Platform Security Requirements 1.0

Document number:	DEN 0106
Release Quality:	Beta
Issue Number:	0
Confidentiality:	Non-Confidential
Date of Issue:	10th July 2024

Contents

About this document

Release Information

The change history table lists the changes that have been made to this document.

Date	Version	Confidentiality	Change
July 2024	BET0	Non-confidential	Beta release. Alignment with text and terms from other PSA and PSA Certified documents.
Oct 2020	ALP-0	Non-confidential	First alpha-quality release.

Platform Security Requirements

Arm Non-Confidential Document License ("License")

This License is a legal agreement between you and Arm Limited ("**Arm**") for the use of Arm's intellectual property (including, without limitation, any copyright) embodied in the document accompanying this License ("**Document**"). Arm licenses its intellectual property in the Document to you on condition that you agree to the terms of this License. By using or copying the Document you indicate that you agree to be bound by the terms of this License.

"Subsidiary" means any company the majority of whose voting shares is now or hereafter owner or controlled, directly or indirectly, by you. A company shall be a Subsidiary only for the period during which such control exists.

This Document is NON-CONFIDENTIAL and any use by you and your Subsidiaries ("Licensee") is subject to the terms of this License between you and Arm.

Subject to the terms and conditions of this License, Arm hereby grants to Licensee under the intellectual property in the Document owned or controlled by Arm, a non-exclusive, non-transferable, non-sub-licensable, royalty-free, worldwide License to:

(i) use and copy the Document for the purpose of designing and having designed products that comply with the Document;

(ii) manufacture and have manufactured products which have been created under the License granted in (i) above; and

(iii) sell, supply and distribute products which have been created under the License granted in (i) above.

Licensee hereby agrees that the Licenses granted above shall not extend to any portion or function of a product that is not itself compliant with part of the Document.

Except as expressly licensed above, Licensee acquires no right, title or interest in any Arm technology or any intellectual property embodied therein.

The content of this document is informational only. Any solutions presented herein are subject to changing conditions, information, scope, and data. This document was produced using reasonable efforts based on information available as of the date of issue of this document. The scope of information in this document may exceed that which Arm is required to provide, and such additional information is merely intended to further assist the recipient and does not represent Arm's view of the scope of its obligations. You acknowledge and agree that you possess the necessary expertise in system security and functional safety and that you shall be solely responsible for compliance with all legal, regulatory, safety and security related requirements concerning your products, notwithstanding any information or support that may be provided by Arm herein. In addition, you are responsible for any applications which are used in conjunction with any Arm technology described in this document, and to minimize risks, adequate design and operating safeguards should be provided for by you.

Reference by Arm to any third party's products or services within this document is not an express or implied approval or endorsement of the use thereof.

THE DOCUMENT IS PROVIDED "AS IS". ARM PROVIDES NO REPRESENTATIONS AND NO WARRANTIES, EXPRESS, IMPLIED OR STATUTORY, INCLUDING, WITHOUT LIMITATION, THE IMPLIED WARRANTIES OF MERCHANTABILITY, SATISFACTORY QUALITY, NON-INFRINGEMENT OR FITNESS FOR A PARTICULAR PURPOSE WITH RESPECT TO THE DOCUMENT. Arm may make changes to the Document at any time and without notice. For the avoidance of doubt, Arm makes no representation with respect to, and has undertaken no analysis to identify or understand the scope and content of, third party patents, copyrights, trade secrets, or other rights.

NOTWITHSTANDING ANYTHING TO THE CONTRARY CONTAINED IN THIS LICENSE, TO THE FULLEST EXTENT PERMITTED BY LAW, IN NO EVENT WILL ARM BE LIABLE FOR ANY DAMAGES, IN CONTRACT, TORT OR OTHERWISE, IN CONNECTION WITH THE SUBJECT MATTER OF THIS LICENSE (INCLUDING WITHOUT LIMITATION) (I) LICENSEE'S USE OF THE DOCUMENT; AND (II) THE IMPLEMENTATION OF THE DOCUMENT IN ANY PRODUCT CREATED BY LICENSEE UNDER THIS LICENSE). THE EXISTENCE OF MORE THAN ONE CLAIM OR SUIT WILL NOT ENLARGE OR EXTEND THE LIMIT. LICENSEE RELEASES ARM FROM ALL OBLIGATIONS, LIABILITY, CLAIMS OR DEMANDS IN EXCESS OF THIS LIMITATION.

This License shall remain in force until terminated by Licensee or by Arm. Without prejudice to any of its other rights, if Licensee is in breach of any of the terms and conditions of this License then Arm may terminate this License immediately upon giving written notice to Licensee. Licensee may terminate this License at any time. Upon termination of this License by Licensee or by Arm, Licensee shall stop using the Document and destroy all copies of the Document in its possession. Upon termination of this License, all terms shall survive except for the License grants.

Any breach of this License by a Subsidiary shall entitle Arm to terminate this License as if you were the party in breach. Any termination of this License shall be effective in respect of all Subsidiaries. Any rights granted to any Subsidiary hereunder shall automatically terminate upon such Subsidiary ceasing to be a Subsidiary.

The Document consists solely of commercial items. Licensee shall be responsible for ensuring that any use, duplication or disclosure of the Document complies fully with any relevant export laws and regulations to assure that the Document or any portion thereof is not exported, directly or indirectly, in violation of such export laws.

This License may be translated into other languages for convenience, and Licensee agrees that if there is any conflict between the English version of this License and any translation, the terms of the English version of this License shall prevail.

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

Arm Limited. Company 02557590 registered in England.

110 Fulbourn Road, Cambridge, England CB1 9NJ.

Arm document reference: PRE-21585

version 5.0, March 2024

References

This document refers to the following documents.

Table 1: References

Ref	Document Number	Author(s)	Title
[1]	JSADEN014	PSA Certified	Platform Security Model 1.1
[2]	DEN 0128	Arm	Platform Security Model 1.1
[3]	DEN 0063	Arm	Firmware Framework for M
[4]	DEN 0077A	Arm	Firmware Framework for Arm® v8-A
[5]	NIST SP 800-90A, NIST SP-800-90B NIST SP-800-90C	NIST	A) Recommendation for the Random Number Generation using Deterministic Randon Bit Generators B) Recommendation for the Entropy Sources Used for Random Bit Generation C) Recommendation for Random Bit Generator (RBG) Constructions
[6]	AIS 20/31	BSI	Random Number Generation
[7]	FIPS 140-3 or ISO/IEC19790	NIST ISO/IEC	Security Requirements for Cryptographic Modules
[8]	DEN 0072	Arm	Platform Security Boot Guide
[9]	SP 800-22	NIST	A Statistical Test Suite for Random and Pseudorandom Number Generators for Cryptographic Applications
[10]	SP 800-57	NIST	Recommendation for Key Management: Part 1 - General
[11]	Semiengineering.com		The Benefits of Anti-fuse OTP
[12]	JSADEN0112	PSA Certified	Platform Threat Model and Security Goals v1.0
[13]	SOG-IS Crypto Working Group		SOG-IS Crypto Evaluation Scheme Agreed Cryptographic Mechanisms
[14]	Arm		Cache Speculation Side channels

Terms and abbreviations

This document uses the following terms and abbreviations.

Table 2: Terms and abbreviations

Term	Meaning
BDK	Boot Decryption Key
Cryptographic hash	A one-way function which maps data of arbitrary size to a bit string of fixed size.
DMA	Direct memory access, meaning some non-CPU mechanism that can read or write to memory, typically, also accessible by the CPU.
DPM	Debug Protection Mechanism
HUK	Hardware Unique Key
IAK	Initial Attestation Key
In-package	In the same physical package. Where package decapsulation and probing attacks are out of scope, this term can be read as on-chip.
JTAG	Joint Test Action Group debug interface
MTP	Multiple-time programmable
NVM	Non-volatile memory
OEM	Original Equipment Manufacturer
On-chip	On the same physical die. Where package decapsulation and probing attacks are out of scope, this term can be read as in-package.
OTP	One-time programmable, for example, using anti-fuse or eFuse technologies
PE	Processing Element, or more generally, a processor
PSA	Platform Security Architecture
RMA	Return Merchandise Authorization
RPMB	Replay Protected Memory Block
RoT	Root of Trust
ROTPK	Root of Trust Public Key (also known as a Boot Validation Key)
SE	Secure Element, typically a discrete chip that provides cryptographic operations and key storage with enhanced security robustness.
SEn	Secure Enclave, typically an on-chip IP with functionality similar to a Secure Element and enhanced robustness.

SBF	Secure Boot Firmware
SoC	System-on-Chip, a single die or multiple die within the same physical package.
SRAM	Static RAM
SWD	Serial Wire Debug Port
TLS	Transport Layer Security
TPM	Trusted Platform Module, typically V2.0 as defined by Trusted Computing Group
TRNG	True Random Number Generator
TRTC	Trusted Real-Time Clock
Trusted world	An isolated environment that supports services that need to be trusted to ensure the security of the platform.
Non-trusted world	An environment that supports services that do need to be trusted for the security of the platform

Potential for change

The contents of this specification are subject to change.

Conventions

Typographical conventions

The typographical conventions are:

Italic Introduces special terminology and denotes citations.

bold Denotes signal names, and is used for terms in descriptive lists, where appropriate.

`monospace`

Used for assembler syntax descriptions, pseudocode, and source code examples.

Also used in the main text for instruction mnemonics and for references to other items appearing in assembler syntax descriptions, pseudocode, and source code examples.

SMALL CAPITALS

Used for some common terms such as IMPLEMENTATION DEFINED.

Also used for a few terms that have specific technical meanings and are included in the Glossary.

Red text

Indicates an open issue.

Blue text

Indicates a link, which can be:

• A cross-reference to another location within the document.

• A URL, for example http://infocenter.arm.com.

Numbers

Numbers are normally written in decimal. Binary numbers are preceded by `0b`, and hexadecimal numbers by `0x`.

In both cases, the prefix and the associated value are written in a monospace font, for example `0xFFFF0000`. To improve readability, long numbers can be written with an underscore separator between every four characters, for example `0xFFFF_0000_0000_0000`. Ignore any underscores when interpreting the value of a number.

Feedback

Arm welcomes feedback on its documentation.

Feedback on this book

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

- The title (Platform Security Requirements).
- The number and issue (DEN 0106 1.0 Beta 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 specifies the minimum security requirements expected of System-on-Chips (SoC) found in many market segments. It is intended for chipset architects, designers, and verification engineers to support security-by-design. It can be used for chipset designers requiring compliance with various security requirements, or for the process of certification schemes, for example PSA Certified™, through security evaluation laboratories.

This document does not specify a specific system architecture or the use of specific components. Other documentation from Arm provides guidance on how to best meet the security requirements using the Arm architecture and system IP.

System designers are encouraged to check conformance to the security requirements so that a specific implementation fulfils the objective.

This document uses the term Trusted world to refer to an isolated environment (enforced by hardware) that hosts trusted services. The term Non-trusted world refers to any environment that hosts services that do not need to be trusted.

Isolation is fundamental to building a secure platform. In some applications a two-way partitioning, for example, to isolate a single Trusted world from a single Non-trusted world, is sufficient. However, the need for multiple Trusted worlds can arise when there is either explicit mistrust between Trusted services, or, more generally, there is no dependency between Trusted services that must co-exist on the platform. In such cases, one Trusted world perceives any other Trusted world, or worlds, as part of the Non-trusted world. The Platform Security Model [1][2], illustrates some possibilities.

Unless necessary to illustrate a specific point, the rest of this document assumes a two-way Trusted and Non-trusted partitioning. The requirements are intended to apply also when there are multiple Trusted worlds, multiple Non-trusted worlds, or both.

2 Security goals

The Platform Security Model [1][2] outlines the important principles of a secure system in the form of ten *security goals*. These goals, which are based on [12], are not Arm specific, but are inherently embodied in various Arm specifications and are used as the basis for developing the detailed security requirements within this document.

2.1 Unique identity

In order to interact with a specific device instance, that instance must be uniquely identifiable. The identity must be attestable and that attestation verifiable as a means of proving the device identity, see section 2.3.

2.2 Security lifecycle

A system must ensure that the protection of assets and the availability of device functions follow a prescribed and constrained path from manufacture to device disposal. Therefore, the system must have a state machine that it can use to make appropriate security decisions within a particular context. This is known as a *security lifecycle*.

