Platform Security Boot Guide

Document number:	DEN 0072
Release Quality:	Release
Release Number:	0
Confidentiality	Non-Confidential
Date of Issue:	30/07/20

Contents

About this document

This document defines the security architecture and technical requirements to create a Trusted Boot process. A Trusted Boot process involves verifying and measuring software in accordance to a chain of trust.

Release Information

The change history table lists the changes that have been made to this document. For a detailed list of changes, see the change history table at the end of this document.

Date	Version	Confidentiality	Change
October 2018	1.0 Beta 0	Non-confidential	First public release
February 2019	1.0 Beta 1	Non-confidential	Second public release
October 2019	1.0 Release 0	Non-confidential	First complete release
March 2020	1.1 Beta 0	Non-confidential	Beta release covering A-class processors.
July 2020	1.1 Release 0	Non-confidential	Second complete release • Fixed broken references • Remove boot seed requirements • Small clarifications about authenticated data • Title renamed from Trusted Boot and Firmware Update.

Platform Security Boot Guide

Arm Non-Confidential Document Licence ("Licence")

the Document or any products based thereon. Visit Arm's website at **https://www.arm.com/company/policies/trademarks** for more information about Arm's trademarks.

The validity, construction and performance of this Licence shall be governed by English Law.

Arm Limited. Company 02557590 registered in England.

110 Fulbourn Road, Cambridge, England CB1 9NJ.

Arm document reference: LES-PRE-21585 version 4.0

References

This document refers to the following documents.

Ref	Document Number	Title
[1]	ARM DEN 0079	PSA Security Model
[2]	ARM DEN 0083A	Arm® Trusted Base System Architecture for M
[3]	ARM DEN 0021D	Arm® Trusted Base System Architecture, Client (4th Edition)
[4]	SEC 2	SEC 2: Recommended Elliptic Curve Domain Parameters. Certicom Corp.
[5]	FIPS PUB 186-4	FIPS PUB 186-4 Digital Signature Standard (DSS). NIST.
[6]	U/OO/814670-15	Commercial National Security Algorithm (CNSA) Suite. National Security Agency.
[7]	RFC 8032	Edwards-Curve Digital Signature Algorithm (EdDSA). IRTF.
[8]	NIST 800-108	NIST Special Publication 800-108 Recommendation for Key Derivation Using Pseudorandom Functions. NIST.
[9]	NIST 800-107	NIST Special Publication 800-107 Revision 1 Recommendation for Applications Using Approved Hash Algorithms. NIST.
[10]	NIST 800-57	NIST Special Publication 800-57 Part 1 Revision 4 Recommendation for Key Management. NIST.
[12]	IETF SUIT	Software Update for Internet of Things (SUIT) manifest format. IETF.
[13]	TCG Client Profile	TCG PC Client Platform Firmware Profile Specification
[14]	DOI 10.1007/3-540-68697-5_9	Timing Attacks on Implementations of Diffie-Hellman, RSA, DSS, and Other Systems
[15]	ARM DEN 0086	Arm® Server Base Security Guide

Terms and abbreviations

This document uses the following terms and abbreviations.

Term	Meaning
AES	Advanced Encryption Standard, a symmetric-key encryption standard
Digest	The output of a hash operation
DoS	Denial of Service
EEPROM	Electrically Erasable Programmable Read-Only Memory
eFlash	See Internal flash

eFuse	OTP memory, available in very limited quantity
HMAC	Hashed Message Authentication Code
HUK	Hardware Unique Key
Internal flash	On-chip embedded flash
KDF	Key Derivation Function
Manifest	Signed metadata for a firmware image
MCU	Micro-controller unit
Measurement	A cryptographic hash of code and/or data
MPU	Memory Protection Unit
MTP	Multi-Time Programmable. A characteristic of some type of NVM
NIST	National Institute of Standards and Technology (**http://www.nist.gov**)
NSPE	Non-Secure Processing Environment (a PSA term)
NSPE-PK	Public Key of the Non-Secure Processing Environment
NVM	Non-volatile memory
OEM	Original Equipment Manufacturer
OTA	Over-The-Air
OTP	One Time Programmable. A characteristic of some types of NVM
PKI	Public Key Infrastructure
PRoT	PSA Root of Trust (a PSA term)
ROM	Read-only memory
ROTPK	Root of Trust Public Key (for firmware verification)
RSA	Rivest, Shamir and Adleman. An algorithm for public-key cryptography.
RSA-PSS	RSA Probabilistic Signature Scheme
Runtime firmware	Generic term to describe the firmware that executes after boot has completed
SE	Secure Element. An example of a secure element is a smart card.
SoC	System on Chip
SPE	Secure Processing Environment. Contains trusted firmware and trusted services.
SPE-PK	Public Key of the Secure Processing Environment
System	Inseparable component integrating all processing elements, bus masters, and secure software. Typically an SoC or equivalent.

SPM	Secure Partition Manager
Security subsystem	A self-contained subsystem providing security functionality e.g. a secure element
XIP	eXecute-In-Place

Feedback

Arm welcomes feedback on its documentation.

Feedback on this book

If you have comments on the content of this book, send an e-mail to **arm.psa-feedback@arm.com**. Give:

- The title (Platform Security Boot Guide).
- The number and release (DEN 0072 1.1 Release 0).
- The page numbers to which your comments apply.
- The rule identifiers to which your comments apply, if applicable.
- A concise explanation of your comments.

Arm also welcomes general suggestions for additions and improvements.

1 Introduction

This document outlines the security requirements for the boot process and for firmware update. It specifically covers both:

- **Secure boot**, which at each stage of the boot process checks that an image is authorized to run before use. Since this occurs recursively it creates a "chain of trust" as shown in Figure 1.
- **Measured boot**, which is the process of cryptographically measuring the code and critical data so that the security state can be attested to later.

This document presents the best security principles for developing a Trusted Boot process across a range of different systems. These principles include:

- A set of mandatory rules.
- A set of recommendations.
- A set of informative notes for specific markets and technologies .

System-on-chips (SoCs) vary in complexity and capability to meet different cost targets. Optional recommendations are given for SoCs that have sufficient resources to provide more robust protection. System developers are encouraged to test the given recommendations to increase the resiliency and longevity of their products.

This document uses terminology defined by the PSA Security Model. This document is also useful to platforms that do not implement full PSA functionality.

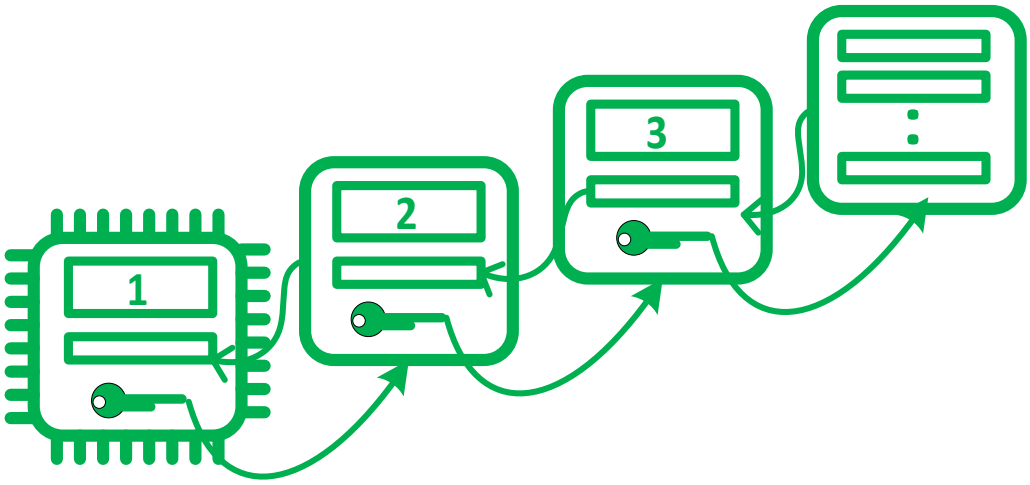

Figure 1: The Chain of Trust approach with a variable number of verification stages, each of which might be encrypted

1.1 Scope

Protection against unauthorized physical access depends on the hardware capabilities and the threat model for a particular product. Therefore, a detailed threat model and security review for a particular product is recommended, which is not within the scope of this document.

The following boot related threats are covered in this specification:

Threat	Summary	Mitigation
T.IMAGE_TAMPERING	An attacker executes an altered firmware or software image to gain control of the system execution and configuration. This can enable an attacker to access some assets.	Image verification Image metadata and parameters Concurrency
T.DATA_TAMPERING	An attacker with physical access alters assets in off-chip storage. Such techniques can include altering or replacing assets in flash chips or removable media. A more sophisticated attacker might also interpose on connected buses.	Secure storage Concurrency
T.ROLLBACK	An attacker downgrades to a flawed version of firmware or software in order to exploit a vulnerability to gain partial or total control of the system. In a similar manner, an attacker can roll back or replay security critical data, such as the boot configuration.	Rollback protection
T.DEBUG_ABUSE	An attacker uses the debug infrastructure to elevate privileges or obtain assets.	Immutable bootloader (security lifecycle enforcement)
T.REVERSE_ENGINEERING	An attacker with physical access reads data in off-chip storage. This enables reverse engineering or cloning of assets to other systems.	Image encryption Secret scrubbing
T.PERSISTENT_MALWARE	An attacker exploits a vulnerability to install malicious software that remains persistent on the system across resets. This can prevent remote recovery of the system.	Immutable bootloader Image verification Reset protection Isolation mechanisms
T.MALICIOUS_PERIPHERALS	An attacker exploits or replaces external off-chip peripherals in order to issue Direct Memory Access transactions. Transactions might be used to escalate privileges or to directly access assets.	DMA protection Secure storage
T.UPDATE_ABUSE	A remote attacker renders a device inoperable by sending an unmodified but incompatible firmware update.	Image metadata and parameters

The following threats are out of the scope of this specification, but might need to be covered for specific markets or certification levels:

Threat	Summary	Mitigation
T.SIDE_CHANNELS	An attacker infers the value of sensitive on-chip code or data by using non-invasive techniques, such as differential power analysis or software observable side channels. An example asset can be a decryption key that is used for decrypting boot content.	Out-of-scope
T.GLITCHING	An attacker glitches the SoC power supply or clock during the boot process in order to bypass authentication checks.	Out-of-scope
T.LAB_EQUIPMENT	An attacker uses specialized equipment to unpackage and probe chips.	Out-of-scope
T.WEAK_CRYPTO	An attacker breaks the cryptography used by the boot process (e.g. hash collision, signature forgery)	See recommendations in Cryptographic algorithms (informational)

Factory provisioning and test modes are also out-of-scope of this specification but might need to be considered by an implementation depending on supply chain relationships.

Attacks on systems continuously evolve, with the effect that old security defenses must be strengthened, and new security defenses must be implemented to maintain the required level of security. The requirements described in this document represent the best practice at the time of writing. Some requirements provide implementation flexibility and improvements when compared to older documentation provided by Arm. In all cases, the differences are in the degree of security that is provided, or that is demanded by other market specifications.

1.2 Assumptions

This document assumes the reader is familiar with standard techniques in cryptography, such as authentication, hashing, encryption, Public Key Cryptography Standards (PKCS), NSA suite B and digital certificates. It does not describe these concepts in detail.

This document assumes the reader is familiar with the *Trusted Base System Architecture (TBSA)*, which includes the necessary hardware features expected for a PSA platform. At the time of writing there are two variants of TBSA:

- *TBSA-M* is the Arm specification for building a microcontroller with best practice security properties
- *TBSA-A* is the Arm specification for building an application processor with best practice security properties. Further segment specific guidance is available in guides, such as the *Arm® Server Base Security Guide.*

This document aims to support a class of system which:

- can perform public key cryptography
- have a boot ROM or a lockable sector of on-chip flash to emulate a boot ROM. The words “boot ROM” and “immutable bootloader” are used interchangeably in this specification
- supports a level of memory isolation that can protect the code and data of security-critical functionality

This document assumes that the reader wants to create firmware that has a Trusted Boot process and a secure update process.

