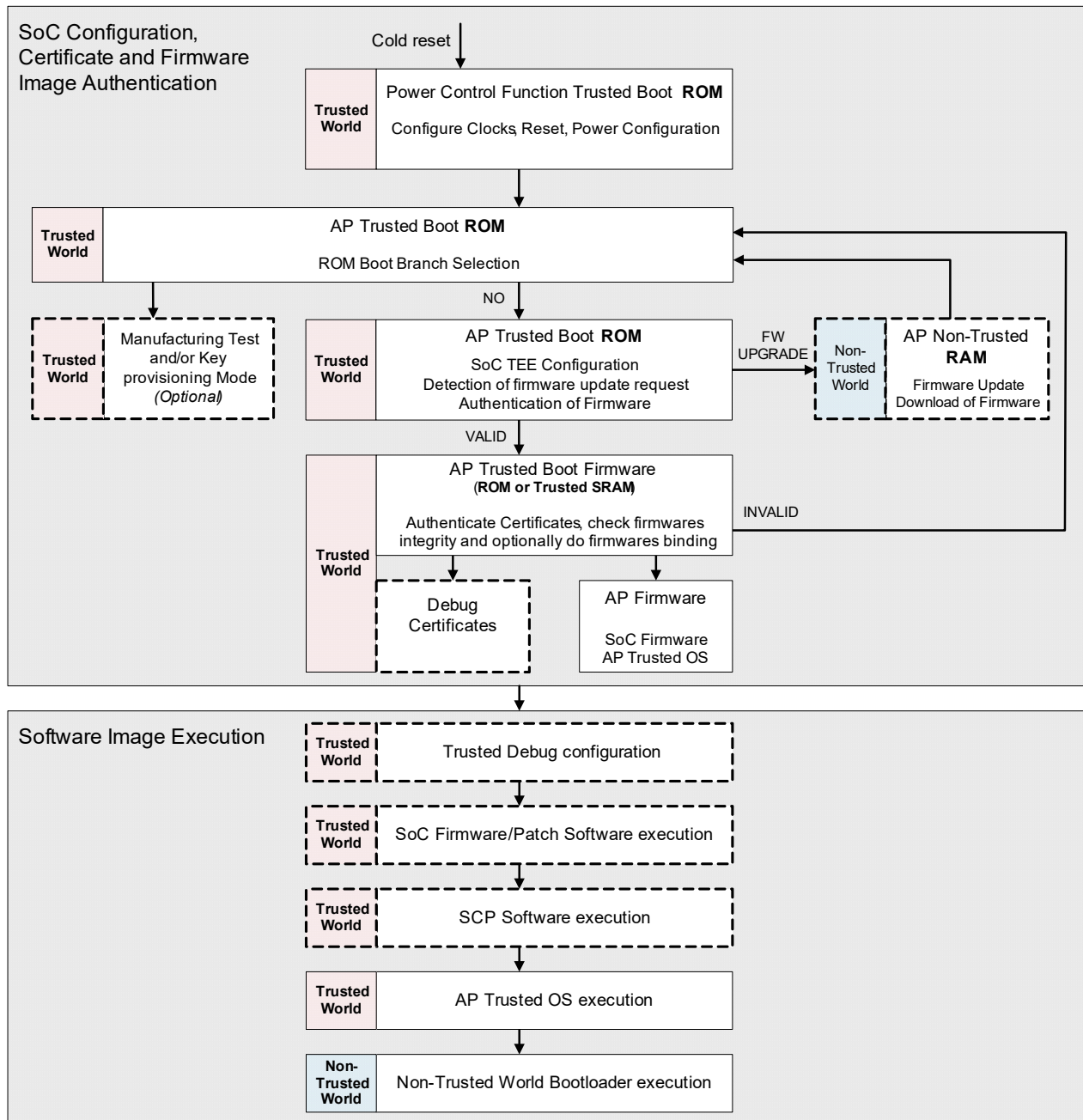
Figure 4: Trusted Boot Process Overview

3.4.3   SoC Wakeup from cold reset

As soon as the SoC comes out of power-on-reset, the Power Control Function begins executing from its reset vector. This is the first software to run on the SoC.

R01 0_TBBR_WAKEUP. As soon as the SoC comes out of power-on-reset:

1. If the device is a "production device", the Power Control Function **must**, when released from reset, begin executing in integrity protected memory such as on-chip Boot ROM or Trusted SRAM.

2. If the device is a "test and development" or "un-initialized" device than it **may** boot on external XIP memory or SRAM and bypass the on-chip Boot ROM. This is motivated by speed binning and ROM debug.

R02 0_TBBR_WAKEUP. The AP Boot ROM may contain more than one initial execution state:

1. The first state **must** be the normal operational model that boots the TEE and executes the Non-Trusted boot loader.

2. The second state **may** be key provisioning software mode and is selected when the device is in an "uninitialized state" and will allow SoC key provisioning.

3. The third state **may** be a manufacturing test mode, entered to enable production tests with minimal boot time to reduce manufacturing time.

Production or diagnostic test modes used in repair and rework, such as the manufacturing and key provisioning test modes described above, are outside the scope of this document. However, it should be noted that such modes must be carefully secured and must have restricted access. Securing the modes could mean, having a fuse that allows provisioning to occur only once in the controlled conditions of a factory, or alternatively for diagnostic software used in repair, and must mean booting the AP into the Non-Trusted world, in a suitably configured state to ensure software cannot access any secure assets,

3.5   Certificates Chaining and Key Infrastructure

Figure 5 explains the certificate key infrastructure and dependencies in place for each certificate and how they are related.
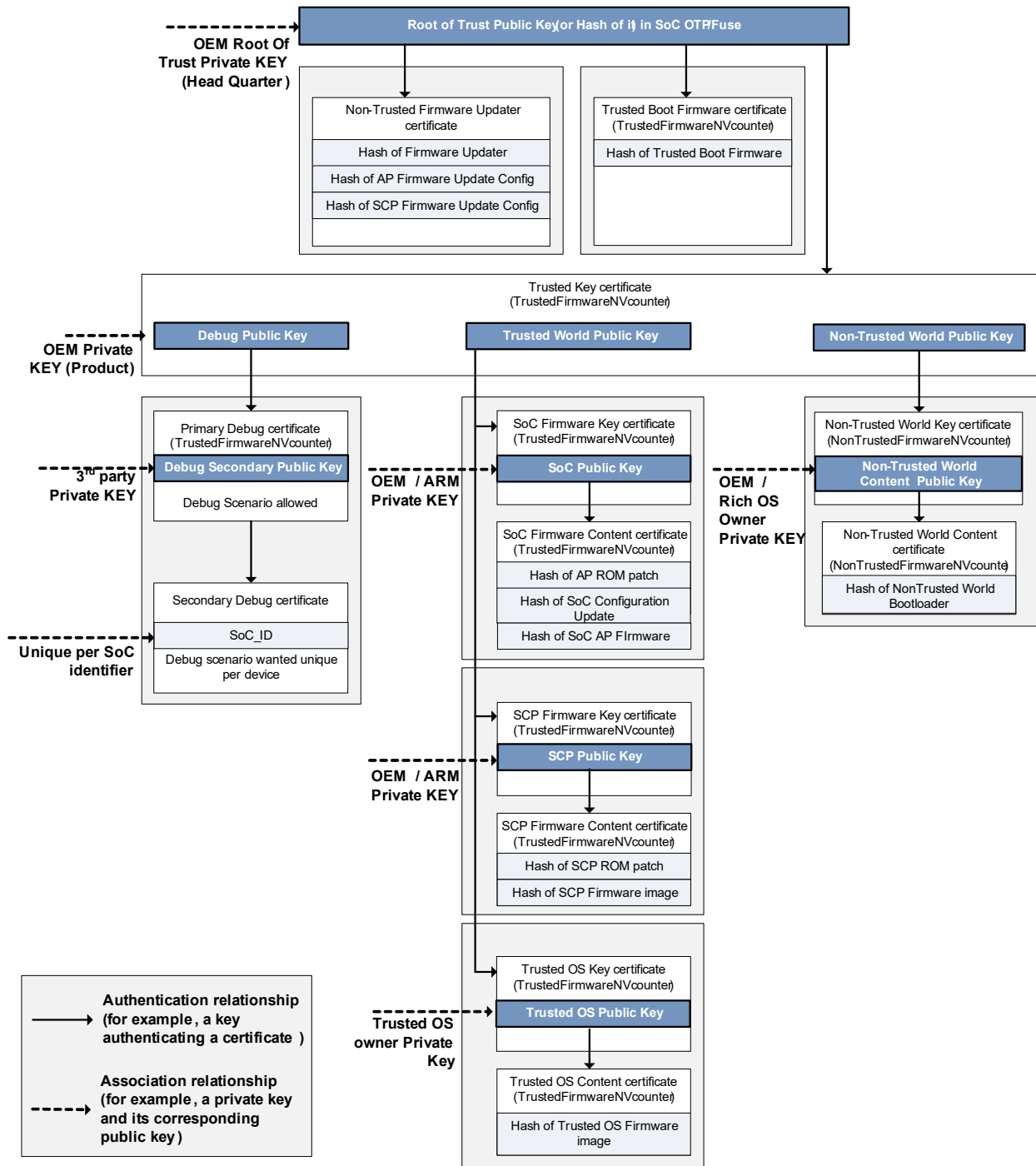
Figure 5: Certificate and Key Relationships

The root of trust private key is the property of the OEM, this key is able to sign the subsidiary public keys in the Key Certificate, owned by the OEM or third parties. The OEM or third parties own the corresponding private keys and use them to sign the subsequent firmware content certificates. The root of trust private key is also used to sign the Firmware Updater and Trusted Boot Firmware Certificates.

3.5.1 TEE configuration and firmware flashing

The primary purpose of this stage is to setup clocks; resets, power configuration as well as any mandatory SoC interconnect QoS function to enable the AP to boot.