The security state of a device within its security lifecycle depends on software versions, run-time measurements, hardware configuration, status of debug ports, and on the product lifecycle phase. Product lifecycle phases include, for example, development, deployment, returns, and end-of-life. Each security state defines the security properties of the device. The security state must be attestable, see section 2.3, and may impact access to bound data, see section 2.9.

2.3 Attestation

A system must be able to provide evidence of its trustworthiness to relying parties. To have validity, the system must be part of a governance program. Such a program includes roles such as evaluation labs, attestation verifiers, and relying parties.

For the trustworthiness of a device to be established, its identity, see section 2.1, and security state, see section 2.2 are proven through attestation.

2.4 Authorized software

A system can only be trustworthy if it runs the software that has been analyzed. Secure boot (also referred to as verified boot) and secure loading processes are necessary to ensure that only authorized software can be executed on the device. See also section 2.6. Allowing unauthorized software is acceptable only if such software cannot compromise the security of the device.

2.5 Secure update

Device software, credentials, programmable hardware configuration, must be updateable to resolve security issues or to provide feature updates. Updates must not compromise the device security. Authentication of an update is required. However, execution of any updated software must be authorized in accordance with section 2.4. The update process itself must be secure against abuse.

2.6 Rollback protection

Updates are necessary to resolve known security issues, or provide feature updates, see section 2.5. Preventing unauthorized rollback, known as anti-rollback, to a previous version with a known (and subsequently fixed) vulnerability is essential. However, authorized rollback for recovery purposes may be allowed.

2.7 Security by isolation

It is probable that software contains flaws that can be exploited to compromise the security of a system (see sections 2.4 and 2.5). Isolation of a trustworthy service from less trusted or untrusted services is essential to protect the integrity of that service. More generally, isolation boundaries aim to prevent one service from compromising other services, for example, between any on-device services and between on-device services and the connected world.

Example software architectures that use security by isolation are the firmware frameworks detailed in [3][4].

2.8 Secure interfaces

Interaction over isolation boundaries, see section 2.7, is essential if isolated services are to serve a purpose. Any such interaction must not be able to compromise the interacting services or device. This will require validation of exchanged data. It may also be necessary to ensure the confidentiality and integrity of any data exchanged.

2.9 Data Binding

Sensitive data, for example, user or service credentials, or secret keys, must be bound to a device to prevent disclosure outside of the device. It may also be required to bind such data to prevent disclosure beyond its owner. Inherently secure storage (typically on-chip with secure access controls) or confidentiality and integrity assured storage (typically off-chip with reliance on cryptography) may be used. Where binding relies on cryptography and keys, see section 2.10, the keys are sensitive data and so must be bound to the device or the data owner. It may also be necessary to bind the data to the security state, for example, to deny access during debug, see section 2.2.

2.10 Trusted services

Trusted services must ensure that other goals are met.

Trusted services may include configuration of the hardware to support security lifecycle (see section 2.2), isolation (see section 2.7), and cryptographic services that may use bound secrets (for example, keys) used to support attestation (see section 2.3), secure boot and secure loading (see section 2.4), and binding of data (see section 2.9). The trusted services must be kept as small as possible to enable analysis and reduce the likelihood flaws.

3 Scope

The classes of threats considered in this document are listed in Table 3, those that are specifically not considered are listed in Table 4. However, when the device is subject to any security certification, such as PSA Certified, the requirements of that scheme take precedence. An Attack Methods document is available on PSACertified.org, which gives examples of the types of attacks that are in scope at the various PSA Certified certification levels. Awareness of these examples can guide the design solutions.

Table 3: In scope threat classes

Threat	Summary
T.ROGUE_CODE	An attacker succeeds in loading and executing rogue code on the device in order to obtain assets or escalate privileges.
T.TAMPERING	An attacker replaces, or tampers with, off-chip storage, memory or peripherals in order to obtain assets or escalate privileges.
T.CLONING	An attacker with physical access reads data in off-chip storage or memory. This enables reverse engineering or cloning of assets to other systems.
T.DEBUG_ABUSE	An attacker succeeds in accessing debug features in order to illegally modify system behavior or access assets.
T.WEAK_CRYPTO	An attacker breaks the cryptography used by the device in order to access assets or impersonate the device. This threat only relates to algorithm strength, key size, and random number generation.
T.IMPERSONATION	An attacker pretends to be the device in order to intercept assets that are provisioned to the device.
T.POWER_ABUSE	An attacker abuses power management controls using software in order to access assets.
T.SOFT_SIDE_CHANNELS	An attacker uses software-observable side channels to infer information about assets.

Table 4: Out of scope threat classes

Threat	Summary
T.INVASIVE_ATTACK	An attacker uses invasive techniques, in which systems are physically unpackaged and probed, in order to recover assets or modify system behavior.
T.GLITCHING	A physically present attacker uses power, clock, temperature, and energy glitch attacks that cause faults such as instruction skipping, malformed data in reads/writes, or instruction decoding errors.

Threat	Summary
T.PHYS_SIDE_CHANNELS	An attacker infers the value of sensitive on-chip code or data by using physical non-invasive techniques, such as differential power analysis or timing attacks. An example asset can be a cryptographic key.
T.DENIAL_OF_SERVICE	An attacker damages an asset or prevents an asset from being accessed.
T.SUPPLY_CHAIN	While the guidance in this document provides mitigation against potential attacks in a supply chain, such as firmware tampering, it does not directly address supply chain security.
T.APPLICATIONS	Threats to the Non-trusted world and general application security.

4 Compliance

To show compliance with this document, there must be evidence-backed documentation that shows the design meets all applicable requirements that this document describes. Typically, compliance can be demonstrated through verified output of a design review. Such compliance documentation may be a valuable input to a security certification scheme such as PSA Certified[1].

It is recommended that the design and any assessment is conducted as part of a Secure Development Lifecycle, which is becoming increasing relevant to demonstrating compliance with regulatory requirements.

The design team must provide evidence of fulfillment for each requirement. This confirmation must include justification for the compliance in the form of a brief outline, 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 system, 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 is necessary to provide more robust security. In these cases, supporting evidence must be documented alongside the requirement.

In several areas, this document provides recommendations. Where possible, these recommendations are provided to give guidance on reasonable default design choices. The threat model and functional requirements of the system are key in determining how requirements are met and the recommendations to follow. The development of a product threat model is beyond the scope of this document; however, example threat models can be found on PSACertified.org. These can freely be used as the starting point for the generation of specific threat models.

[1] https://www.psacertified.org/

5 Security requirements

At an abstract level, a system comprises a collection of assets, alongside operations that act on those assets. In this context, an asset is defined as code or data that has an owner and an intrinsic value, for example, a monetary value. All data sets are assets that are associated with a value, even if that value is zero. A data set can be any stored or processed information.

High-value assets that require protection belong to a Trusted world, while low-value assets that do not require protection should belong to a Non-trusted world. The classification, ranking, and mapping of assets to worlds depends on the product requirements, and is beyond the scope of this document.

This section describes the security requirements that an SoC meets. The requirements are described in tables and are distinct from the supporting text. The text provides additional context to ease comprehension of the rationale for each requirement. Normative requirements are described within tables. Text outside of the tables is informative.

5.1 Security Lifecycle

During its creation and use, the system progresses through a series of states. These states indicate the assets present in the system and the functionality that is available or has been disabled. With the exception of entering and exiting any debug state, progression through the states is usually controlled using a write-once mechanism.

A generic minimal security lifecycle is illustrated in Figure 1. It is expected that an actual product will contain the actual states and transitions specific to that product, Original Equipment Manufacturer (OEM) manufacturing, or market requirements.

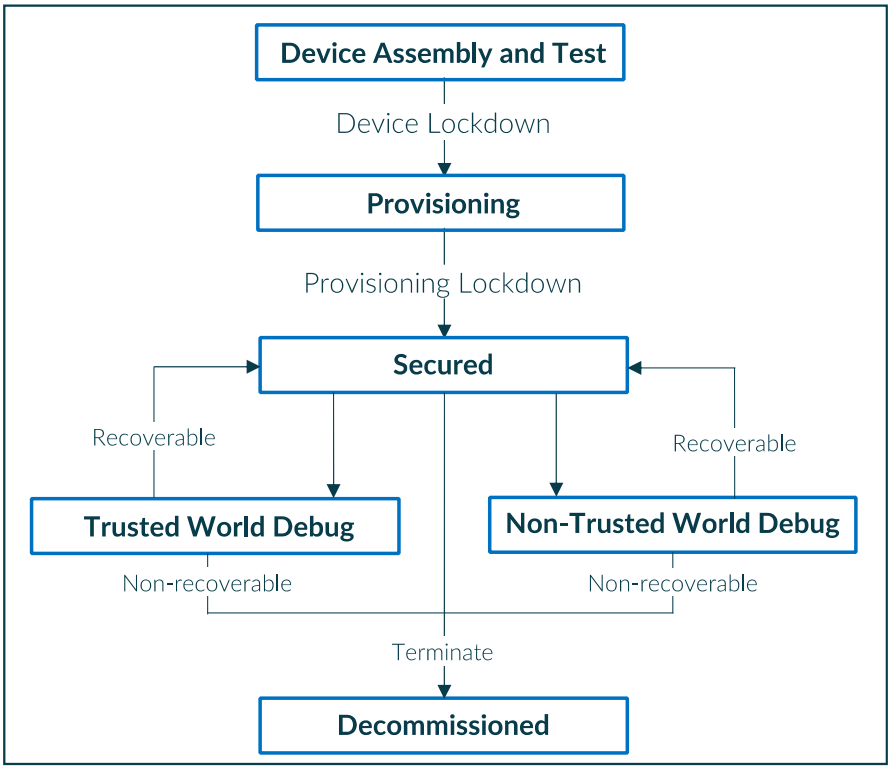

Figure 1: Generic security lifecycle

As a minimum, a system compliant with this document provides a lifecycle control mechanism in which:

- The lifecycle state is held in, or derivable from, protected or one-time programmable memory.
- All lifecycle state transitions are restricted to a designated set that includes at least:
    - A designated initial state from which the system starts.
    - A designated deployed state (Secured) which mandates the use of the system security features.
    - A designated terminal state (Decommissioned) from which no further transitions are permitted. This is also known as *Return Merchandise Authorization* (RMA).
- A transition into the Decommissioned state should put beyond use all secret and private keys through, for example, physical or logical protection or some means of atomic zeroization. The transition must also be authorized by the Root of Trust owner to prevent an attacker from erasing important secrets.
- Some systems might have the capability to hide secret and private keys during an invasive debug state such as Trusted World Debug (see Figure 1). This makes it possible for the system to go back to a 'secured' state and is represented by the "Recoverable" transition in Figure 1.
- Booting, debugging, and scan-chain access are governed by a secure lifecycle policy.

Debug support in deployed devices is not considered to be mandatory because it is common for hardware-supported debug, for example, via a JTAG or SWD port, to be permanently disabled. Where debug on a deployed device is required, the requirements in section 5.5.2 are applicable.

Table 5: Life-cycle state requirements

R010_PSR_LCYC	The system must enforce a security lifecycle.
R020_PSR_LCYC	The security lifecycle must have a designated initial state.
R030_PSR_LCYC	The security lifecycle must have a designated secured state which enforces the security requirements.
R040_PSR_LCYC	The security lifecycle must have a designated terminal state from which no further transitions are allowed.
R050_PSR_LCYC	A transition into the terminal state must put secrets and private cryptographic keys beyond use.
R060_PSR_LCYC	A transition into the terminal state must be authorized by the owner of the security lifecycle.
R070_PSR_LCYC	Where the security lifecycle does not include any debug state then any debug capability must be absent or permanently disabled.

It should be noted that the system can also contain other lifecycles that are specific to a market, application, or supply chain. These lifecycles are expected to be orthogonal and complementary to the security lifecycle described here.

5.2 Reset and Secure Boot

The secure configuration of a system depends on trusted software that forms part of a chain of trust that begins with the secure boot of the SoC. Secure boot, also known as verified boot, ensures the integrity of firmware and critical data by detecting tampering or unauthorized changes. Further details on secure boot and authentication

mechanisms can be found in the Platform Security Boot Guide [8], and in the reference implementation provided by the *Trusted Firmware*[1] project.

