Arm® Platform Security Architecture Security Model 1.0

Architecture & Technology Group

Document number:	DEN 0079
Release Quality:	Alpha
Release Number:	2
Confidentiality	Non-Confidential
Date of Issue:	21/02/2019

Abstract

Summary

This document defines the overall security model for *Platform Security Architecture* (PSA) compliant devices.

The *PSA Security Model* (SM) defines the key goals for designing devices with essential security properties. It ties together the entities, capabilities and processes required to deploy secure services across the IoT.

The PSA SM is primarily concerned with robustness requirements, expressed in this document as robustness rules. Interfaces and technical implementation requirements for the hardware and software services and features identified by the PSA SM will be specified in separate technical specifications.

Purpose

There needs to be mutual trust between devices and *Cloud Service Providers* (CSP).

The PSA Security Model defines the foundation for establishing that trust by:

- Defining the security capabilities that CSPs can rely upon
- Providing technical input for the business commitment between different ecosystem entities
- Establishing common technical definitions and terminology

Target audience

Security communities in service providers, silicon design and manufacture, and end product design and manufacture

Product security architects

Contents

About this document

Release Information

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

Date	Version	Confidentiality	Change
June 2018	1.0 Alpha-0	Confidential	First alpha release
October 2018	1.0 Alpha-1	Non-Confidential	Second alpha release
February 2019	1.0 Alpha-2	Non-Confidential	Third alpha release

Arm® Platform Security Architecture Security Model

Non-Confidential Proprietary Notice

Document outline

Section 1: Overview

Introduces the *Platform Security Architecture* (PSA) and gives an overview of the *PSA Security Model* (PSA SM), with its goals and objectives.

Section 2: PSA Root of Trust

Introduces the PSA Root of Trust, and defines concepts, terms, architecture and requirements.

Section 3: PSA security lifecycle

Defines a generic security lifecycle for the PSA Root of Trust, and associated requirements on processes for manufacture and debug or repair.

Section 4: Boot

Defines the PSA boot process.

Section 5: Initial attestation

Introduces and defines the initial attestation service, and the associated initial attestation token, and how they can be used to bind arbitrary attestation protocols to the attested boot state of a PSA-compliant device.

Section 6: Storage

Introduces and defines the PSA RoT security enablers for storage, and how they can be used to build arbitrary secure storage solutions at the ARoT level.

Section 7: Cryptographic services

General requirements and recommendations for cryptographic services and algorithms for PSA-compliant devices.

Section 8: PSA key management

Summary of keys, sizes and algorithms used in the PSA SM.

Section 10: Appendices

Worked examples for reference, and other supporting information.

- Mappings from TMSA security objectives to PSA SM security features
- Example system realization

Potential for change

The contents of this specification are subject to change.

In particular, the following may change:

- Feature addition, modification, or removal
- Parameter addition, modification, or removal
- Numerical values, encodings, bit maps

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.

Used for a few terms that have specific technical meanings.

Red text

Indicates an open issue.

Blue text

Indicates a link. This 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.

Pseudocode descriptions

This book uses a form of pseudocode to provide precise descriptions of the specified functionality. This pseudocode

is written in a monospace font. The pseudocode language is described in the Arm Architecture Reference Manual.

Assembler syntax descriptions

This book is not expected to contain assembler code or pseudo code examples.

Any code examples are shown in a `monospace` font.

Rules-based writing

This specification consists of a set of individual rules. In this document, each rule is clearly identified in tables, together with rationale and notes for context and clarification.

Rules must not be read in isolation, and where more than one rule relating to a particular feature exists, individual rules are grouped into sections and subsections to provide the proper context. Where appropriate, these sections contain a short introduction to aid the reader. An implementation which is compliant with the architecture must conform to all of the rules in this specification.

Some architecture rules are accompanied by rationale statements which explain why the architecture was specified as it was. In this document, rationale is provided in the rules tables alongside each rule.

Some sections contain additional information and guidance that do not constitute rules. This information and guidance is provided purely as an aid to understanding the architecture. Information statements in this document are provided in the rules tables alongside each rule, or as introductory text in a section.

Arm strongly recommends that implementers read *all* chapters and sections of this document to ensure that an implementation is compliant.

Rules, rationale statements, information statements, implementation notes and software usage statements are collectively referred to as *content items*.

Identifiers

Each content item may have an associated identifier which is unique within the context of this specification.

When the document is prior to beta status:

- Content items are assigned numerical identifiers, in ascending order through the document (`0001, 0002, . . .`).
- Identifiers are volatile: the identifier for a given content item may change between versions of the document.

After the document reaches beta status:

- Content items are assigned random alphabetical identifiers (HJQS, PZWL, . . .).
- Identifiers are preserved: a given content item has the same identifier across versions of the document.

Examples

Below are examples showing the appearance of each type of content item.

Identifier	Rule	Rationale	Information
R1-1	This a rule statement.	This is a rationale statement	This is an information statement.

Current status and anticipated changes

First draft, major changes and re-writes to be expected.

Feedback

Arm welcomes feedback on its documentation.

Feedback on this book

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

- The title (Arm® Platform Security Architecture Security Model).
- The number and release (DEN 0079 1.0 Alpha 2).
- 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.

Open issues

Key	Description
	Populate remaining chapters and sections
	Reformat rule tables
	Update cross references

1 Overview (informative)

1.1 PSA overview

The *Platform Security Architecture* (PSA) program takes a holistic view of device security, recognizing that security must be addressed at both hardware and software levels – hardware security alone is not enough.

Premise:	Most connected devices contains complex code from multiple sources
Premise:	Complex code can be hard or unfeasible to prove to be free from errors, whether malicious or accidental, even with good design processes
Premise:	Compromised software can compromise hardware as well – "hidden secrets" or not, compromised software can enable full exploit of hardware, including cloning, impersonation, and access to data, even if underlying hardware protected root secrets are not directly compromised

PSA assumes that software is buggy, and that reducing risks from potentially unreliable software must be a corner stone in any device security architecture. This means extending the notion of root of trust from pure hardware to a combination of hardware and software, and isolating more trusted software components from less trusted software components.

PSA aims to help partners and industry stakeholders develop and deploy secure products based on formal security analysis.

Products will be expected to comply with a variety of functional and security requirements, for example technical, legal, commercial, regulatory, from many ecosystem stakeholders, for example service providers, end users, industry bodies, and government regulators. These requirements are expected to vary depending on use cases and ecosystems.

A service provider or an ecosystem is expected to ultimately define the overall requirements for a product, based on technical, commercial, and regulatory requirements.

PSA defines a common hardware and software security platform, providing a generic security foundation and allowing secure products and features to be developed on top. It is expected that PSA compliance will be an essential and required cornerstone towards achieving overall product certification for large classes of products.

Compliance comes in two main forms:

- *Functional Compliance*

  Functional compliance deals with interfaces, functional behavior, and interoperability. Functional compliance may cover both general product features and security features.

- *Robustness Evaluation and Certification*

  Robustness evaluation and certification deals with implementation security requirements and governance, based on threat models and security analyses. It defines required measures and processes ensuring that a functionally compliant product, including its critical assets, is not vulnerable to identified threats.

Note:

The PSA SM is primarily concerned with robustness requirements, expressed in this document as *robustness rules*.

Interfaces and technical implementation requirements for the hardware and software services and features identified by the PSA SM will be specified in separate technical specifications.

The PSA program recognizes that there will be different security requirements and different cost and security trade-offs for different applications and ecosystems. This is reflected in specifications by introducing:

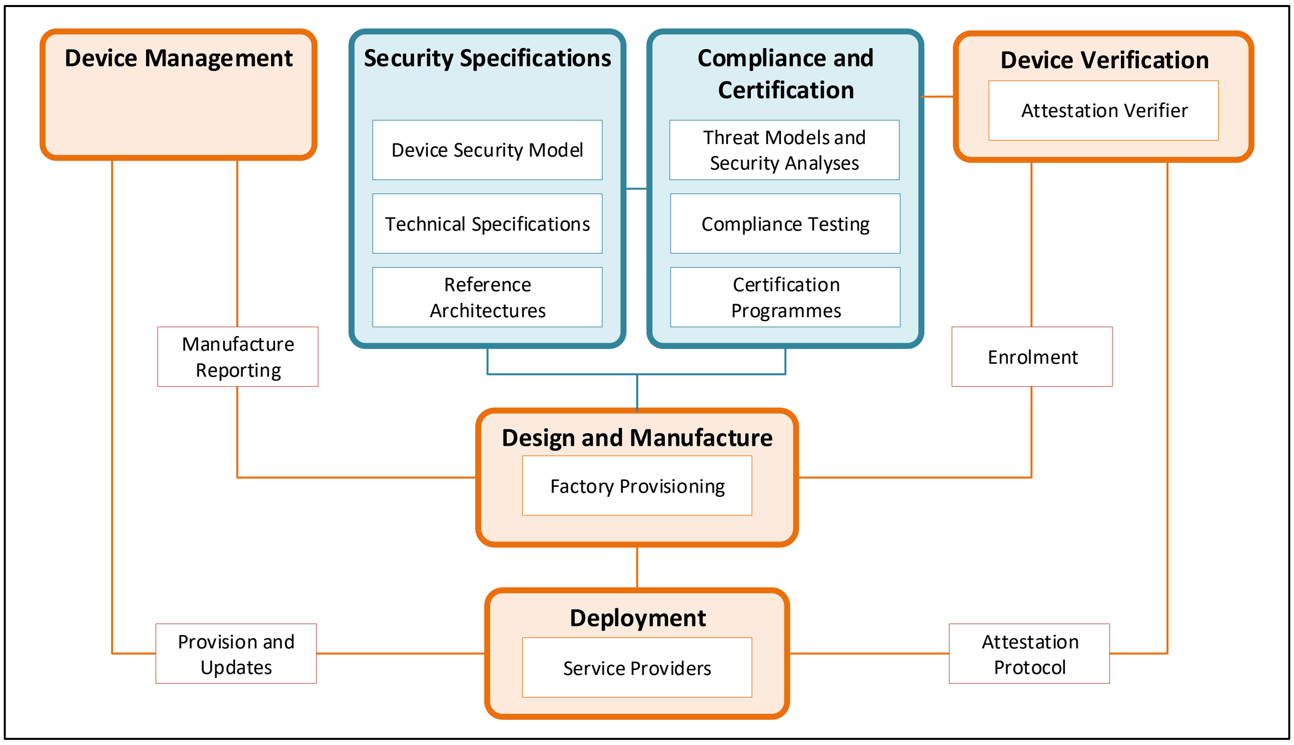
- *Reference architectures*

  Different technical reference realizations of security features, intended as design patterns with different cost and robustness properties

- Range of *robustness levels*

  Defining different robustness properties for different applications and different cost and security trade-off

1.2 PSA ecosystem



PSA-compliant devices are expected to be deployed in the context of an ecosystem with supporting security processes. Examples of such ecosystems could be:

- Vertical models, in which individual device manufacturers (OEM) or service providers create ecosystems around their own products
- Walled garden models, in which a service provider serves as a single point of entry to end customers, fronting an ecosystem of partners
- Horizontal or open models, in which a consortium of service providers, OEM, and industry or regulatory bodies form an open ecosystem of competing entities

A generic reference model for such ecosystems is outlined in the preceding figure.

- Threat Models and Security Analyses (TMSA)

  Threat models and security analyses (a simplified protection profile) identify and motivate security requirements based on a range of industry use cases. TMSA will be developed and promoted by Arm together with industry stakeholders.

- PSA Compliance and Certification

  Derived security requirements get translated into functional compliance requirements, and robustness evaluation and certification requirements, for different applications. It is expected that there will be a range of robustness levels for different use cases and different cost and security trade-offs.

  Compliance testing programs and certification programs define the processes for asserting PSA functional compliance and the PSA robustness level of a product. These may be self-assessment processes or may require third party audit and testing, depending on use case and ecosystem requirements.

- Security Specifications

  Security specifications define technical architectures and requirements for security features, providing necessary mitigations identified by TMSA requirements and enabling design and deployment of PSA-compliant devices.

  The PSA SM – this document – can be viewed as the top-level security specification, identifying and defining a PSA Root of Trust and associated root of trust services and features.

  Other specifications provide detailed hardware and software functional and robustness requirements, and standardized functional interfaces.

- Design and Manufacture

  Devices get designed and manufactured against security specifications. It is expected that common security services, interfaces, and reference implementations will simplify integration tasks.

  Device manufacture involves provisioning of root secrets and other sensitive information in factory provisioning and initialization processes. These processes must be controlled, ensuring that critical assets always remain protected.

  Device manufacture is complex, involving multiple steps and actors, and is in general out of scope of this document. However this specification does define generic robustness requirements on secure factory provisioning and manufacture processes.

- Device attestation and verification

  Manufacturers enrol devices in a device verification system, supporting attestation verification, and in doing so make a commitment to an appropriate robustness level and to play by the rules.

  Attestation and verification services are expected to be deployed by manufacturers, service providers, or by industry consortia depending on ecosystem requirements.

- Device Management

  Manufacturers provide device manufacture data, firmware updates, provisioning services, and other support functions through a device management system. Device management caters for devices throughout their life-time, from factory provisioning, through production use, field debug and repair, to retirement.

- Deployment

  Service providers can deploy secure services by authenticating devices and identifying their security properties (robustness level) through an appropriate device trust model and device verification service.

  Service providers can manage and support deployed devices through appropriate device management frameworks.

1.3 PSA SM security goals

A primary objective for PSA-compliant devices is to allow deployment of secure services using devices with known security properties.

A generic device trust model can be summarized in the steps outlined in the table below. The term *validating entity* is used here to denote any entity that needs to establish the trustworthiness a device. In the case of PSA, this is typically a service provider or cloud service, but it could also be, for example, other devices in a mesh or hive network.

Step	Requires	Motivation
Allow a validating entity to identify a device, and attest its security properties	A unique instance ID and device attestation	A unique instance ID enables identification of: • the device instance • its runtime configuration (software versions, signer(s), measurements, and security state) Following completed attestation, the validating entity will typically have established a secure and authenticated communications link with an attested end-point, which can be used for further service interaction including provisioning of service and user specific credentials and data as required.
Allow a validating entity to identify security properties of device implementations	Robustness certification A unique implementation ID for the underlying hardware (immutable components).	Verified compliance with robustness rules and certification criteria for different applications and robustness levels. A unique implementation ID allows identification of: • the device origin (for example manufacturer, model, and version) • its security implementation and associated robustness level
Provide assurance to a validating entity that its robustness criteria have been met	Manufacturer commitment before participating in the ecosystem	An established governance model for an ecosystem.
On-line verification and attestation services	Trusted verification system	Allows verification of devices against robustness criteria, and ecosystem governance processes, for example advisories and revocation.

Goal 1: PSA SM requires all devices to be uniquely *identifiable*.
To have meaning, identities should be issued in the context of a suitable governance model and a trusted verification system.

Goal 2:	PSA SM requires all devices to support *attestation*.

Application-specific secrets, for example user generated data, user credentials, and service credentials, must be protected and be stored securely on a device ensuring they cannot be accessed by unauthorized agents, and cannot be cloned to other devices.

Application level secure storage of data, keys and credentials can be implemented in many ways depending on application and device requirements, for example blob stores, encrypted file systems and databases. Common to all is a requirement to uniquely bind stored data to a specific device instance.

Goal 3:	PSA SM requires all devices to support unique *binding* of application data and secrets to a device and its security configuration.

The device configuration must be protected to ensure that only authorized software can run on a device. Unauthorized software has unknown properties and hence may leak secrets or data, and otherwise compromise the security of the device, a user, or a service.

Goal 4:	PSA SM requires all devices to support a *secure boot* process, ensuring that only authorized software can be executed on the device.

All software can and should be expected to contain errors and design flaws which may be exploited in order to compromise the security of a device. The more complex the software, the more likely it is to contain exploitable issues.

Software needs to be separated into more trusted and less trusted components. Less trusted components may include software that is difficult or impossible to prove to be without exploitable issues. A secure boot process on its own is not sufficient. Software isolation is required on top to separate software components so that exploitable issues in a less trusted component cannot compromise a more trusted component.

This document defines several isolation levels, guided by the following security goal:

Goal 5:	*Isolation* must always be provided. At a minimum between non-secure application software and root of trust software. For many certification profiles, it is expected that isolation must also be provided between components implementing the root of trust itself.

Finally, it must be possible to update devices to correct exploitable security issues, and to provide feature updates. Software updates must not compromise the integrity of the device, and the update process itself must be robust. Specifically, it must not be possible to abuse the update process to install arbitrary data on a device. All updates must be validated.

Goal 6:	PSA SM requires that all devices support a *secure update* process.

Goal 7:	PSA SM requires that any updates must be validated before being installed.