The Power Control Function can be conducted by the SCP or by the AP ROM depending of the SoC architecture.

> R010_TBBR_AFM_BOOTROM. The Power Control Function **must** configure the SoC clocks, reset, power domains as well as any mechanism that will be needed for the Trusted Boot to proceed.

Its secondary purpose is to setup the AP TrustZone technology such as the exception vectors, MMU, system registers and the EL3 monitor within the SoC AP firmware and to configure any SoC system security settings that have not been initialized by hardware, for example, the Snoop Control Unit.

> R020_TBBR_AFM_BOOTROM. The AP firmware **must** configure the TrustZone technology extensions. This corresponds to the exception vectors, stacks, system registers and MMU mapping of the EL3 monitor firmware.

> R030_TBBR_AFM_BOOTROM. The AP firmware **must** configure the SoC functional and QoS security mechanisms.

> R040_TBBR_AFM_BOOTROM. The AP firmware **must** erase the Trusted RAMs before using them, or check that the hardware has truly erased them.

> R050_TBBR_AFM_BOOTROM. The AP firmware **must** invalidate the L1/L2 cache to prevent false cache line hits.

In a heterogeneous or in a multi-cluster Cortex-A series processor implementation the Trusted Boot process is conducted on a single Cortex-A processor core.

> R060_TBBR_AFM_ BOOTROM. All processor cores of the AP cluster except the core executing the Trusted Boot Process **must** stay powered off or put in WFI, WFE state until the Trusted Boot Process is finished.

The third purpose of this stage is to check if the SoC input pins are setup to request a SoC firmware update and to check if a valid SoC firmware Trusted Table of Contents (ToC) is present in the NVM memory.

> R070_TBBR_AFM_BOOTROM. The AP firmware **must** access SoC read only register to reflecting the status of the input pins to detect if a firmware update is requested

> R080_TBBR_AFM_BOOTROM. The AP firmware **must** configure any TrustZone Address Space Controller (e.g. TZASC) for accessing the NVM firmware images.

> R090_TBBR_AFM_BOOTROM. The AP firmware ROM'ed branch **must** check if the SoC input pins are setup to request a SoC firmware updated.

> R0100_TBBR_AFM_BOOTROM. If a firmware update is requested, the firmware **must** launch the SoC firmware update routine. If not requested the Table of Contents (ToC) of current firmware must be loaded into Trusted SRAM and its validity checked.

> R0110_TBBR_AFM_BOOTROM. If the ToC is not valid, the SoC firmware update routine **must** be launched.

If a firmware update is requested, the following requirements must be followed:

> R0120_TBBR_AFM_BOOTROM. Before switching to the Non-Trusted world, the AP firmware **must** ensure that there is enough Non-Trusted RAM available to host the firmware loader.

> R0130_TBBR_AFM_BOOTROM. Before switching to the Non-Trusted world, the Trusted watchdog timer **must** be refreshed and reprogrammed with the 256s value to ensure the flashing process have enough time to complete.

> R0140_TBBR_AFM_BOOTROM. The AP firmware **must** switch to the Non-Trusted world for downloading the SoC firmware loader.

3.5.2   **AP Non-Trusted firmware updater download and its authentication by the Trusted world**

The AP Non-Trusted firmware updater is responsible for loading the complete SoC firmware, which includes SCP firmware (if present), AP firmware, Trusted OS, Non-Trusted world images from an external interface such as USB, UART, SD-MMC, NAND, NOR, Ethernet to SoC NVM memories such as NAND Flash, LPPDR2-NVM or any memory as indicated by SoC inputs pins.

To download all the firmware images into the SoC NVM, public drivers held in ROM that are specific to the SoC may need to be used, for example, USB or UART.

> R010_TBBR_AFM_FLASHING. The SoC **may** implement a Non-Trusted ROM that contains the necessary drivers and discovery mechanisms to download in Non-Trusted SRAM a SoC firmware updater or the complete firmware from external interfaces such as USB, UART, SD-MMC, NAND, NOR or Ethernet.

> R020_TBBR_AFM_FLASHING. The Non-Trusted ROM **must** be able to send through its public interface at least the ROTPK, and SoC_ID and manufacturing mode value where appropriate, to allow the OEM to identify which firmware to provide.

> R030_TBBR_AFM_FLASHING. If a Non-Trusted ROM requires downloading first a firmware updater in Non-Trusted RAM for installing public drivers (for example, the USB stack) then the downloaded updater **may** need to be authenticated by the TEE. In that case, The Non-Trusted ROM must switch back to the Trusted world.

> R040_TBBR_AFM_FLASHING. The firmware updater (ROM or Non-Trusted RAM based) **must** download the complete SoC firmware, which includes SCP (if present), Trusted OS, Non-Trusted world images from an external interface such as USB, UART, SD-MMC, NAND, NOR, Ethernet to SoC NVM memories such as NAND Flash, LPPDR2-NVM or any memory as indicated by SoC inputs pins.

> R050_TBBR_AFM_FLASHING. The firmware updater **must** switch back to the Trusted world for authentication of the downloaded SoC firmware. Failure to do so may be detected by a Trusted Watch Dog timer overflow.

The Non-Trusted world images include the Non-Trusted boot loader. They might also include, for example, the modem software and/or multimedia processor images.

In the instance where the Non-Trusted ROM must first download a firmware updater to install other drivers, then the following requirements also apply:

> R060_TBBR_AFM_FLASHING. The AP firmware **must** authenticate the SoC firmware updater certificate in Trusted RAM using the ROTPK key contained in the certificate. It must verify this key (or the hash of it) to be equal to the eFused ROTPK value.

> R070_TBBR_AFM_FLASHING. The AP firmware **must** calculate the hash of the SoC firmware loader and it must check it is equal to the one contained in the certificate payload.

> R080_TBBR_AFM_FLASHING. The AP firmware updater **may** be protected against rollback.

> R090_TBBR_AFM_FLASHING. If the authentication is successful the AP firmware **must** refresh the Trusted watchdog timer with the value defined in the Non-Trusted Firmware Updater Certificate to ensure the flashing process will have enough time to complete, otherwise use the standard value (i.e. 256s)

> R0100_TBBR_AFM_FLASHING. The AP firmware **must** switch to the Non-Trusted world for executing the SoC firmware loader.

In order to minimize the flashing process time and the execution of the firmware update software, a small set of AP and Power Control functions may be loaded to configure the SoC clocks, power domains and physical interface to provide optimum performances. The following firmware functions are optional:

> R0110_TBBR_AFM_FLASHING. The AP firmware **must** calculate the hash of the SoC AP firmware update configuration and it must check it is equal to the one contained in the certificate before installing it and running it.

> R0120_TBBR_AFM_FLASHING. The AP firmware update configuration **must** be protected against rollback.

> R0130_TBBR_AFM_FLASHING. The AP firmware **must** calculate the hash of the SoC SCP firmware update configuration and it must check it is equal to the one contained in the certificate before installing it and running it.

> R0140_TBBR_AFM_FLASHING. If present, the SoC SCP firmware update configuration **must** be protected against rollback.

3.5.3   Firmware Table of Contents (ToC)

In order to find and download from the SoC NVM the different firmware images that are needed to continue the Trusted boot process, a firmware ToC is required.

R010_TBBR_TOC. The ToC **may** be authenticated (in the next table it is not).

R020_TBBR_TOC. The ToC **may** be encrypted (in the next table it is not)

R030_TBBR_TOC. The ToC **may** be roll-back protected (in the next table it is not).

R040_TBBR_TOC. The ToC **must** indicate which certificate/content is encrypted and with which key.

R050_TBBR_TOC. The ToC **may** be compliant with the following template:

Table 3: Table of Content Fields

ToC Field	Size (bits)	Description
ToC Identifier	32	This may be a 4 byte ASCII name to identify the table of content. If the the bytes are all 0, the ToC is invalid
ToC Serial Number	32	The serial number of this ToC
Firmware Encryption Status	2	0b00: No encryption used 0b01: Encryption is done with SSK 0b10: Encryption is done with BSSK 0b11: Reserved for future use
Reserved	62	Reserved for future use
<Template for Key/Content Certificate or Path/Firmware Image of TOC Entry>		
Name	128	UUID of the <Key/Content Certificate or Path/Firmware Image> Must be 0x0 for the last ToC entry. UUID must be calculated as described in 0 i.e. ca2dfdbe-0bad-48a0-a952-cf84f8ad5bb8 = FirmwareUpdaterCertificate 7c6fbab2-da36-4459-8f91-cb3caa2a0b02 = APFirmwareUpdaterConfig
Offset Address	64	Offset to be added to ToC base address
Size	64	Size of the material
Encryption	1	0b0: Not Encrypted / 0b1: Encrypted
Reserved	63	Reserved for future use

R060_TBBR_TOC.The SoC TOC **must** contain at least the following items:

The Trusted Key certificate.

The Trusted SoC certificate and firmware.

The Trusted OS certificate and firmware.

The Non-Trusted world bootloader certificate and firmware.

R070_TBBR_TOC. The SoC TOC **may** contain the following items:

The Trusted SCP certificate and firmware.

The Trusted Primary Debug certificate.

The Trusted Secondary debug certificate.

R080_TBBR_TOC. The ToC **must** be located in the NVM memory at a fixed address or block location (ROM'ed or tied by hardware in a readable register by the SoC firmware).

In some implementations, it is expected that the Trusted Boot software will parse the TOC file by TOC Entry Name (i.e. FirmwareUpdaterCertificate) to find the key/content certificate and patch, or the firmware image required for each step.

3.5.4 AP Trusted Boot Firmware Trusted RAM based installation

To minimize the ROM dependencies and risk associated with ROM patching;

R010_TBBR_AFM_TRUSTED_RAM. A Trusted RAM based Trusted Boot Firmware **may** be used instead of stored in ROM. This firmware contains the next steps of the Trusted Boot.