Secure boot is based on an immutable secure boot image. It is the first code to run on the processor core and it is responsible for verifying and launching the next stage of boot. The secure boot image is referred to by the generic term Boot ROM[2] in this document. Boot ROMs are typically implemented as some combination of mask ROM, or embedded flash or one-time-programmable memory with hardware support to ensure that once programmed cannot be altered. The executed image is inherently trusted provided the Boot ROM is on the same chip as the core that executes it. It may be acceptable for the Boot ROM to be on a separate die within the same package as the boot processor core if decapsulation and probing attacks are out of scope.

The Boot ROM contains the boot vectors for the main processing elements as well as the secure boot image. Typically, the boot loader is divided into several stages, the first of which is the Boot ROM. Later stages will be loaded from non-volatile storage into, ideally, secure RAM and executed there. In this document, the second stage boot loader is referred to as Secure Boot Firmware[3] (SBF).

Table 6: Immutable boot code requirements

R010_PSR_BROM	The SoC must have an on-chip Boot ROM with the initial code that is needed to perform a Secure Boot. Where package decapsulation and probing attacks are out of scope, the term “on-chip” can be read as in-package.

An on-chip security subsystem (see section 5.6.3) with its own private Boot ROM may be provided to co-ordinate the boot of the system. When a system reset occurs the security subsystem performs the required verification and any decryption stages prior to releasing the main application processor from reset. The main application processor should execute an image verified by the security subsystem. The main application processor may then extend the chain-of-trust by authenticating further executable images.

Careful analysis of Boot ROM code is essential because a vulnerability can undermine the entire system security. Committing all of the first stage of secure boot to immutable storage is a risk as any change requires a hardware revision. Extending the capability of the Boot ROM via an image that it loads from non-volatile storage, ideally into on-chip memory, and then authenticates is an accepted practice[4].

The Boot ROM contains sensitive code that verifies and optionally decrypts the next stage of the boot. For some devices, if an attacker were able to read and disassemble the ROM image, they could gain valuable information that could be used to target an attack that circumvents the verification mechanism. For example, timing information can be used to target a fault injection attack.

Contingent on the threat model, it might aid robustness if the Boot ROM code and data is accessible only during boot. Device designers should consider implementing a non-reversible mechanism which prevents access by, for example, erasing any volatile Boot ROM state and making the Boot ROM code and the ROTPK inaccessible using a sticky register bit that is activated by the boot software. This is an example of Temporal Isolation, a topic covered in [1] and [2].

[1] https://www.trustedfirmware.org
[2] BL1 in the Trusted Firmware project.
[3] BL2 in the Trusted Firmware project.
[4] This is called split bootloaders in the Trusted Firmware project. BL1_1 is the immutable on-chip image executed first, BL1_2 is the image loaded and verified by BL1_1. Together, these constitute the “Boot ROM”.

5.2.1 Boot keys

The Secure Boot Firmware must be authenticated by the Boot ROM using an on-chip immutable public key, which is here referred to as the Root of Trust Public Key (ROTPK)[1, 2]. The specific public key algorithm used for authentication is defined by the implementation but subject to the security requirements defined in section 5.11, or any market requirements, security certification scheme, or regulatory requirements applicable in a geographical area.

To minimize the amount of required on-chip immutable storage, an SoC may instead store the cryptographic hash of the public key, enabling the larger public key to be held in external storage. On each boot, the Boot ROM can then calculate the hash of the public key read from the external storage and compare it with the hash in on-chip memory to ensure it is the correct key.

Encryption of Secure Boot Firmware images, where needed, requires a secret Boot Decryption Key (BDK) that should be available only to the Immutable Boot ROM. It is typical for the image to be encrypted and then signed; thus decryption is only necessary if the image is successfully verified. This ordering also supports the use case where the content is considered confidential and should not be visible to the signing authority. Signing followed by encryption can be necessary where the operations are performed by different entities at different times in the supply chain; for example, where encryption is unique for each target device and is only performed when the image is to be delivered but issuing the signing requests at that time impacts the delivery timing.

Table 7: Boot key requirements

R010_PSR_BKEY	The SoC must either contain an on-chip ROTPK, or the information that is needed to securely verify the ROTPK. Such information must be immutable.
R020_PSR_BKEY	If a cryptographic hash of the ROTPK is stored in on-chip non-volatile memory, rather than the key itself, it must be immutable.
R030_PSR_BKEY	A secret Boot Decryption Key only accessible to the Immutable Boot ROM will be required if it is necessary to encrypt the Secure Boot Firmware.

It is recommended that a signature of the end-result is included where firmware images are delivered as a delta/diff. If the on-device firmware must be encrypted, then that would be applied via on-device encryption of the end-result image.

5.2.2 Boot types

A cold boot is a boot that is not based on any previous system state, and occurs on power-up, or, if already powered up, on a hard-reset input signal generated by a reset circuit, or by a software initiated reset. A warm boot is a boot that is based on previous system state in order to achieve a more rapid activation of the system than might be possible with a cold boot. A warm boot typically occurs when the SoC is powered-up and a

[1] Some markets require the ROTPK to be inaccessible once it has been used, thus reducing the amount of information exposed to potential attackers.

[2] Some markets recommend encrypting all data in external storage, including public keys, to reduce the amount of information exposed to potential attackers.

software trigger occurs, or via an input signal from, for example, a peripheral (on-chip or off-chip) monitoring relevant events.

Note that an implementation may leave some state undefined on power-up, and that a powered-on reset may not result in all state being reset. It should not be assumed that the reset types have exactly the same result.

Where a warm reboot is required, it is necessary to deploy some method to signal the use of stored state, examples include:

- The Boot ROM can distinguish between a warm boot and a cold boot via a status register.
- The SoC can use an alternate reset vector for a warm boot, causing the Boot ROM to execute warm boot specific code. Directing the SoC to use a specific may be via a status register.

Typically, any storage needed to support these mechanisms is implemented within an always-on power domain.

Table 8: Warm boot requirements

R010_PSR_BWRM	If the system supports warm boot, a flag or register must exist to distinguishing between a warm and cold boot.
R020_PSR_BWRM	Where a flag or register is used to distinguish between cold and warm boot, it must be programmable only by a Trusted world.
R030_PSR_BWRM	Where a flag or register is used to distinguish between cold and warm boot, it must be set after a cold or a warm boot has started to cold boot.
R040_PSR_BWRM	Where a flag or register is used to distinguish between cold and warm boots, the default should be for cold boot, and should use a value that any unauthorized perturbation will result in a cold boot.

Implementing a warm boot brings significant security challenges if any of the necessary retained state is security sensitive and is held in off-chip storage when the system is suspended. The threat model for the platform needs to be considered. See also sections 5.3 and 5.14.

A boot status register can be implemented to indicate the boot state of each processor. For example, the boot status register enables the application processor to check whether other processors booted up correctly and be used for attestation purposes. The register must be either immutable if accessible by a non-trusted world or accessible only by a trusted world, including any secure debug.

Table 9: Boot status register requirements

R010_PSR_BSTR	If a boot status register is implemented, it must either be accessible only by a Trusted world, including secure debug, or immutable if accessible to an un-trusted world.

5.2.3 Boot parameters

Some Boot ROM implementations can be influenced by additional configuration information stored in on-chip one-time programmable memory (OTP). Examples of configuration information include:

- Selection of the device containing the first loadable firmware image, e.g. NOR, NAND or eMMC flash.
- Storage of the ROTPK, or a hash of the ROTPK.
- Storage of a Boot Decryption Key for boot image decryption.

The effect of these parameters on Boot ROM behavior must be carefully considered for each state in the security lifecycle. Some parameter values might need to be disallowed depending on the state of the security lifecycle. For example, some factory test parameters are expected to be disabled once the provisioning lifecycle state is reached, and options to select the boot device from chip pins should de disabled when the device is considered to be in a secure life-cycle state.

Table 10: Boot parameter requirements

R010_PSR_BPRM	The Boot ROM must be aware of the current security lifecycle state.
R020_PSR_BPRM	Any Boot ROM configuration outside of on-chip OTP memory must be authenticated using an on-chip immutable public key, or on-chip immutable hash of an external public key.
R030_PSR_BPRM	It must not be possible to boot the first loadable firmware from any other storage device unless a Trusted Debug mode permits this (see section 5.5.2).

5.2.4 Boot ROM execution

Execution of the Boot ROM must not be perturbed by other agents, for example, by other processors, Direct Memory Access (DMA) mechanisms or via external interfaces (for example, PCI, JTAG, interrupts). Therefore, other processors, DMA mechanisms and external interfaces must be appropriately restricted during a secure cold or warm boot. This helps prevent secure boot from being bypassed.

Other processors, DMA and external interfaces can be re-enabled by boot software when it is safe to do so, for example after secure boot and security protections have been configured.

5.2.4.1 Secondary processors

If the SoC implements multiple processing elements (PE), the designated boot PE is called the primary boot PE. After the de-assertion of a reset the primary boot PE executes the Boot ROM code, but the remaining – the secondary - PEs should be held in reset, or a safe platform-specific state, until the primary boot PE initializes and releases them from reset. There are at least a few possible examples:

- The platform power controller can hold all secondary PEs in a reset state, while the primary boot PE executes the Boot ROM until it requests for the secondary PEs to be released.
- All PEs execute from the Boot ROM from the same boot vector. However, the Boot ROM identifies the primary boot PE and boots using the secure boot image, while the secondary PEs are made inactive in some way.

Table 11: Secondary processor requirements

R010_PSR_BSPE	All secondary PEs must remain inactive until permitted to boot by the primary PE.

5.2.4.2 DMA and External interfaces

Disabling all DMA capable IP and interrupts on or before the start of the Boot ROM secure boot process, typically on a reset, is recommended. This is a simple way to ensure compliance with Table 12 requirements, however, it is recognized that this might be too restrictive for some system designs.

Table 12: DMA and external interface requirements

R010_PSR_BEXE	Secure boot execution state must be protected from DMA reads and writes.
R020_PSR_BEXE	Secure boot execution state must be protected from external interfaces.

5.3 Clocks and power

Platforms with a high degree of power control might integrate an advanced power management subsystem using dedicated hardware, and possibly executes a small software stack from local RAM. In such cases, the management subsystem must be trusted and have control over trusted assets, for example:

- Reset examples include:
  - State machines that sequence the assertion and de-assertion of resets in relation to the reset hierarchy, the system clocks, and any power states.
  - Re-synchronization of resets at clock boundaries.
- Clock generation and selection:
  - Registers to enable or disable clocks.
  - Registers that manage clock glitch and/or frequency detectors.
  - Configuration of clock sources, including any phase-locked-loops.
  - Clock dividers and other glitch-less clock switching and clock gating mechanisms.
- Power control examples include:
  - Access to power controllers, switches, or regulators.
  - State machines for sequencing when changing power states.
  - Logic or processing to intelligently apply power states either on request, or dynamically.
- State saving and restoration. To dynamically apply power states, some subsystems can also perform saving and restoration of system states without the involvement of the main application processor.

Unrestricted access to this functionality is a security risk because it could be used by an attacker to induce a fault that targets a Trusted service by, for example, perturbing a system clock. To mitigate this threat, the advanced power mechanism belongs in a Trusted world. The system must also include a Trusted management function, to perform policy checks on any requests from any Non-trusted world before they are applied.

This approach still permits execution of most Non-trusted complex peripheral wake up code from the Non-trusted world.

If the system can be suspended, various system state will need to be saved. To prevent an attacker with physical access from modifying or reading the saved state, it must be protected using authenticated encryption. See also sections 5.2.2 and 5.14.

Table 13: Clock and power requirements

R010_PSR_PWR	Advanced power control mechanisms must integrate a Trusted management function to control clocks and power.
R020_PSR_PWR	It must not be possible to directly access reset, clock, and power management mechanisms from a Non-trusted world.

R030_PSR_PWR	If suspend to RAM is supported (see also warm boot in section 5.2.2), any protection keys for external memory need to be saved and restored. These operations must be handled by a Trusted service and the keys must be stored in either on-chip Trusted storage or wrapped using a key derived from an on-chip Hardware Unique Key (HUK).
R040_PSR_PWR	Security critical suspend state information that is stored in memory accessible to an attacker (typically off-chip or off-package) must be encrypted and authenticated using a key that is not accessible to the attacker.

5.4 Memory system

Isolation between the operations and assets of a Trusted and any other world(s) follows from the Isolation security goal, section 2.7. Operations and assets are connected by transactions, where a transaction represents an instruction fetch, a data read or write to storage containing the asset. In such a system, storage comprises registers, random access memory, and non-volatile memory. Note that the processor is not necessarily the only component on an SoC that may need to distinguish between trusted and non-trusted operations. For example, there may be DMA operations that perform operations that are specific to the security state.