It is expected that device updates should be progressive in the sense that a more recent version is better than an older version. An older version may contain known functional or security issues which might compromise the device in some way. But at the same time, in normal operation, a device might sometimes need to fall back to some earlier last known good version should there be a problem with an update. This could be locally if, for example, an update should cause a device to become unstable. Or it could be global if, for example, a new feature is rolled out but causes problems for the service.

This means the device must prevent unauthorized rollback, such that it cannot be forced back to some earlier version with known vulnerabilities but it may roll back to some last known good version.

Goal 8:	PSA SM requires that a device must prevent unauthorized rollback– *anti-rollback*.

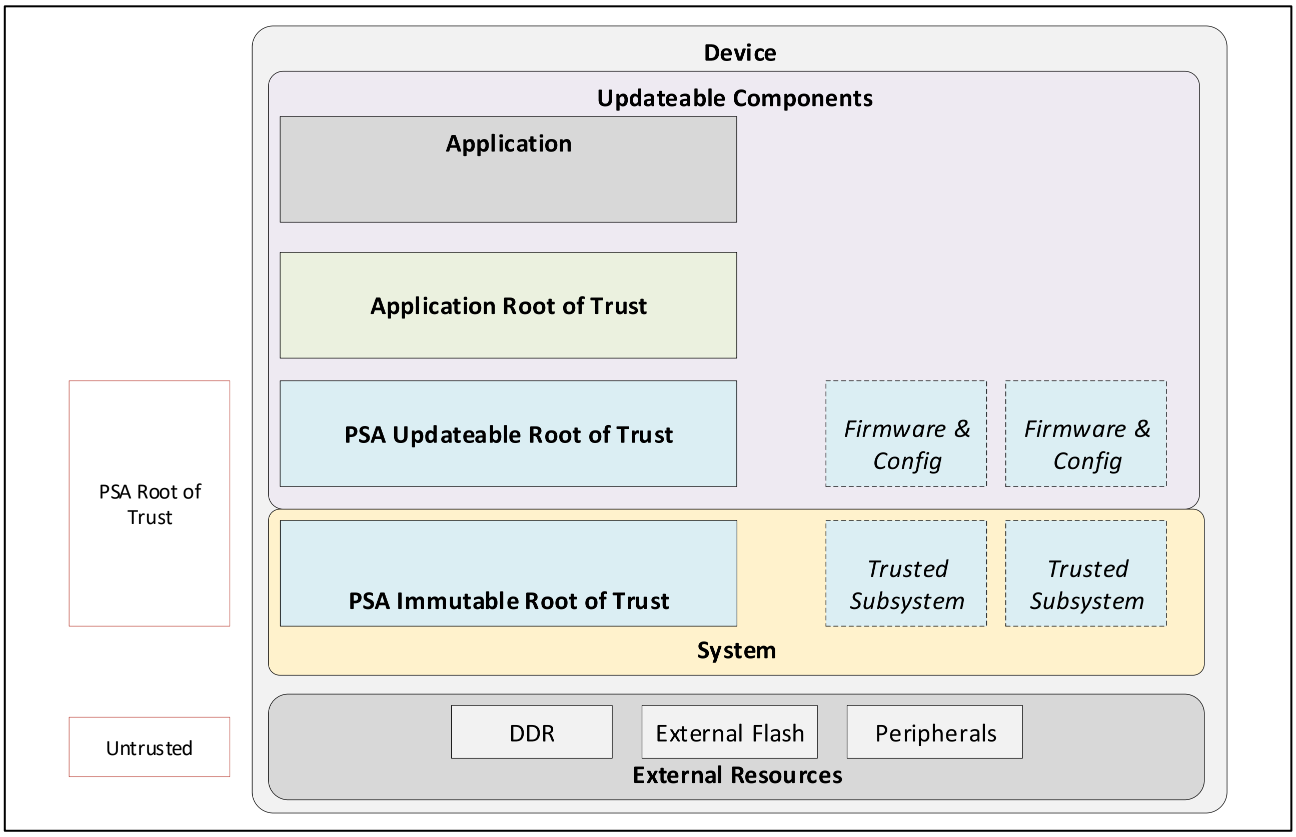
The above security goals will ensure a robust device capable of recovering from a wide range of security issues, and protecting device and user secrets. But in real deployments it must also be possible to support development and debug of device features, including security features, and to support repairs, service development and debug. For these kinds of scenarios it may be necessary to enable debug or development modes which may effectively reduce the trustworthiness of the device.

Goal 9:	PSA SM requires that devices support a *security life cycle* as defined later in this document. The current security lifecycle state of a device must be attestable.

In addition to the above core security features of PSA SM, secure devices are also expected to support a minimal set of trusted cryptography services in support of secure management of secrets and keys.

Goal 10:	PSA SM requires that all devices implement a generic cryptographic service at the most trusted level of the system.

1.4 Generic PSA device model



The figure above outlines a generic device model for PSA devices. It is a reference model for discussing device security properties in this document and actual designs are expected to be mappable to this reference model.

PSA has been designed to enable ecosystems of co-operating parties, allowing applications to be developed on top of a common security framework:

1. Silicon manufacturers provide PSA-compliant hardware
2. A PSA Root of Trust, defined in this document, provides a common security foundation for application developers

   It is expected that PSA-certified implementations of the PSA Root of Trust may be provided by security specialists, and ported to a range of PSA-compliant hardware.
3. Secure applications, including application-specific root of trust extensions, can now be built on top of a common PSA Root of Trust supporting a wide range of PSA-compliant hardware

For the purpose of this document, the following terms are used to describe a generic PSA-compliant product:

Component	Description	Notes
Device	Final end product.	For example, a networked security camera or a tracking device for asset management.
System	Inseparable component integrating all processing elements, bus masters, and PSA Immutable Root of Trust.	Typically an SoC or equivalent. But could also include, for example, an external SIM or TPM device which is inseparably bound to the rest of the system by cryptographic or physical means.

Component	Description	Notes
PSA Immutable Root of Trust	Immutable hardware components required to build and maintain trust chains throughout the device and out to service providers and users of the device. For example: • Boot ROM • Root secrets and IDs • Isolation hardware • Security lifecycle management and enforcement	PSA Immutable Root of Trust should only be accessible to the most trusted software on the system (PSA Updateable Root of Trust). The PSA Immutable Root of Trust is defined by this specification.
Trusted Subsystem	Any trusted component outside of the functional scope of the PSA Root of Trust (that is, components which are not defined in this specification) but within the trust boundary of the PSA Root of Trust: Their configuration (software and parameters) must be attestable by the PSA Root of Trust.	For example: • DDR protection system • Trusted peripherals • SE, for example a SIM or TPM Trusted subsystems are not defined by this specification (not in the functional scope of the PSA root of trust), but any such subsystem must be protected by the PSA Root of Trust and attestable.
External Resources	Hardware resources provided outside the boundary of the system, for example DDR, external flash, devices and peripherals, communications ports.	In the context of the PSA SM, external resources are defined as untrusted as they are outside the trust boundary of the PSA Root of Trust.

Updateable components consist of all software, firmware, and other updateable components in the context of the system. Any PSA device can be said to be made up of the following updateable components:

Component	Description	Notes
PSA Updateable Root of Trust	The most trusted software on the system, implementing generic and self-contained stateless services operating directly on PSA Immutable Root of Trust, for example: • Software isolation framework, protecting more trusted software from less trusted software • Generic services for example binding, initial attestation, generic crypto services, FW update validation	The purpose of the PSA Root of Trust is to provide a generic and portable interface for a set of generic services, which can be provided on top of proprietary hardware. Application-specific root of trust services can then be built on top of a portable interface, rather than having to be ported directly to proprietary hardware.
Updateable components and configuration of trusted subsystems	Any updateable components of a trusted subsystem, and any configurations of trusted subsystems which affect their operation.	

Component	Description	Notes
Application Root of Trust	Any application-specific root of trust services required by a device, for example remote attestation protocols or secure storage services required by a service provider. Application Root of Trust services should not be able to directly access immutable hardware.	The Application Root of Trust is likely to be more complex than the PSA Updateable Root of Trust, and may include third party code. Its services should be built on top of the PSA Updateable Root of Trust, enabling isolation and hence a path to update and recovery. Application-specific root of trust services are out of scope of this specification.
Application	All untrusted (from the point of view of the root of trust) application level functions and services, responsible for managing all application data and resources. Application services should not be able to directly access any root of trust software or PSA Immutable Root of Trust resources, or any secrets protected by the root of trust.	Application software can be expected to contain third party code and libraries, open source code, proprietary clients, and other components that may be difficult to certify. From the point of view of PSA, all application software should be considered as untrusted. Regular updates can be expected to fix issues and to add or modify features.

Finally, updateable components are expected to be packaged, distributed, deployed and stored on devices as images. Depending on use case, device design and operational requirements, packaging may be done in several different ways, for example:

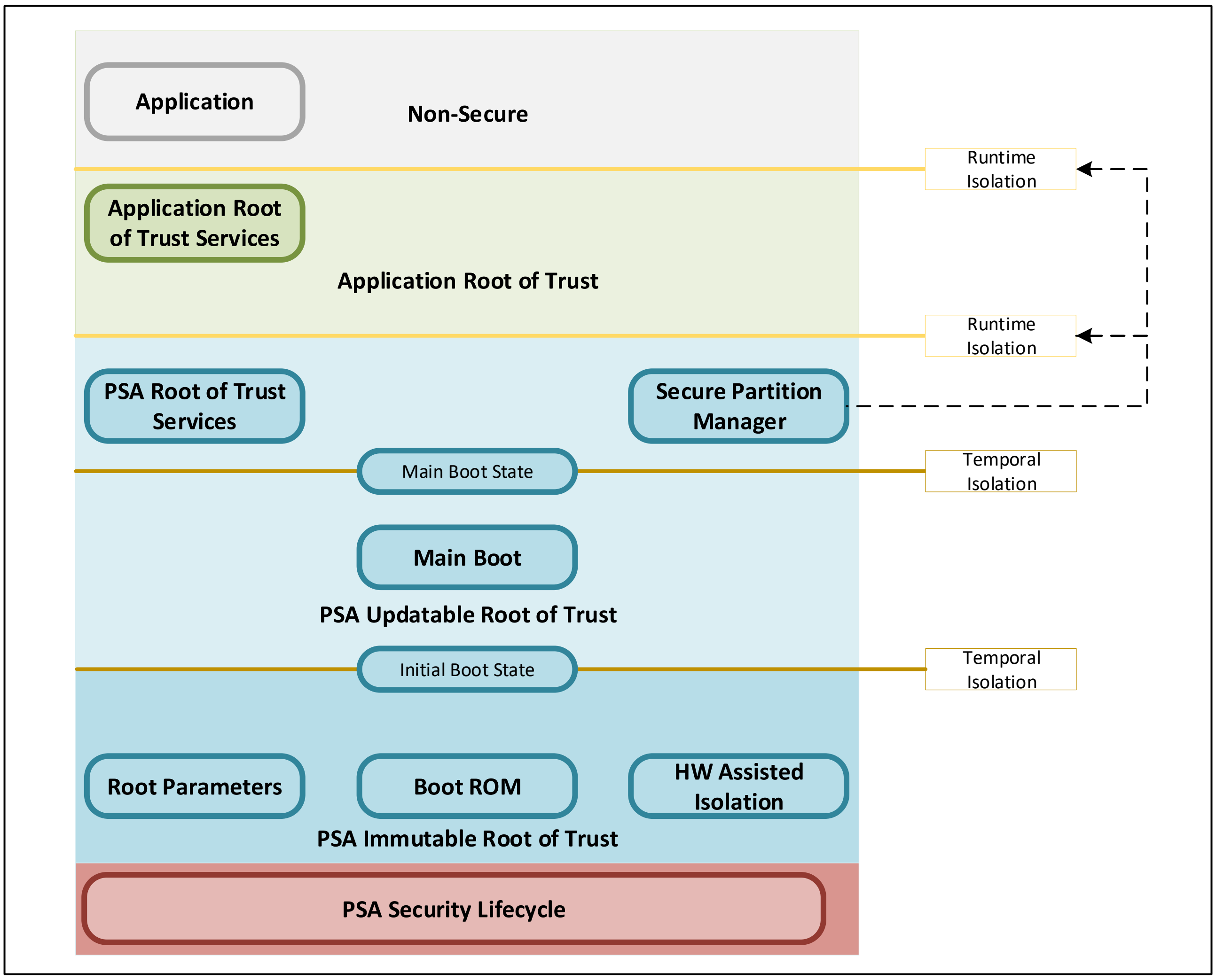
- A single device image containing all updateable components
- Multiple images containing different sets of updateable components, for example a firmware image containing root of trust components, and a separate application image containing all application software

Regardless of packaging, the following generic requirements should always be true:

Identifier	Rule	Rationale	Information
	All updateable components must be measured and validated.	It must only be possible to load signed and verified code.	Measurements are calculated during boot and during updates, and enforced during boot.
	Root of trust software must be measured separately from application software.	This allows root of trust software to be validated and attested separately from application software.	This requirement does not imply separate images or packages, only that root of trust software must be measured separately. Separate attestation of root of trust software is required to allow the trustworthiness of the security implementation to be asserted.

2 PSA Root of Trust

2.1 Overview (informative)



The root of trust of a PSA device is a multi-tier root of trust made up of Immutable and updateable components working together to ensure:

- The integrity of the device and its updateable components
- The integrity of trust chains, both within the device and within an ecosystem
- The privacy and integrity of secrets, and of operations performed using secrets
- Separation and isolation of more trusted components from less trusted components

The central property of PSA isolation is to protect trusted hardware from less trusted software, and to ensure that a compromise of less trusted software does not automatically compromise more trusted software. On networked connected devices this property is essential because any access to operations on secrets by unauthorized software will compromise those secrets, even if the actual values are not exposed.

The PSA Root of Trust acts as a *primary root of trust* on a PSA-compliant device. Implementations of a PSA Root of Trust may or may not choose to incorporate *secondary roots of trust*, for example SE (SIM or TPM) devices, to realise some or all of the PSA security properties. This would be an implementation choice and not discussed in this document (see [TBSA]).

The PSA Root of Trust is itself divided into an *Immutable* portion and an *updateable* portion. The Immutable PSA Root of Trust is the *initial root of trust* for all PSA Root of Trust services and never changes on a production device. The Updateable PSA Root of Trust represents all of the most trusted software components, providing a common trusted platform.

The *Application Root of Trust* represents any application or use case specific security service implementations on top of the generic PSA Root of Trust.

For example, the PSA Root of Trust provides a generic attestation token, which can be used to bind any application-specific attestation protocol implemented in the Application Root of Trust.

Finally, the overall state and functionality of the PSA Root of Trust is governed by a *PSA security lifecycle*. The PSA security lifecycle defines rules for when the security lifecycle of the PSA Root of Trust may change, from manufacture through production use to repairs and debug, and how such lifecycle changes affect secrets managed by the PSA Root of Trust.

This separation of a generic trusted platform from application-specific features and protocols enables a wide variety of ecosystems and supply chains to be built around the PSA security model.

The root of trust architecture of PSA has been designed to map to standard root of trust architectures for example TCG and GP, and this mapping is indicated by the Mapping column in the table.

Component	Description	Mapping	Notes
PSA Immutable Root of Trust	Immutable and tamper resistant hardware security resources, for example boot ROM and root parameters.	Initial Root of Trust	
PSA Updateable Root of Trust	Most trusted software on the system, implementing generic security services directly operating on trusted hardware or device secrets. No other software on the system should directly access rusted hardware or device secrets.	Enhanced Root of Trust	PSA Updateable Root of Trust services are identified and defined in this specification, including initial attestation, binding, FW validation, and isolation.
PSA Root of Trust	The combination of the Immutable and Updateable PSA roots of trust.	Primary Root of Trust	The PSA Root of Trust is expected to be provided by silicon vendors, or by security specialists and ported to silicon.
Application Root of Trust	Implements application-specific security services, extending the functionality of the PSA Root of Trust to provide higher level application-specific security services. The Application Root of Trust should not be able to directly access the PSA Immutable Root of Trust, and should not directly access any private resources of the PSA Updateable Root of Trust.	Enhanced Root of Trust	Application Root of Trust services represent trusted services used by application software, for example secure storage and attestation end points and protocols. The Application Root of Trust is expected to be application-specific and more complex than the PSA Root of Trust. The Application Root of Trust is expected to use interfaces provided by the PSA Updateable Root of Trust and never directly access the PSA Immutable Root of Trust.

Component	Description	Mapping	Notes
PSA security lifecycle	Defines rules for when the security lifecycle of the PSA Root of Trust may change, from manufacture through production use to repairs and debug, and how such lifecycle changes affect secrets managed by the PSA Root of Trust	N/A	See **PSA security lifecycle**.

2.2 Isolation (mandatory)

One of the main goals of PSA is to provide a generic hardware enforced isolation framework, encouraging design of robust software and ensuring that less trusted software cannot compromise more trusted software in normal operation.