If the Trusted Boot is not completely implemented in the ROM and consequently a part of it need to be downloaded and executed from the Trusted RAM then the following requirements apply;

R020_TBBR_AFM_TRUSTED_RAM. The AP firmware **must** download from the SoC NVM the Trusted Boot Firmware certificate and image into Trusted SRAM.

R030_TBBR_AFM_TRUSTED_RAM. The AP firmware **must** authenticate the Trusted Boot firmware certificate in Trusted RAM using the ROTPK key contained in the certificate. It must verify this key (or the hash of it) to be equal to the eFused ROTPK value.

R040_TBBR_AFM_TRUSTED_RAM. The AP firmware **must** calculate the hash of the Trusted Boot firmware image and it must check it is equal to the one contained in the certificate.

R050_TBBR_AFM_TRUSTED_RAM. The AP firmware **must** check that the value of TrustedFirmwareNVcounter read in the certificate is equal or higher than the one stored in SoC NVM, for rollback protection. If the checks fail the device must be reset by the Trusted Watchdog Timer as described in chapter 3.5.7.

R060_TBBR_AFM_TRUSTED_RAM. If the value of TrustedFirmwareNVcounter read in the certificate is higher compared to the value stored in SoC NVM then the SoC NVM counter **must** be incremented accordingly.

As soon as the integrity checking of the firmware is successfully performed, the Trusted Boot code firmware can be executed.

3.5.5 AP Trusted Boot Firmware execution and firmware authentication and integrity check

The Trusted Key certificate is used to install subsidiary keys in the Trusted world and to delegates the OEM signing authority to other OEM teams or third party actors. The keys contained in the Trusted key certificate are used to verify later certificates in the chain of trust process.

R011_TBBR_AFM_TRUSTED_RAM The AP firmware must download the Trusted key certificate from the SoC NVM into the Trusted RAM.

R020_TBBR_AFM_TRUSTED_RAM. The Trusted key certificate **must** be verified in Trusted RAM using the ROTPK key contained in the certificate. This key (or hash of it) must be verified to be equal to the eFused ROPTK value. Anti-roll back protection must be performed using the TrustedFirmwareNVcounter. This Trusted key certificate contains the second level public keys (i.e. debug public key, trusted world public key, Non-Trusted world public key) needed to verify all the later certificates of the Trusted Boot.

During normal operation, the SoC AP Firmware is always present in Trusted RAM. It contains the AP ROM patch if implemented, EL3 monitor, OEM or silicon vendor specific functions, OEM patches, and SoC configuration update that can be applied only in the Trusted world. The SoC Firmware can contain OEM functions, such as advanced power management mechanisms, that are desirable to be kept independent of the Trusted OS and always present in the Trusted world.

It is expected that the AP firmware authenticate and install the following software into the Trusted RAM before releasing the AP from the WFI/WFE state (ref. *R060_TBBR_AFM_ BOOTROM):*

> AP ROM patch if implemented
>
> SoC AP firmware
>
> authenticated certificates

R031_TBBR_AFM_TRUSTED_RAM. The AP firmware **must** download from the SoC NVM and into the AP Trusted RAM:

        the SoC firmware key certificate

        the AP ROM patch if present

        the OEM and SoC configuration update

        the SoC AP image

R040_TBBR_AFM_TRUSTED_RAM. The SoC firmware key certificate **must** be verified in AP Trusted RAM using the Trusted world public key contained in the certificate. This key **must** be verified to be equal to the Trusted world public key contained in the Trusted Key certificate. Anti-roll back protection performed using TrustedFirmwareNVcounter.

R051_TBBR_AFM_TRUSTED_RAM. The AP firmware **must** calculate the hash of the AP ROM patch if present, the SoC configuration update and the SoC AP firmware image and it must check they are equal to the hashes contained in the certificate.

R062_TBBR_AFM_TRUSTED_RAM. The SoC configuration update software image **may** also contain a primary OEM application that is independent of the Trusted OS and always present in the Trusted world. This OEM application executing in a privileged state is aimed to implement OEM specific functions such as SoC run-time security configurations and OEM anti-cloning protection mechanism.

The Trusted ROM should contain the minimum functionality, but if it contains more software code than for just the Trusted boot code (for example, if it also includes cryptographic libraries and power management mechanisms), the ROM may need to be patchable. Note, the mechanism to patch the ROM is not described in this document.

R070_TBBR_AFM_TRUSTED_RAM. The AP Trusted ROM patch (when supported) **must** always be resident in the Trusted RAM and must be applied as early as possible during the boot process.

After these stages, the Trusted OS can be initialised:

R080_TBBR_AFM_TRUSTED_RAM. The AP firmware **must** download from the SoC NVM the Trusted OS certificate and image into the Trusted RAM.

R090_TBBR_AFM_TRUSTED_RAM. The Trusted OS certificates **must** be verified in Trusted RAM by the AFM using the Trusted world public key contained in the certificate. This key must be verified to be equal to the Trusted world public key contained in the Trusted Key certificate. Anti-roll back protection must be performed using TrustedFirmwareNVcounter.

R0100_TBBR_AFM_TRUSTED_RAM. The AP firmware **must** calculate the hash of the Trusted OS firmware image and it must check it is equal to the one contained in the certificate.

The Non-Trusted world signed bootloader is installed as follow:

R01 0_TBBR_AFM_NON-TRUSTED_RAM. The AP firmware **may** download from the SoC NVM the Non-Trusted world certificate and the Non-Trusted world Bootloader into the Non-Trusted RAM.

R02 0_TBBR_AFM_NON-TRUSTED_RAM. The Non-Trusted world Bootloader certificate **must** be verified by the AP firmware using the Non-Trusted world public key contained in the certificate. This key **must** be verified to be equal to the Non-Trusted world public key contained in the Trusted Key certificate. Anti-roll back protection **must** be performed using NonTrustedFirmwareNVcounter.

R03 0_TBBR_AFM_NON-TRUSTED_RAM. The AP firmware **must** calculate the hash of the Non-Trusted world Bootloader and check it is equal to the ones contained in the certificate.

3.5.6 **SW image execution requirements**

The execution of the different firmware needs to be performed in a specific ordered sequence, and must comply with the following:

R010_TBBR_FIRMWARE_EXEC. The firmware of the SCP, when present, **may** be first run to setup the SoC which can then be followed by the loading or execution of:

1. Trusted Boot Firmware

2. SoC configuration update

3. SCP firmware

4. AP ROM patch (if used)

5. AP firmware.

Once done, the AP Trusted OS is executed and starts the Non-Trusted world boot loader which may boot the Rich operating system or a hypervisor.

The Power Control Function is responsible for the setup of the SoC operating point and therefore the AP Trusted Boot Process must wait until the completion of this step before continuing its operations.

R020_TBBR_FIRMWARE_EXEC. The AP **must** wait for the Power Control Function to setup the clocks, power and complete any necessary resets before continuing.

The Trusted Boot firmware is responsible for initialising the Trusted OS which must be started before the Non-Trusted world bootloader.

R030_TBBR_FIRMWARE_EXEC. The Trusted OS **may** start the Non-Trusted boot loader.

The Non-Trusted world bootloader may load a hypervisor or alternatively the Rich operating system.

R040_TBBR_FIRMWARE_EXEC. The Non-Trusted boot loader **may** be used for all booting steps of a hypervisor and its guest operating systems, or the main operating system.

R050_TBBR_FIRMWARE_EXEC. The Non-Trusted world firmware images **may** be copied from NVM to the Non-Trusted RAM

R060_TBBR_FIRMWARE_EXEC. The Non-Trusted world firmware images **may** be authenticated and integrity checked in Non-Trusted RAM.

R070_TBBR_FIRMWARE_EXEC. The Non-Trusted world firmware images **may** be protected against rollback using the NonTrustedFirmwareNVcounter or counters stored by the Trusted OS in the Trusted OS Secure Storage (and eMMC RPMB).

3.5.7 **Trusted Boot protection and Firmware binding**

If an error is detected during the Trusted Boot Process, the Trusted watchdog timer is not refreshed and the firmware ToC is made invalid (by overwriting the ToC structure with zeros). This ensures that the AP will perform a warm reset due to the Trusted watchdog timer and that a firmware update request will be made (recovery/update mode).

R010_TBBR_ PROTECTION. The Trusted watchdog timer **must** not be refreshed if:

1. a certificate is not successfully authenticated or,

2. the integrity of a software image has not been successfully verified or,

3. the anti-roll back counter is lower than the counter stored in SoC NVM

R020_TBBR_PROTECTION. The Trusted watchdog timer when expired **must** generate a warm reset of the SoC to re-start the Trusted boot procedure and act in a recovery/update mode.

R030_TBBR_PROTECTION. The Trusted watchdog timer **may** be disabled on test and development device and when the secure privilege debug mode is enabled. It **must** be active in other cases at reset with a 256s default value.

R040_TBBR_PROTECTION. The ToC **may** be overwritten with zeros in order to re-start the Trusted Boot procedure and act as a recovery/update mode when the boot fails repeatedly because:

1. a certificate is not successfully authenticated or,

2. the integrity of a SW image has not been successfully verified or,

3. the value of the anti-roll back counter is lower than the value of the counter stored in SoC NVM

4. the SoC_ID provided by the certificate does not correspond to the device (except for Trusted debug Certificates which are silently ignored)

Optionally, the certificates may be protected against cloning by binding the SoC firmware certificates and firmware images uniquely per SoC.

R050_TBBR_PROTECTION. The SoC **may** be protected against software cloning by binding the certificates and software images specifically to the SoC. The binding is made by encrypting the certificate with SSK or using a unique per device key called the Binding Secret Symmetric Key (BSSK).

R060_TBBR_PROTECTION. The BSSK **must** be created using AES_{HUK}(Fixed pattern) to be reproducible after cold boot without NVM.