1.3 Compliance

A claim of compliance to this specification is an evidence-backed assertion that the design meets all applicable requirements that are described in this document. The assertion is normally made by the design team and takes the form of documented output of a design review of the device. Arm recommends that this assessment is made as part of a Secure Development Lifecycle (SDL).

The design team shall confirm, for each requirement, whether the requirement is fulfilled. This confirmation shall include a brief description of why the design is compliant and references to the relevant detailed specifications. In general, requirements might not be applicable if the threats that they mitigate can be shown to not form part of the threat model of the device, or that any vulnerabilities that might result from not meeting a requirement can be demonstrated to be mitigated in another way. In some cases, it will be necessary to provide stronger security than is anticipated by these requirements. In these cases, evidence shall be documented to

support this approach alongside the requirement. The Appendix of this document includes a checklist to assist in this activity.

In several areas, this specification provides recommendations. Where possible, these recommendations are provided to give guidance on reasonable default design choices. The threat model and functional requirements of the device is key in determining how requirements are met and which recommendations are followed. This is beyond the scope of this document.

2 Terminology

If the hardware supports isolation of software, then the software can be split into two security domains:

- **Secure Processing Environment (SPE),** which contains the:
    - **Trusted firmware** for providing roots of trust to the system. These roots of trust are used for securing sensitive data such as secrets, platform state, and cryptographic key material. This is typically the Trusted Computing Base (TCB) for the SPE software. It is comprised of:
        - one or more trusted bootloaders
        - a Trusted OS or a Secure Partition Manager (SPM)
        - one or more security subsystems *(see 2.1 Security subsystem*).
    - **Trusted services** for providing application-specific security services while remaining isolated from the non-secure applications. It provides security functionality to the non-secure applications, and typically depends on the primitives provided by the trusted firmware.
- **Non-Secure Processing Environment (NSPE)** for general purpose functionality that is not security critical. This includes communication stacks, device drivers, task management and application software. It is unable to undermine the integrity and confidentiality of the SPE. The NSPE might be isolated on a separate processor from the SPE. On an Arm processor with TrustZone security extensions the NSPE is known as the “Normal World”. On other systems, the NSPE might be a separate processor that is unable to access SPE resources.

This specification refers to the SPE and NSPE when describing roles and responsibilities of components in the firmware update process. It is possible for the SPE and NSPE to be executing on separate CPUs within the same SoC, providing that the NSPE cannot interfere with SPE resources and execution.

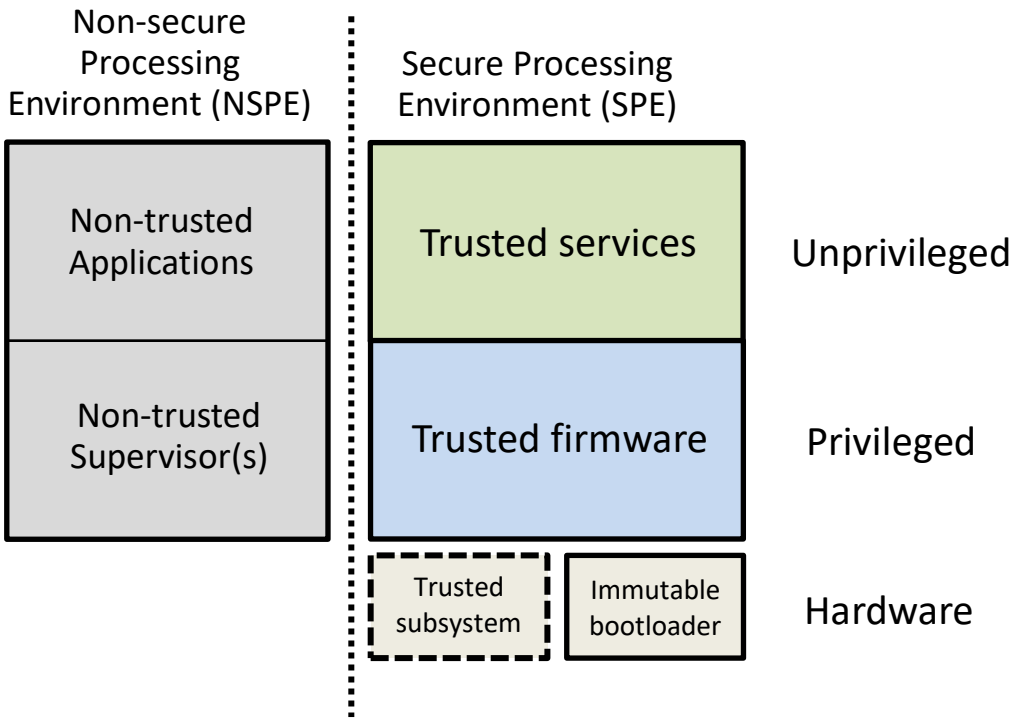

Figure 2: Illustration of various security domains described in PSA (agnostic of hardware architecture)

Each of these subsystems must be securely loaded, starting with the trusted firmware.

A platform might have several images depending on the system design. The code in the SPE should be small in size and limited in purpose such that the software can be better validated for programming errors or design mistakes.

2.1 Security subsystem

Security subsystems are blocks of security IP and might also provide RoT security services to the system. They might be integrated, or external (but bound) to the SoC. For example, these might include:

- Trusted peripherals which support cryptographic operations
- Secure elements and enclaves, each of which contains its own local RoT and its own local security lifecycle
- System Control Processor (SCP)
- Manageability Control Processor (MCP)
- DRAM protection systems
- Trusted real-time clock

Secure Elements are independent subsystems which provide a set of RoT services for the system. There are many variations that might be on-chip or off-chip, have market-specific features and degrees of programmability. These are all collectively referred to as security subsystems.

If a security subsystem is off-chip, it is expected that the communication channel between the SoC and the security subsystem are cryptographically paired. Cryptographic pairing enables secure communications between components and prevents unauthorized replacement of a security subsystem.

2.2 Trusted memory

Each system has physical trust boundaries based on the threats they are designed to mitigate. Memory that is within the trust boundary is described as trusted.

Specifically, trusted memory refers to connected RAM and NVM that are trusted to be sufficiently secure against a common set of threats and adversaries. NVM includes all types of persistent storage.

The SPE uses trusted memory to initialize and load SPE components. SPE resources must be protected from the NSPE.

- If RAM is shared between SPE and NSPE then they must be logically partitioned such that only the SPE can access SPE memory. This might involve an on-chip mechanism such as an address space controller or a security attribution unit.
- If the SPE and NSPE do not share RAM, then the SPE memory must be physically isolated from the NSPE such that only the SPE can access SPE memory.

The requirements within focus on protecting easily accessible interfaces and supporting software countermeasures against typical concurrency issues.

Designers are encouraged to evaluate the default profiles and validate whether they are sufficient against any relevant certification or compliance programs.

Table 1: Default trust boundaries assumed for different device profiles

System class	Trusted memory	Rationale
Microcontroller	On-chip* memory	Off-chip memory, such as external flash, might be easily accessible to a locally present attacker, and might be accessible during manufacturing or deployment.
Endpoint	On-chip* memory + trusted off-chip RAM	Off-chip memory, such as external flash, might be easily accessible to a locally present attacker, and might be accessible during manufacturing or deployment.
Infrastructure	On-chip* memory + trusted off-chip RAM	Off-chip storage, such as flash, might be accessible to a locally present attacker or an external system, such as a networked Baseboard Management Controller (BMC), during manufacturing or deployment.

[*] Arm recommends using on-chip RAM. However, SRAM can be used on a separate die, provided it is within the same package as the main SoC.

Advanced hardware invasive attacks, in which the attacker has access to laboratory equipment that probes on to silicon metal layers, infers fuse settings, or performs differential power analysis are out of scope.

A certain class of attacker might be able to read the contents of off-chip RAM given physical access to the system and appropriate tools. This might involve an attacker that can freeze the memory to manually read the contents

or by replacing the memory with a more advanced component that persists data after the system has been turned off. If a security objective needs to address this then all off-chip memory must be considered untrusted.

For information about the requirements for trusted peripherals see Section 2.1.

2.3 Image

An image contains one or more of the following artifacts:

- Software executables
- Firmware executables
- Patches
- Configuration data and parameters

2.4 Image manifest

Security critical metadata that is associated with an image.

If a system is sufficiently capable of processing certificate chains using the X.509 standard, such as an A-class processor, then all instances of manifest in this document might use X.509 content certificates as a manifest. Moreover, the X.509 standard is well documented, and free tools and source code are available. However, the parsing of encoded formats can introduce complexity that might outweigh the need for interoperability, especially on constrained devices.

3 Security requirements

To be compliant with this specification, the rules specified in this section must be met. Rules are denoted with rule IDs and highlighted in blue. The surrounding text provides examples and rationale. Subsections titled with "optional" contain optional functionality that might increase robustness of implementations.

3.1 Boot flow (informational)

The boot and firmware update cycle can be summarized in a small number of steps:

1. The first Trusted Boot code is a bootloader that is embedded in a boot ROM, a write-protectable equivalent, or a security processor. This is referred to as the immutable bootloader in this specification. The immutable bootloader is a hardware Root of Trust that executes from reset, containing the minimal functionality required to check the authenticity of the Trusted Boot software.
2. The Trusted Boot software authenticates the trusted firmware and possibly also the NSPE software. The entire Trusted Boot sequence might consist of multiple stages, each of which must be authenticated before execution.
3. The SPE initializes different Roots of Trust for the system as well as generic security services, known as Trusted Services. It might also provide bootloader functionality, such as backup and test features.
4. The NSPE receives firmware updates Over the Air (OTA) or from a local peripheral.
5. A runtime service or a bootloader within the SPE will check the update authenticity against a local public key and against a policy to see if it should be installed. If permitted to be updated, then the SPE will provision the update to storage, which is typically performed by a bootloader on reset.

Depending on system resources and supply chain factors, the SPE might consist of multiple images, such as a mutable bootloader, runtime services and recovery software. Multiple bootloaders might also be used to separate board-specific code from chip-specific code.

3.2 Chain of Trust (informational)

A chain of trust ensures that each loaded component on the system has not been tampered. The chain of trust begins on reset whereby a hardware component authenticates the first stage of software. The chain continues when the authenticated software loads additional software.

When the SoC hardware is powered on the CPU automatically executes from the start address (such as the reset vector). The start address and the boot selectors are configured to a predefined location of the internal flash address space or to masked ROM. The code at this start address must be considered immutable once provisioned in the factory. These characteristics provide strong assurance that the immutable bootloader cannot easily be bypassed.

The immutable bootloader uses a public key to check the first stage of code is authentic before executing it:

- In a single stage boot process, the immutable bootloader might validate a single image or simultaneously validate multiple images.
- In a multistage boot process, there might be numerous loaders, each of which uses keys to verify and load other images.

An implementation must decide how many boot stages will be needed based on product requirements. Regardless of the number of stages, each boot stage must authenticate all the software it loads.