The system memory map should be divided into at least two partitions, one in which Trusted world assets are held, and another in which Non-trusted world assets are held. Each transaction belongs to either a Trusted world or the Non-trusted world[1]. In some systems there may need to be more than two partitions, see section 1 and [1][2].

A Non-trusted operation must not be able to access trusted assets. However, to build a useful system it is necessary to communicate between the two worlds, usually through shared memory. Therefore, a Trusted operation must be able to access (some) Non-trusted assets, in addition to trusted assets. However, it is a security risk for a Trusted operation to fetch code belonging to a Non-trusted world, and therefore should not be permitted. This can be enforced using one of the following methods:

- Disabling or faulting[2] on instruction fetches from a Trusted world into the Non-trusted world. This can be fixed in hardware or configurable by Trusted firmware.
- Careful code review of Trusted operations to ensure secure transactions never fetch instructions from Non-secure memory.

Table 14: Trusted and Non-trusted isolation requirements

R010_PSR_MSYS	The SoC must provide a hardware-based mechanism for isolating the memories of a Trusted world from any Non-trusted world.
R020_PSR_MSYS	A Trusted world operation can access Trusted world assets and might be able to access Non-trusted world data assets.
R030_PSR_MSYS	A Trusted world operation must not fetch Non-trusted world instructions. Where hardware mechanisms to prevent such fetches exist they should be controlled only from a Trusted world.
R040_PSR_MSYS	A Non-trusted world operation must only access Non-trusted world assets.

[1] The terms Secure and Non-Secure/Normal are used in Arm TrustZone and mean trusted and non-trusted respectively.
[2] Where provided in Arm processors, the Secure Instruction Fetch bit (SIF) can be set to cause a fault.

Designs that use a network-on-chip type interconnect might have mechanisms that allow the routing of packets to be dynamically configured so that they arrive at a different interface even though the access address remains unchanged. This is a security risk as it can open the possibility of exploits. Any such configuration of routing must only be possible from a Trusted world.

It is possible to have world-aware peripherals, in which the peripheral is visible in both a Trusted world(s) and a Non-trusted world(s) at the same time. This may be achieved through address aliasing or hardware signals.

Peripheral address space must be in a non-executable area of memory.

Table 15: Programmable address mapping requirements

R010_PSR_PAM	If programmable address remapping logic is implemented in the interconnect, then its configuration must be possible only from a Trusted world.
R020_PSR_PAM	If target-side filtering is implemented to identify Trusted and Non-trusted world transactions, it must only permit Trusted or all Non-trusted transactions to any one region. Trusted and Non-trusted aliased accesses to the same address region are not permitted.
R030_PSR_PAM	The target-side transaction filters configuration space must only be accessed from a Trusted world.
R040_PSR_PAM	Configuration of the on-chip interconnect that modifies routing or the memory map must only be possible from a Trusted world unless it is not possible for such modifications to affect Trusted world transactions.

Assets from different worlds can at different times occupy the same physical storage locations. This is called shared storage. The underlying storage can be volatile, for example, on-chip RAM, external RAM, or peripheral space. The shared storage can also be non-volatile, such as flash or, if available, Magneto-resistive RAM (MRAM). Before any shared storage can be reallocated from one world to another, the asset must be securely removed, unless explicitly required to be shared. This process is called scrubbing, and can be performed by a Trusted world, using either trusted hardware or trusted software. Typically, one of the following methods:

- Overwritten with a pre-defined constant value.
- Overwritten with a random value.
- Indirectly changed to a random value, for example, by changing the secret key used to decrypt the content.

Immediately after the scrubbing process the storage contains no information, therefore, it must not be treated as data or as executable instructions.

When a copy of a Trusted world asset is held in a processor cache, it is important that the implementation does not permit any mechanism that provides any Non-trusted world with access to that asset. In effect, any cached copy needs also to be scrubbed. Typically, this means that the cache line holding the copy should be invalidated to ensure no post-scrub write-back to memory. If a hardware engine is used for scrubbing memory, careful attention must be given to the sequence to make sure that the relevant cached data is invalidated before the scrubbing operation.

Similarly, assets can be shared between software at different privilege levels within the same world. Software at each privilege level in each world is referred to as a security domain.

Table 16: Scrubbing of shared storage requirements

R010_PSR_SSS	Shared storage must be scrubbed before it can be reallocated to a different world or security domain.
R020_PSR_SSS	Shared storage must not be executable immediately after allocation from a different security domain.
R030_PSR_SSS	Assets held in a processor cache must be invalidated to ensure no post-scrubbing write-back.

5.5 Processing elements (Processors)

Most security breaches are caused by software vulnerabilities. Therefore, a key aspect of hardware system architecture is selecting and configuring security features of a host processor. The goal is to support a secure software framework which minimizes the likelihood of threats identified in the security development lifecycle of the product combining with vulnerabilities in software being exploited by an attacker during product deployment.

It is recommended that hardware security features are selected according to the software architecture and threat model of the product.

At a minimum, the SoC must ensure that:

- the execution state of a Trusted world cannot be tampered by a Non-trusted world. The implications for memory transactions and interrupts are covered in other sections.
- there is suitable hardware support to ensure that writable data in memory is never executable. This mitigates common “shellcode” exploits.
- controls are implemented to appropriately disable speculative execution[1] on processors that have this characteristic.

Table 17: Processor (PE) requirements

R010_PSR_PE	The processor must provide a hardware-based mechanism(s) for isolating the execution contexts of a Trusted world from the Non-trusted world(s).
R020_PSR_PE	The processor must provide a hardware-based mechanism(s) that ensures runtime data in memory is never executable.
R030_PSR_PE	If a processor implements features to prevent the isolation mechanisms being bypassed, they should be used and must be controlled by trusted software. Examples include speculative execution.
R040_PSR_PE	If a processor implements features to prevent side channel leakage, they should be used where leakage is identified as a concern and must be controlled by trusted software. Examples include the caches and the memory management system, and instructions that act on security critical assets where the timing is data dependent.

Other PE features should be considered beyond the base security requirements in this document, such as:

[1] Some background on this can be found in the Cache Speculation Side-channels whitepaper [14].

- Protection against *return-oriented programming* (ROP)[1] and *jump-oriented programming* (JOP)[2] attacks. The protection should prevent malicious code from executing illegal subsets of code functions.
- Detection of *use-after-free* (UAF) vulnerabilities[3]; a memory safety issue in some programming languages.

5.5.1 Interrupts and Exceptions

Each world may receive interrupts and exceptions. An interrupt or exception that is only meant to be received by a Trusted world is referred to as a Trusted interrupt (or exception). In most cases, a Trusted interrupt (or exception) must only be visible to the intended trusted world, and not be visible to any Non-trusted world. This aims to prevent information leaks that might be useful to an attacker. Consequently, the on-chip interrupt network must be able to route any interrupt to any world. However, the routing of Trusted interrupts must only be configured from a Trusted world.

When a memory access violation occurs, such as when a Non-trusted world tries to access a Trusted asset, a *security exception* or *security interrupt* is raised.

Table 18: Interrupt and exception handling requirements

R010_PSR_IEH	An interrupt or exception originating from a Trusted operation must by default be mapped only to a Trusted handler.
R020_PSR_IEH	Security interrupts or exceptions should only be handled by a Trusted world, However, where there is no security risk, security interrupts may be handled by a Non-trusted world provided R030_PSR_IEH is met.
R030_PSR_IEH	Any configuration to mask or route a Trusted interrupt or exception must only be carried out from a Trusted world.
R040_PSR_IEH	Any status flags recording Trusted interrupt events must only be readable from a Trusted world, unless specifically configured by a Trusted world to be readable by the Non-trusted world.

These requirements permit a Trusted operation to deliver a Trusted Interrupt to a Non-trusted handler, for example, to signal to the Non-trusted world the end of an operation performed by a trusted world. The configuration of the interrupt must be performed by a Trusted world before or during a Trusted operation.

Where a Non-trusted world is permitted by the trusted world to raise Secure interrupts, the Secure interrupt handler must be written carefully in order to avoid denial of service and other attacks that may lead to leakage of sensitive data.

5.5.2 Debug

A processor typically supports at least two types of debug modes:

[1] For example, Pointer Authentication is available for Arm PEs from Armv8.3-A onwards.
[2] For example, Branch Target Identification is available for Arm PEs from Armv8.5-A onwards.
[3] For example, the Memory Tagging Extension available for Arm PEs from Armv8.5-A onwards.

- External debug: The debugging occurs either on-chip (for example, in a second processor) or off-chip (for example, a debugger connected via JTAG or SWD).
- Self-hosted debug: The processor itself hosts a debugger. Developer software and debugger run on the same processor.

Deployed products may have debug via hardware interfaces permanently disabled and may prevent self-hosted debug by excluding the necessary device-side software in the production build. The implications of not being able to use code debugging on a deployed device must be considered. Note that this does not prevent the ability to host device-level diagnostics.

All active debug mechanisms need access control to prevent abuse. Access rights must be based on:

- The requestor of the debug access.
- The type of debug capability being requested.
- The current security lifecycle state.

The enforcement of these properties must be provided by an on-chip component, which is referred to as a *debug protection mechanism (DPM)*. An SoC can include one or more DPMs. A DPM authenticates each requestor using one of the following methods:

- Token-based authentication. A token containing unlock information that is signed by a trusted authority is sent to the device. The device uses a public key to check if the signature is valid before enabling debug access.
- Password-based authentication. A password is sent to the device which checks the value before enabling debug access. If an attacker is able to extract the password stored in the device then the device should instead store a cryptographic hash of the password. The device should limit the authentication attempt rate to deter a brute force attack.

Which method to use often depends on the trade-off between complexity on the device and complexity of the external debug server. For example, it is more complicated to implement signature checking on a device than to compare passwords, but managing a database of unique passwords may be more complicated than a small number of private keys on a server.

To prevent the leak of a secret from affecting multiple devices, tokens or passwords used to authenticate to DPMs should be unique for each instance.

Table 19: Debug requirements

R010_PSR_DBUG	All external debug functionality must be protected by a DPM so that only an authorized external entity can access the debug functionality.
R020_PSR_DBUG	A DPM must be implemented either solely in hardware or together with software running in a Trusted world.
R030_PSR_DBUG	A DPM must be aware of the current security lifecycle state
R040_PSR_DBUG	A DPM unlock password must be at least 128 bits in length.

Complex SoCs often include extra debug functionality beyond the main processor. Examples of this are initiators on the interconnect, which are controlled directly from an external debug interface, and system trace modules.

Care must be taken to make sure that they are controlled by the correct DPM. They must be evaluated based on their access to assets that belong to each world and assigned the corresponding DPM.

A scan chain is a mechanism to test all the flip-flops in an SoC. Scan chains are a form of debug and need to be governed by a security lifecycle to ensure they can never be accessed after a certain point. While scan chains are expected to be disabled in the factory, the specific requirements will be determined by the product.

Table 20: Scan chain requirements

R010_PSR_SCCN	Access to scan chains must be security lifecycle aware.
R020_PSR_SCCN	The coverage of a scan chain must be security lifecycle aware.

5.6 Peripherals and Security subsystems

A peripheral or subsystem is hardware that is not part of a processor (PE). It can be an integral part of the SoC or external, in which case it will be connected via an off-chip bus. In many cases the hardware is an isolated system with its own local resources, configuration and, possibly, firmware. It has an interface to receive commands and data from one or more processors (PEs) and might be capable of direct memory access (DMA).

5.6.1 Peripherals

Peripherals offer an interface allowing commands to be received that cause the peripheral to operate on assets. These might be assets of a Trusted world or a Non-trusted world depending on the functionality provided. A Trusted peripheral is one that operates on assets belonging to a Trusted world. A few types of peripheral mapping topologies are possible, see also Figure 2:

- A peripheral is mapped exclusively into one world or the other depending on its role.
- A Trusted peripheral might only act on Trusted world assets.
- A Trusted peripheral might act within both worlds, supporting both Trusted and Non-trusted operations as determined by the specific operation. An implementation-specific policy manages the separation, which might be fixed in hardware or configurable by a Trusted service.