R070_TBBR_PROTECTION. The Trusted boot firmware **may** do the binding of software image updates at run-time by decrypting the updated SoC certificates and software images using the OTP/Fuse Secret Symmetric Key (SSK), followed by the re-encrypting these SoC certificates and software images using a reproducible secret unique per device symmetric key (BSSK), and then updating the ToC correspondingly.

R080_TBBR_PROTECTION. The binding of software updates **must** be carried out only after a successful firmware update process as a secondary step.

3.5.8 Trusted Debug

In order to re-enable the debug on mass production device, the following requirements must be met:

R010_TBBR_DEBUG. Trusted debug certificates **may** be included in the SoC firmware image or be optionally downloaded using test and diagnostic interfaces, such as JTAG (IEEE1149.x) or TEST (IEEE1500) interfaces in trusted RAM. They can be processed by the processor itself or using a dedicated hardware subsystem such as the Authentication debug Module (ADM).

R020_TBBR_DEBUG. The trusted debug certificates **must** be composed of a primary certificate and a secondary certificate. A device may contain no certificates, a primary certificate or both a primary and secondary certificate. Where no valid certificates are present, no debug scenario can be available.

R030_TBBR_DEBUG. The primary Trusted debug certificate **may** be signed for a class of device, and must restrict what the capabilities of the secondary certificate.

R040_TBBR_DEBUG. The secondary Trusted debug certificate **may** contain the SoC_ID of the device to ensure it is unique per device.

R050_TBBR_DEBUG. The primary and secondary Trusted debug certificate **may** allow configuration of the following invasive and non-invasive debug scenarios:

1. No debug
2. Public debug (Self-hosted debug only)
3. Public debug
4. Secure privilege debug

When the Public debug mode is set the Trusted world is executable but not debuggable.

Debug scenarios and and usage models are described in in reference **Error! Reference source not found.**.

R060_TBBR_DEBUG. The primary Trusted debug certificate **must** be verified using the root of trust public key (ROTPK) of which the hash or the full length value is stored in OTP/Fuse or using a subsidiary debug public key provided in the Trusted key certificate.

R070_TBBR_DEBUG. The secondary Trusted debug certificate **may** be verified using a subsidiary public key provided in the key certificate.

R080_TBBR_DEBUG. If the trusted debug certificate is not included in the SoC firmware, the SoC **must** keep the hardware default configuration given by the manufacturing mode (MM):

1. secure privilege debug enabled for test and development device
2. public debug (Self-hosted debug only) for mass production device.

R090_TBBR_DEBUG. The Trusted debug certificate **may** optionally be protected against rollback.

R0100_TBBR_DEBUG. If the Trusted debug certificate verification is not successful, this error is silently ignored, the hardware default debug configuration given by the Manufacturing Mode is kept and the Trusted boot process continues without interruption.

R0110_TBBR_DEBUG. The Trusted watchdog timer **may** be suspended if necessary such as when the processor is halted in debug on a uniprocessor system, or when the secure privilege debug is configured. The Trusted watchdog timer must be running after cold reset with a default value of 256s.

R0120_TBBR_DEBUG. The primary and secondary debug certificates **may** allow the configuration of the JTAG taps of all the platform subsystems (i.e. the Modem, DSP).

R0130_TBBR_DEBUG. Booting on external XIP memory to replace and **debug** the Boot ROM software must be possible only on development and test device, and un-initialize device.

R0140_TBBR_DEBUG. The SoC_ID **must** be calculated with SHA256(ROTPK || AES_{HUK}(Fixed Pattern)).

R0150_TBBR_DEBUG. When the Trusted debug certificates are processed the register used to configure at SoC level the global debug mode **must** be locked until next cold reset.

As an example, the OEM could sign a primary debug certificate which allows only the enabling of 'public debug' and some spare bits (assigned to modem debug for example) on a class of device without knowing which device is to be debugged. The third party could then select accordingly to its need which particular device is publicly debuggable(e.g. development) or not debuggable (beta testing).

Another example would be to have the OEM signing a primary debug certificate that allows enabling the secure privilege debug to the Trusted OS owner. This processing can be done for whole lifecycle of the device without the OEM intervention.

3.6 Certificate Structure and Content

This section describes the structure and content of certificates.

3.6.1 Non-Trusted Firmware Updater certificate

The root of trust public key (ROTPK) is used to authenticate the Non-Trusted Firmware Updater certificate, the Trusted Boot Firmware certificate and the Trusted Key certificate.

This key is contained in each certificate as per X.509 standard, and this key (or hash of it) must be equal to eFused ROTPK value contained in the SoC (provisioned at manufacture) for verifying that the certificate has truly been issued by the OEM.

This certificate does not need to be protected against replay attacks.

In order to speed up the flashing process a small AP and SCP firmware, (if present) may be loaded to configure the SoC clocks, power domains and physical interface to provide optimum performances.

The certificate optionally contains a value (in seconds) specifying the maximum time firmware update process should take.

Table 4 explains the X.509 v3 extension fields required and their content:

Table 4 Non-Trusted Firmware Updater Certificate X.509 fields

Version of the certificate (v3=2)
Certificate Serial Number (Integer)
Certificate Algorithm Identifier for certificate Issuer's Signature =1.2.840.10045.4.3.2 (ecdsa-with-SHA256 as defined in RFC 5758 0)
Certificate Issuer name = CN=FirmwareUpdaterCertificate
Validity Period
Subject Name = CN=FirmwareUpdaterCertificate

Subject Public Key Information:		
AlgorithmIdentifier:algorithm = 1.2.840.10045.2.1 (ecPublicKey)		
AlgorithmIdentifier:parameters = 1.2.840.10045.3.1.7 (secp256r1 as defined in SECG 0)		
SubjectPublicKey = ROTPK 528 bits (First byte is 0x00 for number of unused bits in last octet; second byte is 0x04 for uncompressed format. Followed by 256bit x coordinate; 256bit y coordinate)		

Certificate extension

Extension Name	Criticality	Value
APFirmwareUpdaterConfigHash	=0	SHA256 (APFirmwareUpdaterConfig)
SCPFirmwareUpdaterConfigHash	=0	SHA256 (SCPFirmwareUpdaterConfig)
FirmwareUpdaterHash	=0	SHA256 (FirmwareUpdater)
TrustedWatchdogRefreshTime	=0	(16 bits)

Signature:

Signature Algorithm Identifier =1.2.840.10045.4.3.2 (ecdsa-with-SHA256 as defined in RFC 5758 0)
Signature value = 256Bits for EC-DSA

3.6.2 Trusted Boot Firmware Certificate

The Trusted Boot Firmware contains the software code that executes in the Trusted Execution Environment to perform the Trusted Boot process.

Table 5 explains the X.509 v3 extension fields required and their content:

Table 5 Trusted Boot Firmware Certificate X.509 fields

Version of the certificate (v3=2)
Certificate Serial Number (Integer)
Certificate Algorithm Identifier for certificate Issuer's Signature =1.2.840.10045.4.3.2 (ecdsa-with-SHA256 as defined in RFC 5758 0)
Certificate Issuer name = CN=TrustedBootFirmwareCertificate
Validity Period
Subject Name = CN=TrustedBootFirmwareCertificate

Subject Public Key Information:		
AlgorithmIdentifier:algorithm = 1.2.840.10045.2.1 (ecPublicKey)		
AlgorithmIdentifier:parameters = 1.2.840.10045.3.1.7 (secp256r1 as defined in SECG 0)		
SubjectPublicKey = ROTPK 528 bits (First byte is 0x00 for number of unused bits in last octet; second byte is 0x04 for uncompressed format. Followed by 256bit x coordinate; 256bit y coordinate)		

Certificate extension

Extension Name	Criticality	Value
TrustedFirmwareNVCounter	=1	(5 bits)
TrustedBootFirmwareHash	=1	SHA256 (Trusted Boot Firmware)

Signature:
Signature Algorithm Identifier =1.2.840.10045.4.3.2 (ecdsa-with-SHA256 as defined in RFC 5758 0)
Signature value = 256Bits for EC-DSA

3.6.3 Trusted Key Certificate

The public keys defined in the Trusted Key certificate are used to verify the later certificates in the chain of trust process. The corresponding private keys are still owned by the OEM but for revocation capability the Trusted Key Certificate acts as an authority transfer to OEM product teams.

The public keys in the extension fields (PrimaryDebugCertificatePK, TrustedWorldPK, NonTrustedWorldPK) are full size keys and not the hash of them. This choice has been made to reduce the processing time by avoiding extra hash calculations.

Table 6 explains the encoding of the key certificate based on the X.509 v3 certificate:

Table 6 Trusted Key certificate X.509 fields

Version of the certificate (v3=2)		
Certificate Serial Number (Integer)		
Certificate Algorithm Identifier for certificate Issuer's Signature =1.2.840.10045.4.3.2 (ecdsa-with-SHA256 as defined in RFC 5758 0)		
Certificate Issuer name = CN=TrustedKeyCertificate		
Validity Period		
Subject Name = CN=TrustedKeyCertificate		
Subject Public Key Information: **AlgorithmIdentifier:algorithm = 1.2.840.10045.2.1 (ecPublicKey)** **AlgorithmIdentifier:parameters = 1.2.840.10045.3.1.7 (secp256r1 as defined in SECG 0)** **SubjectPublicKey = ROTPK** 528 bits (First byte is 0x00 for number of unused bits in last octet; second byte is 0x04 for uncompressed format. Followed by 256bit x coordinate; 256bit y coordinate)		
Certificate extension		

Extension Name	Criticality	Value
TrustedFirmwareNVCounter	=1	(5 bits)
PrimaryDebugCertificatePK	=1	(512 bits for EC-DSA) // uncompressed
TrustedWorldPK	=1	(512 bits for EC-DSA)
NonTrustedWorldPK	=1	(512 bits for EC-DSA)

Signature:
Signature Algorithm Identifier =1.2.840.10045.4.3.2 (ecdsa-with-SHA256 as defined in RFC 5758 0)
Signature value = 256Bits for EC-DSA

3.6.4 Trusted Debug Certificates

The debug scenarios are enabled though the chaining of two certificates;

a primary debug certificate which is signed for a class of device

a secondary debug certificate which is unique per device.