The PSA Root of Trust relies on two types of isolation boundaries:

Isolation type	Description	Notes
Temporal Isolation	Code running before a temporal boundary must be the only code executing at that time. Code running before a temporal boundary is expected to run once, complete its task in a predictable and repeatable manner, and then handover to the next stage after the temporal boundary. Code executing before a temporal boundary may leave state behind for use by the next stage code executing after the boundary. Code executing after a temporal boundary should not access private resources of the code executing before the boundary, only a well-defined state left on the boundary by the previous stage.	Temporal isolation boundaries in PSA are used in the boot stages.
Runtime Isolation	In normal operation software executes concurrently. The PSA isolation model divides code into multiple *secure partitions*. A secure partition provides an isolated execution environment, protecting the code and data within a secure partition from access by code executing in other partitions. Secure partition access control policy is enforced by a dedicated PSA RoT service – the *secure partition manager (SPM)*.	Runtime isolation applies to concurrently executing components following completed boot (following initialization of SPM). See [FF].

Typically, a secure partition is expected to host one or more root of trust services. Related services that share underlying functionality or data may be implemented within the same secure partition for efficiency, but unrelated services should be kept in separate partitions.

However, a secure partition must only implement services that are either completely within the PSA Root of Trust, or completely within the Application Root of Trust, or completely within the *Non-secure* (NS) space. A secure partition cannot cross those boundaries.

Depending on the underlying hardware isolation mechanism available, [FF] defines three isolation levels. The level of isolation supported on a device depends on the level of hardware assisted isolation that can be provided.

Software for PSA devices should always be designed to assume that maximum isolation applies, in order to provide robust and portable code.

The isolation levels defined by [FF] are:

Isolation level	Purpose	Description
Level 1	*Secure Processing Environment* (SPE) isolation	PSA RoT and ARoT are isolated from non-secure applications, but not from each other.
Level 2	PSA RoT isolation	PSA RoT and ARoT are isolated from each other, and from non-secure applications.
Level 3	Maximum isolation	Individual secure partitions are isolated from each other even within a particular security domain (PSA RoT, ARoT, NS).

The following general isolation requirements always apply to any PSA device:

Identifier	Rule	Rationale	Information
	The Application Root of Trust should not be able to directly access internal resources of the PSA Updateable Root of Trust, nor directly operate on PSA Immutable Root of Trust.	The PSA RoT contains the most trusted security services, and should be protected from all other code.	Without hardware enforced isolation at this level, the device may not be able to recover from a compromise at the Application Root of Trust level.
	Application software should not be able to directly access any internal resources of the Application Root of Trust, nor of the PSA Root of Trust.	Non-secure applications include the least trusted code, and should not be able to compromise the ARoT or the PSA RoT.	
	Any PSA-compliant device must support at least isolation level 1.	It must always be possible to recover a device following a compromise of non-secure application level software.	
	Level 2 isolation is a recommended level whenever possible.	It should be possible to recover a device following a compromise in either ARoT or non-secure applications.	

2.3 Trusted subsystems (mandatory)

A system may provide additional hardware security features not defined in this document, for example cryptographic acceleration, secure time source, trusted graphics. Any such additional hardware security features can only be considered part of the Immutable PSA Root of Trust if they contain no updateable components or configurations.

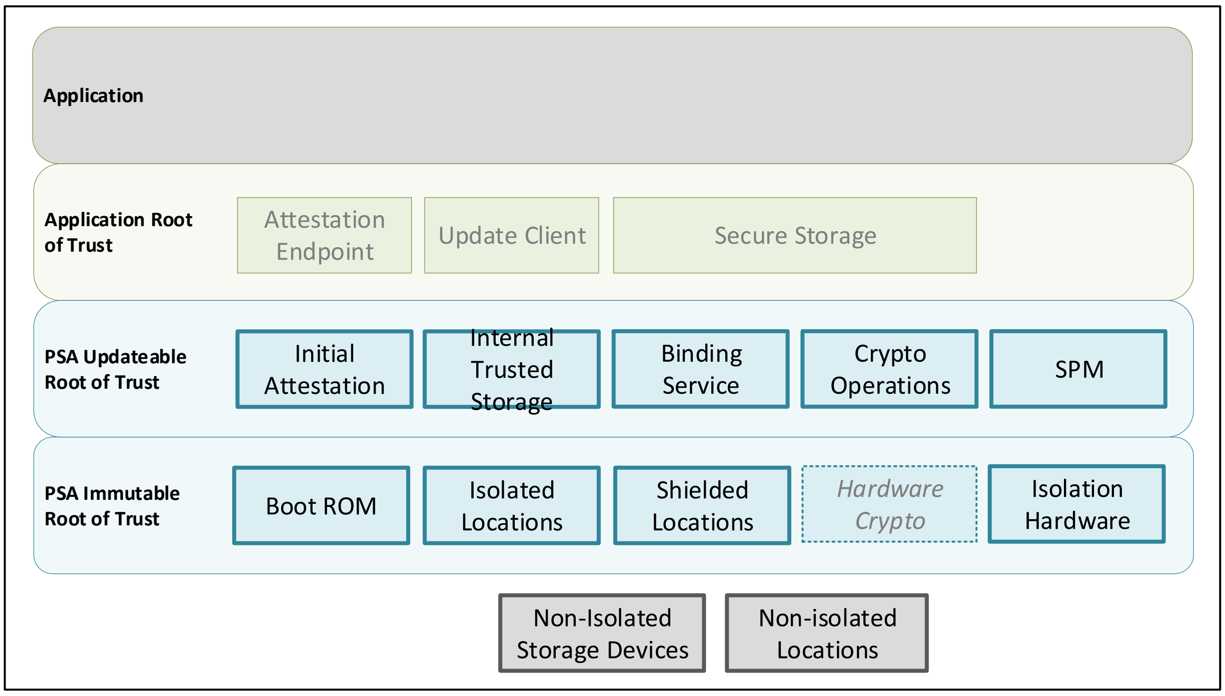
Any additional hardware security features which contain updateable components or configurations must be treated as *trusted subsystems*.

Trusted subsystems are defined as any system features which are functionally out of scope of the PSA Root of Trust, but are within the *trust boundary* of the PSA Root of Trust – they are functionally separate and cannot directly access any PSA Root of Trust resources, but their correct implementation and configuration must be attestable by the PSA Root of Trust.

A trusted subsystem may itself implement its own secondary root of trust and its own secondary security life cycle, but it must at all times be subordinate to the PSA Root of Trust in the sense that its configuration and state must always be attestable by the PSA Root of Trust.

Identifier	Rule	Rationale	Information
	Any system feature which contributes to the overall security and trustworthiness of a device must be attestable by the PSA Root of Trust	A validating entity must be able to attest the device, including any trusted subsystems.	
	The security status of the PSA Immutable Root of Trust is implicit, and asserted by the attestation identity of the system.	The attestation identity of the system identifies the implementation of the immutable root of trust.	system security features may be considered part of the immutable root of trust if they are fixed at manufacture and cannot be modified or updated on production devices. Example: A cryptographic accelerator running from its own ROM may be considered part of the immutable root of trust
	Any system security features which include updateable components must be treated as trusted subsystems, and their security status and configuration must be validated and attestable by the PSA Root of Trust.	A validating entity must be able to verify the complete security configuration of a device.	Example: Graphics hardware or any other on-system trusted devices which accept firmware updates must be treated as a trusted subsystem and their security configuration must be included in attestation
	The security configuration of system security features must only be directly accessible by the PSA Root of Trust, and must be attestable.	Where there are options for how system security features operate, including enabling or disabling for all or part of a device, or selecting operating modes with different security properties, then such configurations must be controlled by the PSA Root of Trust and included in attestation.	For example, the configuration of a memory protection engine.

2.4 PSA Root of Trust services (mandatory)



This document defines a minimum set of PSA RoT services which must always be present in any PSA-compliant implementation.

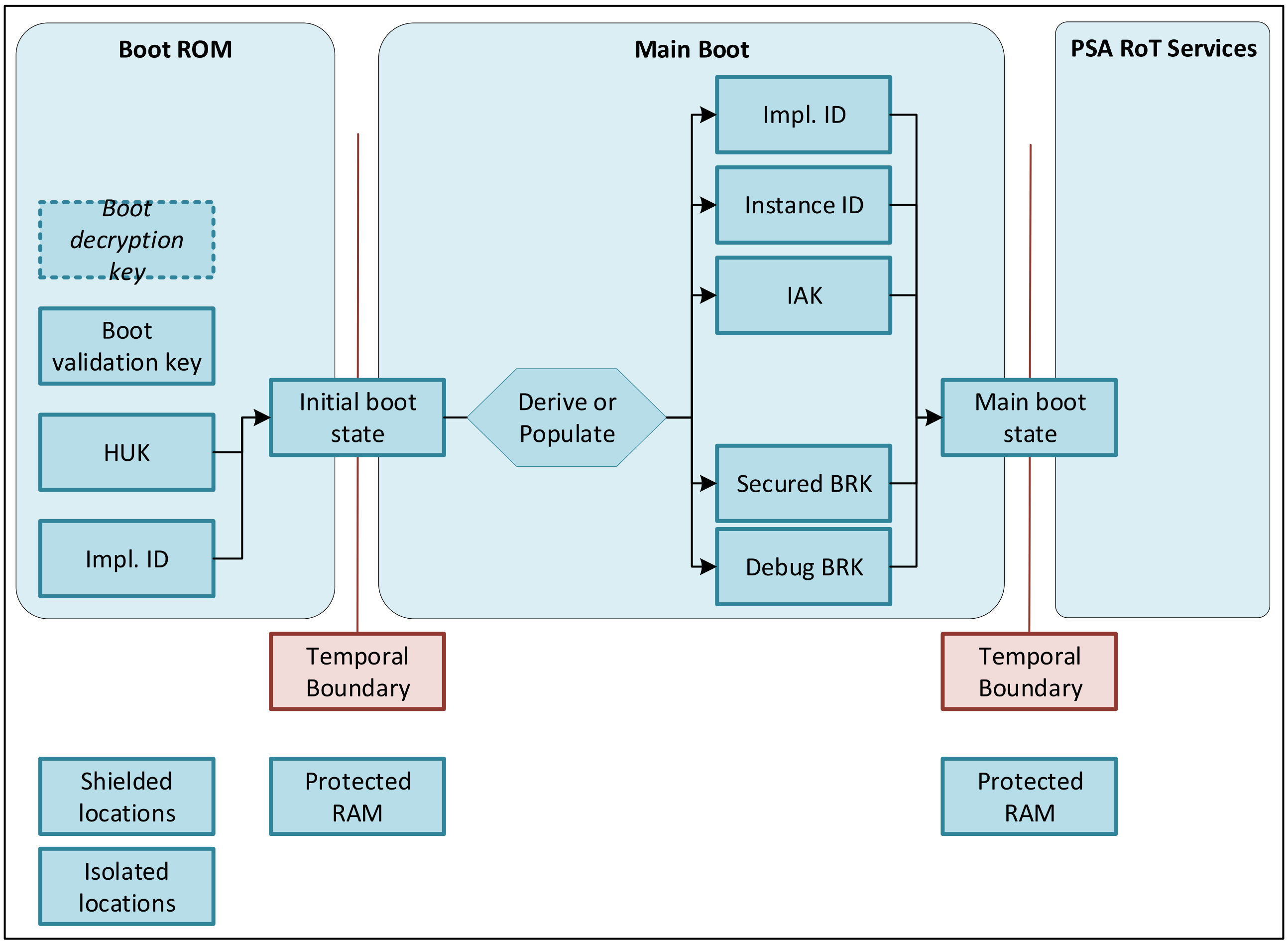
They do represent a minimal set. Implementations may define additional implementation specific services at this level if required. Future versions of this specifications may mandate additional services.

This document defines these services in terms of their required security properties and functionality. Software interfaces are defined separately in technical specifications.

Identifier	Rule	Rationale	Information
	A boot ROM must be implemented.	Enforce secure boot. Manage security lifecycle states and state changes.	**Boot**
	Secure Partition Manager (SPM) must be implemented.	Enforce partition level isolation access control policies.	**Isolation**
	Crypto must be implemented.	Provide general crypto services, with the ability to hide secret and to keep actual values of secrets away from less trusted code.	**Cryptographic services**
	Binding must be implemented.	Allow data and secrets to be bound to a partition, a device instance, and the security state of the device.	**Storage**
	Internal trusted storage must be implemented.	Manage isolated locations – locations with physical isolation properties, and which can only be accessed the PSA RoT.	**Storage**

	Initial attestation must be implemented.	Issue Initial Attestation Tokens (IAT), attesting to the current boot state of the device.	Initial attestation
	Additional implementation specific PSA RoT services may be provided as long as: 1. All mandatory PSA RoT services are always provided 2. Any such extensions do not compromise the integrity, security or function of the mandatory PSA RoT services	For example, services managing access control for implementation specific trusted hardware, for example a secure time source, or a power management controller.	

2.5 PSA RoT secrets, identities, and other parameters (informative)



A PSA-compliant device needs a number of immutable parameters, including identities and secrets, defined at device manufacture in a secure provisioning process.

For the purpose of the PSA security model, the following security classes of PSA parameters are defined:

Security class	Properties	Information
Private	Secret values which must not be accessible to unauthorized agents, including external agents, debug agents, and unauthorized code.	For example, an attestation private key.
Public	Values which may be shared both inside and outside the device.	For example, a boot validation key, or an instance ID.

Depending on implementation such parameters may be either:

- Provisioned directly in *isolated locations* – storage locations protected by physical isolation and only accessible to trusted code
- Provisioned directly in *shielded locations* – isolated locations which also have a degree of tamper resistance
- Derived at boot from seeds provisioned in shielded locations

Private values should be either provisioned in shielded locations, or derived at boot. Public values should be either derived at boot, or at least be provisioned to isolated locations to prevent against modification (for example cloning or substitution).

Depending on certification profile, directly provisioned parameters may be stored using different technologies, for example:

- In a protected read-only flash section
- In protected separate on-chip OTP
- In an SE, for example a SIM or TPM

Regardless of provisioning model, PSA specifies a boot architecture with defined temporal isolation boundaries, and defined *boot state* stored in *protected RAM* (on-chip, or encrypted) on each temporal boundary.

Code following a temporal isolation boundary should only operate based on boot state left by code running before the boundary. Only boot code should directly operate on PSA parameters directly provisioned in isolated or shielded locations, and PSA RoT code should only operate on boot state.

This model provides an additional isolation layer for the PSA RoT itself, and facilitates a variety of strategies for managing and protecting directly provisioned PSA parameters.

Within this model, PSA defines the following types of PSA parameters:

Parameter type	Properties	Information
Initial parameters Parameters directly provisioned in shielded locations.	Should only be directly accessible to boot code. May be used to either directly populate boot state, or as seeds for derivation of boot state.	Typically boot validation keys, boot encryption keys (if used), and at least a *Hardware Unique Key* (HUK).
PSA RoT parameters Parameters required by PSA RoT services.	Either populated in boot state from initial parameters, or derived from a seed for example a HUK.	For example Initial attestation key, Instance ID, Implementation ID, and binding root keys.

This document specifies PSA RoT parameters as required by the PSA RoT services.

The provisioning model – direct provisioning, or a derivation scheme – is largely implementation specific.

In general a derivation scheme may be more flexible and future proof, can support additional strategies for protecting PSA RoT private parameters, and may reduce the risk of exposure of private parameters during device manufacture. But it also may require additional computing resources at boot.

2.6 Hardware security features (mandatory)

Any PSA-compliant device must provide at least the following hardware security features:

Service	Explanation	Notes
Boot ROM	Boot ROM is the first software executing on a device, and as such is the ultimate root of trust for all software which follows.	Boot ROM may be realized as actual ROM, or locked on-chip flash, depending on certification profile.
HW Assisted Isolation	All hardware required to implement appropriate isolation between PSA software components, as defined in this document.	Examples of hardware assisted isolation include, TZ-style isolation, and physical isolation (co-processor, or an SE capable of hosting applications). See [TBSA-M].