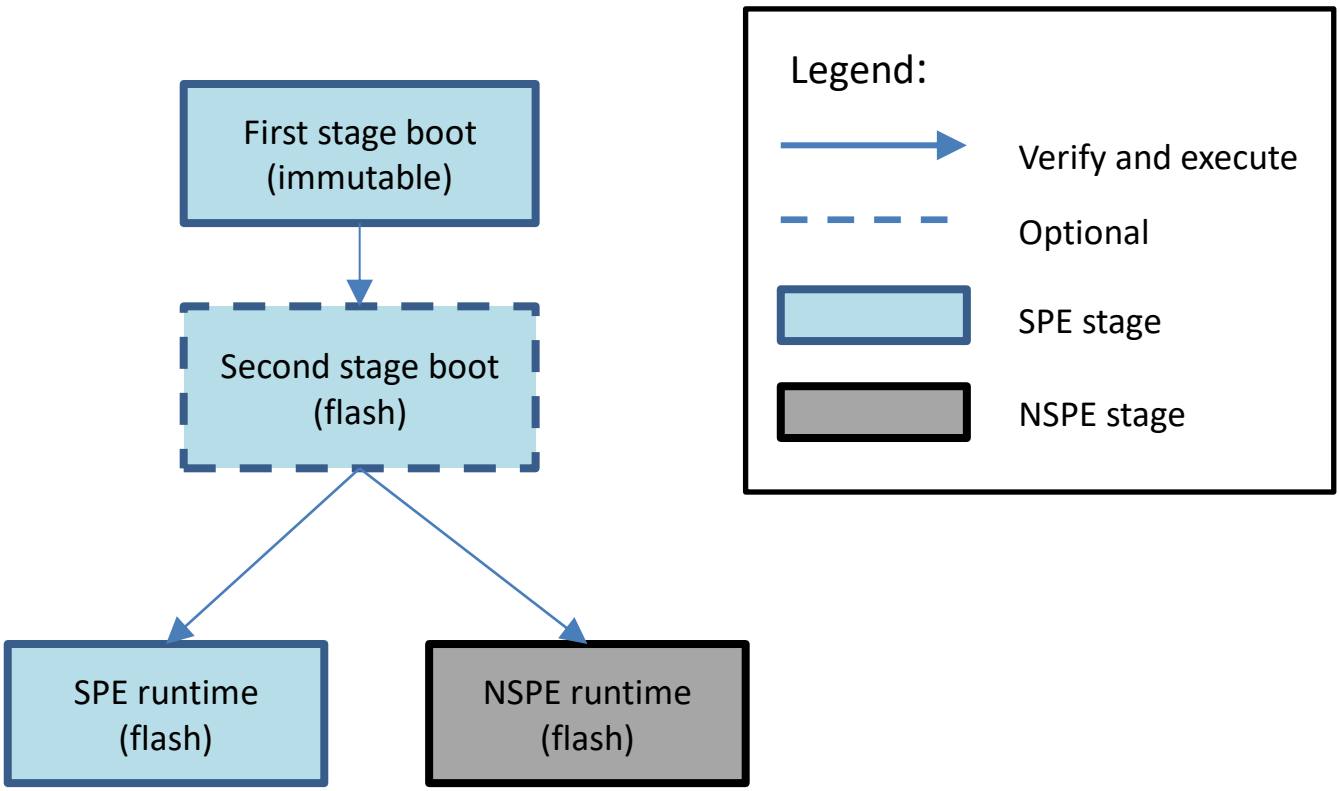

Figure 3: Generic boot process with multiple stages

It is common for a boot loader to be divided into several stages, the first of which is the immutable bootloader. The latter stages might be loaded from non-volatile storage into RAM and executed there or executed directly from eFlash. Splitting the bootloader into at least two stages, immutable and mutable, has some advantages:

- It minimizes the risk of problems in the immutable code, as it allows for updates or errata to be handled by the mutable stage at provisioning time or later. As an example, image backup and recovery functionality can be non-trivial to implement in code for certain storage types and is likely to require bug fixes after product release
- It separates concerns between silicon vendor and board maker, who might be different parties. For instance, the SoC manufacturer might provide code for basic initialization and protection of certain assets whereas the Original Equipment Manufacturer (OEM) might customize board specific features, such as external flash and advanced recovery capabilities. In this scenario, the SoC manufacturer's code would authenticate the OEM's initial code.

The most constrained system might only contain a single stage bootloader within locked flash.

At each step in the boot chain, each stage must verify the next, and verification of an image is based on a combination of hashing and asymmetric cryptography. Since asymmetric cryptographic algorithms are CPU-intensive, hardware acceleration can be employed. For instance, the RSA cryptosystem might be accelerated using a Montgomery Multiplier hardware unit. Where possible, image authentication should use any available secure elements or accelerators to reduce memory footprint and accelerate the boot process.

It is possible for the boot sequence to fail at any stage due to a faulty component or restricted functionality. If the signature of a component fails verification, then the platform must not execute that component. The platform might want to perform one or more of the following actions:

- log the event
- recover the component
- reset the system
- load other components

3.3 Use cases (informational)

The size of the chain of trust is dependent on supply chain constraints and system complexity. A product analysis must determine the number of potential actors that will maintain the components of a product. Examples might include:

- A Silicon Provider (SiP) that customizes SoC-specific code or errata fixes before the boot process begins.
- A manufacturer who adds board-specific customizations based on product requirements (for example, OEM firmware and management software).
- An operator (or owner) might configure an update policy, install additional software, or add additional identities on the system before deployment.
- A repair company that might be authorized with certain debug capabilities.
- An owner who might wish to replace the firmware after a system is no longer supported by a manufacturer.

The number of distrustful actors will determine the number of keys the system needs to support and the required security lifecycle.

3.4 Supply chain scenarios (informational)

This specification requires at least one public key known as the Root of Trust Public Key (ROTPK), which is responsible for securely authenticating the first stage of code using public key cryptography. It might also be used to verify certificates of delegated signing keys

Either the ROTPK hash or the ROTPK itself must be in an immutable part of the SoC NVM or within a security subsystem. A hash is often preferable because it requires less space in immutable storage.

Multiple ROTPKs might be included for different vendors in the supply chain. For instance, the SoC vendor might be able to securely boot their own code using an in-built ROTPK, and then verify OEM firmware using a separate OEM ROTPK (see Section 3.4.3). The OEM ROTPK might be provisioned at a different point in the supply chain, where there might be operational processes in place to reduce risks in the provisioning process. The specific factory provisioning methods are outside the scope of this specification.

R10_PSBG_KEYS: The system must include at least one firmware verification public key known as a Root of Trust Public Key (ROTPK) and a Hardware Unique Key (HUK)

The private key associated with the immutable ROTPK cannot be revoked. This private key might be used to sign an OEM generic second stage boot loader or SPM, which will perform board-specific configuration and continue the Trusted Boot process. As such the ROTPK private key is hardly used (a few times per OEM) and can be kept in a highly secure Hardware Security Module (HSM).

R20_PSBG_KEYS: Each ROTPK must be immutable. These can be stored using a locked on-chip flash sector, a secure element, or on-chip OTP memory. It is permitted to store an immutable hash of each ROTPK to check the integrity of ROTPKs in untrusted storage.

3.4.1 Single provider

Figure 4 shows the simplest case, such as a constrained or embedded device, where a manufacturer controls the entire software stack and signing process. The manufacturer provides the ROTPK. No revocation is possible in this scenario.

This scenario is only beneficial to ultra-constrained MCUs or simple peripherals.

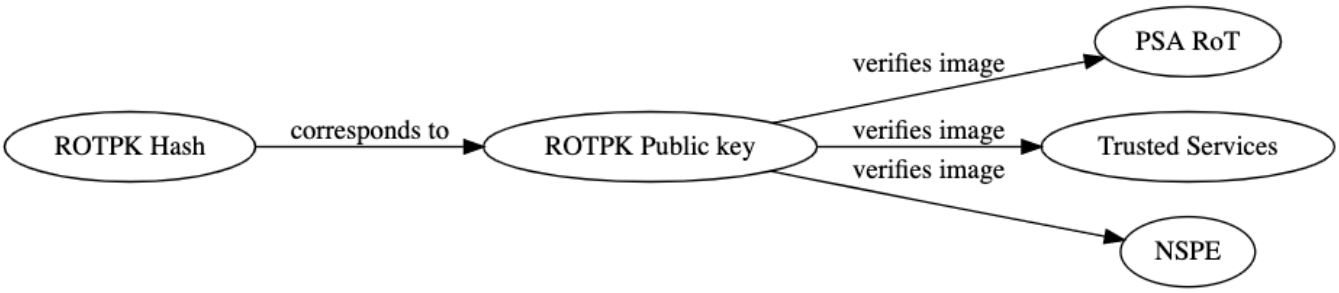

Figure 4: Single provider with a single signing key

If the system has multiple software providers and has sufficient computational ability to validate a certificate chain, then it is recommended that separate signing keys are created to mitigate any loss of a private image signing key. This is discussed in the following sections.

3.4.2 Multiple dependent providers

The first link in the chain of trust is the management of the Root of Trust public key, which is used to verify all subsequent certificates and images. An example scenario is the following:

- The OEM provides their own ROTPK.
- The OEM signs a certificate belonging to a software vendor's credentials. There could be multiple software vendors.
- The software vendor uses their certified credentials to sign:
    - Production image signing credentials (unique for a particular model line)
    - Debug image signing credentials (unique for a particular model line)
- The SoC ships with the OEM's ROTPK provisioned into OTP memory

If an image signer needs to replace their image signing key, then they must contact the owner of the ROTPK (the Root CA in Figure 4). This requires a continuous business relationship between the image signer and the Root CA (the ROTPK owner).

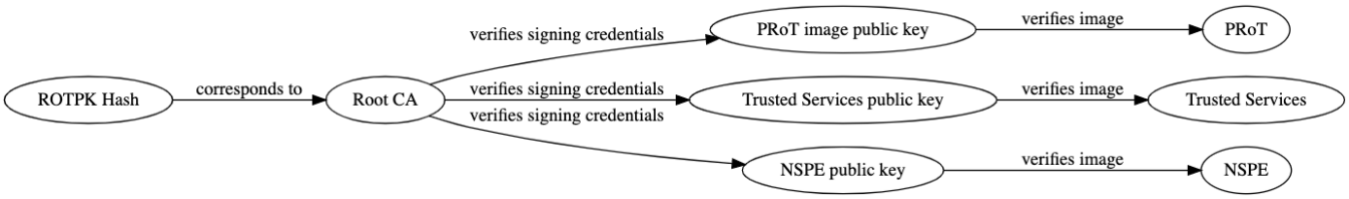

Figure 5: Example of software signers dependent on a Root CA (the owner of the ROTPK).

If SPE and NSPE are separate systems with separate images, then Arm recommends signing these images with separate signing keys. If SPE and NSPE come from the same vendor, then the signing key for the SPE should have more levels of operational protection. Since the SPE must provide secure services to the NSPE, it must be more rigorously tested and should not require frequent updates.

3.4.3 Multiple independent providers

The system might include enough on-chip storage to hold multiple discrete ROTPKs. Each ROTPK provides an independent chain of trust, which allows for different manufacturers to authorize and revoke firmware signing keys independently of each other. For instance, one ROTPK can correspond to the SoC vendor whereas another might correspond to the OEM. Alternatively, an OEM owns one ROTPK while allowing a customer to install a separate ROTPK for NSPE firmware. Figure 6 shows a simplified chain of trust for independent providers.

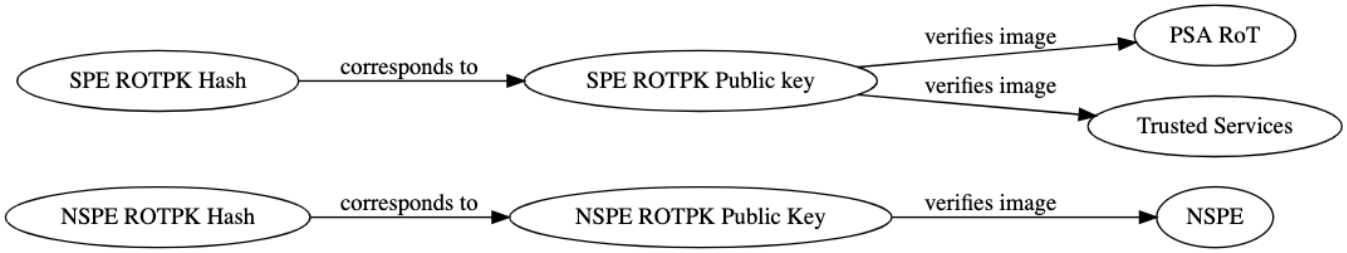

Figure 6: Example of multiple providers with independent roots of trust

It should be noted that the example NSPE chain in Figure 5 can be arbitrarily long and complex depending on the number of actors signing NSPE components (multiple dependent providers within the NSPE domain).

The NSPE ROTPK (or the hash used to identify it) can be stored in one of the following permitted implementations:

- On-chip OTP memory
- A secure element or security processor
- Secure storage that is only writeable by the SPE. Any update of that key after provisioning must either:
  - Be rejected by the SPE (one-time provisioning)
  - Be replaced with a newer key only if it is signed by the currently stored NSPE ROTPK

Implementation note

Sufficient OTP space might allow for two truncated ROTPK cryptographic hashes to be stored. If this is desired, then it must conform to NIST's recommendations as specified in NIST 800-107 [9]. For example, it is common for the most significant 128 bits from a SHA-256 digest to be used when truncated.