The primary certificate is owned by the OEM and its purpose is to specify what scenarios are allowed to be configured by the secondary certificate that is signed by a third party. It contains a spare field of 64 bits and the public key needed to verify the secondary debug certificate (SecondaryDebugCertPK).

Table 7 explains the X.509 v3 extension fields required and their content:

Table 7 Primary debug certificate X.509 fields

Version of the certificate (v3=2)		
Certificate Serial Number (Integer)		
Certificate Algorithm Identifier for certificate Issuer's Signature =1.2.840.10045.4.3.2 (ecdsa-with-SHA256 as defined in RFC 5758 0)		
Certificate Issuer name = CN=PrimaryDebugCertificate		
Validity Period		
Subject Name = CN=PrimaryDebugCertificate		
Subject Public Key Information:		

AlgorithmIdentifier:algorithm = 1.2.840.10045.2.1 (ecPublicKey)
AlgorithmIdentifier:parameters = 1.2.840.10045.3.1.7 (secp256r1 as defined in SECG 0)
SubjectPublicKey = PrimaryDebugCertificatePK 528 bits (First byte is 0x00 for number of unused bits in last octet; second byte is 0x04 for uncompressed format. Followed by 256bit x coordinate; 256bit y coordinate)

Certificate extension

Extension Name	Criticality	Value
DebugScenario	=1	0b000= No Debug 0b001= Public Debug (Self-hosted debug only) 0b010= Public Debug *0b011= Secure User Debug (deprecated)* 0b100= Secure Privilege Debug
SoC Specific	=0	(64bits)
SecondaryDebugCertPK	=1	(512 bits for EC-DSA) // uncompressed

Signature:

Signature Algorithm Identifier =1.2.840.10045.4.3.2 (ecdsa-with-SHA256 as defined in RFC 5758 0)
Signature value = 256Bits for EC-DSA

The secondary debug certificate purpose is to select which debug scenario to enable on the platform and to specify the unique identifier per device (SoC_ID).

Table 8 explains the X.509 v3 extension fields required and their content:

Table 8 Secondary debug certificate X.509 fields

Version of the certificate (v3=2)
Certificate Serial Number (Integer)
Certificate Algorithm Identifier for certificate Issuer's Signature =1.2.840.10045.4.3.2 (ecdsa-with-SHA256 as defined in RFC 5758 0)
Certificate Issuer name = CN=SecondaryDebugCertificate
Validity Period

Subject Name = CN=SecondaryDebugCertificate		

Subject Public Key Information:		
AlgorithmIdentifier:algorithm = 1.2.840.10045.2.1 (ecPublicKey)		
AlgorithmIdentifier:parameters = 1.2.840.10045.3.1.7 (secp256r1 as defined in SECG 0)		
SubjectPublicKey = SecondaryDebugCertPK 528 bits (First byte is 0x00 for number of unused bits in last octet; second byte is 0x04 for uncompressed format. Followed by 256bit x coordinate; 256bit y coordinate)		

Certificate extension

Extension Name	Criticality	Value
DebugScenario	=1	0b000= No Debug 0b001= Public Debug (Self-hosted debug only) 0b010= Public Debug *0b011= Secure User Debug (deprecated)* 0b100= Secure Privilege Debug
SoC_ID	=0	SHA256(ROTPK || AESHUK(Fixed pattern))
SoC Specific	=0	(64 bits)

Signature:

Signature Algorithm Identifier =1.2.840.10045.4.3.2 (ecdsa-with-SHA256 as defined in RFC 5758 0)
Signature value = 256Bits for EC-DSA

The SoC Specific extension fields of the Primary and Secondary debug certificate are not defined in this document and may contain no security related information. The Primary SoC Specific extension defines which bits of the Secondary SoC Specific field are allowed to be set following the same model as the debug scenarios.

3.6.5 **Trusted SoC firmware certificates**

The Trusted SoC firmware certificates are composed of two certificates; the SoC firmware key certificate which contains the Public key (SoCFirmwareContentCertPK) needed to verify the content certificate that contains the hash of the Application processor (AP) ROM patch and the hash of the SoC configuration update.

Note: SoC configuration update may be used for Cortex-A errata corrections (these may not be security related but the correction may be applied only using secure accesses).

The SoC firmware key certificate purpose is to optionally transfer the OEM signing authority to a third party company.

Table 9 explains the X.509 v3 extension fields required and their content:

Table 9 SoC Firmware Key Certificate X.509 meaningful fields

Version of the certificate (v3=2)
Certificate Serial Number (Integer)
Certificate Algorithm Identifier for certificate Issuer's Signature =1.2.840.10045.4.3.2 (ecdsa-with-SHA256 as defined in RFC 5758 0)
Certificate Issuer name = CN=SoCFirmwareKeyCertificate
Validity Period
Subject Name = CN=SoCFirmwareKeyCertificate

Subject Public Key Information:
AlgorithmIdentifier:algorithm = 1.2.840.10045.2.1 (ecPublicKey)
AlgorithmIdentifier:parameters = 1.2.840.10045.3.1.7 (secp256r1 as defined in SECG 0)
SubjectPublicKey = TrustedWorldPK 528 bits (First byte is 0x00 for number of unused bits in last octet; second byte is 0x04 for uncompressed format. Followed by 256bit x coordinate; 256bit y coordinate)

Certificate extension		
Extension Name	**Criticality**	Value
TrustedFirmwareNVCounter	=1	(5 bits)
SoCFirmwareContentCertPK	=1	(512 bits for EC-DSA)

Signature:
Signature Algorithm Identifier =1.2.840.10045.4.3.2 (ecdsa-with-SHA256 as defined in RFC 5758 0)
Signature value = 256Bits for EC-DSA

Table 10 explains the X.509 v3 extension fields required and their content:

Table 10 SoC Firmware Content Certificate X.509 meaningful fields

Version of the certificate (v3=2)
Certificate Serial Number (Integer)
Certificate Algorithm Identifier for certificate Issuer's Signature =1.2.840.10045.4.3.2 (ecdsa-with-SHA256 as defined in RFC 5758 0)
Certificate Issuer name = CN=SoCFirmwareContentCertificate
Validity Period
Subject Name = CN=SoCFirmwareContentCertificate

Subject Public Key Information:
AlgorithmIdentifier:algorithm = 1.2.840.10045.2.1 (ecPublicKey)
AlgorithmIdentifier:parameters = 1.2.840.10045.3.1.7 (secp256r1 as defined in SECG 0)
SubjectPublicKey = SoCFirmwareContentCertPK 528 bits (First byte is 0x00 for number of unused bits in last octet; second byte is 0x04 for uncompressed format. Followed by 256bit x coordinate; 256bit y coordinate)

Certificate extension		
Extension Name	**Criticality**	Value
TrustedFirmwareNVCounter	=1	(5 bits)
APRomPatchHash	=0	SHA256(AP ROM Patch Payload)
SoCConfigHash	=1	SHA256(SoC configuration update)
SoCAPFirmwareHash	=1	SHA256(SoC AP Firmware)

Signature:
Signature Algorithm Identifier =1.2.840.10045.4.3.2 (ecdsa-with-SHA256 as defined in RFC 5758 0)
Signature value = 256Bits for EC-DSA

3.6.6   **Trusted OS firmware certificates**

The Trusted OS firmware certificates are composed of two certificates; the Trusted OS key certificate which contains the Public key (TrustedOSFirmwareContentCertPK) needed to verify the Trusted OS content certificate. The later certificate contains the hash of Trusted OS firmware for integrity checking. The Trusted OS firmware key certificate purpose is to transfer the OEM signing authority optionally to a third party company.

Table 11 explains the X.509 v3 extension fields required and their content:

Table 11 Trusted OS Firmware Key Certificate X.509 fields

Version of the certificate (v3=2)		
Certificate Serial Number (Integer)		
Certificate Algorithm Identifier for certificate Issuer's Signature =1.2.840.10045.4.3.2 (ecdsa-with-SHA256 as defined in RFC 5758 0)		
Certificate Issuer name = CN=TrustedOSFirmwareKeyCertificate		
Validity Period		
Subject Name = CN=TrustedOSFirmwareKeyCertificate		
Subject Public Key Information:		
AlgorithmIdentifier:algorithm = 1.2.840.10045.2.1 (ecPublicKey)		
AlgorithmIdentifier:parameters = 1.2.840.10045.3.1.7 (secp256r1 as defined in SECG 0)		
SubjectPublicKey = TrustedWorldPK 528 bits (First byte is 0x00 for number of unused bits in last octet; second byte is 0x04 for uncompressed format. Followed by 256bit x coordinate; 256bit y coordinate)		
Certificate extension		
Extension Name	**Criticality**	Value
TrustedFirmwareNVCounter	=1	(5 bits)
TrustedOSFirmwareContentCertPK	=1	(512 bits for EC-DSA)
Signature:		
Signature Algorithm Identifier =1.2.840.10045.4.3.2 (ecdsa-with-SHA256 as defined in RFC 5758 0)		
Signature value = 256Bits for EC-DSA		

Table 12 explains the X.509 v3 extension fields required and their content:

Table 12 Trusted OS Firmware Content Certificate X.509 fields

Version of the certificate (v3=2)
Certificate Serial Number (Integer)
Certificate Algorithm Identifier for certificate Issuer's Signature =1.2.840.10045.4.3.2 (ecdsa-with-SHA256 as defined in RFC 5758 0)
Certificate Issuer name = CN=TrustedOSFirmwareContentCertificate
Validity Period
Subject Name = CN=TrustedOSFirmwareContentCertificate

Subject Public Key Information:		
AlgorithmIdentifier:algorithm = 1.2.840.10045.2.1 (ecPublicKey)		
AlgorithmIdentifier:parameters = 1.2.840.10045.3.1.7 (secp256r1 as defined in SECG 0)		
SubjectPublicKey = TrustedOSFirmwareContentCertPK 528 bits (First byte is 0x00 for number of unused bits in last octet; second byte is 0x04 for uncompressed format. Followed by 256bit x coordinate; 256bit y coordinate)		

Certificate extension		
Extension Name	**Criticality**	Value
TrustedFirmwareNVCounter	=1	(5 bits)
TrustedOSFirmwareHash	=1	SHA256(Trusted OS firmware)

Signature:	
Signature Algorithm Identifier =1.2.840.10045.4.3.2 (ecdsa-with-SHA256 as defined in RFC 5758 0)	
Signature value = 256Bits for EC-DSA	

3.6.7 Non-Trusted world firmware certificate

The Non-Trusted world firmware certificates are composed of two certificates; the Non-Trusted world firmware key certificate which contains the Public key needed to verify the content certificate that contains the hash of the Non-Trusted world firmware.