2.7 Hardware robustness rules (mandatory)

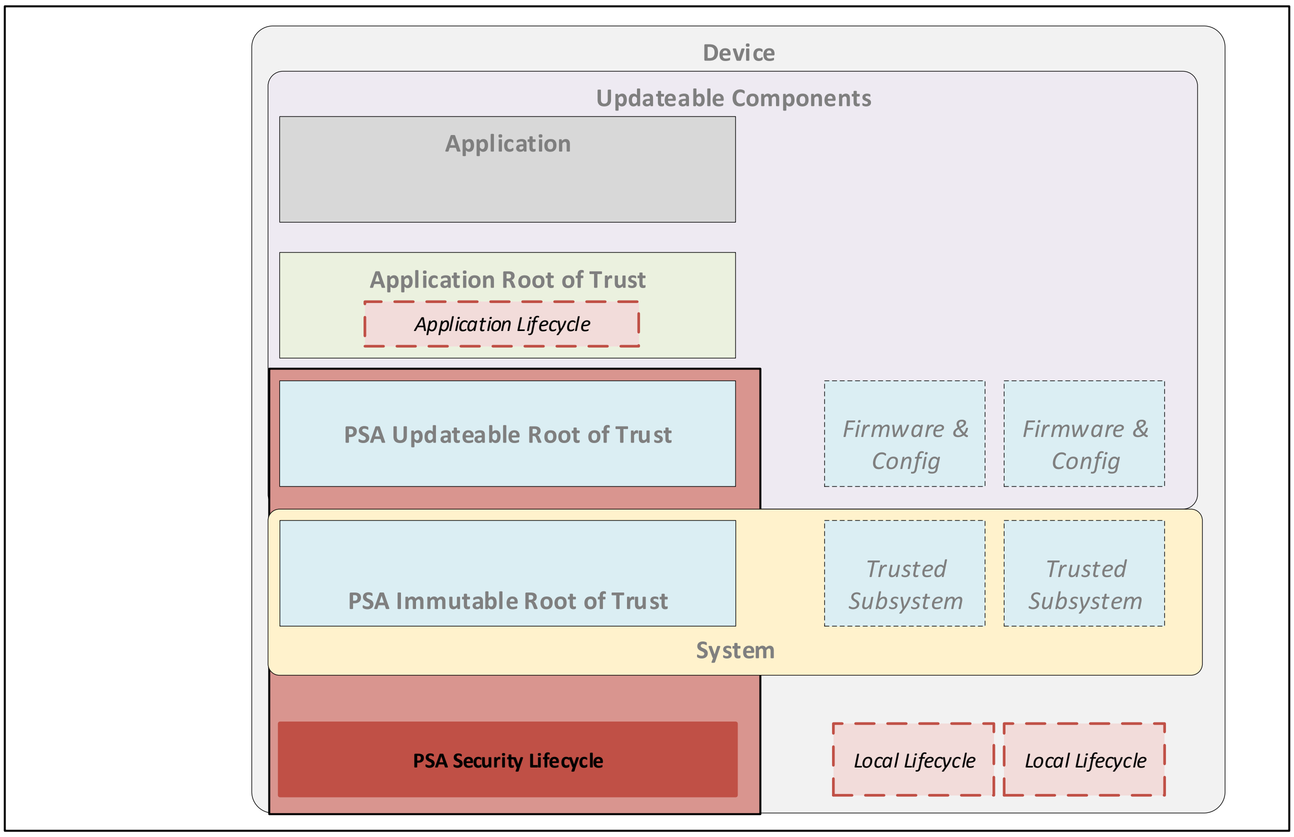
The following general hardware robustness rules apply for the PSA Immutable Root of Trust:

Identifier	Rule	Rationale	Information
	Tamper resistance: Shielded locations must provide a degree of tamper resistance.	Shielded locations hold provisioned secrets, including PSA RoT secrets. A degree of tamper resistance should be applied to protect from attempts to extract such secrets.	Depending on certification profile and deployment requirements, tamper resistance may address a number of issues, including: • Physical access control (debug interfaces, external interfaces, physical tamper proofing) • Side-channels (for example power and timing analysis) • Active probing (for example physical disassembly, access to internal buses and interfaces, or debug interfaces) • Passive probing (for example x-ray, or electron microscopes)
	Access control and isolation: PSA initial parameters should only be directly accessible by boot code. PSA RoT parameters should only be directly accessible to the PSA RoT (following completed boot).	Protect parameters from unauthorized access and modification.	Depending in certification profile and hardware capabilities this requirement may be met by either software convention, PSA isolation access control, or by lockable hardware registers.

Identifier	Rule	Rationale	Information
	Immutable	PSA parameters must be fixed, and must not change in normal operation.	Depending in certification profile and hardware capabilities this requirement may be met by either software convention, enforced by the PSA security lifecycle, or by lockable hardware registers.
	Debug protection	Device support for debug and repair must not compromise PSA parameters.	

3 PSA security lifecycle

3.1 PSA security lifecycle overview (informative)



A lifecycle tracks the state of an object through its life-time – from development and manufacturing, through use in the field, to debug and repair states. Depending on its lifecycle state an object will have different security properties, for example:

- In early development and manufacture states, secrets and identities may not have been provisioned and debug ports may not yet have been locked down
- In some debug and repair states secrets could potentially be compromised, or boot state and attestation might not be trustworthy any more

On a device there may be many objects with their own lifecycles, for example:

- Some trusted subsystem, for example SIM and TPM style devices, will have their own local life cycles and provisioning processes
- The application itself may have an application lifecycle, tracking it through logistics and distribution chains, service onboarding and activation, to de-activation and re-assignment

For the most part these objects and lifecycles are implementation or application-specific and out of scope of this specification.

This chapter defines a generic PSA security lifecycle for the PSA Root of Trust on PSA-compliant devices. As such, all other objects and any associated local or application lifecycles are subordinate to the PSA Root of Trust and its PSA security lifecycle for the purpose of establishing trust.

> Any local or application lifecycle must never be in a state which conflicts with the PSA security lifecycle of the PSA Root of Trust.

3.2 Manufacture and Factory Provisioning (mandatory)

Manufacture and factory provisioning are activities that take place while a device is in the process of being assembled and security provisioned, before being packaged and shipped to end customers down the path of distribution, retail and on-boarding.

For PSA-compliant devices, the PSA RoT must be provisioned and made fully operational as part of this process. This involves steps including provisioning of PSA RoT parameters, provisioning of PSA RoT firmware, and lockdown of the system such that debug interfaces are disabled and secure boot is enabled.

Depending on application requirements, additional application level provisioning may also be required either at the manufacture stage, or later in the distribution and on-boarding chain. Provisioning of application level data and secrets should only take place once the PSA RoT is fully provisioned and operational, and must always be protected by the PSA RoT by using PSA RoT services, for example storage and initial attestation.

Identifier	Rule	Rationale	Information
	The PSA RoT must always be made fully provisioned and operational as part of the manufacture process.	Only fully secured devices should enter the distribution, retail and on-boarding chains.	This is expected to require manufacture reporting for tracking and identification of fully secured devices.
	Application level provisioning should only take place once the PSA RoT is fully provisioned and operational.	The device must be attestable and all PSA RoT security services in a trustworthy state before additional application level secrets are provisioned.	Provisioning of application level secrets may be done, for example, by an ARoT level provisioning client.

3.3 Debug and Repairs (mandatory)

As part of repair and servicing scenarios, for example *Return Merchandise Authorisation* (RMA) processes, a fully secured device may need to be opened up for debug and repairs. Debug and repairs may be more or less intrusive depending on the access level made available to the debugging agent. In the most intrusive cases in which the PSA RoT itself has been compromised it may not be possible to return the device to a secure operational state.

The PSA security lifecycle does not distinguish between how debug has been enabled, for example by enabling a hardware debug interface, or by enabling software debug features.

Intrusive debug modes should require a secure authorisation process with appropriate governance. Again, the PSA security model does not specify the method, although other PSA specification may provide some guidance.

The PSA security lifecycle is concerned with establishing the intrusiveness of any enabled debug, defining how data and secrets can be protected when debug is enabled, and ensuring debug states are attested so that validating entities can take appropriate action depending on the actual state of the device.

For the purpose of this document, debug intrusiveness can be classified as follows:

Level	Security implications	Information
Non-revealing diagnostics	Does not reveal any sensitive user data and secrets, application level secrets, or PSA RoT level secrets. Does not affect the trustworthy operation of any aspect of the device software.	Standard logging, basic diagnostics, and similar device management functions.
NS debug	Intrusive debug which compromises NS-level operation, including access to NS-level data and secrets. ARoT, PSA RoT, and boot remain intact and trustworthy. This level of debug must be attested.	Active debugging of non-secure application software which does not cross the ARoT isolation boundary.
ARoT debug	Intrusive debug which compromises ARoT operation, including access to ARoT level data and secrets. PSA RoT and boot remain intact and trustworthy.	Active debugging which crosses the ARoT isolation boundary, but does not cross the PSA RoT isolation boundary.
PSA RoT debug	Intrusive debug which compromises PSA RoT operation, or compromises secure boot.	Active debugging which crosses the PSA RoT isolation boundary. With this level of debug enabled the device is no longer attestable.

Depending on hardware isolation implementation, a device may support some or all debug levels.

For example, a TrustZone based device may support a debug boundary between NS code and ARoT and PSA RoT code, but may not support a debug boundary between ARoT and PSA RoT code. In this case it is possible to support NS-level debug without compromising ARoT and PSA RoT, but enabling debug at the ARoT level will also expose the PSA RoT.

As long as the PSA RoT is not compromised, the device remains in an attestable state in the sense that the initial attestation remains valid and trustworthy. The initial attestation must always reflect the true state of the device, including any enabled debug. This in turn means that the debug state must not change after the boot code has completed execution, and hence changing a debug state requires a device reset.

Only changing debug state following reset is also important for cases in which sensitive data and secrets should not be available to a debugging agent. In this case it is essential that such data is made inaccessible by taking account of debug states in key derivations for storage. If a debug state were to change at runtime without reset then such data might still be exposed through cached data in memory, for example.

If the PSA RoT itself, or the boot process, would be compromised by enabling a certain level of debug, then all PSA root parameters must be made inaccessible. Such debug gives unrestricted access to the device and its capabilities. For example it may compromise attestation, or allow secure boot to be bypassed, leaving the device in an un-attestable state.

In this case, all PSA root parameters must be made inaccessible, making it impossible to issue a valid attestation using a production IAK, or derive production binding keys for storage.

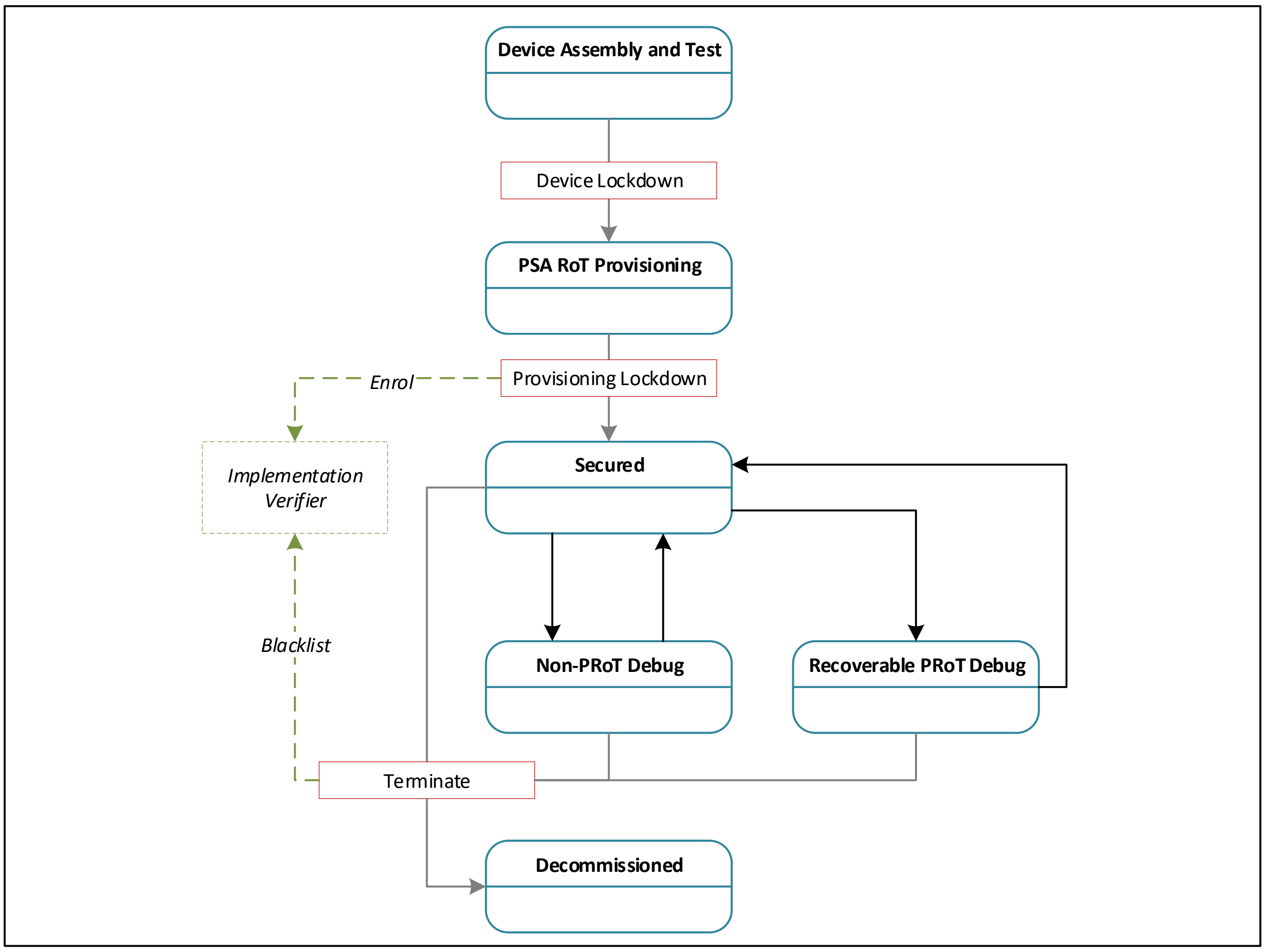
Entering a PSA RoT debug state is typically unrecoverable. Once secure boot and attestation have been compromised it is normally not possible to return the device to a trustworthy state. In this case PSA root parameters must be made permanently inaccessible, meaning the original PSA RoT levels keys and identities cannot be recreated or recovered again.

This is a terminal state for that particular set of PSA root parameters. The device may be capable of being given a new set of PSA root parameters by effectively reusing the hardware and putting it back at the start of the manufacture process. From a PSA security model point of view such repurposing of hardware must result in a new device instance.

Some devices with specialised hardware may be able to independently detect if the device enters such a debug state, protect PSA root parameters while the device is in such a state, and then verify that the device has been restored to a fully trustworthy state before making PSA root parameters available again. In this case the original PSA root parameters may be recovered as the device has some other means of ensuring the device – boot and PSA RoT – have been returned to a trustworthy state.

Identifier	Rule	Rationale	Information
	Non-revealing diagnostics must not expose any secrets, sensitive data, or affect the normal operation of a device.	The device must remain in a fully secured, attestable state.	
	Enabling NS debug, ARoT debug, and PSA RoT debug must only be possible following a device reset.	To protect sensitive secrets and data, the debug state must never change at runtime. The attestable state must not change at run-time, and hence any debug must only be enabled by boot ROM.	In NS debug or ARoT debug the debug state must be explicit in the initial attestation report. In PSA RoT debug, the debug state is implicit by any initial attestation report not being signed by production keys and identities.
	Enabling intrusive debug should require a secure authorisation process with appropriate governance.	Intrusive debug should be restricted to authorized agents, and user permission should be obtained.	Depending on certification profile.

3.4 Generic PSA security lifecycle (mandatory)



The generic PSA security lifecycle is outlined in the figure above and is intended to capture the minimum lifecycle states and transitions for the PSA RoT.

In practice, it is expected that many of the steps identified above will be expanded in any actual process. For example, factory provisioning may involve separate provisioning of a SIM or TPM style device with its own processes. Likewise, depending on isolation hardware capabilities, Non-PSA RoT debug may refer to NS debug, or to either NS debug or ARoT debug. Or even to debug only enabled for specific secure partitions at either level.

As such, the states in the generic PSA security lifecycle are defined as *main states*, and the security properties of all main states are defined in this specification as they relate to the PSA RoT, and to the attestable state of the device.

Implementations may define additional *sub states* to a main state depending on, for example, isolation capabilities and application requirements, or certification profile.

A sub state must always retain the properties of its associated main state, but may add finer granularity.

Both main states and sub states must always be attestable while a device is in an *attestable state*, and when a device enters an attestable state. This means such state changes must not be possible after the boot code has completed execution, and hence a device reset is required for such state changes.