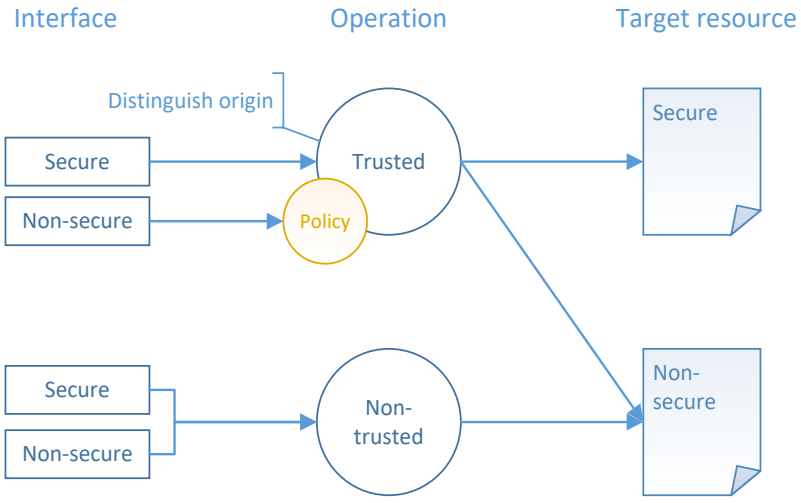

Figure 2: Peripheral operations

Interfaces can be implemented fully in hardware or mediated by a service in a Trusted world. These interfaces permit software to request operations on data. Care must be taken by the interface designer to ensure that Trusted assets and operations are isolated from the Non-trusted assets and operations.

Table 21: Peripheral requirements

R010_PSR_PER	If access to a peripheral, or a subset of its operations, is dynamically switched between a Trusted world and any Non-trusted world, then this must only be done under the control of a Trusted world.
R020_PSR_PER	A Trusted peripheral must be able to distinguish whether commands and data were received at an interface accessible to a Trusted world only, or at an interface accessible to the Non-trusted world.
R030_PSR_PER	If a Trusted peripheral stores Trusted-world assets within the peripheral, it must not be possible for a Non-trusted world to perform operations on those assets.
R040_PSR_PER	A Trusted peripheral that exposes a Non-secure interface must apply a policy check to the Non-trusted commands and data before acting on them. The policy check must be atomic and, following the check, it must not be possible to modify the checked commands or data.
R050_PSR_PER	All DMA transactions from any Non-trusted peripheral must be constrained using an on-chip mechanism configured by a Trusted-world.

When data is processed on behalf of multiple worlds, a policy is needed to constrain privileges based on the accessor. An example policy for a cryptographic accelerator peripheral would cover at least:

- The world the input data can be read from.
- The world the output data can be written to.
- Whether encryption is permitted.
- Whether decryption is permitted.

Figure 2 shows an illustration with a policy in place, where requests can be rejected if they do not comply with the policy.

5.6.2 External peripherals

SoCs will often need to communicate with external - off-chip - peripherals. Examples of such peripherals include secure elements, displays, network controllers, and interface controller hubs. Some interfaces are via simple interfaces such as I2C, SPI or UART, whereas others may be via high bandwidth controllers within the SoC, for example, PCIe or USB.

Like external storage, see Section 5.14, external peripherals may be subject to physical probing of the interface and to replacement by local attackers.

It is important to protect SoC assets from DMA and transactions that originate from outside the SoC, e.g. PCIe. Therefore, transactions must be constrained using an on-chip mechanism. The configuration of such a constraining mechanism can be fixed in hardware or configurable by firmware. The precise constraints will vary depending on the context. For instance, the boot process must configure the constraining mechanism to protect its own assets, while the runtime firmware or OS will have need to reconfigure the mechanism to protect a

different set of assets. Note that protecting SoC assets from external DMA operations is a system-integrity issue as well as security problem.

Some designs are subject to threat models in which particular hardware IP blocks could have unknown or undesirable behaviors. In these cases, additional initiator-side filters should be implemented and under sole control of a Trusted world to ensure that such IP cannot access Trusted world assets beyond that authorized by a Trusted world *policy*.

Table 22: External peripheral requirements

R010_PSR_XPER	When an external peripheral can receive commands from an external system, for example PCIe, then the system must enforce a policy to check that those commands do not breach the security of the SoC.
R020_PSR_XPER	If an external peripheral is used to send or receive clear or unauthenticated Trusted world assets, then it must meet the requirements for Trusted operations.

5.6.3 Security subsystems

A security subsystem is a peripheral that is used to operate on or store high value Trusted assets. The security services provided by a security subsystem might include one or more of the following services:

- A key for a unique, unclonable identity that is bound to hardware.
- Counters.
- Key storage and management.
- Cryptographic operations where the keys are never visible outside the security subsystem.
- Key derivation.
- True random number generation.
- Secure storage of boot measurements used as the basis for a system to perform secure attestation.
- Transparent encryption of RAM or storage

A security subsystem must be managed by a Trusted world. This ensures that the security subsystem is always available for use by a Trusted world.

If a Trusted world does not intend to use a particular security subsystem, or make some uses, it might choose to delegate the subsystem to the Non-trusted world, subject to the threat model of the final product.

If the security subsystem is off-chip then it is susceptible to bus interposition attacks or physical replacement. A compliant platform must ensure that the communication path is protected from eavesdropping. Communication may also require replay protection to ensure that an attacker cannot record and replay bus traffic. To ensure that an off-chip attack does not reduce security, there are two methods to consider:

- The SoC and security subsystem are physically tied during manufacture. For example, the subsystem can be placed within the same physical packaging as the SoC, though this depends on the physical attacks in scope in the threat model.
- The SoC and security subsystem are cryptographically bound during manufacture time. For example, during device assembly an off-chip security subsystem is connected to the host system and unique shared keys established and stored. The SoC must authenticate the security subsystem before any use. Where supported, the security subsystem should authenticate the SoC in order to prevent extraction of

stored secrets. Authentication failure is used to detect replacement of the security subsystem or the SoC.

A product threat model shall dictate more specific requirements of a security subsystem. For instance, there might be requirements necessitated by the chosen operating system vendor, by market, or by region in which the system is to operate in.

Table 23: Security subsystem requirements

R010_PSR_SUB	An off-chip security subsystem must be physically or logically inseparable from the host system. Separation must not reduce system security.
R020_PSR_SUB	Communication to and from an off-chip security subsystem must be protected against eavesdropping.
R030_PSR_SUB	Communication to and from an off-chip security subsystem must be able to detect tampering and replay attacks.
R030_PSR_SUB	A security subsystem key must not be directly accessible by any software unless a policy explicitly allows the key to be exported.
R040_PSR_SUB	A Trusted world must be able to enforce a usage policy for any security subsystem key that can be used for Non-trusted world cryptographic operations.

Examples of security subsystems include, but are not limited to, Security Enclaves (SEn), Secure Elements (SE) or Trusted Platform Modules (TPMs), DRAM protection subsystems, and security sensitive accelerators.

It is recommended that security subsystems are managed by a Trusted world to ensure that Trusted services can safely use them.

Some security subsystems can also offer increased tamper resistance against a variety of side channel attacks. Increasing protection for cryptographic keys in the system by providing a security subsystem with a hardware key store that prevents the keys from being read by both Non-trusted and Trusted software is recommended.

5.7 Invasive subsystems

Invasive subsystems include any hardware system feature or interface which could be used to compromise security properties, such as:

- JTAG debug interface.
- Boundary scan interface.
- I2C interface with access to on-chip resources.
- *Reliability, Availability and Serviceability* (RAS) and other fault detection and recovery technologies.
- Interfaces to a power management subsystem.

Table 24: Invasive subsystem requirements

R010_PSR_ISUB	An Invasive subsystem must only be controllable from a Trusted world.

5.8 Platform identity

To meet the Attestation security goal in section 2.3, the system must include an attestation key. An attestation key is a cryptographic key that proves identity, and therefore trustworthiness, to the external world. The attestation key might be used to initially provision credentials of a market specific attestation scheme, and is, therefore, called the Initial Attestation Key (IAK). An Initial Attestation Key might also be known as an *endorsement key* on some systems. It is strongly recommended to use public key cryptography, whereby the attestation key is a keypair consisting of a (secret) private key and a public key.

The manufacturer is expected to issue information about the key for the purposes of proving that a platform is genuine during remote attestation. The manufacturer vouches that the key is protected in a platform that they have manufactured. For example, the manufacturer can produce a public key certificate signed by their own certificate authority. The manufacturer in this context is the company who provisioned or generated the key. The certificate should be made available to the platform owner in order to participate with remote attestation services.

The Initial Attestation Key must be protected against cloning. If an attacker can copy the key to another platform, then they will be able to impersonate the device. This means that the key must be safely stored in a Trusted world. It is acceptable to provision the IAK through manufacturing processes or to derive it at run-time from a HUK.

For privacy sensitive deployments, such as personal devices, it is permitted for a group of devices to share the same attestation key. This provides a degree of anonymity for device owners. However, keys should only be shared within small groups to reduce the impact of a leaked key. The particular group size might depend on the size of production batches or industry standards.

Table 25: Platform identity requirements

R010_PSR_PID	The SoC must include an Initial Attestation Key that is either held within secure storage controlled by a Trusted world or held within a Security subsystem.
R020_PSR_PID	The Initial Attestation Key must be unique per instance or per batch of devices.
R030_PSR_PID	In an implementation that uses a Security subsystem for cryptographic identities, the Initial Attestation Key must only be visible to the Security subsystem.
R040_PSR_PID	The Initial Attestation Key must be protected by a security lifecycle.

Additional keys for firmware decryption and provisioning may also be included. These keys are either unique to the device or are class keys that are common across a family of devices.

5.9 Random number generation

Many cryptographic protocols depend on challenge-response mechanisms that need truly random numbers. This makes a *true random number generator* (TRNG) an important element of a secure system. There is normally a requirement that specifies the quality of the source or a set of tests that must be passed. The quality of a random source is normally described in terms of entropy. For any string of bits provided by a TRNG, the maximum entropy is achieved if all bit combinations are equally probable. Recommendations can be found in [5].

A hardware realization of a TRNG typically consists of two main components: an entropy source and a digital post-processing block, as illustrated in Figure 3.

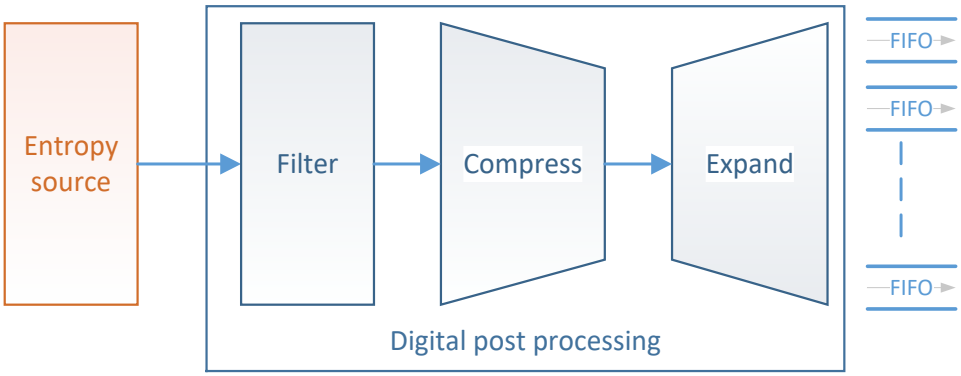

Figure 3: Entropy source top level

The entropy source incorporates the non-deterministic, entropy-providing circuitry. Constructing an on-chip entropy source might exploit die thermal noise or manufacture process variations.

Digital post-processing is responsible for collecting entropy from the source, for monitoring the quality of the data, and for filtering it to ensure a high level of entropy. For example, repeated periodic sequences are predictable and must be rejected. This is important because fault injection techniques can be used to induce predictable behavior in a TRNG.

Although a filtering scheme can remove predictable patterns in an entropy source, other more complex patterns might degrade the available entropy. The extent of any such degradation depends on the quality of the source, and in some cases additional digital processing might be required to compensate for it. A common compensation technique utilizes a cryptographic hash function to compress a long bit string of lower entropy into a shorter bit string of higher entropy. However, this comes at the expense of available data rate. To counter this, a digital post processing stage can expand the entropy source to provide a greater number of bits per second by using the filtered or compressed source to seed a cryptographically strong pseudo random sequence generator with a large period. Recommendations can be found in [5].

Each random bit generated should be used no more than once, which ensures statistical independence between samples. This applies to consecutive reads on any one interface, and for reads via different interfaces.