The Non-Trusted world firmware key certificate purpose is to transfer the OEM signing authority optionally to a third party company.

Table 13 explains the X.509 v3 extension fields required and their content:

Table 13 Non-Trusted Firmware Key Certificate X.509 fields

Version of the certificate (v3=2)		
Certificate Serial Number (Integer)		
Certificate Algorithm Identifier for certificate Issuer's Signature =1.2.840.10045.4.3.2 (ecdsa-with-SHA256 as defined in RFC 5758 0)		
Certificate Issuer name = CN=NonTrustedFirmwareKeyCertificate		
Validity Period		
Subject Name = CN=NonTrustedFirmwareKeyCertificate		
Subject Public Key Information:		
AlgorithmIdentifier:algorithm = 1.2.840.10045.2.1 (ecPublicKey)		
AlgorithmIdentifier:parameters = 1.2.840.10045.3.1.7 (secp256r1 as defined in SECG 0)		
SubjectPublicKey = NonTrustedWorldPK 528 bits (First byte is 0x00 for number of unused bits in last octet; second byte is 0x04 for uncompressed format. Followed by 256bit x coordinate; 256bit y coordinate)		

Certificate extension		
Extension Name	**Criticality**	Value
NonTrustedFirmwareNVCounter	=1	(8 bits)
NonTrustedFirmwareContentCertPK	=1	(512 bits for EC-DSA)

Signature:
Signature Algorithm Identifier =1.2.840.10045.4.3.2 (ecdsa-with-SHA256 as defined in RFC 5758 0)
Signature value = 256Bits for EC-DSA

If NonTrustedBootFirmwareContentCertPK or the Non-Trusted Boot Firmware Hashes are equal to zero, it means that the Rich OS firmware can be freely replaced on the SoC. When this is done, TBBR recommends that the Primary and secondary debug certificate allow public debug only to guarantee that OEM with TEE enabled on the field are not endangered by cloning.

Table 14 explains the X.509 v3 extension fields required and their content:

Table 14 Non-Trusted Firmware Content Certificate X.509 fields

Version of the certificate (v3=2)		
Certificate Serial Number (Integer)		
Certificate Algorithm Identifier for certificate Issuer's Signature =1.2.840.10045.4.3.2 (ecdsa-with-SHA256 as defined in RFC 5758 0)		
Certificate Issuer name = CN=NonTrustedFirmwareContentCertificate		
Validity Period		
Subject Name = CN=NonTrustedFirmwareContentCertificate		
Subject Public Key Information:		
AlgorithmIdentifier:algorithm = 1.2.840.10045.2.1 (ecPublicKey)		
AlgorithmIdentifier:parameters = 1.2.840.10045.3.1.7 (secp256r1 as defined in SECG 0)		
SubjectPublicKey = NonTrustedFirmwareContentCertPK 528 bits (First byte is 0x00 for number of unused bits in last octet; second byte is 0x04 for uncompressed format. Followed by 256bit x coordinate; 256bit y coordinate)		
Certificate extension		
Extension Name	**Criticality**	Value
NonTrustedFirmwareNVCounter	=1	(8 bits)
NonTrustedWorldBootloaderHash	=1	SHA256(NonTrustedWorldBootloader)
Signature:		
Signature Algorithm Identifier =1.2.840.10045.4.3.2 (ecdsa-with-SHA256 as defined in RFC 5758 0)		
Signature value = 256Bits for EC-DSA		

3.7   Certificate Creation and Key Management

As described above, the trusted boot process requires a series of certificates which have been cryptographically signed.

It's recommended that for the creation of certificates for use in production and development devices, users should consider appropriate processes for the handling, and management of keys used in the signing process, that should consider the physical security and storage of keys, controlled access to those keys, auditing of access and so on.

A description of such a process is outside of the scope of this document, but is a significant security issue that must be considered as part of the implementation of this document.

3.8 Trusted Board Boot Requirements Boundaries mapping

In the following table, Boundary1 (B1) is the minimal feature set defining the invariant in SoC that expect the Trusted Boot, Trusted OS and Trusted Apps developers for payment and financial use cases. Boundary1 is also the recommended feature set for applying to security certification such as the one defined by the GlobalPlatform in 0.

Boundary2 (B2) is the standard feature set capable of providing flexible anti-rollback protection, dynamic trusted debug capability, trusted display, denial-of-service protection, UHD video protection, DRM and enterprise support.

All Boundaries have the same security strength and level of protection and just describe different device functionality. Some requirements do not need to be present to meet either current functional boundaries and may be considered best practice and / or future looking; these are marked as (optional).

Table 15 Trusted Board Boot Requirements Boundaries mapping

Trusted Board Boot Requirement ID	Use Cases	B1	B2
Fundamental Features			
R010_TBBR_FUNCTION.1	UC-ALL	✓	✓
R010_TBBR_FUNCTION.2	UC-ALL	✓	✓
R010_TBBR_FUNCTION.3	UC-ALL	✓	✓
R010_TBBR_FUNCTION.4	UC-ALL	✓	✓
R020_TBBR_FUNCTION.1	(Optional)	·	·
R020_TBBR_FUNCTION.2	(Optional)	·	·
R020_TBBR_FUNCTION.3	(Optional)	·	·
R020_TBBR_FUNCTION.4	(Optional)	·	·
R030_TBBR_FUNCTION	UC-ALL	✓	✓
R040_TBBR_FUNCTION	(Optional)	·	·
R050_TBBR_FUNCTION	UC-ALL	✓	✓
R060_TBBR_FUNCTION	(Optional)	·	·
R070_TBBR_FUNCTION	(Optional)	·	·
Cryptography			
R010_TBBR_CRYPTO.1	UC-ALL	✓	✓
R010_TBBR_CRYPTO.2	UC-ALL	✓	✓
R010_TBBR_CRYPTO.3	UC-ALL	✓	✓
R020_TBBR_CRYPTO	UC-ALL	✓	✓
R030_TBBR_CRYPTO	(Optional)	·	·
R040_TBBR_CRYPTO	UC-ALL	✓	✓
Boot certificates			
R010_TBBR_CERTIFICATE	(optional)	·	·
R020_TBBR_CERTIFICATE	(optional)	·	·

R030_TBBR_CERTIFICATE	(optional)	·	·
R040_TBBR_CERTIFICATE	(optional)	·	·
R050_TBBR_CERTIFICATE	(optional)	·	·
SoC Cryptographic Keys			
R010_TBBR_KEY_STORAGE	UC-ALL	✓	✓
R020_TBBR_KEY_STORAGE	(Optional)	·	·
Key security robustness property			
R010_TBBR_KEY_REVOCATION	UC-ALL	✓	✓
R020_TBBR_KEY_REVOCATION	UC-ALL	✓	✓
R030_TBBR_KEY_REVOCATION	UC-ALL	✓	✓
R040_TBBR_KEY_REVOCATION	UC-ALL	✓	✓
R050_TBBR_KEY_REVOCATION	UC-ALL	✓	✓
R060_TBBR_KEY_REVOCATION	UC-ALL	✓	✓
SoC Wakeup from cold reset			
R010_TBBR_WAKEUP.1	UC-ALL	✓	✓
R010_TBBR_WAKEUP.2	(optional)	·	·
R020_TBBR_WAKEUP.1	UC-ALL	✓	✓
R020_TBBR_WAKEUP.2	(optional)	·	·
R020_TBBR_WAKEUP.3	(optional)	·	·
TEE configuration and firmware flashing			
R010_TBBR_AFM_BOOTROM	UC-ALL	✓	✓
R020_TBBR_AFM_BOOTROM	UC-ALL	✓	✓
R030_TBBR_AFM_BOOTROM	UC-ALL	✓	✓
R040_TBBR_AFM_BOOTROM	UC-ALL	✓	✓
R050_TBBR_AFM_BOOTROM	UC-ALL	✓	✓
R060_TBBR_AFM_BOOTROM	UC-ALL	✓	✓
R070_TBBR_AFM_BOOTROM	UC-ALL	✓	✓
R080_TBBR_AFM_BOOTROM	UC-ALL	✓	✓
R090_TBBR_AFM_BOOTROM	UC-ALL	✓	✓
R0100_TBBR_AFM_BOOTROM	UC-ALL	✓	✓
R0110_TBBR_AFM_BOOTROM	UC-ALL	✓	✓
R0120_TBBR_AFM_BOOTROM	UC-ALL	✓	✓
R0130_TBBR_AFM_BOOTROM	UC-ALL	✓	✓
R0140_TBBR_AFM_BOOTROM	UC-ALL	✓	✓

AP Non-Trusted firmware updater download and its authentication by the Trusted world