3.4.1 Main States

The generic model defines the following main states:

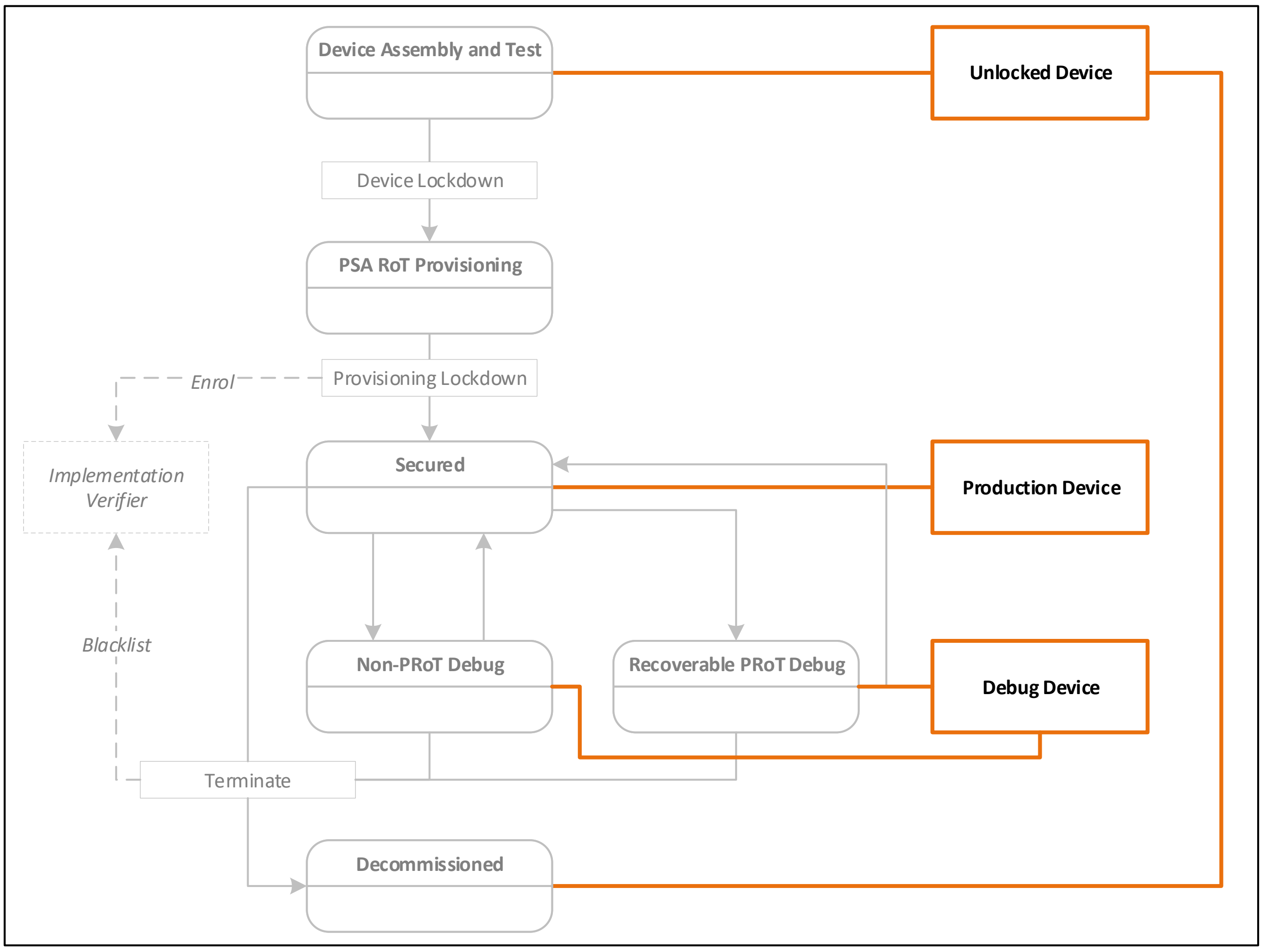
Main State	Explanation	Notes
Device Assembly and Test	During device assembly and test it is expected that the device will not be in a secure state, hardware debug and diagnostics interfaces may be open, and the device will be running manufacture and diagnostics software which must not be present on production devices.	
PSA RoT provisioning	The device must be in a secure state before any root parameters are provisioned or become accessible to any device software. Depending on certification profile and device capabilities, root parameters may be generated on device or off device. If root parameters are generated on device then care must be taken to ensure the device has access to sufficient entropy during manufacture.	If root parameters and identities are provisioned separately, for example on a an SE, then it must not be possible for any device software to access or perform any operations on them until the device has reached this state.
Secured	In secured state the PSA Root of Trust is fully operational and secured. All PSA root parameters have been provisioned and locked.	Depending on device capabilities and certification profile, locking of PSA root parameter values may be enforced by hardware (hard locking) or by PSA isolation (soft locking). Additional application level data and secrets should only be provisioned after the device gets to this state.
Non-PSA RoT debug	Non-PSA RoT debug is any debug which does not compromise the PSA Root of Trust. A device may return from Non-PSA RoT Debug to secured state as long as the PSA Root of Trust has not been compromised.	
Recoverable PSA RoT debug	Recoverable PSA RoT Debug is any debug which compromises the PSA RoT, but protects PSA root parameters such that they are inaccessible while the device remains in debug state, but may be recovered if the device is returned to a verifiable secured state.	Requires dedicated hardware capable of: 1. Detecting that the device is entering such a state 2. Hiding PSA root parameters while the device remains in such a state 3. Detecting *and* verifying that the device has been returned to a secured state A device while in this state is not capable of generating an initial attestation report signed by production keys, and hence is not attestable.
Decommissioned	Any debug state in which all PSA root parameters have been made permanently inaccessible. This is the only state in which PSA RoT debug is possible on production devices not supporting recoverable PSA RoT debug.	A device in this state is not capable of generating an initial attestation report signed by production keys, and hence is no attestable.

3.4.2 Rules

Identifier	Rule	Rationale	Information
	No production root parameters or other sensitive data should be present during device assembly and test.	During device assembly and test the device is typically unprotected and hence no root secrets and identities can be present. Hardware debug ports and diagnostics interfaces may be open.	Test parameters may be used.
	Factory test tools and firmware should not be signed or validated by the same authority as final production images.	Manufacture software may contain potentially revealing test and diagnostics modes which should never be present in any secured device. It should not be possible to load such software on a device other than in the device assembly and test state.	
	When entering PSA RoT Provisioning state, the following security properties must be enabled: 1. Secure boot enabled and boot ROM locked 2. No unsigned manufacture or diagnostics software present 3. All hardware debug and diagnostics interfaces disabled or locked 4. Only signed production software present	Depending on certification profile, debug and diagnostics interfaces may be locked with password or key, or be permanently disabled.	Depending on device capabilities and certification profile, locking of boot ROM may be enforced by hardware (hard locking) or by PSA isolation (soft locking).
	When entering secured state, the device must have the following security properties: 1. All PSA root parameters and identities provisioned 2. All PSA root parameters and identities locked	Once the device enters the secured state it must not be possible to update or modify root parameters.	Depending on device capabilities and certification profile, locking of PSA root parameter values may be enforced by hardware (hard locking) or by PSA isolation (soft locking).
	A device must only enter a debug state, or the decommissioned state, following reset. A reboot must ensure that all volatile memory is cleared and reset.	This rule ensures that there is never any intermediary state, which might reveal secrets or sensitive user data, left in memory when entering any debug mode.	
	All PSA root parameters must be disabled before a device enters the recoverable PSA RoT debug or the decommissioned state.	In these states the PSA Root of Trust is compromised and cannot be trusted. It must not be possible to derive production binding keys, or to sign attestation reports using production keys.	

	It must only be possible to return from recoverable PSA RoT debug to secured state if the device has been returned to a verified attestable state.		Typically requires dedicated hardware capable of detecting and verifying that the device has been returned to an attestable state.
	It must not be possible for a device to leave the decommissioned state. It must not be possible to recover PSA root parameters while a device is in the decommissioned state.		Depending on hardware capabilities and certification profile, and with appropriate governance, it may be possible to repurpose decommissioned device hardware by effectively putting the hardware back through manufacture and giving it a fresh set of PSA root parameters (a new identity). This is out of scope of the PSA security lifecycle and such a repurposed device should be treated as a new device.

3.5 Device types and properties (informative)



Linked to the PSA security life cycle are distinct device types in different security states which may be used for different purposes:

1. *Production device*

   A production device has been fully provisioned and locked down. In particular, the PSA Root of Trust is fully provisioned and operational and in full control of security policy on the device, and the device can be attested.

2. *Unlocked device*

   An unlocked device can be a device taken off the production line before lockdown, a recoverable PSA RoT debug device, or a device which has been decommissioned.

   Typical uses include, for example, device and service development and testing, security development and testing, and some repair and debug scenarios.

   Devices of this type cannot be trusted with production secrets or actual customer data, and should never be enrolled as trustworthy devices with any implementation verifier. But they may, for example, be manually enabled by a service provider against dedicated test accounts for test and debug purposes.

3. *Debug device*

   Depending on isolation level and method devices may support active debug in which the PSA Root of Trust is not compromised.

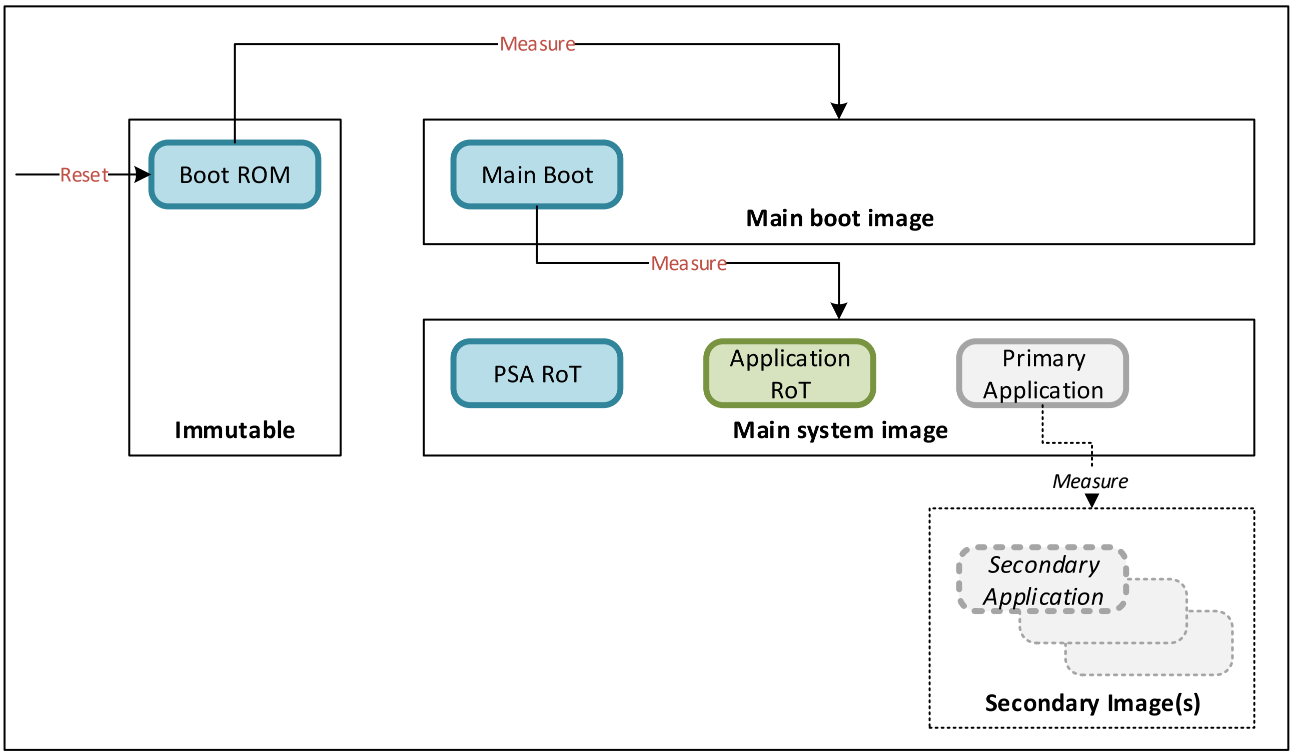
   Since at least the PSA Root of Trust is still operational and in full control of its security policy, the device remains attestable, and a service can apply its own policy accordingly.

   Further, since the PSA security lifecycle state is included in binding key derivation, any production secrets and actual customer data held on the device and protected by binding keys cannot be revealed while the device is in debug mode.

   A debug device can be returned to a production device state as long as the PSA Root of Trust has not been compromised.

4 Boot

4.1 General (informative)



All PSA devices must support a secure boot flow to ensure only authorized software can be loaded on the device.

The figure above outlines a generic secure boot flow for the purpose of this document. This is intended only as a reference for the purpose of defining security requirements.

In general it is recommended that the boot process is split up in a small, simple and verifiable boot ROM with all complex steps in the boot process contained in an updateable main boot image. This essentially moves all complex steps of the boot process into the boot phase of the PSA RoT (SPM), using the boot ROM only as trust root for the boot validation chain.

Actual implementations may use more complex models. For example, some solutions may support a boot chain with additional steps, for example an initial boot image loading the RoT code and an initial application loader (*primary application*), where the application loader in turn then loads NS side kernel and application images separately (*secondary application* images). Or an implementation might support a main system image split into multiple sub-images to allow for incremental updates rather than updating the whole image, or to support more complex supply chain models with different signing entities.

Any such variation must still meet the generic security properties as defined in this document.

1. A device must always boot from a Boot ROM following reset

   Boot ROM forms part of the PSA Immutable Root of Trust on a system. Being immutable it cannot change, and hence must be minimal, certifiable and generic.

   Boot ROM measures and validates a main boot image before transferring execution to the main boot code.

2. A main image includes main boot code

   Main boot code is responsible for the majority of the boot process, bringing up the primary software of the device, including root of trust components. The main boot image may be updateable.

   Main boot code measures the components of the RoT, and the primary application of the device, before transferring execution.
3. Typically on a PSA system main boot code will transfer execution to SPM, which then creates and enforces the isolation environment for the rest of the code on the system.
4. Some devices may include additional boot steps for loading user applications, downloadable applications, kernel file systems, and other later stage images and components

   Any loaded components at any point in the execution chain should be validated, and must not break the secure boot chain of the device.

4.2 Image signing and validation (mandatory)

All images loaded on a PSA device must be signed using asymmetric keys (RSA or ECC), and must be validated before installation on a device.

Once installed on a device, images may be locally hash locked to avoid asymmetric validation on each boot (hybrid validation).

Identifier	Rule	Rationale	Information
	All images loaded on a PSA device must be signed using asymmetric keys (RSA or ECC)	A device must be able to ensure that only authorized software is installed and loaded.	Symmetric signing is not sufficient for PSA-compliant devices.
	An image signature must cover at least: 1. Image size 2. All image content (code and data) 3. Critical parameters, for example location and launch address, where applicable 4. The image manifest, including component version, measurement		

	Each software component must specify a software version. The software version must cover at least the following components either together (release version or image version), or individually (component version): 1. Boot ROM 2. Main Boot 3. PSA Root of Trust 4. Application Root of Trust 5. Primary application 6. Any updateable components for trusted subsystems	Both the boot process and an attestation verifier needs to be able to identify the version of software currently loaded in the device. The version should uniquely identify an iteration of the associated component(s), where a higher version represents a more recent iteration of the component.	Included in initial attestation. Used to enforce anti-rollback semantics.
	If individual component versions are not specified then for the purpose of attestation and anti-rollback they shall be reported as having the same version as the overall image version.		Captured in boot state.
	Full asymmetric validation must be performed before an image is installed on a device.		For example, during firmware updates.
	Full asymmetric validation should be performed before an image is loaded for execution.		For example, at boot.
	Images may be locally hash locked (symmetric) on installation.	Allow a device to only perform full asymmetric validation of an image when it is installed (for example, firmware update), and perform a simpler local symmetric validation when it is loaded for execution (for example, at boot), to improve performance.	
	Any local has locking must be integrity and replay protected.	It must not be possible to use a local hash locking mechanism to circumvent the secure boot chain.	See **Storage**.
	Boot ROM must be associated with an immutable root validation key.	The boot ROM must be able to validate any images it loads against an immutable root.	The root validation key may be part of the boot ROM, or stored separately in protected **Storage**. Either way, it must not be possible to modify or replace the root validation key on a production device.
	The immutable root validation key should not be used to directly sign and validate images.	The immutable root validation key is a critical asset and should only be used to sign delegated validation keys in order to minimise signing events for the root validation key.	See **Validation and supply chains**, and [TBFU].

4.3 Component measurements (mandatory)

Regardless of packaging in images, software components must be measured when loaded for execution for the purpose of attestation.

Identifier	Rule	Rationale	Information
	At a minimum, the following components must be measured individually when loaded for execution: 1. Main Boot 2. PSA updateable Root of Trust 3. Application Root of Trust 4. Main application	A validating entity must be able to determine the full security configuration of a device.	If either of the listed components are loaded as multiple sub-components, then each sub-component must be measured individually.
	If either of the components listed above are updated as multiple individual sub-components, then each sub-component must be measured individually.	Some device architectures support fine grained updates to minimise the size of updates.	
	A measurement for a component must be calculated as a hash of at least: • All loaded content (code and data)		Critical parameters, for example location and launch address are covered by signature validation and do not need to be measured separately.