One or more suitably sized First-in-First-out (FIFO) buffers might be implemented to ensure short-term peak demands are met. Where there is no FIFO or the TRNG is too slow to cover peak demand, use of the TRNG to occasionally seed a Pseudo Random Number Generator (PRNG) is common.

Table 26: Random number requirements

R010_PSR_RNG	The entropy source and post processing must be an integrated hardware block.
R020_PSR_RNG	It must not be possible to monitor the entropy source output on production parts.
R030_PSR_RNG	It must not be possible to halt the entropy source output on production parts.
R040_PSR_RNG	Each bit from the entropy source must be used no more than once.
R050_PSR_RNG	Each bit derived in post-processing must be used no more than once.

There are many possible choices for measuring entropy. The required methods will be determined by the applicable certification scheme protection profile, industry or government regulations. The NIST 800-22 test suite is commonly referenced [9], but see also [6].

Although some or all of the digital post processing can be performed in software by a Trusted Service, a full hardware design is recommended.

5.10 Trusted Clock, Timer, Watchdog Timer and Real-time Clock

Various forms of trusted timer (e.g. for scheduling a lockout), trusted watchdog timer functionality (e.g. to counter denial of service) and wall clock time are typically required. Trusted timers and watchdog timers are required to provide time-based triggers to Trusted services. All the timers in this section require a trusted clock source, and in the case of wall-clock timers, a trusted source of date and time.

5.10.1 Trusted Clock Source

A trusted clock source must only be configured by Trusted software and be resistant against tampering to ensure timing validity. Clock sources can be internal or external, requiring different approaches to ensure either tamper resistance or to provide tamper detection:

- Internal clock source: the clock source is an integrated autonomous oscillator(s) on the die that cannot be easily altered or stopped without deploying invasive techniques. Such clocks must not rely on any external input.
- External clock source: the clock source is connected to the SoC via an I/O pin or pins, for example, from a clock module circuit or a crystal. In this case, an attacker can easily stop the clock or alter its frequency. Where this is the case and that threat is in scope, the SoC must implement monitoring hardware that can detect when the clock frequency is outside its acceptable range.

It is recommended that where clock monitoring hardware is implemented, the hardware provides a status register to indicate if the associated clock source is compromised. This register must be readable only from a Trusted world to prevent leakage or modification of information that may assist an attacker. To signal a clock frequency violation, it might useful if a Trusted clock monitoring hardware can generate a Trusted interrupt.

Table 27: Trusted clock source requirements

R010_PSR_TCLK	Clock sources used by a Trusted timer must be exclusively controlled by Trusted software.
R020_PSR_TCLK	Clock sources used by a Trusted timer must be resistant against tampering.
R030_PSR_TCLK	If a Trusted clock source is external, then monitoring hardware must be implemented that reports via a trusted register that the clock frequency is within acceptable bounds.

5.10.2 Trusted Timer

Trusted timers are needed to provide time-based triggers to Trusted services. The SoC must support at least one Trusted timer.

Table 28: Trusted timer requirements

R010_PSR_TTME	At least one Trusted timer must exist.
R020_PSR_TTME	A Trusted timer must only be modifiable by Trusted software. Examples of modifications include refresh, suspension, or reset.
R030_PSR_TTME	The clock source that drives a Trusted timer must be exclusively controlled by Trusted software.
R040_PSR_TTME	A Trusted timer must only produce Trusted interrupts.

5.10.3 Trusted Watchdog

Trusted watchdog timers are useful to protect against attacks. For example, where trusted service execution depends on the non-trusted scheduler, or to set a limit on how long the system can remain in an update state so that it is not used as a foothold for an attack.

In such cases, if the Trusted world task is not performed within a pre-defined time limit, some corrective action is taken, for example, a hard reset issued and the SoC restarted (see section 5.2.2).

It is recommended that a Trusted watchdog timer has the ability to signal an interrupt in advance of the corrective action, for example, to permit software to save any necessary state before the reboot.

Table 29: Trusted watchdog requirements

R010_PSR_TWDG	At least one Trusted watchdog timer must exist.
R020_PSR_TWDG	A Trusted watchdog time must only be configured by Trusted software. Examples of configuration are time intervals, corrective action options, e.g. refresh, or hard reset.
R030_PSR_TWDG	Before needing a refresh, a Trusted watchdog timer must be capable of running for a time that is long enough to complete critical pre-corrective action saving of state.
R040_PSR_TWDG	A Trusted watchdog timer must be able to trigger a reset of the SoC, after a pre-defined time. This value may be fixed in hardware or programmed by Trusted software.
R050_PSR_TWDG	A Trusted watchdog timer must implement a flag that indicates the occurrence of a timeout event that causes a warm reset, to allow post-reset software to distinguish this from a powerup cold boot. See also warm boot in section 5.2.2.
R060_PSR_TWDG	The clock source driving a Trusted watchdog timer must be exclusively controlled by Trusted software.
R070_PSR_TWDG	A Trusted Watchdog must only produce Trusted interrupts.

After a system reset, it is recommended that a Trusted watchdog timer should be started before execution of the immutable boot code transfers control to the next firmware stage.

5.10.4 Trusted Real-time Clock

Some trusted services rely on the availability of Trusted real-time clack (also known as wall-clock time), for example, it is common for Digital Rights Management (DRM) systems to authorize access to streamed media until a certain time/date, or a for a number of days.

A Trusted real-clock time (TRTC) is typically implemented using an on-chip trusted real-time counter that is synchronized securely with a remote time server.

An implementation of a TRTC might consist of a continuously powered counter driven by a continuous and accurate clock source, with Trusted time programmable only from a Trusted world. However, systems that may lose power must deal with power outages. A suitable solution can be realized by implementing a counter together with a status flag. The valid flag is set when the Trusted timer has been updated by a Trusted service and is cleared when power is removed from the timer.

When the Trusted time is lost due to a power outage, the response depends on the target specifications. For example, it might be acceptable to restrict specific Trusted services until the TRTC has been updated by the appropriate Trusted service.

Table 30: Trusted real-time clock requirements

R010_PSR_TRTC	A TRTC must be configured only by a Trusted world access.
R020_PSR_TRTC	All components of a TRTC must be implemented within the same power domain.
R030_PSR_TRTC	On initial power-up, and following any other power outage to the TRTC, a validity mechanism must indicate that the TRTC is not trusted.
R040_PSR_TRTC	The TRTC must be exclusively controlled by a Trusted world.

5.11 Cryptography

The cryptographic algorithms that are used must be strong against networked adversaries and local attackers. The specific choice of algorithms depends on the target market and any applicable regulations.

The security strength of a cryptographic algorithm is determined by the number of operations that is required to break it in some way. If the security strength associated with an algorithm or system is S bits, then it is expected that (roughly) 2^S basic operations are required to break it. It must be noted that S bits does not refer to the key length. For example, to meet 128 bits of security strength, an RSA-based key must be at least 3072 bits and an elliptic-curve-based key must be at least 256 bits.

Further information can be found in externally published documents from the cryptographic community and governments, for example, it is recommended to use approved algorithms from [10]. Alternatively, refer to the approved cryptographic algorithm lists that SOG-IS [13], IPA, and Common Criteria (CC) have published for the EU, Japan, and China.

It is strongly recommended that implemented algorithms execute in data independent time, for example, constant or random time, in order to prevent, or make harder, timing-based attacks.

Table 31: Cryptography strength requirements

R010_PSR_CRSS	Unless defined by a national or sector standard, all use of cryptography must use an algorithm that meets at least 128 bits of security.

It is important that a key is treated as an atomic unit when it is created, updated, or destroyed. This applies at the level of the requesting entity. Replacing part of a key with a known value and then using that key in a cryptographic operation makes it easier for an attacker to discover the key using a divide and conquer brute-force attack. This is especially relevant when a key is stored in memory units that are smaller than the key; for example, a 128-bit key that is stored in four 32-bit memory locations. Entities, such as trusted firmware functions, which implement creation, updating or destruction services for keys should ensure that it is not possible for their clients to observe or use keys in a manner which breaks the assumption of atomicity.

Table 32: Key atomicity requirements

R010_PSR_KATM	A key must be treated as an atomic unit. It must not be possible to use a key in a cryptographic operation before it has been fully created, fully updated, or during its destruction.
R020_PSR_KATM	Any operations on a key must be atomic. It must not be possible to interrupt the creation, update, or destruction of a key.

A cryptographic scheme provides one or more security services and is based on a purpose and an algorithm requiring specific key properties and key management. Keys are characterized depending on their classification as private, public, or symmetric keys and according to their use.

Broadly, each key should only be used for a single purpose, such as encryption, signature generation (signing), verification (integrity) check, and key wrapping. The main motivations for this principle are:

1. Limiting the uses of a key limits the potential harm if the key is compromised.
2. The use of a single key for two or more different cryptographic schemes can reduce the security provided by one or more of the processes.
3. Different uses of a single key can lead to conflicts in the way each key should be managed. For example, the different lifetime of keys used in different cryptographic operations may result in keys having to be retained longer than is best practice for one or more uses of that key.

In cases where a scheme can provide more than one cryptographic service, this principle does not prevent use of a single key. For instance, when a symmetric key is used both to encrypt and authenticate data in a single operation or when a digital signature is used to provide both authentication and integrity.

Re-using part of a larger key in a scheme that uses a shorter key, or using a shorter key in a larger algorithm and padding the key input, can leak information about the key. Such usage is prohibited.

Table 33: Key use requirements

R010_PSR_KUS	A key must only be used by the cryptographic scheme for which it was created.
R020_PSR_KUS	A key must only be used by the purpose for which it was created.

A secure SoC will need a number of keys during its operation, each with potentially different lifespans:

- A *static key* is a key that cannot change after it has been introduced to the device. It might be stored in an immutable structure like a ROM (therefore, set by the SoC Vendor) or a set of fuses programmed at the required time in the manufacture or deployment of the product. Although a static key cannot have its value changed, that does not preclude it from being revoked or made inaccessible.

- An *ephemeral key* is a key that has a short lifespan. Such keys exist only when they are required, for example, a TLS session. In many cases they will not be retained over a power or reset cycle of the device (see section 5.2). Ephemeral keys are created in the device in several ways, such from a TRNG source or via a key derivation algorithm. The use of ephemeral keys can give better protection by generating keys that are unique for every boot cycle, or each session.

A *hardware key*, either static or ephemeral, is a key that is visible only to hardware, so invisible to software. Typically, these are used for Trusted world cryptographic operations, but usage by a Non-trusted world must be subject to a trusted usage policy.

A *temporally* isolated key, either static or ephemeral, is a key that is only available at a specific point in time. For example, a bootloader can derive a key from source material, e.g., a static key, on each system reset, use the key, erase the key, and finally trigger a hardware mechanism to hide the source material until the next system reset. This allows the bootloader to have a secret that cannot be derived or used by software that is later loaded.

Table 34: Key lifetime requirements

R010_PSR_KLT	When a key is no longer required by the system, it must be put beyond use to prevent it from being revealed at a later time.
R020_PSR_KLT	A static key must be stored in an immutable structure, for example a ROM or a set of bulk-lockable fuses.
R025_PSR_KLT	Revocation of (or making inaccessible), and any re-enablement, of a static key must only be possible by trusted software.
R030_PSR_KLT	To prevent the re-derivation of previously used keys, the source material used in the derivation must be hidden either by Trusted code or Trusted hardware.
R040_PSR_KLT	If an ephemeral key is stored in memory or in a register in clear text form, the storage location must be scrubbed before being used for another purpose.
R050_PSR_KLT	A key that is accessible to, or generated by, a Non-trusted world must only be used for Non-trusted cryptographic operations. These are operations that are either implemented in Non-trusted software or have both clear text and cipher text in the Non-trusted world.
R060_PSR_KLT	A key that is accessible to, or generated by, a Trusted world can be used for operations in both Non-trusted and Non-trusted worlds, and even across worlds, if a Non-trusted world cannot access the key directly, or a Trusted world can control the use of the key through a policy.
R070_PSR_KLT	A Trusted hardware key must not be directly accessible by any software.
R080_PSR_KLT	A Trusted world must be able to enforce a usage policy for any Trusted hardware key that can be used for Non-trusted world cryptographic operations.