The NSPE ROTPK might be used to authenticate a service within the SPE provided that it is sufficiently isolated from the Trusted Services and the trusted firmware.

3.4.4 Certificate creation and key management

It is recommended that for the creation of certificates for use in production and development systems, 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.

3.5 Image metadata and parameters

A manifest contains security critical metadata about an image and its runtime parameters. A manifest is cryptographically signed to ensure integrity and authenticity.

R10_PSBG_MANIFEST: Each image must be associated with a signed manifest. It is permitted for the manifest to be appended to the image itself and then signed together.

Implementations might wish to transfer manifests in encrypted form to provide privacy for system updates.

Attribute	Description
Manifest format version	Version of the manifest format used
Signer ID	Identifies the signing key for the image. This must be the hash of the signer's public key. This field is not required if there is only one image signer for the device software.
Image hash algorithm	The cryptographic algorithm used to calculate the image hash value. This field is not required if only one algorithm is supported.

Attribute	Description
Image hash value	The expected hash value of the image. If the image is encrypted, it is IMPLEMENTATION DEFINED whether the hash value is of the ciphertext or the plaintext.
Image size	The image size in bytes.
Image type	Specifically identifies the boot stage or peripheral. Also known as a purpose identifier. This field is not required if only one type of image is supported.
Product class	This can refer to a vendor ID, product ID, or a more specific identifier, such as the Instance ID.
Image version	The version that is used to for rollback protection. The specific versioning scheme is IMPLEMENTATION DEFINED. An implementation might separate image versioning from security versioning for deployment flexibility.
Manifest signature algorithm	Identifies the digital signature algorithm used for the “Manifest signature” field. This field is not required if only one algorithm is supported.
Manifest signature	Digital signature that signs all the manifest fields. The manifest might be signed using an industry-standard container format, Cryptographic Message Syntax (CMS), previously known as PKCS-7. It is recommended that the manifest format supports multiple signatures to support counter signing.

Table 2: Required manifest attributes

More attributes can be added to the manifest to fit different use cases. The following contains a non-exhaustive list of additional attributes:

- The full public key might need to be included (instead of the Signer ID) if the full key is not provisioned on the device.
- If a firmware image has multiple dependencies, then a manifest can specify multiple images. For instance, a device might be a collection of different processing units, each of which requires specific firmware versions.
- A set of conditions might also be included, such as a pre-condition or a post-condition.
- A value (in seconds) specifying the maximum time the firmware update process should take to flash
- Firmware storage location in terms of absolute or relative addressing
- A device ID for special images that might be created for a specific platform instance. This might be necessary to provide specialized builds for specific customers, which should not be transferable to other platforms not owned by the customer.
- Information on how to decompress a compressed image.
- Parameters that affect runtime security, such as load addresses or stack pointer address.
- A nonce might be included to make the manifest unique.

R20_PSBG_MANIFEST: Each image manifest must contain the properties specified in Figure 2. It is permitted for additional information to be included.

The serialization format of a manifest might benefit from Concise Binary Object Representation (CBOR) or Abstract Syntax Notation One (ASN.1) encoding, depending on processing or interoperability requirements.

For constrained devices, such as microcontrollers, Arm recommends using the IETF SUIT manifest format for signed manifests [12]. An example manifest is provided in the Appendix of this specification.

For rich devices, such as application processors, Arm recommends the X.509v3 content certificate format for signed manifests. This enables interoperability with existing tools and software stacks. Additional fields might be specified using X.509v3 extensions. An example manifest is provided in the Appendix of this specification. Parsing of encoded formats can introduce complexity that outweighs the need for interoperability with popular formats.

Systems might support delta encoded patches to reduce bandwidth and storage of firmware updates. The image manifest must contain the expected hash value of the image after the patch has been applied.

R40_PSBG_MANIFEST: For delta updates, the image manifest must also contain the hash of the final expected image state.

A system might have many modules that require updating individually. It might also need to trust several different actors to authorize an update. For example, a firmware author might not have the authority to install firmware on critical infrastructure without the authorization of an operator. In this case, the firmware should reject firmware updates unless they are signed both by the firmware author and by the operator.

R50_PSBG_MANIFEST: All data in the image manifest must be digitally signed using asymmetric cryptography.

R60_PSBG_MANIFEST: Image updates that include security enhancements or vulnerability fixes must increase the software version when signing the manifest.

3.6 Image verification

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. Each image has either a lightweight manifest or a certificate, which is authenticated by a public key. This public key must be traced back to an ROTPK.

At the point where a new image is to be installed the associated signature must first be verified against a public key.

R00_PSBG_EXEC: Each loaded SPE image must be authenticated before use. It is recommended to also authenticate NSPE images.

Signature verification might be computationally expensive to perform on each boot. Verifying a large number of components might have a significant effect on the time it takes to boot the system. An implementation can optimize the Trusted Boot process at the expense of simplicity: once an image manifest has been successfully verified against a public key, it is permitted for an implementation to use one of the following mechanisms to speed up subsequent boots:

- The image manifest can be locally authenticated with a Message Authentication Code (MAC). In this specification this process is referred to as *rekeying*. This avoids having to perform an asymmetric signature check again on subsequent boots of the same image and therefore can speed up the boot process. It is recommended that the key of the MAC be derived from the HUK using a KDF. It is then safe for the manifest and the MAC to be placed in untrusted storage. When using this option. implementations must consider protection against side channel attacks that can extract the key.

R10_PSBG_EXEC: Any use of a MAC to re-key and authenticate a firmware image manifest must be in the form of a HMAC, CMAC, or GMAC signature. The key might either be in trusted memory or in a security subsystem.

- The manifest of the verified image can be placed in on-chip storage that is write protected from the next stage of components. On subsequent reboots the calculated hash of the image can then be directly compared with the expected hash within the stored manifest without re-authenticating the manifest. It is permitted for a subset of the manifest information to be stored providing that there is sufficient information to provide integrity verification and rollback protection.

  R20_PSBG_EXEC: Any caching of authenticated manifests must be held in trusted memory and write protected from untrusted components.

3.6.1 Secret scrubbing

Memory might be shared between components at different stages of the Trusted Boot process. Secrets that are resident in memory, such as decryption keys, might need to be removed prior to executing a newly loaded component.

R30_PSBG_EXEC: Secrets used by a trusted component X must be scrubbed from volatile memory and registers before ownership of the memory is transferred to a component not trusted by X.

Scrubbing a memory location can mean any of the following methods:

- Overwritten with a pre-defined constant value (for example, zero)
- Overwritten with a random value
- Indirectly changed to a random value, for example by changing a key which is used to decrypt the memory contents

3.6.2 Concurrency restrictions

By loading components from untrusted storage to trusted memory prior to cryptographic validation, it is not possible to bypass verification with an authorized copy of the firmware and then substitute an unauthorized version at runtime.

R40_PSBG_EXEC: Each loaded SPE image must be verified in trusted memory before execution. It is permitted for NSPE images to be loaded into untrusted memory.

To prevent interference from code running on other processors, and to eliminate time-of-check-time-of-use (TOCTOU) exploits, concurrent execution must be disabled when validating images.

R50_PSBG_EXEC: The boot process must be uninterruptible during signature verification to prevent race conditions.

When signatures fail to verify or a roll back attack is detected, then this event is considered a security violation. The component that causes the security violation must not be executed. If the component is critical to the system functionality then a recovery mode might be entered, or the system might log diagnostics and reboot depending on the severity of the violation.

R60_PSBG_EXEC: The boot process must not execute a component if a security violation occurs.

Updating an image in persistent storage can be a complex process. Care must be taken to avoid partial or undefined system states.

R70_PSBG_EXEC: The update process must be an atomic operation. If interrupted, then the update process must either revert to the prior state or enter a recovery mode.

Recovery modes are IMPLEMENTATION DEFINED. As an example, the mode might restore a stored backup of the image or listen on a wired interface awaiting a new image. Any restored image must still be subject to all the rules defined in this specification, including rollback protection.

R80_PSBG_EXEC: If the NSPE software is to be executed on a secondary processor, the secondary processor must be kept in reset or halted until the NSPE firmware has been verified by the SPE.

This rule includes implementations with a secondary processor that has minimal ROM code waiting for an SPE notification.

3.6.3 Isolation mechanisms

If the SPE and the NSPE share compute resources, such as CPU or memory, then isolation mechanisms must be configured to protect SPE resources before the NSPE begins execution. For instance, an isolation mechanism might be one of or a combination of Address Space Controllers, Security Attribution Units, Peripheral Protection Units, Memory Protection Units, etc.

R85_PSBG_EXEC: Isolation mechanisms must be correctly configured before the NSPE begins execution.

3.6.4 DMA protection

Direct Memory Access (DMA) is a common way for peripherals to transfer data to and from SoC memory. It is possible for an attacker controlled peripheral to interfere with the Trusted Boot process using DMA. Some SoCs include special functionality for restricting DMA transactions to specific memory regions, ensuring that trusted components are integrity protected during the boot process.

R95_PSBG_EXEC: Any available I/O protection mechanisms must be enabled to integrity protect loaded images from untrusted peripherals.

An example of an I/O protection mechanism is a System Memory Management Unit (SMMU) or a Peripheral Protection Unit (PPU).

3.6.5 Secure storage

The following requirements address problems when handling images and authenticated data in untrusted memory.

SPE firmware must not be executed directly from external flash memory. Instead it must either be copied into trusted RAM and verified and executed from there or executed in-place in secure internal flash memory.

R10_PSBG_STORAGE: An SPE image in untrusted memory must be copied to trusted memory for authentication.

R20_PSBG_STORAGE: When an image in untrusted memory is copied into trusted memory, it must be integrity checked after the copy has completed. The integrity check must match the authentication data for the image.

R30_PSBG_STORAGE: Authentication data used to verify images must be in trusted memory before use.

R40_PSBG_STORAGE: Encrypted images in untrusted memory must be decrypted into trusted memory and authenticated in trusted memory.

When an image is created by a vendor and needs encryption, the point of encryption might vary depending on supply chain logistics:

- If a signed image is subsequently encrypted before distribution, then the image must be decrypted by the device before it is authenticated.
- If an image is encrypted before it is signed and distributed, then the device must authenticate the image before decrypting it.

Arm recommends that SPE images be protected from NSPE access. This can be achieved by providing the SPE with exclusive access to some on-chip or off-chip non-volatile storage.

3.7 Immutable bootloader

The first code of the Trusted Boot process is an immutable bootloader placed in an on-chip boot ROM or a locked eFlash sector that can emulate a boot ROM. An example of an emulated boot ROM would be an area of flash with an OTP lock that permanently disables write and erase accesses to the flash area, while also disabling debug access (e.g. JTAG/SWD).

The purpose of the immutable bootloader should be solely to validate the signature of the next stage against the ROTPK. The next stage is expected to be a second stage bootloader or a SPM, which provide richer functionality. As a minimum the hash of the ROTPK is either embedded in the bootloader or provisioned into OTP NVM. The hash must be used to verify the integrity of the full public key, which might be included in the mutable next stage image.

Code that performs more complex functions will naturally have a greater attack surface or risk of bugs. Therefore, the immutable bootloader should only contain the flash and cryptographic primitives necessary to read and validate software. Additional functionality should be a part of the SPE software, either another bootloader or an SPM.

R20_PSBG_BOOT: The immutable bootloader must verify loaded images using the ROTPK itself or a delegated key. It is permitted for the immutable bootloader to only verify a single firmware image containing all remaining verification functionality

Implementation note:

It is advised that most functionality is placed into a loadable module to ensure that there is never a need for the immutable bootloader to be updated. This module can be verified using RSA/ECC authentication or using a hash in on-chip OTP memory. A silicon provider might use this to deliver errata to the SoC and to defer certain ROM functionality to a later stage of manufacturing.

When using locked eFlash instead of ROM, the immutable bootloader must be placed at the reset vector to ensure that this stage always boots first. This might be at the beginning of the address space on some platforms. The regions must be configured in such a way that they are protected from program and erase operations.