R010_TBBR_AFM_FLASHING	(optional)	·	·
R020_TBBR_AFM_FLASHING	UC-ALL	✓	✓
R030_TBBR_AFM_FLASHING	(optional)	·	·
R040_TBBR_AFM_FLASHING	UC-ALL	✓	✓
R050_TBBR_AFM_FLASHING	UC-ALL	✓	✓
R060_TBBR_AFM_FLASHING	(optional)	·	·
R070_TBBR_AFM_FLASHING	(optional)	·	·
R080_TBBR_AFM_FLASHING	(optional)	·	·
R090_TBBR_AFM_FLASHING	(optional)	·	·
R0100_TBBR_AFM_FLASHING	(optional)	·	·
R0110_TBBR_AFM_FLASHING	UC-ALL	✓	✓
R0120_TBBR_AFM_FLASHING	UC-ALL	✓	✓
R0130_TBBR_AFM_FLASHING	UC-ALL	✓	✓
R0140_TBBR_AFM_FLASHING	UC-ALL	✓	✓

Firmware Table of Content (ToC)

R010_TBBR_TOC	(optional)	·	·
R020_TBBR_TOC	(optional)	·	·
R030_TBBR_TOC	(optional)	·	·
R040_TBBR_TOC	UC-ALL	✓	✓
R050_TBBR_TOC	(optional)	·	·
R060_TBBR_TOC.1	UC-ALL	✓	✓
R060_TBBR_TOC.2	UC-ALL	✓	✓
R060_TBBR_TOC.3	UC-ALL	✓	✓
R060_TBBR_TOC.4	UC-ALL	✓	✓
R070_TBBR_TOC.1	(optional)	·	·
R070_TBBR_TOC.2	(optional)	·	·
R070_TBBR_TOC.3	(optional)	·	·
R080_TBBR_TOC	UC-ALL	✓	✓

AP Trusted Boot Firmware Trusted RAM based installation

R010_TBBR_AFM_TRUSTED_RAM	(optional)	·	·
R020_TBBR_AFM_TRUSTED_RAM	UC-ALL	✓	✓
R030_TBBR_AFM_TRUSTED_RAM	UC-ALL	✓	✓
R040_TBBR_AFM_TRUSTED_RAM	UC-ALL	✓	✓

R050_TBBR_AFM_TRUSTED_RAM	UC-ALL	✓	✓
R060_TBBR_AFM_TRUSTED_RAM	UC-ALL	✓	✓
AP Trusted Boot Firmware execution and firmware authentication and integrity check			
R011_TBBR_AFM_TRUSTED_RAM	UC-ALL	✓	✓
R030_TBBR_AFM_TRUSTED_RAM	UC-ALL	✓	✓
R031_TBBR_AFM_TRUSTED_RAM	UC-ALL	✓	✓
R040_TBBR_AFM_TRUSTED_RAM	UC-ALL	✓	✓
R041_TBBR_AFM_TRUSTED_RAM	UC-ALL	✓	✓
R042_TBBR_AFM_TRUSTED_RAM	(optional)	·	·
R070_TBBR_AFM_TRUSTED_RAM	UC-ALL	✓	✓
R080_TBBR_AFM_TRUSTED_RAM	UC-ALL	✓	✓
R090_TBBR_AFM_TRUSTED_RAM	UC-ALL	✓	✓
R0100_TBBR_AFM_TRUSTED_RAM	UC-ALL	✓	✓
R010_TBBR_AFM_NON-TRUSTED_RAM	(optional)	·	·
R020_TBBR_AFM_NON-TRUSTED_RAM	UC-ALL	✓	✓
R030_TBBR_AFM_NON-TRUSTED_RAM	UC-ALL	✓	✓
SW image execution requirements			
R010_TBBR_FIRMWARE_EXEC	(optional)	·	·
R020_TBBR_FIRMWARE_EXEC	UC-ALL	✓	✓
R030_TBBR_FIRMWARE_EXEC	(optional)	·	·
R040_TBBR_FIRMWARE_EXEC	(optional)	·	·
R050_TBBR_FIRMWARE_EXEC	(optional)	·	·
R060_TBBR_FIRMWARE_EXEC	(optional)	·	·
R070_TBBR_FIRMWARE_EXEC	(optional)	·	·
Trusted Boot protection and Firmware binding			
R010_TBBR_ PROTECTION	UC-ALL	✓	✓
R020_TBBR_ PROTECTION	UC-ALL	✓	✓
R030_TBBR_ PROTECTION	(optional)	·	·
R040_TBBR_ PROTECTION	(optional)	·	·
R050_TBBR_ PROTECTION	(optional)	·	·
R060_TBBR_ PROTECTION	(optional)	·	·
R070_TBBR_ PROTECTION	(optional)	·	·
R080_TBBR_ PROTECTION	(optional)	·	·
Trusted Debug			
R010_TBBR_DEBUG	(optional)	·	·
R020_TBBR_DEBUG	(optional)	·	·

R030_TBBR_DEBUG	(optional)	·	·
R040_TBBR_DEBUG	(optional)	·	·
R040_TBBR_DEBUG	(optional)	·	·
R060_TBBR_DEBUG	(optional)	·	·
R070_TBBR_DEBUG	(optional)	·	·
R080_TBBR_DEBUG	(optional)	·	·
R090_TBBR_DEBUG	(optional)	·	·
R0100_TBBR_DEBUG	(optional)	·	·
R0110_TBBR_DEBUG	(optional)	·	·
R0120_TBBR_DEBUG	(optional)	·	·
R0130_TBBR_DEBUG	(optional)	·	·
R0140_TBBR_DEBUG	(optional)	·	·
R0150_TBBR_DEBUG	(optional)	·	·
SCP - **Fundamental Features**			
R010_TBBR_SCP_FUNCTION	(optional)	·	·
R020_TBBR_SCP_FUNCTION	(optional)	·	·
SCP Requirements - **First Stage Boot**			
R010_TBBR_SCP_ BOOTROM	(optional)	·	·
R020_TBBR_SCP_ BOOTROM	(optional)	·	·
SCP Requirements - Firmware Execution			
R030_TBBR_SCP_ TRUSTED_RAM	(optional)	·	·
R040_TBBR_SCP_ TRUSTED_RAM	(optional)	·	·
R050_TBBR_SCP_ TRUSTED_RAM	(optional)	·	·
R060_TBBR_SCP_ TRUSTED_RAM	(optional)	·	·
R070_TBBR_SCP_ TRUSTED_RAM	(optional)	·	·
END			

Table 16 Trusted Board Boot feature Boundaries

Field	Description
·	Support for this requirement is not required to meet this functional Boundary.
Y or ✔	This is a requirement of this functional boundary
No	This solution is NOT acceptable in systems at this functional boundary
(<comment>)	See requirement for explanation

APPENDIX A   SYSTEM CONTROL PROCESSOR

The System Control Processor (SCP) is a trusted on-chip MCU that performs power management functions, and has responsibility for power management of the SoC. The SCP controls clocks and reset of each core, and is responsible for power state transitions for the power regions on the SoC.

This section describes the additional requirements necessary if the SoC design incorporates a SCP as part of its design, to take the role of the Power Control Function.

A.1   SCP Requirements

A.1.1   Fundamental Features

As a Trusted element of the system, the SCP must provide security equal to any other part of the Trusted world in the SoC design.

R010_TBBR_SCP_FUNCTION. The SCP Boot ROM **may** be patchable.

R020_TBBR_SCP_FUNCTION. In order to provide key revocation capability in the Trusted boot process, the certificate of the SCP Trusted OS **must** be signed by a different private/public keys pair and must contain the TrustedFirmwareNvCounter.

A.1.2   First Stage Boot

The following requirements apply to the first stage of the boot

R010_TBBR_SCP_BOOTROM. The SCP **must** configure the Cortex-A series processor clocks and reset, power domain as well as any mechanism to ensure the AP boot on its Trusted ROM.

R020_TBBR_SCP_BOOTROM. The SCP, when present, **must** initiate a Trusted channel with the AP, such as using a Trusted mailbox.

A.1.3   Firmware Execution

R030_TBBR_SCP_TRUSTED_RAM. The SCP ROM patch (when supported) **must** always be resident in the Trusted RAM and must be applied as early as possible during the boot process.

The SCP Trusted firmware is always present in SCP Trusted RAM. It will contain the software responsible for the system and power management of the SoC.

The SoC architecture can be made in such a way that the SCP itself authenticates its firmware, but it is also possible to have the AP authenticate the SCP firmware.

In both cases, the AP can upload the SCP Trusted RAM with the firmware or the AP can request through a mailbox the SCP to get its firmware from the NVM or the AP Trusted RAM.

R040_TBBR_ SCP_TRUSTED_RAM. The AP or alternatively, the SCP firmware **must** download from the SoC NVM the SCP firmware key certificate, ROM patches (when supported) and SCP firmware image into SCP Trusted RAM.

R050_TBBR_ SCP_TRUSTED_RAM. The SCP firmware key certificate **must** be verified in SCP Trusted RAM by the SCP or AP using the Trusted world public key contained in the certificate. This key **must** be verified to be equal to the Trusted world public key contained in the Trusted Key certificate. Anti-roll back protection must be performed using TrustedFirmwareNVcounter.

R060_TBBR_ SCP_TRUSTED_RAM. The firmware **must** calculate the hash of the SCP ROM patch if present and SCP Firmware image and it must check they are equal to the ones contained in the certificate.

R070_TBBR_ SCP_TRUSTED_RAM. The AP **must** synchronize with the SCP to download or make the SCP upload the software image when the SCP is ready to boot its firmware. This synchronization **may** be carried out using a Trusted mailbox.

A.2   Trusted SCP firmware certificates

The SCP firmware certificates are composed of two certificates; the SCP firmware key certificate which contains the Public key (SCPFirmwareContentCertPK) needed to verify the SCP firmware content certificate that contains the hash of the SCP ROM patch and the hash of SCP firmware. The SCP firmware key certificate purpose is to transfer the OEM signing authority optionally to a third-party company.