5.12 Secure storage

Trusted assets, such as firmware images and sensitive data, often need to be stored in external storage. The threat model may require the external memory to be protected from attackers who may try to read, modify or clone the trusted assets. Therefore, a compliant SoC must provide a secure storage solution by embedding:

- A *hardware unique key (HUK)* as the root key for encrypting and decrypting data held in external storage. The key is hardware unique so that assets cannot be cloned or decrypted on another platform. The actual key used for such encryption and decryption should be derived from the HUK.
- An on-chip Trusted non-volatile counter is required for version control of firmware and trusted data held in external storage. An important property of these counters is that it must not be possible to roll them back, to prevent replay attacks. There must be at least one counter for Trusted firmware use and at least one counter for Non-trusted firmware use.

An implementation of secure storage can be made with a Trusted service or by using a hardware approach. A hardware implementation of secure storage can be transparent to software and provide increased throughput compared to a software solution. However, a hardware implementation must conform to the requirements described in section 5.6.3.

Table 35: Secure storage requirements

R010_PSR_SST	Any sensitive data that needs to be stored must be stored in Secure storage.
R020_PSR_SST	The SoC must embed at least one hardware unique key (HUK) of at least 128 bits of entropy.
R030_PSR_SST	The HUK used as a root key for secure storage must have at least 128 bits of entropy.
R040_PSR_SST	The HUK used for secure storage must only be accessible by Trusted code or Trusted hardware that acts on behalf of Trusted code.
R050_PSR_SST	An on-chip non-volatile Trusted firmware version counter implementation must provide a counter range sufficient for the expected number of updates over the planned lifetime of the product. Where the end application is not known then 0-63 is typical.
R060_PSR_SST	An on-chip non-volatile Non-trusted firmware version counter implementation must provide a counter range sufficient for the expected number of updates over the planned lifetime of the product. Where the end application is not known then 0-255 is typical.
R070_PSR_SST	It must only be possible to increment a version counter through a Trusted access.
R080_PSR_SST	It must only be possible to increment a version counter; it must not be possible for it to be decremented.
R090_PSR_SST	When a version counter reaches its maximum value, it must not roll over, and no further changes must be possible.
R100_PSR_SST	A version counter must be non-volatile, and the stored value must survive a power down period up to the lifetime of the system.

Ideally, an SoC implements secure storage and version counters using on-chip non-volatile storage. It is recognized that Multi-Time-Programmable (MTP) storage is currently not economically viable for smaller process nodes. However, One-Time Programmable (OTP) storage, based on anti-fuse or e-fuse technology, is widely available and cost-effective, but supporting *n* updates requires *n* bits. The advantages of anti-fuse OTP technology are discussed in [11].

Flash memory devices that support Replay Protected Memory Block (RPMB) technology can provide a route to replay protection from an external storage device. Secure use of such storage requires many of the requirements in section 5.6 to be met.

5.13 On-chip Secure memory

Trusted code is expected to execute from, and store high value assets in, Secure memory. Secure memory provides confidentiality, integrity and replay protection, properties that can be provided by physical means or by cryptographic means.

A typical implementation is on-chip Static RAM (SRAM). It may be acceptable to use SRAM on a separate die within the same package as the main SoC if decapsulation and probing attacks are out of scope.

Example Secure memory use cases are:

- Secure boot code and data.
- Secure Monitor code.
- A Trusted OS or a Secure Partition Manager.
- Cryptographic services.
- Trusted services.

Secure memory refers to one or more dedicated regions that are mapped onto one or more physical RAM implementations. The mapping of regions for use by trusted code can be static and fixed by design, or programmable at runtime. When a physical RAM is not entirely dedicated to Secure memory, it can be configured to be shared between worlds. However, the underlying locations are not classified as shared volatile storage unless they are reallocated from a trusted world to a non-trusted world, see also section 5.4.

The size of the Secure memory depends on the target requirements and is therefore not specified in this document.

Table 36: Secure memory requirements

R010_PSR_SMEM	The SoC must integrate Secure memory.
R020_PSR_SMEM	Secure memory must be mapped into a Trusted world only.
R030_PSR_SMEM	If the mapping of Secure memory into regions is programmable, then configuration of the regions must only be possible from a Trusted world.

5.14 External Secure memory

Some SoC designs rely on external memory, typically DRAM, for sensitive code and data. External memory is vulnerable to probing attacks, which can be used to:

- Directly recover sensitive assets.
- Subvert the behavior of the system to extract assets.
- Use the system for illegitimate purposes.

To mitigate these risks, the protections covered in sections 5.14.1 to 5.14.3 can be applied to an asset before it is stored in external memory. The type of protection required depends on the nature of the stored asset in the context of the deployed target system. See also section 5.2.2 on warm boot implications.

5.14.1 Confidentiality protection

An attacker that can freeze external memory or use a battery-backed DRAM module can directly recover any asset in main memory. This is known as a “cold boot” attack.

Encryption ensures that assets in main memory cannot be read by physical attackers. Encryption can be transparently provided through performance-optimized on-chip cryptographic hardware blocks, each of which receives a symmetric key. Alternatively, the encryption might be provided by software that executes in on-chip Secure memory (see section 5.13), at the expense of performance and coverage. The required level of cryptographic protection depends on the target requirements and is not specified here.

When encryption is implemented, it must not be possible to decrypt a copy of the memory contents on a different device. Therefore, the keys used for encryption must be unique to the SoC. It is recommended that the keys are randomly generated on each system reset.

5.14.2 Integrity protection

Integrity protection enables the detection of external modification of DRAM content, enabling execution to be halted to prevent an attacker from exploiting any such modifications. However, cryptographic hashes need to be generated and stored (and themselves be integrity protected) on write and validated on read. Alternatively, a keyed-hash could be used provided the key is stored securely. Either way, additional storage is required and the processing degrades performance.

Integrity protection is unlikely to be practical to perform for all the memory, and so should be restricted to integrity protection of very specific assets.

The required level of cryptographic protection depends on the target requirements and is not specified here.

5.14.3 Replay protection

An attacker with the right specialized equipment might be able to capture and reproduce memory content, either by directly altering the contents in physical memory, or by interposing on bus transactions between the SoC and the external memory. However, the data necessary to detect a replay needs to be generated and stored (also integrity protected) on write and checked on read. This will consume secure storage and degrades performance.

With this capability an attacker can force an SoC to accept a piece of captured memory that passes integrity checks and correctly decrypts. For an attacker to exploit this vulnerability:

1. They must capture memory content at an address that is known to contain an insecure value or insecure configuration.
2. The attacker then “replays” this memory content when the SoC requests for this memory and at a point in time that is suitable for the attacker. This is a form of “timing attack”.
3. Once the SoC receives the memory transaction, it may operate on it, which may reduce or deactivate software defenses. For example, the memory content might contain system configuration that was previously safe but is now unsafe, which is then written into a privileged configuration register.

With the addition of replay protection, attackers cannot use captured memory to mount a timing attack.

Replay protection is unlikely to be practical to perform for all the memory, and so should be restricted to protection of very specific assets. Depending on the threat model, the use of memory encryption, section 5.14.1, and Error Correction Codes in DDR memory may provide reasonable benefit.

5.14.4 External Secure Memory Protection

The choice of confidentiality, integrity and replay protection depends on the threat model of the final system. However, there are some common rules that apply in all cases, which are listed in Table 37.

Table 37: External memory protection requirements

R010_PSR_EXTM	Keys used by a memory protection block must be unique to the SoC.
R010_PSR_EXTM	If the mapping of cryptographic hardware into the memory system is configurable, then it must only be possible to perform the configuration from a Trusted world.
R010_PSR_EXTM	The activation and deactivation of external memory protection must only be possible from a Trusted world.
R010_PSR_EXTM	If a memory region is configured for encryption, then there must not exist any alias in the memory system that can be used to bypass the encryption/decryption mechanism.

The addition of cryptographic hardware in the data path to the memory system often carries performance penalties that are typically proportional to the cryptographic strength.

The threat model should indicate if DRAM integrity protection is required. A row hammer attack exploits unintended physical side-effects of DRAM memory to change the values stored in other memory cells. Target Row Refresh (TRR) as defined by JEDEC, should be implemented to add resistance to such attacks.

Appendix A: Requirement Checklist

Reference	Section 5.1 Security Lifecycle
R010_PSR_LCYC	The system must enforce a security lifecycle.
R020_PSR_LCYC	The security lifecycle must have a designated initial state.
R030_PSR_LCYC	The security lifecycle must have a designated secured state which enforces the security requirements.
R040_PSR_LCYC	The security lifecycle must have a designated terminal state from which no further transitions are allowed.
R050_PSR_LCYC	A transition into the terminal state must put secrets and private cryptographic keys beyond use.
R060_PSR_LCYC	A transition into the terminal state must be authorized by the owner of the security lifecycle.
R070_PSR_LCYC	Where the security lifecycle does not include any debug state then any debug capability must be absent or permanently disabled.

Reference	Section 5.2 Reset and Secure Boot
R010_PSR_BROM	The SoC must have an on-chip Boot ROM with the initial code that is needed to perform a Secure Boot. Where package decapsulation and probing attacks are out of scope, the term “on-chip” can be read as in-package.

Reference	Section 5.2.1 Boot keys
R010_PSR_BKEY	The SoC must either contain an on-chip ROTPK, or the information that is needed to securely verify the ROTPK. Such information must be immutable.
R020_PSR_BKEY	If a cryptographic hash of the ROTPK is stored in on-chip non-volatile memory, rather than the key itself, it must be immutable.
R030_PSR_BKEY	A secret Boot Decryption Key only accessible to the Immutable Boot ROM will be required if it is necessary to encrypt the Secure Boot Firmware.

Requirement	Section 5.2.2 Boot types
R010_PSR_BWRM	If the system supports warm boot, a flag or register must exist to distinguishing between a warm and cold boot.

Requirement	Section 5.2.2 Boot types
R020_PSR_BWRM	Where a flag or register is used to distinguish between cold and warm boot, it must be programmable only by a Trusted world.
R030_PSR_BWRM	Where a flag or register is used to distinguish between cold and warm boot, it must be set after a cold or a warm boot has started to cold boot.
R040_PSR_BWRM	Where a flag or register is used to distinguish between cold and warm boots, the default should be for cold boot, and should use a value that any unauthorized perturbation will result in a cold boot.
R010_PSR_BSTR	If a boot status register is implemented, it must either be accessible only by a Trusted world, including secure debug, or immutable if accessible to an un-trusted world.

Requirement	Section 5.2.3 Boot parameters
R010_PSR_BPRM	The Boot ROM must be aware of the current security lifecycle state.
R020_PSR_BPRM	Any Boot ROM configuration outside of on-chip OTP memory must be authenticated using an on-chip immutable public key, or on-chip immutable hash of an external public key.
R030_PSR_BPRM	It must not be possible to boot the first loadable firmware from any other storage device unless a Trusted Debug mode permits this (see section 5.5.2).

Requirement	Section 5.2.4 Boot ROM execution
R010_PSR_BSPE	All secondary PEs must remain inactive until permitted to boot by the primary PE.
R010_PSR_BEXE	Secure boot execution state must be protected from DMA reads and writes.
R020_PSR_BEXE	Secure boot execution state must be protected from external interfaces.

Requirement	Section 5.3 Clocks and power
R010_PSR_PWR	Advanced power control mechanisms must integrate a Trusted management function to control clocks and power.
R020_PSR_PWR	It must not be possible to directly access reset, clock, and power management mechanisms from a Non-trusted world.
R030_PSR_PWR	If suspend to RAM is supported (see also warm boot in section 5.2.2), any protection keys for external memory need to be saved and restored. These operations must be handled by a Trusted service and the keys must be stored in either on-chip Trusted storage or wrapped using a key derived from an on-chip Hardware Unique Key (HUK).

Requirement	Section 5.3 Clocks and power
R040_PSR_PWR	Security critical suspend state information that is stored in memory accessible to an attacker (typically off-chip or may be off-package) must be encrypted and authenticated using a key that is not accessible to an attacker (typically on-chip, may be in-package).