The immutable bootloader must be able to securely read the ROTPK in on-chip memory. Typically, there are a number of ways to achieve this:

- The ROTPK might be embedded in the immutable bootloader itself. This is the least flexible option because the ROTPK owner might be a different party to the owner of the immutable bootloader.
- The ROTPK might be provided with an SPE image. In this case, the immutable bootloader must calculate the hash of it and compare it against an embedded provisioned hash in immutable memory before using it.
- The ROTPK or its hash might be stored in separate on-chip OTP NVM (i.e. for example eFuse or a flash sector that was locked during factory provisioning).
- The ROTPK or its hash might be provided by a security subsystem.

The immutable bootloader itself might execute within a security subsystem provided that it is on-chip and has sufficient capability to verify the SPE software while the main cores of the processor are held in reset (s*ee 4.2.2 Boot using an on-chip security subsystem*).

Manufacturers might want confidentiality of bootloader secrets after initialization. Hiding the bootloader after the boot process requires a non-reversible mechanism, for example a sticky register bit that is activated by the boot software. It is recommended that the bootloader stages are only readable during the Trusted Boot process to prevent reverse engineering and code re-use attacks after boot.

A security lifecycle is essential for ensuring that the debug infrastructure cannot be used to compromise the integrity of the boot process or leak assets. The security lifecycle can have many states, such as assembly and test, provisioning, production, return, and more. Once the system is in a secure production state, the immutable bootloader (or debug authentication module) must disable and lock debug capabilities.

R40_PSBG_BOOT: It must not be possible to debug the immutable bootloader while Trusted Boot is enabled

The detailed security requirements and lifecycle for debug are described in the *Trusted Base System Architecture* set of specifications.

An advanced attacker with physical access to the system and specialized equipment might attempt to analyze the boot process before performing a non-invasive timing attack. Such a non-invasive timing attack might introduce intentional faults using clock, power or thermal means, which could result in instruction skipping, decoding errors, or malformed data accesses. Attackers might use these side-effects to target specific areas of code execution to bypass authentication checks during the Trusted Boot process. Since these attacks rely on reliable and precise timing, early stage firmware might employ an on-chip hardware random number generator to randomize the execution time of boot, making timing attacks infeasible for adversaries.

3.7.1 Reset protection

Some platforms provide a register for programming an address for the processor to jump to on reset. To make sure the trusted boot cannot be bypassed on reset, this must be protected from untrusted components.

R60_PSBG_BOOT: If the platform has a programmable reset address, then the trusted firmware must protect this from untrusted components. This might be achieved using a locking or memory protection mechanism.

Any processor cache used by the immutable bootloader needs to be invalidated before it is used.

R70_PSBG_BOOT: Either the hardware or the immutable bootloader must invalidate caches before using them

In the case where the immutable bootloader must invalidate the processor cache, the specific sequence of actions to invalidate it are typically provided in the corresponding technical reference manual.

If a warm reset is supported, then the immutable bootloader should be able to determine whether the system performed a warm reset or a cold reset. This might be achieved by checking a variety of platform specific registers.

Arm recommends enabling an available watchdog timers as soon to detect prolonged .

3.8 Unlocking

The image signer of the NSPE might decide to disable Trusted Boot of the NSPE either at factory provisioning stage or at a later point in the product lifecycle.

R80_PSBG_MANIFEST: If either the NSPE-PK or the NSPE image hash are authentic and equal to zero, then the NSPE image does not require authentication in order to execute.

The SPE runtime software might require information about the status of the NSPE. For instance, a Trusted service might use the verification status of the NSPE as a factor in an access control policy.

3.9 Image encryption

Images might contain secrets that must not be readable by attackers. When encryption is desired there are three options for decryption keys:

- A full pre-shared symmetric key
- A symmetric key derived using a key derivation function (KDF), which is based on a small pre-shared secret
- An asymmetrically encrypted symmetric key, provisioned dynamically with the encrypted payload

The choice of key and algorithm is IMPLEMENTATION DEFINED. Arm recommends an authenticated encryption algorithm, such as AES-GCM, which has the additional advantage of detecting tampering at the point of decryption.

Implementations must also consider protections against side-channel attacks.

R10_PSBG_ENCRYPTION: Each image containing secrets must be encrypted.

Arm recommends that each image is encrypted with a separate key.

3.10 Rollback protection

If a firmware image is updated to fix security vulnerabilities, and the system permits the firmware image to be “rolled back” to a previous, insecure version, then a security risk exists. Therefore, firmware must use non-volatile (NV) version counters to protect against rollback attacks.

For validating untrusted software, each Trusted Boot stage must use one of the following mechanisms for providing rollback protection for images:

- **Secure on-chip NVM**: Counters can be implemented in on-chip storage only accessible to the SPE.

- **On-chip OTP memory**: Counters can be implemented using on-chip OTP memory if enough individual eFuses are available. Care should be taken to ensure that enough updates can be supported for the lifetime of the product.
- **Security subsystem:** Counters might be provided by a security subsystem. If the security subsystem is off-chip then the security subsystem must support an MTP counter that is cryptographically paired with the SoC. The pairing is required in case an attacker replaces the security subsystem with one that contains lower counter values.
- **Authenticated off-chip memory**: Implementations might use a single on-chip counter to secure off-chip version counters. This can be achieved by storing an authenticated table of counters in untrusted storage. The table must be authenticated using a key derived from the HUK (to prevent cloning) and combined with the on-chip counter (for replay protection). The on-chip counter must be incremented on every update to the table. This option is not robust against flash erasure or corruption.

Counters must never be updatable to a value less than their current value. A counter must also never overflow. If the maximum value is reached it must remain at that value.

After a firmware image is verified, the image version number taken from the signed manifest and is compared with the corresponding stored counter. If the value is:

- Less than the 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 with the higher value

R10_PSBG_ROLLBACK: Only images of a higher version or the same version can be executed.

It is recommended to implement as many version counters as there are images, where each image can use a separate counter without affecting other images. However, the number of rollback counters that can practically be supported is implementation dependent.

R20_PSBG_ROLLBACK: Each software stage must not be able to decrease their corresponding rollback counter. It is permitted for the trusted firmware to be exempt from this rule if it is responsible for preserving the integrity of counters.

On TBSA complaint hardware it is expected for boot software to use the provided NV counters. On systems that are not TBSA complaint, trusted on-chip NVM might be used for counter values providing that the trusted firmware can ensure their correct operation. This protects the counters against modification from low skilled physical attacks but not from a runtime error or software attack that can revert the counter value.

R30_PSBG_ROLLBACK: Rollback counters must be implemented either with on-chip OTP memory, a security subsystem, or a private on-chip NVM region which is only write-accessible to the Trusted Boot software.

Rollback counters might also be required to support version control of other software. A suitable implementation might employ one counter per software instance, or group together a list of version numbers inside a database file, which is itself versioned using a single counter.

R40_PSBG_ROLLBACK: Rollback counters must never overflow. If the maximum value is reached it must remain at that value.

If a rollback counter is implemented using on-chip OTP memory, such as eFuses, then a lower bound on the number of supported updates must be specified.

R50_PSBG_ROLLBACK: Each rollback counter used to validate SPE software must support at least 64 values. If the SPE consists of multiple boot stages, then it is recommended that each stage has a dedicated counter for each verification step.

A rollback counter is increased when newer software is loaded. In some markets it can be desirable to perform a boot test of the image before increasing the version counter. One simple example of a boot test mechanism is a watchdog that tests whether an update is unresponsive. A more complex example might involve testing the network stack or update system to ensure that network connectivity has not been broken.

R60_PSBG_ROLLBACK: If the counter value in the manifest is greater than the rollback counter, and if the manifest is authentic, then the rollback counter must eventually be increased to match the counter value in the manifest.

It is IMPLEMENTATION DEFINED when a rollback counter is increased:

- An implementation can decide to perform a boot test of a new image before incrementing the rollback counter. When implementing staged update with failure rollback, counters must only be updated when the installed image has been successfully tested.
- An implementation can decide to support a mechanism to control rollback protection by remote messaging. However, it must not be possible to use any remote management feature to increase an rollback counter on a device to a value beyond the highest version of any images currently loaded on the device. Such a mechanism requires:
    - Authentication of the command issuer with at least the same cryptographic properties as that used for image signing.
    - Replay protection, ensuring that any issued command instance can only be acted on only once by a device.

It is IMPLEMENTATION DEFINED when a rollback counter is reset:

- An implementation can decide to reset the anti-rollback mechanism following factory reset. This allows devices to be recovered if the anti-rollback mechanism becomes desynchronized with the signer.
- An implementation can decide to use a remote messaging protocol as described above.

R70_PSBG_ROLLBACK: Any implemented mechanism to reset rollback protection must be at least as secure as the image signing mechanism

3.11 Measured boot and attestation

An SoC might need to prove the integrity of its software to a remote party or to local systems on the same board.

A prerequisite for attesting the platform state is to create measurements of loaded code and data on each boot. The measurements are then securely stored either in the trusted firmware or a security subsystem. This is known as a measured boot. Any measured boot mechanism must assure the integrity of such firmware and make it part of an overall chain of trust. Each stage of the chain of trust must accurately and robustly measure all the critical code and data that will be loaded. This also includes:

- Loadable modules (including dynamic patches and code loaded from peripherals)
- Parameters that influence boot behavior (for example, flags or variables that might change the control flow of the loaded program)

Each stage of the chain of trust must store the measurements in a local root of trust. The measurements might be held in a security module or by a trusted firmware. A remote party can use the list of measurements to help validate the specific software identity of the platform.

A platform might have one or more security subsystems, such as a Secure Element, Trusted Platform Module, or a cryptographic accelerator, for example. They might include updateable firmware, which must be measured.

R30_PSBG_ATTEST: All firmware loaded into security subsystems must be measured and verified at boot. The measurements must be included in the boot state.

The boot state must be accessible to a trusted service. This might be stored in on-chip NVM or on-chip RAM. The specific format and way to pass information is considered IMPLEMENTATION-DEFINED.

R60_PSBG_ATTEST: The boot state must be stored in an on-chip memory area, which is only accessible to a trusted service. It is also permitted for the boot state to be stored within a security subsystem.

Implementation note:

A security subsystem, such as a Trusted Platform Module (TPM), can store measurements made by the boot software. With a TPM this is achieved by programming the TPM's Platform Configuration Registers (PCRs) using the *PCR extend* TPM operation. The convention for using PCRs is specific to the device class and is being standardized by the Trusted Computing Group (TCG).

For implementations based on the Unified Extensible Firmware Interface (UEFI) the convention for using TPM PCRs is described in the TCG Client profile [13]. Servers and embedded systems might also use the same profile to ease interoperability.

It is possible for systems to provide multiple images in the form of a signed firmware image package or a single image. A firmware image package allows for packing bootloader images (and potentially other payloads) into a single archive that can be loaded. Nevertheless, each component must be measured independently. This is necessary for a remote party to easily verify a remote attestation.

R70_PSBG_ATTEST: All images and configuration files must be measured separately, even if they are in one firmware image package.

The state of the hardware might have a direct effect on the security of the system. As an example, a bootloader stage that supports an "unlocked" state might permit third party images to be loaded after an appropriate authorization procedure. This is considered a change of state because it affects and might be recorded in the boot state for remote attestation purposes.

R80_PSBG_ATTEST: In addition to measuring the next stage, each stage must measure parameters that might influence the behavior of software.

Examples of sensitive configuration data are boot parameters and configuration data, which might be stored separately from the images. For example, the boot parameters might be loaded from a file or from a connected input device.

4 Architectural variants

The implementation of a Trusted Boot process largely depends on the security properties of various subsystems and non-volatile memories.

This section presents the common variants in two categories, the *baseline architecture* and the *assisted architecture,* as also described in the TBSA documents. The figures:

- are not to scale
- omit redundant boots stages
- do not show multiple images per stage. It is expected that more complex systems have multiple stages that contain multiple images per stage
- do not distinguish between volatile and non-volatile memory

4.1 Baseline architecture

The Baseline Architecture performs the majority of the security functions within Trusted world software in the processor. It is supported by a minimum set of required security hardware, for example:

- Immutable bootloader.
- Trusted RAM and/or Trusted External Memory Partitioning.
- Trusted peripherals
    - OTP Fuses
    - Entropy Source
    - Watchdog

The Baseline architecture focuses on ensuring that the SPE has access to all the assets it requires, and has the underlying mechanisms to protect the integrity, confidentiality, and authenticity of the SPE.