4.4 System reset (mandatory)

In this document, the term *system reset* is used to describe a complete system reset, including any trusted subsystems.

Following system reset, the system and any trusted subsystems should be in a fresh state. No runtime state from before the reset should be retained or used.

Identifier	Rule	Rationale	Information
	Reset of a PSA system shall include reset of any trusted subsystem.	The system must start in a fresh state, including all trusted subsystems.	
	The boot state must be freshly evaluated and verified on every boot.	The boot chain and the trustworthiness of the system must be re-established following every reset.	
	The system must only launch from boot ROM following system reset.		
	The boot ROM, and the entry point following reset, must be immutable and not updateable or modifiable in any way on a production device.		See **PSA security lifecycle**.

4.5 System hibernation (optional)

The term *system hibernation* is used to describe a system reset event which can preserve enough state to restart execution from a known point prior to the reset.

System hibernation is an optional feature. If supported, hibernation typically involves hibernation code saving runtime state to persistent storage before powering down the system. On system reset, the boot code is then able to detect and restore the previously saved state as part of the system boot process.

From a security point of view, PSA-compliant devices supporting system hibernation should ensure that any saved and restored hibernated state is appropriately protected:

- Privacy to protect runtime secrets and confidential data contained in stored state
- Integrity to prevent modification to stored state
- Replay protection to prevent replacement of stored state

Identifier	Rule	Rationale	Information
	Hibernation state must be protected, including privacy, integrity, and replay protection.	It must not be possible to extract, modify, or replace the hibernation state, including runtime secrets and data.	Depending on device architecture, this requirement may be met by using on-chip storage resources, or through cryptographic protection. See **Storage**.
	A system which has been hibernated and then restored must be indistinguishable from if the hibernation event never took place.	It must not be possible to use a hibernation function to affect or modify the runtime state or data of a system.	Indistinguishable from a security point of view. For example, network connections may have been dropped by the other side as a result of a hibernation event.
	Hibernation code must be implemented within the PSA Root of Trust.	The act of hibernating a system must be treated as part of the most trusted code on the device.	General power management code, including the decision whether to hibernate or not, may be outside the PSA RoT. The actual hibernation code must be inside the PSA RoT.

4.6 System suspend (optional)

The term *system suspend* is used to describe any low-power state in which the system has not been reset or power-cycled but in which most resources have been suspended.

Note: Any power management state requiring the system to be reset or completely powered down must be treated as system hibernation.

System suspend is an optional feature. If supported, suspend typically involves suspend code ensuring orderly suspend of the current execution state, and then power management hardware halting and powering down system resources, maintaining only a small internal power management state and keeping DDR refreshed to maintain the runtime state of the system.

On resume, power management hardware resumes system resources, and suspend code ensures an orderly resumption of the suspended execution state.

Identifier	Rule	Rationale	Information
	The suspend state should be protected, as far as possible, including privacy, integrity, and replay protection.	The suspend state contains both runtime state and data held in memory.	In general, there is a trade-off between security and suspend performance, with short expected suspend and resume times. Depending on certification profile and hardware capabilities, suspend protection may be by physical inaccessibility, or may cover a partial critical portion of the execution state.
	A system which has been suspended and then restored must be indistinguishable from if the system suspend event never took place.	It must not be possible to use a system suspend function to affect or modify the runtime state or data of a system.	Indistinguishable from a security point of view. For example, network connections may have been dropped by the other side as a result of a system suspend event.
	Suspend code must be implemented within the PSA Root of Trust.	Any suspend code required to support system suspend must be treated as part of the most trusted code on the device.	General power management code, including the decision whether to suspend or not, may be outside the PSA RoT. The actual suspend code must be inside the PSA RoT.

4.7 Anti-rollback (mandatory)

Anti-rollback is a mechanism ensuring that a device only accepts newer versions of software. This is a general robustness feature ensuring the smooth operation of a service, intended to prevent devices in normal operations from loading earlier versions of software containing known errors or vulnerabilities.

PSA allows either of the following two methods of anti-rollback protection to be supported on PSA-compliant devices:

1. Local update with reset function

   In this model, the boot ROM automatically updates an anti-rollback counter whenever it has successfully loaded a new version of software. 'Successfully' in this context may be strict, as in using "the image passed authentication" as the only test.

   However, some implementations may have the ability to detect (by boot ROM) at a following reboot whether the reboot was caused by later code causing a reset (crash). For example, detecting if a watchdog failure was the cause, or using more elaborate checkpoint schemes. The boot ROM may then defer updating the anti-rollback counter until at least one successful boot.

   In the latter case, and before the boot code has updated the anti-rollback counter, then the boot code may fall back to an earlier last known good version if a boot error is detected.

   In case of an error with the anti-rollback process itself, such that the anti-rollback counter ends up out of synch leaving the device unable to boot, it may be necessary to support a reset mechanism for the anti-rollback counter.

   For example, a factory reset operation may also reset the anti-rollback counter. Or a device management protocol may include a secure messaging feature instructing a device to reset its anti-rollback counter. In the latter case, supporting both source validation and replay protection for such messages, enforced by Boot ROM is obviously essential.

2. Update on command

   An alternative approach is for the boot R not to update its anti-rollback counter on its own accord, and only update it when it receives a secure message from the device management service to do so.

   This requires a secure messaging feature in the device management protocol, supporting at least source validation and ideally replay protection as well.

   It also requires a secure messaging feature to signal to devices which software version to load out of potentially a number of possible options (basically any version newer than the current version of the anti-rollback counter on the device).

Which method to use depends on operational requirements.

The first option may be more robust at the device level but may make it harder to manage roll-back scenarios at the service level in case there is a service level problem with a newly deployed image – it might boot correctly, but some aspect of its function is broken and the service provider decides to roll back its population of devices. Further, if a reset message is used then it may pose a security risk in itself if an appropriate and effective message replay protection scheme is not in place.

The second option may be more robust at the service level as the service provider decides when to invoke anti-rollback after rolling out a new software version in its network. But it also potentially leaves devices able to be rolled back to a previous software version for longer. It requires a secure messaging feature ensuring messages to update the counter can only come from a trusted source, but it is still potentially (depending on protocol implementation) susceptible to denial of service attacks by blocking the message to update the anti-rollback counter, leaving the device exposed to rollback attacks.

Either way, for both types of semantics the following generic features are required:

1. An anti-rollback counter managed exclusively by the boot code – only images that are of the same or a more recent version than the anti-rollback counter can be loaded on the device
2. The anti-rollback counter may be updated either automatically by the boot code following successful boot of a new image, or on command from a device management service
3. The anti-rollback counter may be reset, either automatically upon factory reset, or by command from a device management service
4. In the case of anti-rollback being controlled by command from a remote service (reset, or update), the command must be protected by source validation and replay protection

Identifier	Rule	Rationale	Information
	The system must only load and install updates that are of the same or higher version than the currently installed version of the same component. It must not be possible to install an older version of any component.	Anti-rollback must be supported on production devices.	Typically requires securely stored anti-rollback counters.

Identifier	Rule	Rationale	Information
	Anti-rollback must be enforced by boot ROM.	The device state, including its boot state, must be attestable at all times and must not change at runtime. All firmware updates must require a device reboot to take effect.	
	The anti-rollback mechanism must only be directly accessible to boot ROM.	No other code on the device should be able to directly modify the anti-rollback state of the device.	Depending on certification profile, this requirement may be met by either isolation, or lockable registers, or similar.
	An anti-rollback mechanism may be reset following factory reset.	Allow devices to be recovered if the anti-rollback mechanism gets out of synch.	
	A device may support a mechanism to control anti-rollback protection by remote messaging – reset message, or update message.	Remote device management messaging.	Such remote mechanisms may also be used, for example, in some device management protocols to support targeted updates. Such additional features are out of scope of this document but in general have similar security requirements.
	Any remote device management mechanism must be protected by at least: 1. Authentication of the command issuer with at least the same cryptographic properties as that used for image signing 2. Replay protection, ensuring that any issued command instance can only be acted on once by a device	It must not be easier to subvert any such messaging protocol than to subvert the secure boot mechanism itself. It must not be possible to use previously issued versions of such commands against a device.	
	It must not be possible to use any remote management feature to force the anti-rollback counter on a device to a value beyond the highest version of any images currently loaded on the device.	It must not be possible to force a device into a state in which it can no longer boot.	

4.8 Boot State (mandatory)

4.8.1 Temporal Isolation

The boot stages before SPM is loaded – boot ROM and main boot – should be isolated by temporal isolation:

1. Code in each stage should execute its tasks, complete, and the terminate and hand over execution to the next stage
2. Code in each stage should leave boot state to carry forward results and data from one stage to the next
3. Private data for one stage should not be directly accessed (or accessible) by later stages

Note that temporal isolation in this context relates to the execution of the boot tasks. On constrained devices re-use of generic boot code functionality, for example cryptographic libraries, may be used to reduce the overall code footprint. Any such reuse must not change or affect the boot state of the system, and must not expose any private boot code secrets or data.

Identifier	Rule	Rationale	Information
	Code executing during one boot stage must completely exit before handing over to the next stage.	Each boot stage (pre SPM) must be in complete control of its own environment and execution context.	
	Data and results from one boot stage may be left as boot state for the next stage.		
	Boot state should be left in on-chip RAM.	Boot state may include root secrets.	Using external DDR for holding boot state may expose root secrets to probing and other external attacks.
	Code from one boot stage should not directly access secrets and private data from a previous boot stage.	A temporal boundary must be respected.	Ideally enforced by hardware, for example invalidating certain storage locations until a full system reset when exiting a particular boot stage. May be enforced by software convention or isolation depending on certification profile.

4.8.2 Initial boot state

The initial boot state is the secrets and data left by the boot ROM before handing over to main boot.

Identifier	Rule	Rationale	Information
	At least the following component information must be captured for each software component validated by the boot ROM, and for itself: a. Signer ID (from manifest) b. Software version (from manifest) c. Actual measurement – the measurement calculated by the boot ROM for this instance of the associated component	Capture the boot state as determined by the boot ROM.	
	Initial parameters should be copied from shielded locations to the initial boot state.	Only the boot ROM should directly access root parameters in shielded locations.	
	The initial boot state should only be directly accessible to the main boot.	Initial parameters should not be available to later stages of software.	This may be implemented, for example, by main boot erasing all or part of the initial boot state when it exits.

	The initial boot state should not be modified once set by the boot ROM.	Make sure that the initial boot state can be trusted by main boot components.	Depending on certification profile, this requirement may be met by isolation and software convention. On devices with stricter certification profiles lockable registers may be required (only writeable by boot ROM).

4.8.3 Main boot state

The main boot state is the secrets and data left by the main boot code.

Identifier	Rule	Rationale	Information
	The main boot state must include all component information captured by boot ROM.	Retain measurements for all components loaded by the boot ROM.	
	At least the following component information must be captured for each additional software component validated by the main boot code: a. Signer ID (from manifest) b. Software version (from manifest) c. Actual measurement – the measurement calculated by the boot ROM for this instance of the associated component	Add measurements for all additional components loaded by main boot.	
	Boot seed The boot state must include a boot seed uniquely generated at each boot event.	The boot seed may be used by later services, for example to allow a validating entity to ensure that attestations for different attestation end points were generated in the same boot session. It must be large enough to make global collisions statistically improbable.	Unique here may be read as statistically unique. A 256-bit size is recommended. This property may be satisfied either by randomly generated if sufficient entropy is available in the boot process, or by generating the seed as a hash from a monotonic boot counter.
	The main boot state must include all PSA RoT root parameters.	All root parameters required by the PSA RoT must be accessible through the main boot state.	Either populated from initial parameters in the initial boot state, or derived from initial parameters, for example a HUK.
	The main boot state must only be accessible to PSA Root of Trust services.	Only the PSA Root of Trust should be able to directly access PSA RoT parameters.	For systems with level 1 isolation this requirement can only be met by software convention. For systems with level 2 isolation or higher, this requirement must be enforced by hardware isolation mechanisms.

Identifier	Rule	Rationale	Information
	The main boot state should not be modified once set by the main boot code.	Make sure that the boot state can be trusted by the PSA RoT.	Depending on certification profile, this requirement may be met by isolation or by software convention. On devices with stricter certification profiles lockable registers may be required (only writeable by main boot and the locked).

4.8.4 Measured trusted subsystems (mandatory)

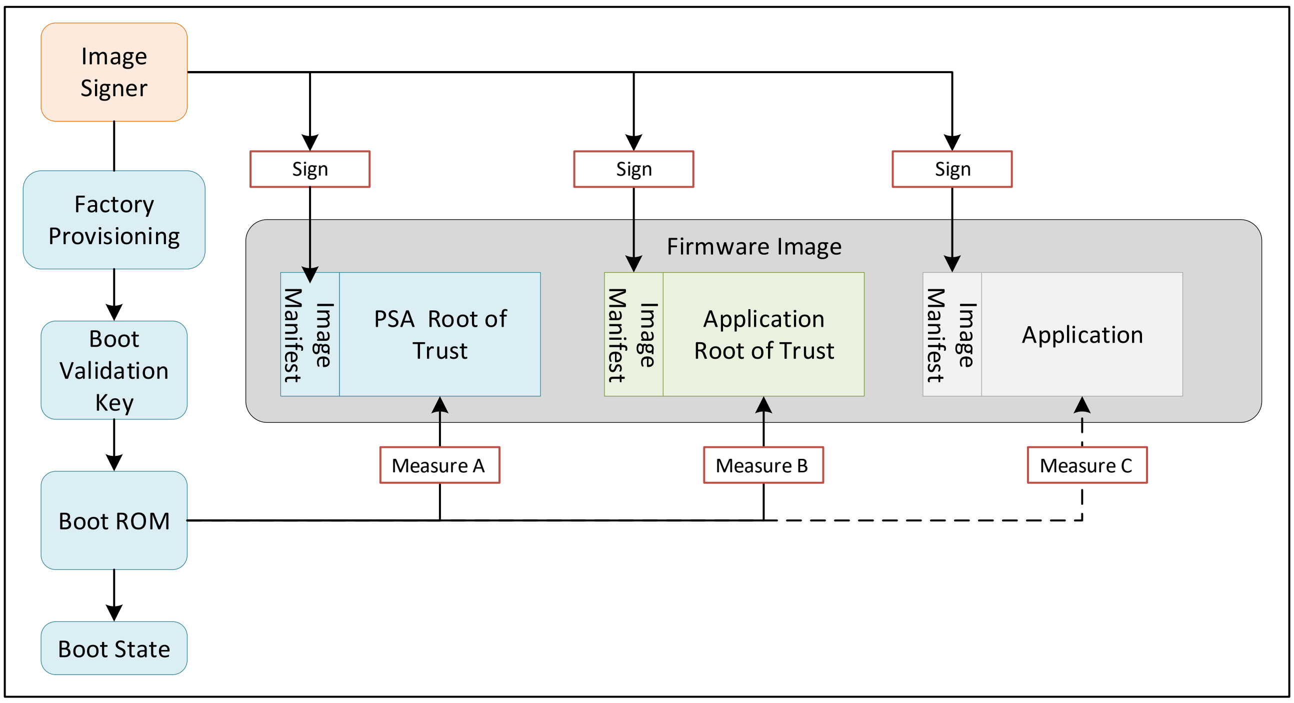
Any updateable components for trusted subsystems must be measured and validated at boot, and included in the boot state.

Identifier	Rule	Rationale	Information
	Any updateable part of each trusted subsystem shall be described by a signed manifest.	Same model as for validated software components.	Updateable parts include, for example, software, firmware, rigidware. Any updateable component which affects the function of the trusted subsystem.
	Each separately updateable component for each trusted subsystem must be measured and validated separately at boot by the boot ROM.	The state and properties of trusted subsystems must be verified.	
	The boot state must be extended to include, for each updateable component of each trusted subsystem: 1. Signer ID (from manifest) 2. Software version (from manifest) 3. Actual measurement – the measurement calculated by the boot ROM for this instance of the associated component	Same model as for validated software components.	