Table 17 explains the X.509 v3 extension fields required and their content:

Table 17 SCP Firmware Key Certificate X.509 fields

Version of the certificate (v3=2)		
Certificate Serial Number (Integer)		
Certificate Algorithm Identifier for certificate Issuer's Signature =1.2.840.10045.4.3.2 (ecdsa-with-SHA256 as defined in RFC 5758 0)		
Certificate Issuer name = CN=SCPFirmwareKeyCertificate		
Validity Period		
Subject Name = CN=SCPFirmwareKeyCertificate		
Subject Public Key Information:		
AlgorithmIdentifier:algorithm = 1.2.840.10045.2.1 (ecPublicKey)		
AlgorithmIdentifier:parameters = 1.2.840.10045.3.1.7 (secp256r1 as defined in SECG 0)		
SubjectPublicKey = TrustedWorldPK 528 bits (First byte is 0x00 for number of unused bits in last octet; second byte is 0x04 for uncompressed format. Followed by 256bit x coordinate; 256bit y coordinate)		
Certificate extension		
Extension Name	**Criticality**	Value
TrustedFirmwareNVCounter	=1	(5 bits)
SCPFirmwareContentCertPK	=1	(512 bits for EC-DSA)
Signature:		
Signature Algorithm Identifier =1.2.840.10045.4.3.2 (ecdsa-with-SHA256 as defined in RFC 5758 0)		
Signature value = 256Bits for EC-DSA		

Table 18 explains the X.509 v3 extension fields required and their content:

Table 18 SCP Firmware Content Certificate X.509 fields

Version of the certificate (v3=2)
Certificate Serial Number (Integer)
Certificate Algorithm Identifier for certificate Issuer's Signature (e.g. sha256WithRSAEncryption, ECDSAwithSHA256)
Certificate Issuer name = CN=SCPFirmwareContentCertificate
Validity Period
Subject Name = CN=SCPFirmwareContentCertificate

Subject Public Key Information:

AlgorithmIdentifier:algorithm = 1.2.840.10045.2.1 (ecPublicKey)	
AlgorithmIdentifier:parameters = 1.2.840.10045.3.1.7 (secp256r1 as defined in SECG 0)	
SubjectPublicKey = SCPFirmwareContentCertPK	
528 bits (First byte is 0x00 for number of unused bits in last octet; second byte is 0x04 for uncompressed format. Followed by 256bit x coordinate; 256bit y coordinate)	

Certificate extension

Extension Name	Criticality	Value
TrustedFirmwareNVCounter	=1	(5 bits)
SCPRomPatchHash	=0	SHA256(SCP ROM Patch Payload)
SCPFirmwareHash	=1	SHA256(SCP Firmware)

Signature:

Signature Algorithm Identifier	
=1.2.840.10045.4.3.2 (ecdsa-with-SHA256 as defined in RFC 5758 0)	
Signature value = 256Bits for EC-DSA	

Cortex-A series processors with TrustZone technology extensions copy the SCP firmware into the SCP on-chip local SRAM before allowing the SCP to boot.

APPENDIX B   LIST OF IMAGES AND PATCH FILES

The following section lists and characterizes each of the images included in the Trusted Boot process.

The format and structure of each file, including the structure of patching files, is outside the scope of this document.

B.1   AP Firmware Updater Configuration

Purpose	A subset of AP and optionally Power Control functions to configure the SoC clocks, power domains and physical interface to minimize flashing and manufacture time.
Certificate Name	Non-Trusted Firmware Updater certificate
Extension Name for hash	APFirmwareUpdaterConfigHash
Ownership	Silicon Vendor
Requirement	Optional

B.2   SCP Firmware Updater Configuration

Purpose	A subset of Power Control functions to configure the SoC clocks, power domains and physical interface to minimize flashing and manufacture time.
Certificate Name	Non-Trusted Firmware Updater certificate
Extension Name for hash	SCPFirmwareUpdaterConfigHash
Ownership	Silicon Vendor
Requirement	Optional

B.3   Firmware Updater Image

Purpose	The AP Non-Trusted firmware updater is responsible for loading the complete SoC firmware, which includes SCP firmware (if present), AP firmware, Trusted OS, Non-Trusted world images.
Certificate Name	Non-Trusted Firmware Updater certificate
Extension Name for hash	FirmwareUpdaterHash
Ownership	Silicon Vendor
Requirement	Optional

B.4   AP Boot ROM Image

Purpose	The initial boot code present in ROM which cold boots the AP.
Certificate Name	n/a
Extension Name for hash	n/a
Ownership	Silicon Vendor
Requirement	Mandatory

B.5 Trusted Boot Firmware Image

Purpose	The main firmware image, to be loaded after ROM.
Certificate Name	Trusted Boot Firmware Certificate
Extension Name for hash	TrustedBootFirmwareHash
Ownership	Silicon Vendor
Requirement	Mandatory

B.6 AP ROM Patch

Purpose	An image to patch the AP ROM
Certificate Name	SoC Firmware Content Certificate
Extension Name for hash	APRomPatchHash
Ownership	Silicon Vendor
Requirement	Optional

B.7 SoC Configuration Update

Purpose	The SoC Configuration update, is an image which may contain data and/or code to resolve hardware issues in the SoC implementation, or any subsystem of the SoC. It may also contain configurations which when applied to the SoC change the nature of its operation to overcome design errors.
Certificate Name	SoC Firmware Content Certificate
Extension Name for hash	SoCConfigHash
Ownership	Silicon Vendor/OEM
Requirement	Optional

B.8 SoC AP Firmware Image

Purpose	The main AP firmware image.
Certificate Name	SoC Firmware Content Certificate
Extension Name for hash	SoCAPFirmwareHash
Ownership	Silicon Vendor
Requirement	Mandatory

B.9 SCP ROM Image

Purpose	The initial boot code present in ROM for the SCP.
Certificate Name	n/a

Extension Name for hash	n/a
Ownership	Silicon Vendor
Requirement	Optional

B.10   SCP Firmware Image

Purpose	The SCP firmware image contains the firmware required by the SCP.
Certificate Name	SCP Firmware Certificate
Extension Name for hash	SCPFirmwareHash
Ownership	Silicon Vendor/OEM
Requirement	Optional

B.11   SCP ROM Patch

Purpose	The SCP ROM patch image contains a patch to the SCP ROM
Certificate Name	SCP Firmware Certificate
Extension Name for hash	SCPRomPatchHash
Ownership	Silicon Vendor/OEM
Requirement	Optional

B.12   Trusted OS Firmware Image

Purpose	The Trusted OS firmware image may contain the TEE or Trusted OS image, or may only contain a stub to load the Trusted OS.
Certificate Name	Trusted OS Firmware Content Certificate
Extension Name for hash	TrustedOSFirmwareHash
Ownership	Silicon Vendor/TEE Vendor
Requirement	Mandatory

B.13   Non-Trusted World Bootloader

Purpose	The Non-Trusted world bootloader is the image to load the next stage, such as a UEFI compliant boot loader to load a kernel image, or similar.
Certificate Name	Non-Trusted World Certificate
Extension Name for hash	NonTrustedWorldBootloaderHash
Ownership	Silicon Vendor/OEM Vendor
Requirement	Mandatory

APPENDIX C   RESERVED OBJECT ID VALUES FOR X.509

C.1   Reserved Object Id Values for X.509

The following table lists the OID value defined and reserved by TBBR, used to define the extension fields described in the certificate structure in section 3.6.

Common fields		
	TrustedFirmwareNVCounter	1.3.6.1.4.1.4128.2100.1
	NonTrustedFirmwareNVCounter	1.3.6.1.4.1.4128.2100.2
Non-Trusted Firmware Updater certificate		
	APFirmwareUpdaterConfigHash	1.3.6.1.4.1.4128.2100.101
	SCPFirmwareUpdaterConfigHash	1.3.6.1.4.1.4128.2100.102
	FirmwareUpdaterHash	1.3.6.1.4.1.4128.2100.103
	TrustedWatchdogRefreshTime	1.3.6.1.4.1.4128.2100.104
Trusted Boot Firmware Certificate		
	TrustedBootFirmwareHash	1.3.6.1.4.1.4128.2100.201
Trusted Key Certificate		
	PrimaryDebugCertificatePK	1.3.6.1.4.1.4128.2100.301
	TrustedWorldPK	1.3.6.1.4.1.4128.2100.302
	NonTrustedWorldPK	1.3.6.1.4.1.4128.2100.303
Trusted Debug Certificate		
	DebugScenario	1.3.6.1.4.1.4128.2100.401
	SoC Specific	1.3.6.1.4.1.4128.2100.402
	SecondaryDebugCertPK	1.3.6.1.4.1.4128.2100.403
SoC Firmware Key Certificate		
	SoCFirmwareContentCertPK	1.3.6.1.4.1.4128.2100.501
SoC Firmware Content Certificate		
	APRomPatchHash	1.3.6.1.4.1.4128.2100.601
	SoCConfigHash	1.3.6.1.4.1.4128.2100.602
	SoCAPFirmwareHash	1.3.6.1.4.1.4128.2100.603
SCP Firmware Key Certificate		
	SCPFirmwareContentCertPK	1.3.6.1.4.1.4128.2100.701
SCP Firmware Content Certificate		
	SCPFirmwareHash	1.3.6.1.4.1.4128.2100.801
	SCPRomPatchHash	1.3.6.1.4.1.4128.2100.802
Trusted OS Firmware Key Certificate		
	TrustedOSFirmwareContentCertPK	1.3.6.1.4.1.4128.2100.901
Trusted OS Firmware Content Certificate		
	TrustedOSFirmwareHash	1.3.6.1.4.1.4128.2100.1001
Non-Trusted Firmware Key Certificate		
	NonTrustedFirmwareContentCertPK	1.3.6.1.4.1.4128.2100.1101

Non-Trusted Firmware Content Certificate	
NonTrustedWorldBootloaderHash	1.3.6.1.4.1.4128.2100.1201