4.1.1 Boot from on-chip storage

This variant does not depend on external NVM. All images in this variant are in internal XIP flash.

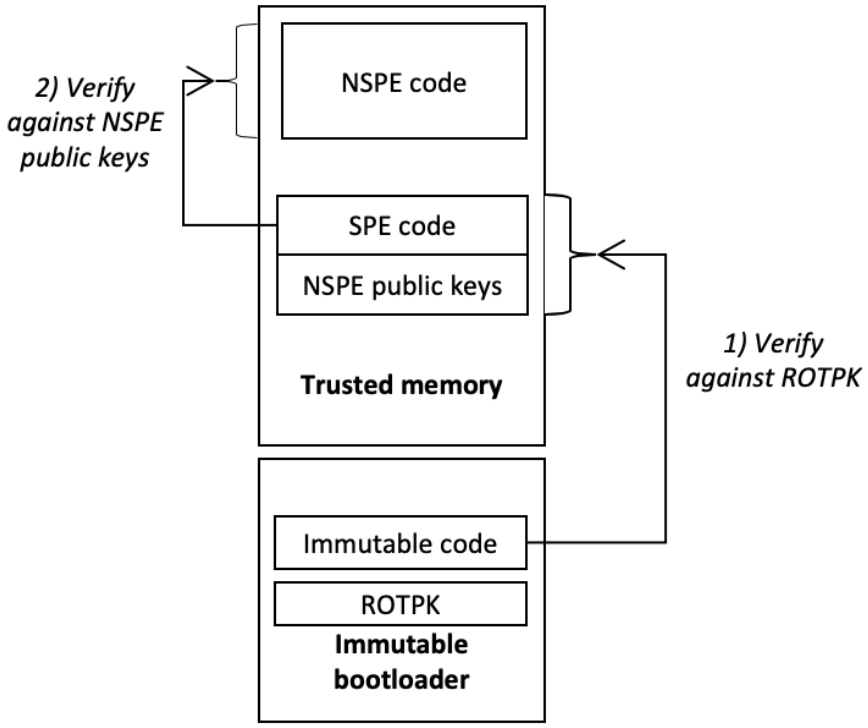

Figure 7: Example boot from trusted storage

4.1.2 Boot from off-chip storage

This variant has no on-chip NVM. Since no on-chip NVM exists for secure variables, both SPE and NSPE must be authenticated using embedded public keys on boot.

The SPE might be composed of multiple images. The SPE and NSPE might be a combined single image. Anti-rollback counters must be implemented in on-chip memory that is only accessible to the SPE.

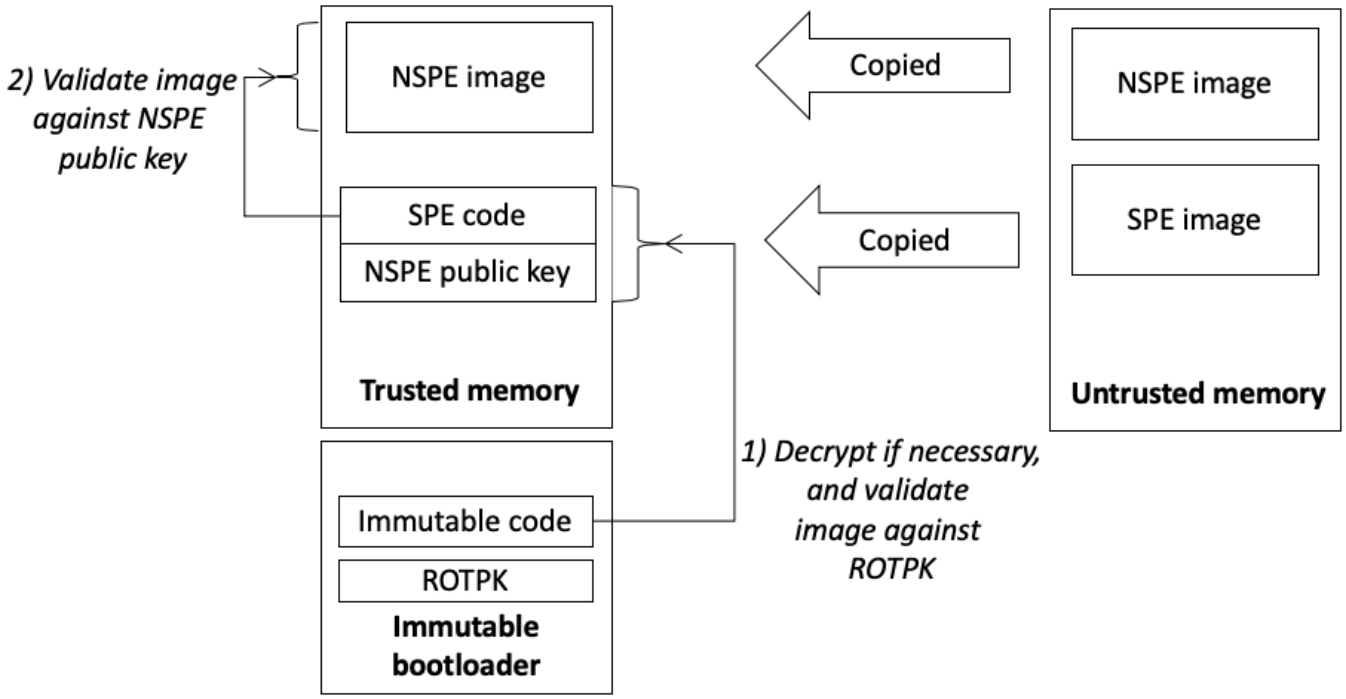

Figure 8: Example boot from untrusted memory

4.2 Assisted architecture

The Assisted Architecture builds on the Baseline Architecture by adding a security subsystem to accelerate and offload some of the cryptographic operations from the SPE software, and to provide increased protection to high value assets, such as root keys.

The cryptographic accelerators are expected to support the most commonly used algorithms for encryption, decryption, and authentication

4.2.1 Boot using a passive security subsystem

The ROTPK(s) and other hardware secrets are contained within a security subsystem.

A security module might be also known as a secure element or a secure enclave. For example, a Trusted Platform Module (TPM), smart card, or a generic security processor. The security module might be on-chip or off-chip.

If the security module is off-chip then it must be cryptographically paired with the SoC during device assembly. Communications over exposed buses must be encrypted using a key established from the cryptographic pairing process.

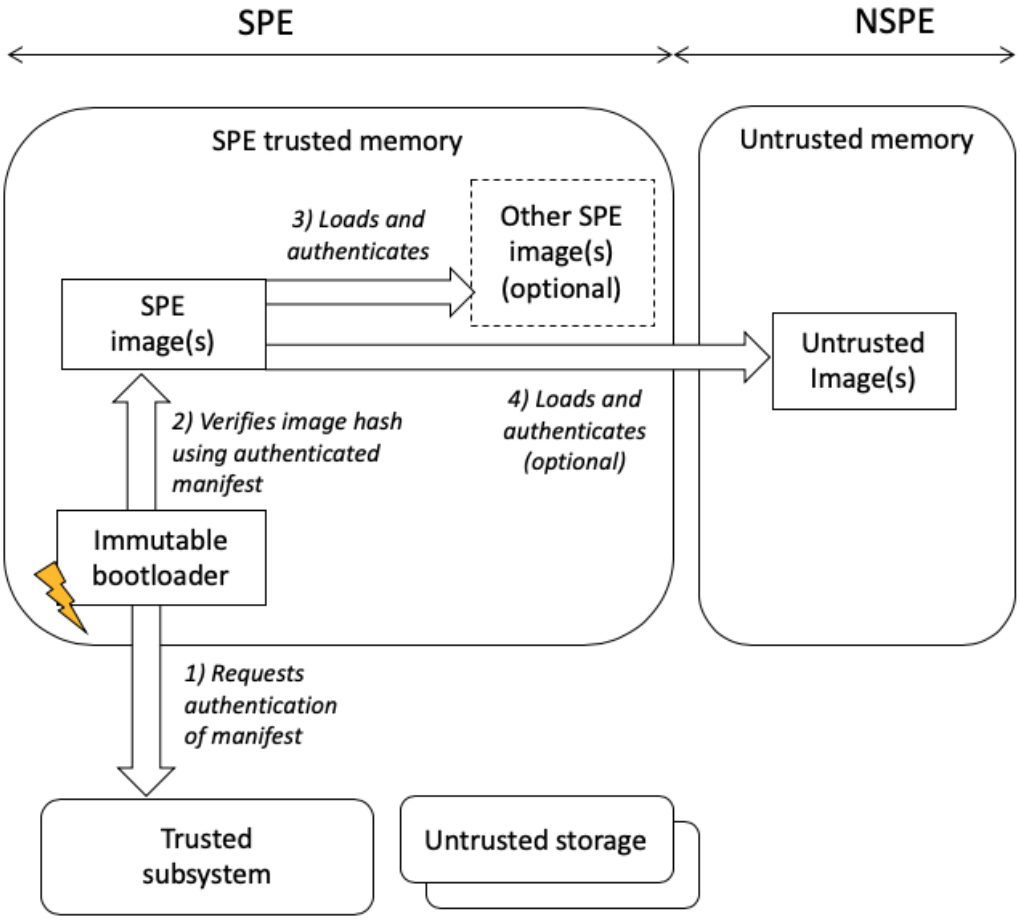

Figure 9: Example boot using a passive security subsystem

4.2.2 Boot using an on-chip security subsystem

The on-chip security subsystem is on-chip and might be, for example:

- An isolated system control processor (SCP)
- Manageability control processor (MCP)
- A Secure Enclave or advanced version of an integrated Secure Element

The ROTPK(s) and other hardware secrets are contained within the security processor.

The security module might be a secure enclave or a separate processor that controls the boot process.

It has the capability to verify the initial SPE image in the internal flash. It is possible for a security module to verify multiple images. If the contents are successfully verified, then the main CPU is released, and execution begins.

The non-volatile counters used for rollback protection of the SPE are stored within the security module. The main CPU then boots the SPE software.