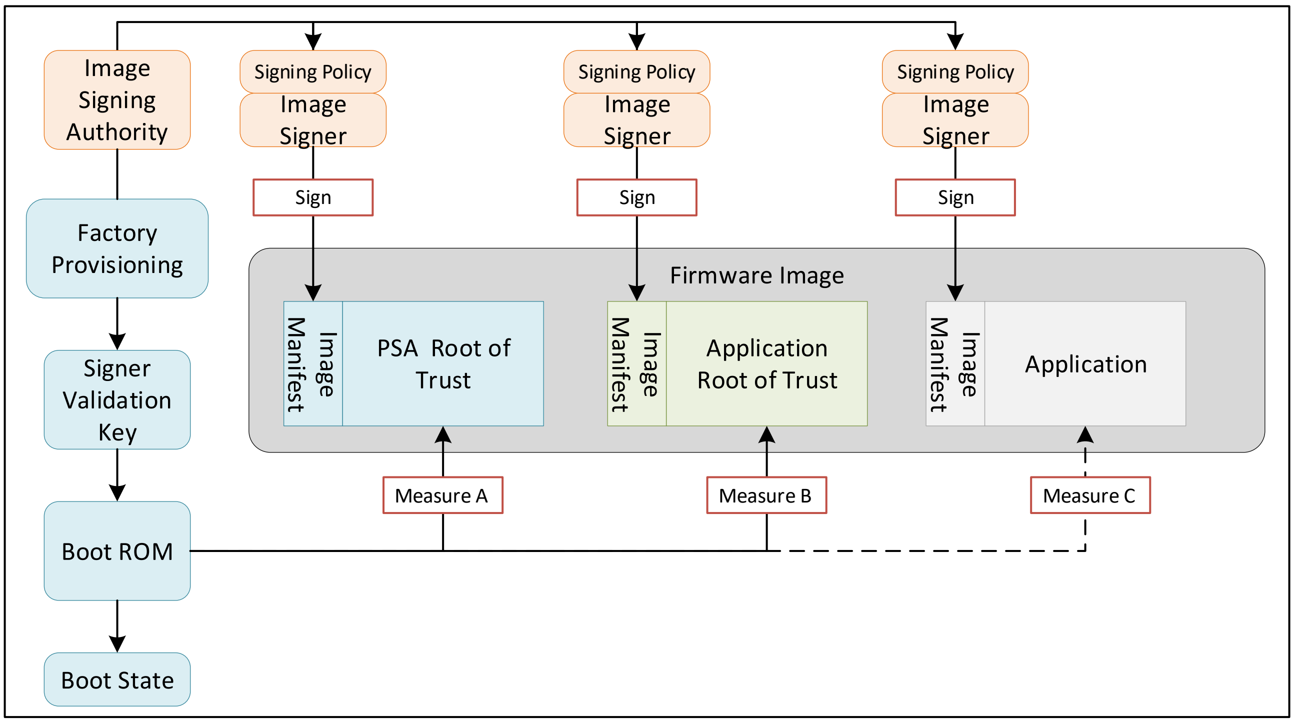
4.9 Validation and supply chains (informative)

[TBD]

4.9.1 Single Signer



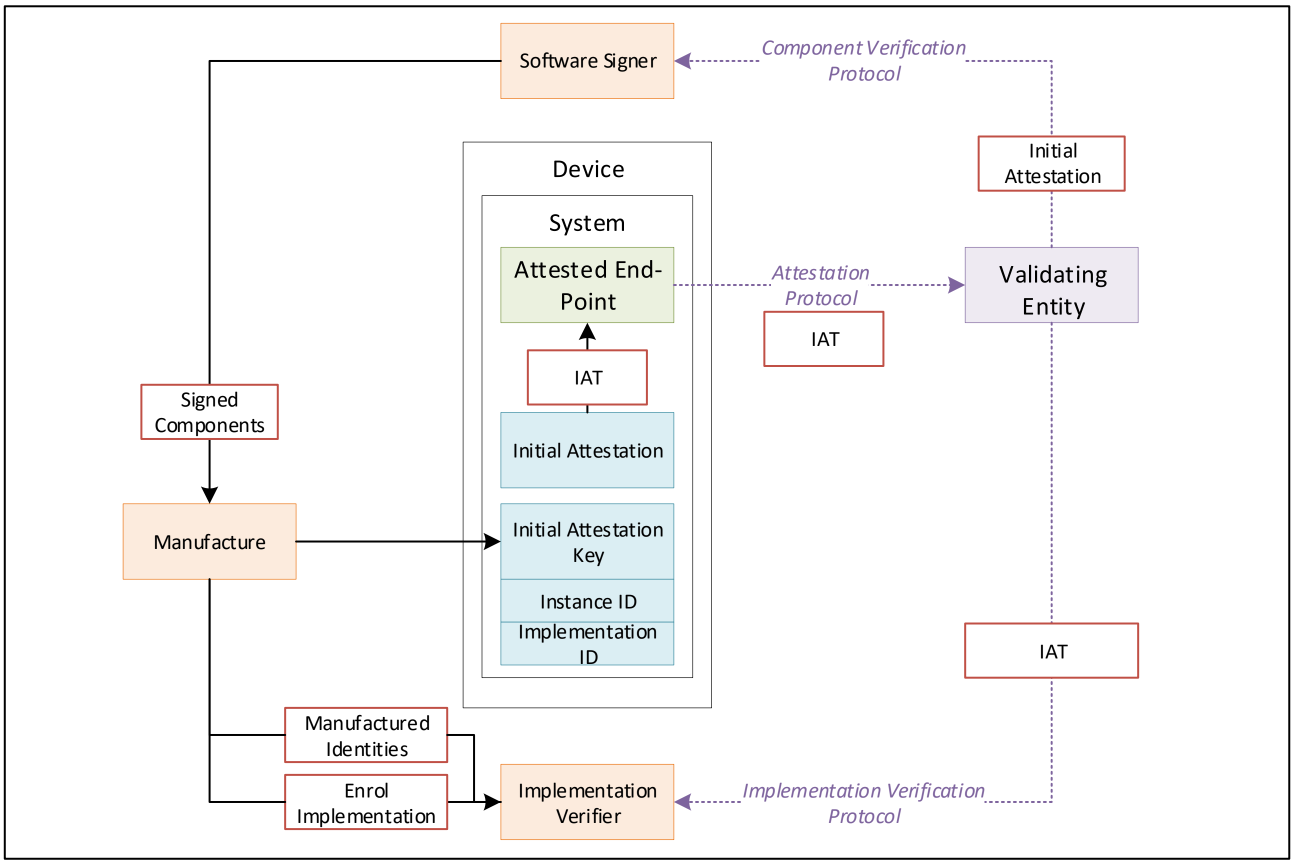
4.9.2 Delegated signers



4.9.3 Signer revocation

5 Initial attestation

5.1 General (informative)



Attestation in general involves several security processes, working together to build a trusted ecosystem. There are many protocols for attestation, from proprietary systems to attempted standardisation by groups such as FIDO, TCG, and Global Platforms.

PSA does not attempt to define or replace any attestation protocol. Instead it provides a framework and the minimal generic security features providing an interoperable and hardware independent way for OEM and service providers to integrate any attestation protocol on top of the PSA Root of Trust.

This is done by introducing the *initial attestation token (IAT)* and the *initial attestation service* at the PSA RoT level.

The figure above outlines a generic example for the purpose of identifying typical roles and separations of concern which apply to virtually any attestation scheme.

A generic workflow for attestation then runs as follows:

1. At manufacture the device is provisioned with immutable hardware root parameters in PSA Immutable Root of Trust, including a unique *initial attestation key* (IAK), *instance ID*, and *implementation ID*:
    a) The IAK must be unique to an instance of an implementation of the PSA Root of Trust
    b) The Instance ID uniquely identifies the IAK
    c) The Implementation ID uniquely identifies the underlying Immutable PSA RoT

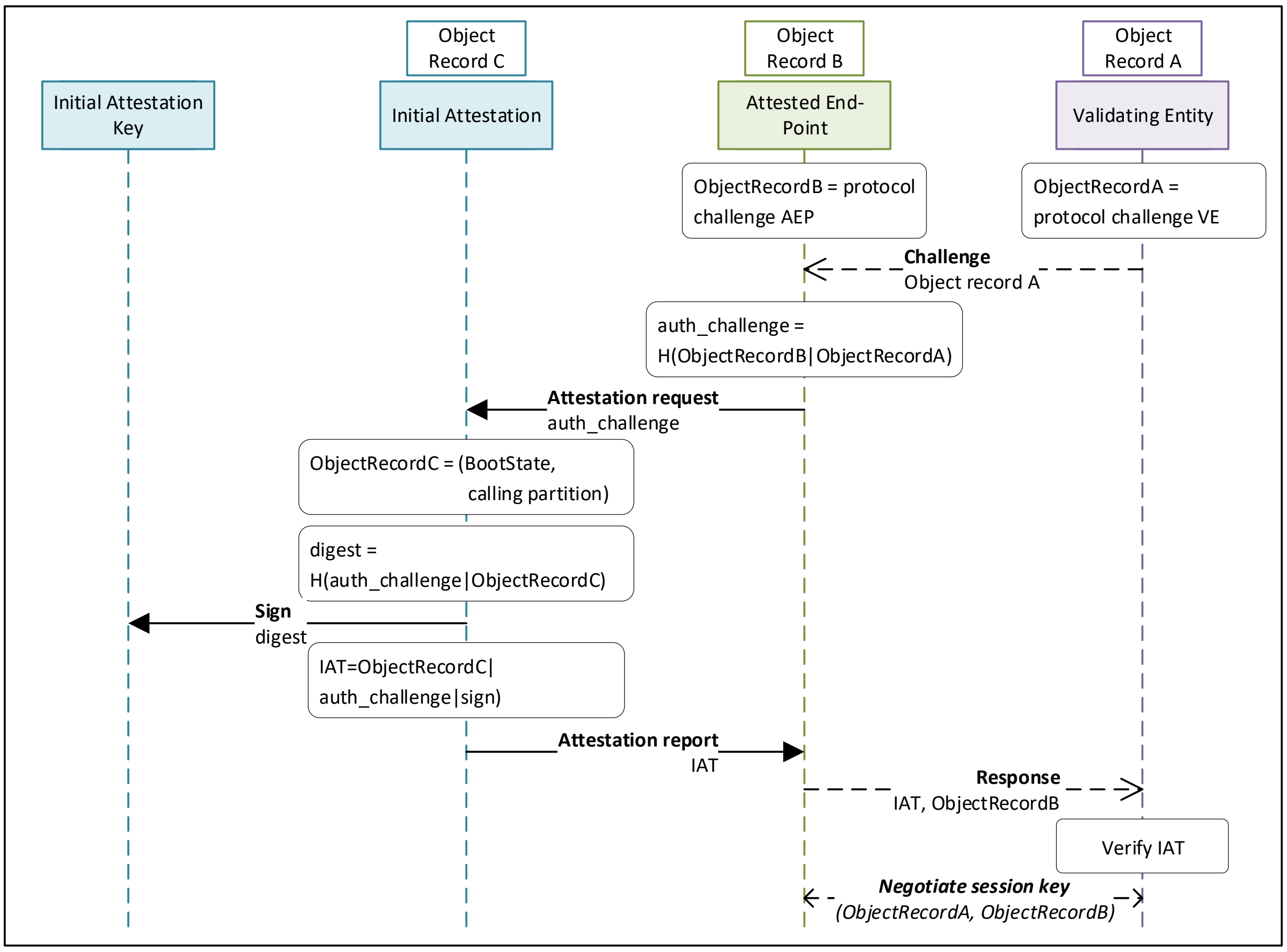
    d)

2. Manufacture reporting must be in place to track issued and manufactured identities, and the status of associated hardware
    a) An *implementation verifier* must be able to track and verify manufactured identities and their associated security properties, for example certification status
3. The PSA Root of Trust provides a generic *initial attestation service*

    The initial attestation service produces an *initial attestation token (IAT)*, allowing an *attested end point* to bind any attestation protocol to:
    a) The boot state of the PSA RoT
    b) The security state of the PSA RoT
    c) The calling partition (the attested end point)
    d) Essential properties of the protocol – an *authentication challenge* supplied by the attested end point
4. The attested end point is expected to implement any application-specific attestation protocol

    An attested end point represents all application-specific logic required for an attestation protocol. It could be, for example:
    a) A device end point for a device attestation and authentication protocol
    b) An application level key attestation service for, for example, user credentials or alias identity keys
    c) An end point for an anonymization service, for example *Direct Anonymous Attestation* (DAA)
5. A validating entity can take the initial attestation report and:
    a) Verify the implementation of the PSA Root of Trust, its manufacture status (for example a production device or a development device), and its certification status
    b) Verify the updateable components currently loaded on the device
    c) Verify the security state of the PSA Root of Trust on the particular device instance being attested

5.2 Basic attestation (mandatory)



Initial Attestation Key
Object Record C
Initial Attestation
Object Record B
Attested End-Point
Object Record A
Validating Entity
ObjectRecordB = protocol challenge AEP
ObjectRecordA = protocol challenge VE
Challenge
Object record A
auth_challenge = H(ObjectRecordB|ObjectRecordA)
Attestation request
auth_challenge
ObjectRecordC = (BootState, calling partition)
digest = H(auth_challenge|ObjectRecordC)
Sign
digest
IAT=ObjectRecordC| auth_challenge|sign)
Attestation report
IAT
Response
IAT, ObjectRecordB
Verify IAT
Negotiate session key
(ObjectRecordA, ObjectRecordB)

The basic attestation model enables attestation of an attested end point on a system, and the state of the system.

In PSA, an attested end point is a secure partition. This allows a verifier to confirm and be confident that the end point is not only a known PSA instance in a known state, but is a trusted component of that instance and not any software.

Any attestation protocol can be represented as starting with a challenge carrying some metadata – Object Record A – from the validating entity. Depending on protocol this might be nothing at all, or for example some identity information for the validating entity, or a nonce, a public key or initial DH parameters.

The attested end point is associated with metadata of its own – Object Record B – representing any client side key exchange information (for example, a public key, or initial DH parameters), and any other metadata required by the chosen attestation protocol.

The attested end-point then requests an initial attestation from the PSA Root of Trust. The initial attestation is constructed as follows:

1. The attested end-point calculates an authentication challenge, auth_challenge, constructed as a hash of object records A and B

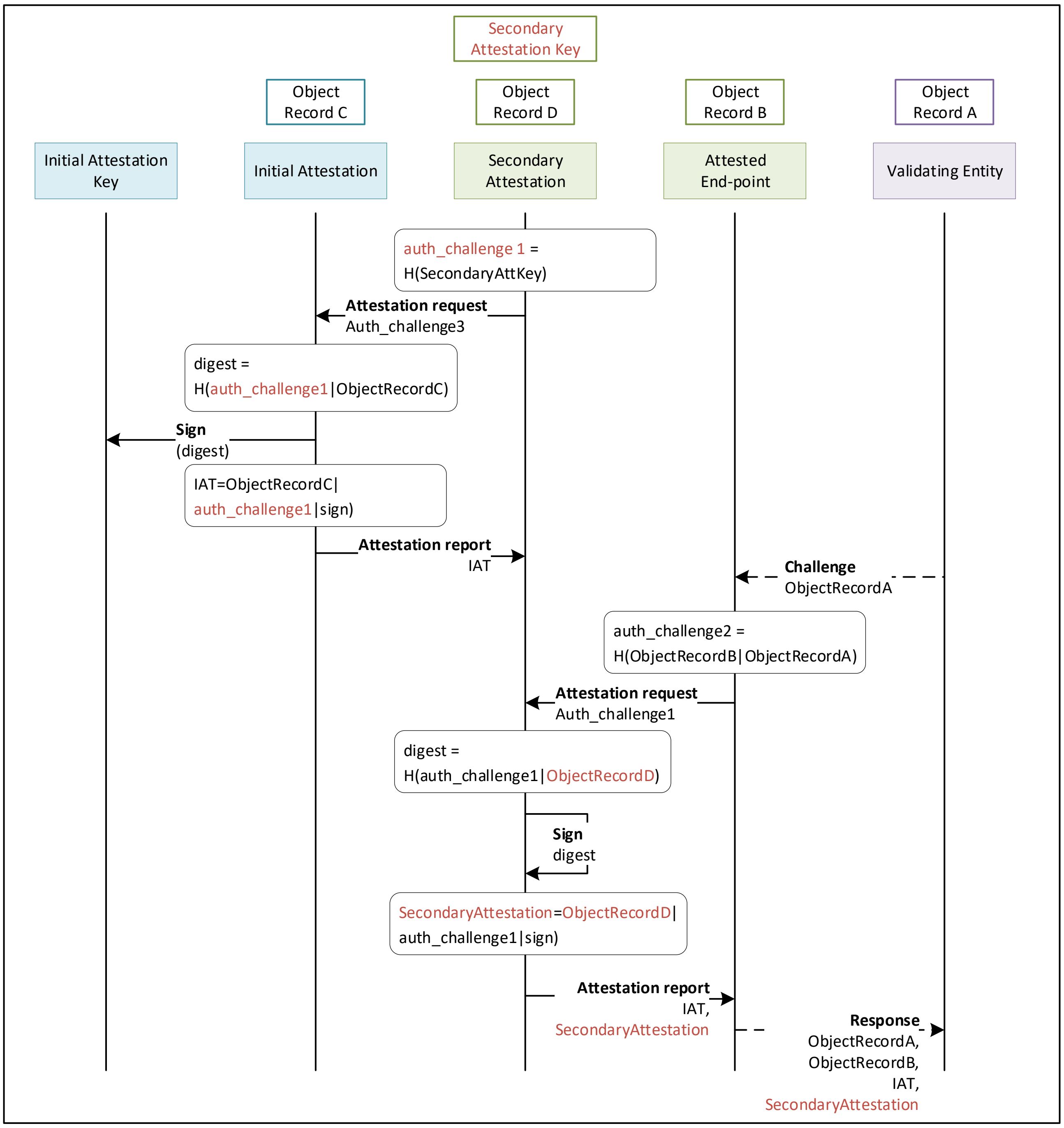
   The hash needs to include at least all metadata that must be validated by the validating entity as part of the chosen attestation protocol.