Requirement	Section 5.4 Memory system
R010_PSR_MSYS	The SoC must provide a hardware-based mechanism for isolating the memories of a Trusted world from any Non-trusted world.
R020_PSR_MSYS	A Trusted world operation can access Trusted world assets and might be able to access Non-trusted world data assets.
R030_PSR_MSYS	A Trusted world operation must not fetch Non-trusted world instructions. Where hardware mechanisms to prevent such fetches exist they should be controlled only from a Trusted world.
R040_PSR_MSYS	A Non-trusted world operation must only access Non-trusted world assets.
R010_PSR_PAM	If programmable address remapping logic is implemented in the interconnect, then its configuration must be possible only from a Trusted world.
R020_PSR_PAM	If target-side filtering is implemented to identify Trusted and Non-trusted world transactions, it must only permit Trusted or all Non-trusted transactions to any one region. Trusted and Non-trusted aliased accesses to the same address region are not permitted.
R030_PSR_PAM	The target-side transaction filters configuration space must only be accessed from a Trusted world.
R040_PSR_PAM	Configuration of the on-chip interconnect that modifies routing or the memory map must only be possible from a Trusted world unless it is not possible for such modifications to affect Trusted world transactions.
R010_PSR_SSS	Shared storage must be scrubbed before it can be reallocated to a different world or security domain.
R020_PSR_SSS	Shared storage must not be executable immediately after allocation from a different security domain.
R030_PSR_SSS	Assets held in a processor cache must be invalidated to ensure no post-scrubbing write-back.

Requirement	Section 5.5 Processing elements (Processors)
R010_PSR_PE	The processor must provide a hardware-based mechanism(s) for isolating the execution contexts of a Trusted world from the Non-trusted world(s).
R020_PSR_PE	The processor must provide a hardware-based mechanism(s) that ensures runtime data in memory is never executable.
R030_PSR_PE	If a processor implements features to prevent the isolation mechanisms being bypassed, they should be used and must be controlled by trusted software. Examples include speculative execution.
R040_PSR_PE	If a processor implements features to prevent side channel leakage, they should be used where leakage is identified as a concern and must be controlled by trusted software. Examples include the caches and the memory management system, and instructions that act on security critical assets where the timing is data dependent.

Requirement	Section 5.5.1 Interrupts and Exceptions
R010_PSR_IEH	An interrupt or exception originating from a Trusted operation must by default be mapped only to a Trusted handler.
R020_PSR_IEH	Security interrupts or exceptions should only be handled by a Trusted world, However, where there is no security risk, security interrupts may be handled by a Non-trusted world provided R030_PSR_IEH is met.
R030_PSR_IEH	Any configuration to mask or route a Trusted interrupt or exception must only be carried out from a Trusted world.
R040_PSR_IEH	Any status flags recording Trusted interrupt events must only be readable from a Trusted world, unless specifically configured by a Trusted world to be readable by the Non-trusted world.

Requirement	Section 5.5.2 Debug
R010_PSR_DBUG	All external debug functionality must be protected by a DPM so that only an authorized external entity can access the debug functionality.
R020_PSR_DBUG	A DPM must be implemented either solely in hardware or together with software running in a Trusted world.
R030_PSR_DBUG	A DPM must be aware of the current security lifecycle state
R040_PSR_DBUG	A DPM unlock password must be at least 128 bits in length.
R010_PSR_SCCN	Access to scan chains must be security lifecycle aware.

Requirement	Section 5.5.2 Debug
R020_PSR_SCCN	The coverage of a scan chain must be security lifecycle aware.

Requirement	Section 5.6.1 Peripherals
R010_PSR_PER	If access to a peripheral, or a subset of its operations, is dynamically switched between a Trusted world and any Non-trusted world, then this must only be done under the control of a Trusted world.
R020_PSR_PER	A Trusted peripheral must be able to distinguish whether commands and data were received at an interface accessible to a Trusted world only, or at an interface accessible to the Non-trusted world.
R030_PSR_PER	If a Trusted peripheral stores Trusted-world assets within the peripheral, it must not be possible for a Non-trusted world to perform operations on those assets.
R040_PSR_PER	A Trusted peripheral that exposes a Non-secure interface must apply a policy check to the Non-trusted commands and data before acting on them. The policy check must be atomic and, following the check, it must not be possible to modify the checked commands or data.
R050_PSR_PER	All DMA transactions from any Non-trusted peripheral must be constrained using an on-chip mechanism configured by a Trusted-world.

Requirement	Section 5.6.2 External peripherals
R010_PSR_XPER	When an external peripheral can receive commands from an external system, for example PCIe, then the system must enforce a policy to check that those commands do not breach the security of the SoC.
R020_PSR_XPER	If an external peripheral is used to send or receive clear or unauthenticated Trusted world assets, then it must meet the requirements for Trusted operations.

Requirement	Section 5.6.3 Security subsystems
R010_PSR_SUB	An off-chip security subsystem must be physically or logically inseparable from the host system. Separation must not reduce system security.
R020_PSR_SUB	Communication to and from an off-chip security subsystem must be protected against eavesdropping.
R030_PSR_SUB	Communication to and from an off-chip security subsystem must be able to detect tampering and replay attacks.
R030_PSR_SUB	A security subsystem key must not be directly accessible by any software unless a policy explicitly allows the key to be exported.

Requirement	Section 5.6.3 Security subsystems
R040_PSR_SUB	A Trusted world must be able to enforce a usage policy for any security subsystem key that can be used for Non-trusted world cryptographic operations.

Requirement	Section 5.7 Invasive subsystems
R010_PSR_ISUB	An Invasive subsystem must only be controllable from a Trusted world.

Requirement	Section 5.8 Platform identity
R010_PSR_PID	The SoC must include an Initial Attestation Key that is either held within secure storage controlled by a Trusted world or held within a Security subsystem.
R020_PSR_PID	The Initial Attestation Key must be unique per instance or per batch of devices.
R030_PSR_PID	In an implementation that uses a Security subsystem for cryptographic identities, the Initial Attestation Key must only be visible to the Security subsystem.
R040_PSR_PID	The Initial Attestation Key must be protected by a security lifecycle.

Requirement	Section 5.9 Random number generation
R010_PSR_RNG	The entropy source and post processing must be an integrated hardware block.
R020_PSR_RNG	It must not be possible to monitor the entropy source output on production parts.
R030_PSR_RNG	It must not be possible to halt the entropy source output on production parts.
R040_PSR_RNG	Each bit from the entropy source must be used no more than once.
R050_PSR_RNG	Each bit derived in post-processing must be used no more than once.

Requirement	Section 5.10.1 Trusted Clock Source
R010_PSR_TCLK	Clock sources used by a Trusted timer must be exclusively controlled by Trusted software.
R020_PSR_TCLK	Clock sources used by a Trusted timer must be resistant against tampering.
R030_PSR_TCLK	If a Trusted clock source is external, then monitoring hardware must be implemented that reports via a trusted register that the clock frequency is within acceptable bounds.

Requirement	Section 5.10.2 Trusted Timer
R010_PSR_TTME	At least one Trusted timer must exist.
R020_PSR_TTME	A Trusted timer must only be modifiable by Trusted software. Examples of modifications include refresh, suspension, or reset.
R030_PSR_TTME	The clock source that drives a Trusted timer must be exclusively controlled by Trusted software.
R040_PSR_TTME	A Trusted timer must only produce Trusted interrupts.

Requirement	Section 5.10.3 Trusted Watchdog
R010_PSR_TWDG	At least one Trusted watchdog timer must exist.
R020_PSR_TWDG	A Trusted watchdog time must only be configured by Trusted software. Examples of configuration are time intervals, corrective action options, e.g. refresh, or hard reset.
R030_PSR_TWDG	Before needing a refresh, a Trusted watchdog timer must be capable of running for a time that is long enough to complete critical pre-corrective action saving of state.
R040_PSR_TWDG	A Trusted watchdog timer must be able to trigger a reset of the SoC, after a pre-defined time. This value may be fixed in hardware or programmed by Trusted software.
R050_PSR_TWDG	A Trusted watchdog timer must implement a flag that indicates the occurrence of a timeout event that causes a warm reset, to allow post-reset software to distinguish this from a powerup cold boot. See also warm boot in section 5.2.2.
R060_PSR_TWDG	The clock source driving a Trusted watchdog timer must be exclusively controlled by Trusted software.
R070_PSR_TWDG	A Trusted Watchdog must only produce Trusted interrupts.

Requirement	Section 5.10.4 Trusted Real-time Clock
R010_PSR_TRTC	A TRTC must be configured only by a Trusted world access.
R020_PSR_TRTC	All components of a TRTC must be implemented within the same power domain.
R030_PSR_TRTC	On initial power-up, and following any other power outage to the TRTC, a validity mechanism must indicate that the TRTC is not trusted.
R040_PSR_TRTC	The TRTC must be exclusively controlled by a Trusted world.

Requirement	Section 5.11 Cryptography
R010_PSR_CRSS	Unless defined by a national or sector standard, all use of cryptography must use an algorithm that meets at least 128 bits of security.
R010_PSR_KATM	A key must be treated as an atomic unit. It must not be possible to use a key in a cryptographic operation before it has been fully created, fully updated, or during its destruction.
R020_PSR_KATM	Any operations on a key must be atomic. It must not be possible to interrupt the creation, update, or destruction of a key.
R010_PSR_KUS	A key must only be used by the cryptographic scheme for which it was created.
R020_PSR_KUS	A key must only be used by the purpose for which it was created.
R010_PSR_KLT	When a key is no longer required by the system, it must be put beyond use to prevent it from being revealed at a later time.
R020_PSR_KLT	A static key must be stored in an immutable structure, for example a ROM or a set of bulk-lockable fuses.
R025_PSR_KLT	Revocation of (or making inaccessible), and any re-enablement, of a static key must only be possible by trusted software.
R030_PSR_KLT	To prevent the re-derivation of previously used keys, the source material used in the derivation must be hidden either by Trusted code or Trusted hardware.
R040_PSR_KLT	If an ephemeral key is stored in memory or in a register in clear text form, the storage location must be scrubbed before being used for another purpose.
R050_PSR_KLT	A key that is accessible to, or generated by, a Non-trusted world must only be used for Non-trusted cryptographic operations. These are operations that are either implemented in Non-trusted software or have both clear text and cipher text in the Non-trusted world.
R060_PSR_KLT	A key that is accessible to, or generated by, a Trusted world can be used for operations in both Non-trusted and Non-trusted worlds, and even across worlds, if a Non-trusted world cannot access the key directly, or a Trusted world can control the use of the key through a policy.
R070_PSR_KLT	A Trusted hardware key must not be directly accessible by any software.
R080_PSR_KLT	A Trusted world must be able to enforce a usage policy for any Trusted hardware key that can be used for Non-trusted world cryptographic operations.

Requirement	Section 5.12 Secure storage
R010_PSR_SST	Any sensitive data that needs to be stored must be stored in Secure storage.
R020_PSR_SST	The SoC must embed at least one hardware unique key (HUK) of at least 128 bits of entropy.

Requirement	Section 5.12 Secure storage
R030_PSR_SST	The HUK used as a root key for secure storage must have at least 128 bits of entropy.
R040_PSR_SST	The HUK used for secure storage must only be accessible by Trusted code or Trusted hardware that acts on behalf of Trusted code.
R050_PSR_SST	An on-chip non-volatile Trusted firmware version counter implementation must provide a counter range sufficient for the expected number of updates over the planned lifetime of the product. Where the end application is not known then 0-63 is typical.
R060_PSR_SST	An on-chip non-volatile Non-trusted firmware version counter implementation must provide a counter range sufficient for the expected number of updates over the planned lifetime of the product. Where the end application is not known then 0-255 is typical.
R070_PSR_SST	It must only be possible to increment a version counter through a Trusted access.
R080_PSR_SST	It must only be possible to increment a version counter; it must not be possible for it to be decremented.
R090_PSR_SST	When a version counter reaches its maximum value, it must not roll over, and no further changes must be possible.
R100_PSR_SST	A version counter must be non-volatile, and the stored value must survive a power down period up to the lifetime of the system.

Requirement	Section 5.13 On-chip Secure memory
R010_PSR_SMEM	The SoC must integrate Secure memory.
R020_PSR_SMEM	Secure memory must be mapped into a Trusted world only.
R030_PSR_SMEM	If the mapping of Secure memory into regions is programmable, then configuration of the regions must only be possible from a Trusted world.

Requirement	Section 5.14.4 External Secure Memory Protection
R010_PSR_EXTM	Keys used by a memory protection block must be unique to the SoC.
R010_PSR_EXTM	If the mapping of cryptographic hardware into the memory system is configurable, then it must only be possible to perform the configuration from a Trusted world.
R010_PSR_EXTM	The activation and deactivation of external memory protection must only be possible from a Trusted world.
R010_PSR_EXTM	If a memory region is configured for encryption, then there must not exist any alias in the memory system that can be used to bypass the encryption/decryption mechanism.