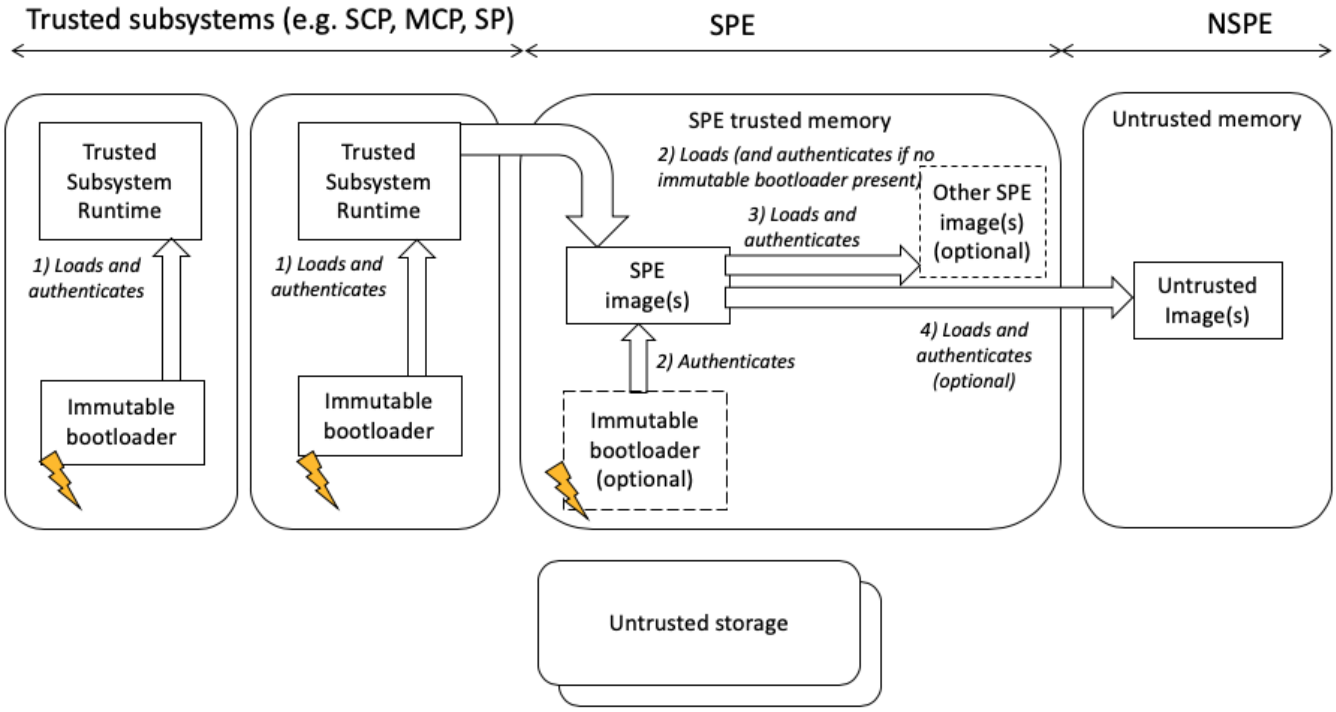

Figure 10: Example boot using a security subsystem that directly loads and verifies images in the SPE

5 eFlash considerations (informational)

SoCs are used in different market segments and have varying security requirements based on their usage models. Usage models can be part of open or closed software ecosystems. System designers must consider which assets they need to protect, and which threats they want to protect against. This specification describes the required rules to reduce the impact of software-based attacks. It also provides recommendations, where appropriate, to prevent scalable low-cost physical attacks.

The requirements for this document are derived from best security practices. To understand threats and general security objectives, refer to the Threat Models and Security Analyses (TMSA) provided by Arm.

Offline modification of off-chip flash memory is a likely risk if an attacker has physical access to the system. It is common for external off-chip storage to be present, such as an SD card or NOR flash.

Many consumer systems include a method of linking with a PC, for example a USB connection. This is an example of simple equipment that any attacker who is local to the system would have. Beyond this, an attacker might utilize more specialized equipment that can be easily acquired and that is relatively inexpensive. Examples of this equipment are JTAG interface controllers, soldering irons, and oscilloscopes. To perform the most sophisticated attacks, an attacker might require expensive laboratory-like equipment or software that must be specifically developed.

Internal on-chip flash that can easily be reprogrammed or erased using programmers is also considered as untrusted storage. Microcontrollers can offer different levels of protection for internal flash. It is important to understand the implications of these protections for both development and production scenarios. Special configuration registers might provide the following modes for the internal flash:

- **Internal protection**: This is typically used to protect a bootloader or any parts of the system to be overwritten either by the debugger or by the application itself. This protection can be reversed either by

a mass erase capability or reprogramming a special configuration register. If the registers cannot be protected from application software, then this does not provide security against software-based attacks.

- **External protection**: Prohibits external tools or debuggers from accessing the flash. Designed to prevent reverse engineering of software. To recover the system a mass erase has to be applied. If the register cannot be protected from application software, then this does not provide security against software-based attacks.
- **Disable mass erase**: Prevents triggering of a mass erase. Combined with **Internal protection** and **External protection**, the system can never be manually reprogrammed. If mass erase is enabled and triggered then all system code, secrets and data are erased. If mass erase is enabled, this marks the end of the security lifecycle.

If the registers for these states cannot be protected from untrusted application software, then this does not provide security against software-based attacks. The configuration registers for these modes must be protected from application software.

6 Delegated signing schemes (optional)

The security of the Trusted Boot process is primarily dependent on the secrecy of the private portion of the ROTPK. If the ROTPK private key is leaked or lost, then no recovery is possible. To mitigate this risk, the private portion of the ROTPK should have limited exposure, while the attack surface is moved to a delegated signing key of lesser authority. By limiting the exposure of the ROTPK's private key, strict operational processes can be put in place around its usage, which reduces the possibilities for attackers.

Implementations with higher robustness requirements should have a scheme in place to limit the exposure of the private key. The signing of key certificates should take place in a secure offline environment. It should be noted that operational processes are very dependent on business processes, which are not within the scope of this specification. However, this section will describe a scheme that implementations might wish to consider.

Figures are provided to help illustrate the dependencies in such schemes. The provided examples only consider one level of delegation. However, the examples can easily be extended to include intermediate certificate authorities.

6.1 Key certificate scheme

In this scheme a key certificate is used. A key certificate is signed metadata that explicitly contains the identities of each delegated signing authority. The identities are represented by the hashes of each signer's public key (see "Public key hash" within the Key Certificate of Figure 12).

The ROTPK private key is used to sign the key certificate. It is expected that this event is very rare.

Figure 11 shows an example of this scheme that includes a root authority with a delegated image signing key. The public key associated with the image in Figure 12 is called the "Image public key". It should be noted that the example could have more intermediary signers to further reduce the privilege of each signer.

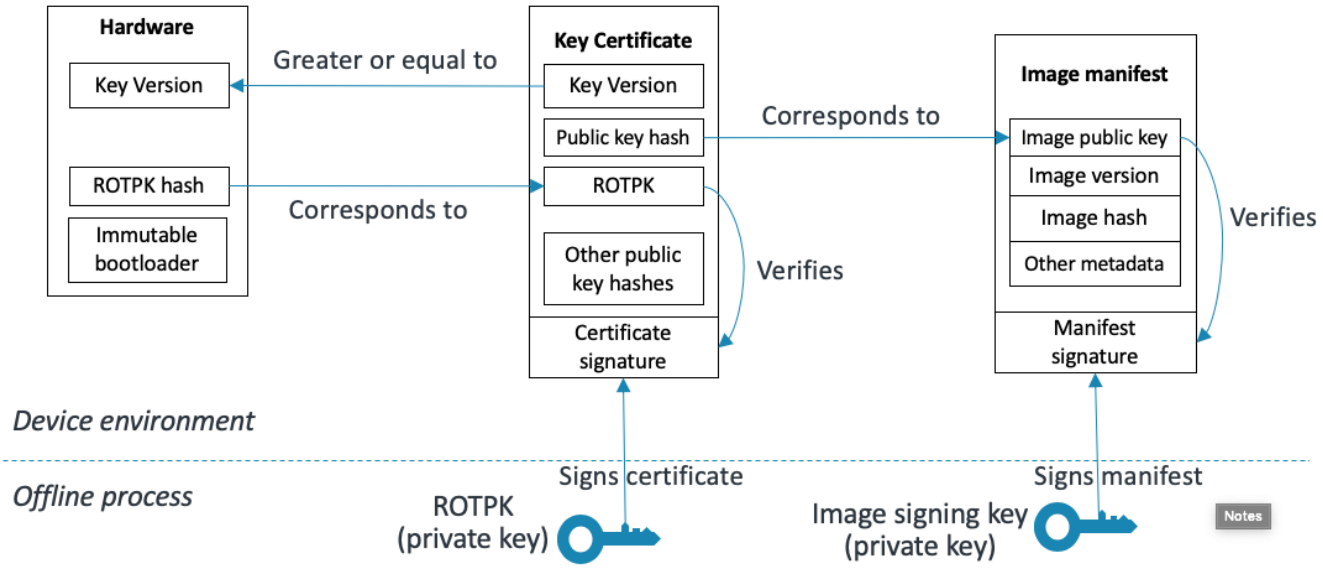

Figure 11: Key certificate example with a single image signer

Implementation note:
The image public key does not need to be embedded in the image manifest. It might be embedded in the Key Certificate itself or a separate file that is checked against the public key hash.

6.1.1 Revocation workflow

If the image signer loses control of the private signing key, then the revocation process is as follows:

1. The image signer must generate a new key pair, keeping the private key secret.
2. The public key of the new key pair is given to the holder of the ROTPK using a secure process. The ROTPK owner validates the image signer using multiple alternative factors of authentication.
3. The holder of the ROTPK creates and signs a new key certificate, which includes:

    - an incremented Key Version number
    - the hashes of all delegated signer identities (their public keys), including the public key hash of the new keypair. The old public key hash associated to the image signer is removed and not included.
    - the policy of each signer (which public key is allowed to sign which type of image)

4. All image signers must be notified by the ROTPK owner of the new changes
5. The new key certificate is sent to all image signers and must be included with the latest firmware update

The process is the same for each signer if they require revocation.

R10_PSBG_DELEGATION: On systems that support subsidiary keys, a compromised image signing key must be revoked as long as the private key of the ROTPK is not compromised, by signing a new key certificate with an incremented key version value. The use of subsidiary keys is not mandatory.

R20_PSBG_DELEGATION: On systems that support subsidiary keys, the key version of the key certificate must be compared to an on-chip non-volatile counter to detect rollback of old keys.

7 Cryptographic algorithms (informational)

The cryptographic algorithms that are needed and their strength depend on the assets and the target requirements. This section describes the recommended cryptographic algorithms and sizes. The specific purpose of the required keys, such as a Root of Trust Public Key (ROTPK) are described in Section 3.2. Cryptographic best practice demands that the same key is not used for multiple purposes.

This document recommends firmware to use one of the public key cryptography algorithms for the ROTPK.

The choice between algorithms for cryptographic signatures might be made for patent related reasons, hardware cost, signature size and time required for authentication. The use of ECC for asymmetric cryptography is often beneficial because its smaller key sizes lessens storage and transmission requirements. System architects should also review the comparative performance of implementations in terms of throughput for each of the relevant key use cases. It should be noted that the speed of a signing operation might significantly differ from the speed of a signature verification operation. For this specification, only the latter is required for Trusted Boot and update. Since the image signing process is likely to be done in an offline environment and on a more capable machine, implementers might favor speed of signature verification since this can significantly affect boot time.

7.1.1 RSA

The RSA-PSS signature scheme signs a hash of a message instead of the message directly. This technique is often 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, and so Arm considers signatures to be the same length as keys.

For RSA signatures, it is recommended that the RSA-PSS signature scheme is used.

In the absence of a compliance profile it is recommended to use the key size defined in the TBSA documents [2,3].

7.1.2 ECC

Any use of ECDSA might want to follow the FIPS 186-4 standard [FIPS PUB 186-4]. The Commercial National Security Algorithm (CNSA) Suite recommends the P-384 curve as a parameter. Alternatively, the Standards for Efficient Cryptography Group (SECG) also provide recommendations for curves and parameters.

In the absence of a compliance profile it is recommended to use the key size defined in the TBSA documents [2,3].

7.1.3 EdDSA

Any use of EdDSA should conform to IETF RFC 8032 [RFC 8032].

In the absence of a compliance profile it is recommended to use the key size defined in the TBSA documents [2,3].

7.1.4 Hashing

In the absence of a compliance profile, all cryptographic hashes in this document must use either:

- Secure Hash Algorithm 2 (at least 256 bits). Systems that are expected to be in the field for a long time are recommended to use the stronger algorithm SHA-384, which is suggested by the CNSA suite.
- SHA-3 (at least 256 bits)

7.1.5 Key derivation

For Key Derivation Functions (KDF) it is recommended to follow NIST's recommendations as specified in NIST 800-108 [8].

7.1.6 Side channels

Shared secrets that are used for encryption and decryption can be vulnerable to side-channel key-recovery attacks. Public key authentication, as required by this specification, is unaffected because no secret is required to authenticate firmware. However, ensuring confidentiality of assets remains an issue.

Cryptographic algorithms can be implemented with hardware dedicated engines or completely in software. Both might be protected against non-invasive side-channel attacks. The software implementation of the cryptographic primitives might want to ensure that their execution have the same processing time and cache footprint for every code path in the cryptographic algorithm.

8 Update process (informational)

Some devices require protection against failure of a new image by retention of a known good image, normally the current image. This implies sufficient NVM to store two images. The simplest case is when both images might be stored on the device in eFlash, in which case the eFlash has to be dimensioned for two image slots, a primary slot and a secondary slot. The same principle can be applied for external flash. Following the download and processing of a new image the update client of the device is responsible for programming the new image into the secondary slot and arranging for the device to be rebooted. Images that have been provisioned to storage are known as candidate images.

The update process might fetch 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.

The update procedure might consist of the following stages:

1. Fetching signed manifests and their corresponding firmware images
2. Authenticating the firmware images using the manifests to check their provenance and integrity
3. Authorizing updates against a device security policy
4. Installing the images into persistent storage

A high availability use case might require candidate SPE images to be write-protected from the NSPE at all times. This ensures that certain images cannot be overwritten. This can be achieved if the SPE configures the appropriate hardware isolation mechanisms correctly.

A device security policy is used by an updater to authorize the firmware update. It might include information about:

- the public keys that must be used to verify specific images

- whether the update is targeted for that specific device
- whether the device has enough power to perform all the required steps
- rate limiting the frequency of updates in order to prevent premature write exhaustion of flash

There might be restrictions set by any installed device management software.

An update server holding firmware images should not be trusted for providing any security. The only data the updater needs to trust is manifest information signed by a trusted authority. Therefore, image updates can be propagated in different fashions. For instance, in a mesh topology of IoT or edge devices it might be convenient to distribute and host image updates via a local available gateway or through a decentralized distribution mechanism. Similarly, a network service provider might have an elaborate and dynamic content distribution network for provisioning updates to different regions resulting in fast updates and less network pressure.

For rich devices, Arm recommends OEMs implement the UEFI Capsule Update interface to give operating systems a standard way of updating firmware.

Appendix A: Example manifest using the IETF SUIT draft

The IETF SUIT working group is developing a standard firmware manifest specifically for constrained devices. Arm recommends this standard for microcontrollers and embedded processors. This following mapping table is based on the **draft-ietf-suit-manifest-03** document available from the IETF. It is recommended to use the latest version published by the IETF.

Table 3 Example mapping of requirements to SUIT fields

PSA-PSBG field	IETF-SUIT field	Description
Manifest version	manifest-version	
Image version	manifest-sequence-number	Used for rollback protection
Image hash	digest-bytes	
Image size	image-size	

Appendix B: Example manifest using the X.509v3 standard

Application processors, such as the Cortex-A series, have sufficient resources to handle industry standard X.509 certificates within a reasonable computation budget. This section provides an example interpretation of the PSA-PSBG manifest requirements using the X.509 standard.

The fields are part of the X.509 content certificate. Some fields defined in this specification are hence represented as X.509v3 certificate extensions.

DER encoding is recommended.

Table 4: Example mapping of requirements to X.509v3 fields

PSA-PSBG field	X.509 field	Extension name	Extension criticality	Extension size (bytes)	Description
Image hash value	Extension	FirmwareHash	1	>=32	SHA-256 is 32 bytes SHA-512 is 64 bytes etc
Image size	Extension	FirmwareSize	0	>=32	
Image type	Extension	FirmwareType	1	1	Identifies the component type. The value is IMPDEF.
Image version	Extension	FirmwareNVCounter	1	4	Used for anti-rollback protection. Legacy implementations might use: "TrustedFirmwareNVCounter" or "NonTrustedFirmwareNVCounter"
Vendor/class ID	Certificate Issuer name Subject name	n/a	n/a	n/a	Used to make sure that an update is not intended for another device.
Manifest version	n/a	n/a	n/a	n/a	An X.509 certificate is already versioned.

Appendix C: Checklist

Ref name	Description
R10_PSBG_KEYS	The system must include at least one firmware verification public key known as a Root of Trust Public Key (ROTPK) and a Hardware Unique Key (HUK)
R20_PSBG_KEYS	Each ROTPK must be immutable. These can be stored using a locked on-chip flash sector, a secure element, or on-chip OTP memory. It is permitted to store an immutable hash of each ROTPK to check the integrity of ROTPKs in untrusted storage.

Ref name	Description
R00_PSBG_EXEC	Each loaded SPE image must be authenticated before use. It is recommended to authenticate NSPE images.

R10_PSBG_EXEC	Any use of a MAC to re-key and authenticate a firmware image manifest must be in the form of a HMAC, CMAC, or GMAC signature. The key might either be in trusted memory or in a security subsystem.
R20_PSBG_EXEC	Any caching of authenticated manifests must be held in trusted memory and write protected from untrusted components.
R30_PSBG_EXEC	Secrets used by a trusted component X must be scrubbed from volatile memory and registers before ownership of the memory is transferred to a component not trusted by X.
R40_PSBG_EXEC	Each loaded SPE image must be verified in trusted memory before execution. It is permitted for NSPE images to be loaded into untrusted memory.
R50_PSBG_EXEC	The boot process must be uninterruptible during signature verification to prevent race conditions.
R60_PSBG_EXEC	The boot process must not execute a component if a security violation occurs.
R70_PSBG_EXEC	The update process must be an atomic operation. If interrupted, then the update process must either revert to the prior state or enter a recovery mode.
R80_PSBG_EXEC	If the NSPE software is to be executed on a secondary processor, the secondary processor must be kept in reset or halted until the NSPE firmware has been verified by the SPE.
R85_PSBG_EXEC	Isolation mechanisms must be correctly configured before the NSPE begins execution.
R95_PSBG_EXEC	Any available I/O protection mechanisms must be enabled to integrity protect loaded images from untrusted peripherals.

Ref name	**Description**
R10_PSBG_STORAGE	An SPE image in untrusted memory must be copied to trusted memory for authentication.
R20_PSBG_STORAGE	When an image in untrusted memory is copied into trusted memory, it must be integrity checked after the copy has completed. The integrity check must match the authentication data for the image.
R30_PSBG_STORAGE	Authentication data used to verify images must be in trusted memory before use.
R40_PSBG_STORAGE	Encrypted images in untrusted memory must be decrypted into trusted memory and authenticated in trusted memory.

Ref name	**Description**

R20_PSBG_BOOT	The immutable bootloader must verify loaded images using the ROTPK itself or a delegated key. It is permitted for the immutable bootloader to only verify a single firmware image containing all remaining verification functionality
R40_PSBG_BOOT	It must not be possible to debug the immutable bootloader while Trusted Boot is enabled.
R60_PSBG_BOOT	If the platform has a programmable reset address, then the trusted firmware must protect this from untrusted components. This might be achieved using a locking or memory protection mechanism.
R70_PSBG_BOOT	Either the hardware or the immutable bootloader must invalidate caches before using them.

Ref name	**Description**
R10_PSBG_MANIFEST	Each image must be associated with a signed manifest. It is permitted for the manifest to be appended to the image itself and then signed together.
R20_PSBG_MANIFEST	Each image manifest must contain the properties specified in Table 2. It is permitted for additional information to be included.
R40_PSBG_MANIFEST	For delta updates, the image manifest must also contain the hash of the final expected image state.
R50_PSBG_MANIFEST	All data in the image manifest must be digitally signed using asymmetric cryptography.
R60_PSBG_MANIFEST	Image updates that include security enhancements or vulnerability fixes must increase the software version when signing the manifest.
R80_PSBG_MANIFEST	If either the NSPE-PK or the NSPE image hash are authentic and equal to zero, then the NSPE image does not require authentication in order to execute.

Ref name	**Description**
R10_PSBG_ENCRYPTION	Each image containing secrets must be encrypted.

Ref name	**Description**
R10_PSBG_ROLLBACK	Only images of a higher version or the same version can be executed.
R20_PSBG_ROLLBACK	Each software stage must not be able to decrease their corresponding rollback counter. It is permitted for the trusted firmware to be exempt from this rule if it is responsible for preserving the integrity of counters.
R30_PSBG_ROLLBACK	Rollback counters must be implemented either with on-chip OTP memory, a security subsystem, or a private on-chip NVM region which is only write-accessible to the Trusted Boot software.

R40_PSBG_ROLLBACK	Rollback counters must never overflow. If the maximum value is reached it must remain at that value.
R50_PSBG_ROLLBACK	Each rollback counter used to validate SPE software must support at least 64 values. If the SPE consists of multiple boot stages, then it is recommended that each stage has a dedicated counter for each verification step.
R60_PSBG_ROLLBACK	If the counter value in the manifest is greater than the rollback counter, and if the manifest is authentic, then the rollback counter must eventually be increased to match the counter value in the manifest.
R70_PSBG_ROLLBACK	Any implemented mechanism to reset rollback protection must be at least as secure as the image signing mechanism

Ref name	**Description**
R10_PSBG_ATTEST	On each reset, the immutable bootloader must create a boot state.
R20_PSBG_ATTEST	The boot state must include calculated cryptographic measurements of loaded images and configuration files prior to execution. The immutable bootloader might also measure the installed NSPE image.
R30_PSBG_ATTEST	All firmware loaded into security subsystems must be measured and verified at boot. The measurements must be included in the boot state.
R60_PSBG_ATTEST	The boot state must be stored in an on-chip memory area, which is only accessible to the trusted firmware. It is also permitted for the boot state to be stored within a security subsystem.
R70_PSBG_ATTEST	All images and configuration files must be measured separately, even if they are in one firmware image package.
R80_PSBG_ATTEST	In addition to measuring the next stage, each stage must measure parameters that might influence the behavior of software.

Ref name	**Description**
R10_PSBG_DELEGATION	On systems that support subsidiary keys, a compromised image signing key must be revoked as long as the private key of the ROTPK is not compromised, by signing a new key certificate with an incremented key version value. The use of subsidiary keys is not mandatory.
R10_PSBG_DELEGATION	On systems that support subsidiary keys, the key version of the key certificate must be compared to an on-chip non-volatile counter to detect rollback of old keys.

Appendix D: Detailed change log

Version	Change
1.0 Beta 0	• IETF SUIT manifest format is now recommended for constrained devices. • X.509v3 certificates are now recommended for application processors. • Clarification - NSPE software is permitted to be loaded into off-chip memory • Update client section has been renamed to Update process, and is now informational • Manifest requirements have been separated from the update section
1.0 Beta 1	• New PSA terminology section introduced as Section 2. • Rules no longer use on-chip/off-chip terminology. Trust boundaries are defined and used in the rules to refer to trusted/untrusted peripherals/memory/storage. • Image hash of encrypted images must be of ciphertext not the plaintext. • Removed Section 6.2 • Removed security epoch as it is no longer defined in the PSA • Updated rollback section to allow for implementation-defined rollback schemes • Introduced *R25_PSBG_EXEC* to be explicit about system behavior during security violations. • Example manifests are included in the appendix for the IETF SUIT and ITU X.509v3 standards. • Simplified the figures in Chapter 4. • Introduced *R80_PSBG_MANIFEST* about disabling Trusted Boot for the NSPE. • *Introduced R30_PSBG_KEYS and R40_PSBG_KEYS*
1.0	• Document restructure • Improved definition of terminology • Scope subsection now clearly describes threats and mitigations that are covered • “Approved algorithms” section has been renamed, relabeled as informational, and moved to the end of the document. Implementers should check local government regulations and advisories on cryptographic best practices. • Update process (informational) section moved to the back of the document. • New section on reset protection, which also introduces R70_PSBG_BOOT to address invalid cache state for processors that include caches • Elaborate Unlocking chapter • Section 4 is now aligned with TBSA’s baseline and assisted architecture definitions • Added R10_PSBG_ENCRYPTION to make encryption requirements explicit • Removed R30_PSBG_BOOT and R90_PSBG_EXEC due to redundancy • Change direction of arrows in Figure 12 to represent a more realistic flow • Rule table added

	• Renamed PRoT to PSA RoT • Reduce emphasis on PSA RoT where not needed • Modified R20_PSBG_ATTEST to explicitly include measurements of configuration files • Removed R50_PSBG_STORAGE
1.1	• ‘Image metadata’ section has been moved to an earlier part of the document • Added cache invalidation on reset • Introduced Invasive subsystems term in PSA terminology section • Relabeled “Scenarios as “Supply chain scenarios” • Relabeled NVCounter to FirmwareNVCounter • Rephrased R40_PSBG_BOOT • Updated SUIT reference and guidance • Ciphertext hashes and plaintext hashes are permitted for image manifests • Added clarifications about encrypted images