2. The initial attestation service is associated with its own metadata – Object Record C – including:
    a) The measured boot state
    b) The current security lifecycle state of the PSA Root of Trust
    c) The partition ID of the calling partition
3. The initial attestation service calculates a digest as a hash of auth_challenge and its own object record C
4. This digest is then signed using the initial attestation key, only accessible to the PSA RoT
5. An initial attestation token (IAT) can now be constructed by combining Object Record C, auth_token, and the signature
6. The attestation protocol challenge can now be completed by the attested end point by combining the initial attestation token with its Object Record B and returning both to the validating entity
7. The validating entity can use the report to validate the trustworthiness of the PSA Root of Trust and its implementation, validate the authenticity of Object Record B, and validate the context of the original challenge (Object Record A).
8. A secure connection can now be established to the attested end point using an appropriate session key negotiation mechanism for the chosen attestation protocol (for example, an SSL or TLS connection, or completing a DH exchange) based on challenges encoded in object records A and B.

Identifier	Rule	Rationale	Information
	The system must be provisioned at manufacture with an immutable initial attestation key.		
	The initial attestation key must be an asymmetric key.	To allow validation by arbitrary entities in an ecosystem.	RSA or ECC.
	The system must be provisioned at manufacture with an immutable Instance ID, uniquely identifying the initial attestation key.	Serves as a unique secure identifier for a PSA RoT instance.	A hash of the public part of the initial attestation key.
	The system must be provisioned with an immutable Implementation ID, uniquely identifying the underlying implementation of the Immutable PSA RoT.	It must be possible to verify a link from an initial attestation key to the underlying implementation of the Immutable PSA RoT, and its properties.	
	A calling partition must be able to make a contribution to the initial attestation – auth_challenge.	Allow a calling partition to bind additional metadata to an initial attestation.	A hash of the metadata to be bound to the initial attestation.

	The object record for the initial attestation service must include at least the following security information from the main boot state: a) The boot state for each updateable component loaded at boot b) The current security lifecycle state of the system c) A boot seed d) The Instance ID of the system e) The Implementation ID of the system		The boot seed may be used by a validating entity to ensure multiple reports were generated in the same boot session. For example, if an Application Root of Trust provides multiple attestation end points, or in extended attestation use cases.
	The initial attestation service shall produce an initial attestation token (IAT) containing: a) Object record C b) Auth_challenge c) Calling partition ID d) Signature using the initial attestation key	This binds application-specific attestation protocol parameters (auth_challenge) to the boot state and identity of the PSA Root of Trust, and to the calling partition	

5.3 Delegated attestation (optional)



Secondary Attestation Key
Object Record C
Object Record D
Object Record B
Object Record A
Initial Attestation Key
Initial Attestation
Secondary Attestation
Attested End-point
Validating Entity
auth_challenge 1 = H(SecondaryAttKey)
Attestation request Auth_challenge3
digest = H(auth_challenge1|ObjectRecordC)
Sign (digest)
IAT=ObjectRecordC| auth_challenge1|sign)
Attestation report IAT
Challenge ObjectRecordA
auth_challenge2 = H(ObjectRecordB|ObjectRecordA)
Attestation request Auth_challenge1
digest = H(auth_challenge1|ObjectRecordD)
Sign digest
SecondaryAttestation=ObjectRecordD| auth_challenge1|sign)
Attestation report IAT, SecondaryAttestation
Response ObjectRecordA, ObjectRecordB, IAT, SecondaryAttestation

The basic attestation model outlined previously can be extended to allow delegated attestation in an attestation chain. This model is expected to be more common on multi-stage boot devices in which each stage attests its own state.

The example in the figure above shows outlines the main differences with a basic two stage delegated attestation for a two-stage boot scenario, but the same pattern can be applied to other configurations. It extends the basic attestation model already described, so all details are not repeated here.

1. The boot ROM validates and measures the boot state of the PSA Root of Trust and the Application Root of Trust (SPE) and records the result in the initial boot record (object record C), including the current boot seed, as defined by the boot ROM
2. A secondary loader stage in the Application Root of Trust validates and measures the application (N-SPE) and records the result in a secondary boot record (object record D), including the current boot seed as defined by the boot ROM
3. A secondary attestation service generates its own secondary attestation key at boot (random), and binds it to the initial attestation token

   This binds the secondary attestation key to the initial boot state of the system (Object record C), and to the partition ID of the secondary attestation service partition (the owner of the secondary attestation key) – an example of *key attestation*.
4. Attestation end points now request attestations from the secondary attestation service
   a) The secondary attestation service generates a secondary attestation report, covering the secondary boot state (object record D)
   b) The secondary report is signed by the secondary attestation key
   c) The secondary attestation service returns both the initial attestation token (key attestation of the secondary attestation key) and the secondary attestation report (signed by the secondary attestation key)
5. A validating entity can now:
   a) Validate that the boot seeds match in both reports (they were both generated in the same boot session)
   b) Verify the implementation and the root of trust software using the initial attestation token
   c) Validate the secondary attestation key using the initial attestation token
   d) Validate the secondary report and object records A and B using the secondary attestation key
   e) Verify the application software using the secondary attestation report
   f) Proceed with the attestation protocol based on object records A and B

The secondary attestation service is now able to produce attestation reports without having to go all the way back to the PSA Root of Trust. The initial attestation report is valid for an entire boot session, as its security state can only change following reset, but those reports are still bound to the initial attestation of the PSA Root of Trust and can be traced to the implementation of the Immutable PSA RoT.

5.4 Note on attestation and freshness (informative)

It is essential for an attestation system to be able to correctly and reliably determine he state of the attested end up and the state of the underlying root of trust. This includes when a device has entered a debug state.

PSA provides two generic mechanisms which can be used to guarantee freshness of attestation reports:

1. Include a nonce or other freshness data from the validating entity in all attestation reports

   Including such a nonce in the authentication challenge guarantees that a report was generated freshly. This method is illustrated in the basic attestation example above.

   If this method is to be applied to a delegated attestation model, then it is essential that the same freshness nonce is included in the auth_challenge for both the initial attestation token and the secondary attestation report. Using a freshness nonce in this way precludes caching of the initial attestation token.

2. In the case of delegated attestation, derive the secondary attestation key from a secured binding root key

   Using this method, it is not possible for the secondary attestation service to gain access to the same secondary attestation key if the device is not in the secured lifecycle state.

   If this method is applied to delegated attestation then the validating entity can be assured that if it can validate a signature it must have been created by the secondary attestation service while it was in the secured security lifecycle state. Since the secondary attestation service can be trusted while the device is in the secured security lifecycle state, the secondary attestation service can cache the initial attestation token without reducing security.

   A freshness nonce may still be included each time a secondary attestation report is generated, if required by the attestation protocol, but it does not need to be included in the initial attestation token.

5.5 Attested trusted subsystems (mandatory)

Trusted subsystems are additional components and security features provided by a device, required by specific certification profiles but not defined in this document.

Any updateable components for such subsystems must be measured and included in the boot state, and must be attested as part of initial attestation.

Identifier	Rule	Rationale	Information
	Boot state recorded by the boot ROM for any validated and measured trusted subsystem must be included in the initial attestation.	Same model as for attesting the boot state of any other updateable component.	

6 Storage

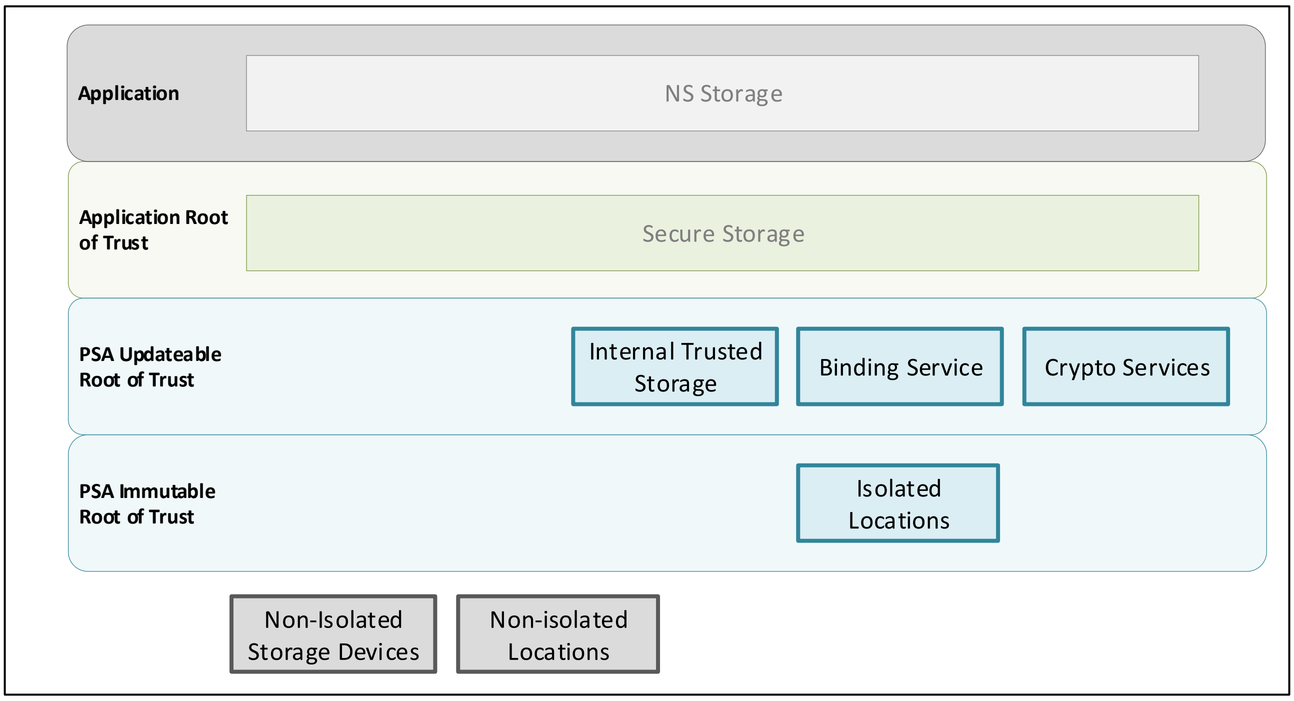
6.1 Overview (informative)

PSA requires secure storage services to be provided for any *sensitive data* stored on the device: for example private data, secrets, keys and key materials. From manufacture provisioned secrets to application generated and service provisioned secrets, and for user generated private data.

Secure storage services in general need to provide:

1. Access control policies – ownership of sensitive data
2. Privacy and integrity protection – prevent sensitive data from being accessed or modified by an unauthorized agent
3. Replay protection – prevent a stored set of sensitive data from being replaced by a previously stored version of the same data set
4. Protection against unauthorized access to sensitive data when the device is in a non-secured state, for example debug modes

Depending on implementation requirements and certification profile these properties may be enforced by PSA isolation, or cryptographically, or in many cases in some combination.



The PSA security model defines common PSA RoT level building blocks for secure storage:

1. *Internal trusted storage* – a simple storage model intended as a basic storage service on devices with limited storage needs, or as storage for root secrets and root metadata for application level (ARoT or NS) storage services
2. A *binding service*, allowing ARoT and NS partitions to bind data to a device instance, a partition, and the security lifecycle state

Using these building blocks, storage services of varying complexity and capability can be built. For example:

- Simple storage, using only Internal Trusted Storage, based on on-chip storage locations protected by PSA isolation and robustness rules
- Application-specific storage services utilising off-chip storage locations or removable storage devices, perhaps with application-specific access control policies, for example protected files systems, secure database or object store.

  In this case, Internal Trusted Storage may be used to keep root metadata for integrity trees, and the binding service used to derive key ladders bound to a device instance and its security lifecycle state.

6.2 Physical storage (mandatory)

Physical storage represents actual persistent storage media. For the purpose of the security model, the PSA architecture defines the following classes of physical storage:

Storage type	**Isolation Property**	**Use cases** (examples)	**Notes**
Isolated locations	Memory locations which are inaccessible to any agent external to the system. Only directly accessible by PSA RoT.	Device configuration data. Simple persistent key and data store. Root metadata for application level storage services.	Typically on-chip flash or NV memory with restricted external access (for example, disabled I2C). Partition based access control enforced by PSA RoT provides additional isolation and access control between secure partitions making use of isolated locations.
Shielded locations	An isolated location or storage device specifically designed and used for the purpose of securely storing and protecting secrets; for example, a tamper resistant memory or register. Only directly accessible by PSA RoT.	PSA root secrets Application-specific provisioned root secrets	For example: inseparable SE, on-chip NV memory, or dedicated sector of on-chip Flash. Shielded locations should incorporate a degree of tamper resistance. The extent of tamper resistance will depend on certification profile.
Non-isolated locations	General purpose memory mapped persistent storage resources. No PSA RoT enforced access control.	Application and user generated data. For example, file systems or object stores.	Typically external flash.
Non-isolated storage devices	General purpose storage devices. No PSA RoT enforced access control.	Application and user generated data. For example, file systems or object stores.	For example, USB storage devices or removable memory cards (other than any devices used to implement a shielded location).

The following rules apply to physical storage:

Identifier	Rule	Rationale	Information
	Tamper resistance: Shielded locations must provide a degree of tamper resistance.	Shielded locations typically hold provisioned secrets, including PSA RoT secrets. A degree of tamper resistance should be applied to protect from attempts to extract such secrets.	Depending on certification profile and deployment requirements, tamper resistance may address a range of issues, including: • Physical access control (debug interfaces, external interfaces, physical tamper proofing) • Side-channels (for example power and timing analysis) • Active probing (for example physical disassembly, or access to internal buses and interfaces, debug interfaces) • Passive probing (for example x-ray or electron microscopes)
	Isolation: Isolated locations and shielded locations must only be directly accessible to PSA RoT.	These types of locations typically relyon inaccessibility for privacy and integrity protection. Access control and ownership policies must be enforced by the PSA RoT.	
	Debug interfaces: Isolated locations and shielded locations must be inaccessible via any external interfaces on devices in the Secured and Non-PSA RoT Debug security lifecycle states.	These types of locations rely on inaccessibility for privacy and integrity protection. Access control and ownership policies must be enforced by the PSA RoT while the device remains in an attestable state.	External interfaces include, for example, USB, I2C, JTAG, EJTAG, Boundary Scan, and serial port debugging.

6.3 Internal trusted storage (mandatory)

Internal trusted storage is a PSA RoT service providing secure partition level access control for data stored in isolated locations. Isolated locations should only be accessible through the internal trusted storage service.

Use cases for internal trusted storage include, for example:

1. A simple store for data on devices with limited storage needs
2. Storage of basic device configuration data
3. Storage for root keys and root metadata on devices with more complex Application RoT or NS level storage services

Individual data objects are associated with an owning partition, and only the owning partition can access or modify data in a stored data object.

In some cases, internal trusted storage may rely solely on the physical inaccessibility property of isolated locations, together with PSA isolation, without requiring additional cryptographic protection.

However, cryptographic protection bound to the security lifecycle state of the device (binding) is mandatory if internal trusted storage is used for sensitive data that should not be accessible to debugging agents.

Identifier	Rule	Rationale	Information
	Isolated locations must only be directly accessible to the PSA RoT.	Ability to enforce partition based access control rules for isolated locations.	Enforced by PSA isolation.
	It shall be possible for secure partitions to store data objects in isolated locations.	Provide simple storage for devices with limited storage needs. Provide root metadata storage for devices with more complex storage requirements.	
	Each data object stored in isolated locations shall be associated with an owning partition.		Enforced by the PSA RoT.
	Only the owning partition shall be able to read, modify, or delete a data object.	Provide access control at secure partition level for data objects stored in isolated locations.	Enforced by the PSA RoT.
	Data which should not be accessible to a debugging agent when the device is in any debug state in the PSA security lifecycle, must be cryptographically protected based on the binding root key.	Debug states potentially expose any data or secrets stored on a device to the debugging agent, either directly by compromising the internal trusted storage service, or indirectly by providing access to stored data from a debugged partition.	See **PSA security lifecycle**. See **Binding root key**.

6.4 Binding (optional)

Binding is a generic PSA RoT service allowing partition to request one or more *partition specific binding keys* for its own unique use.

Partition specific binding keys are intended to be used in cases in which a secure partition needs to store data in non-isolated storage while ensuring that:

- Other secure partitions on the system cannot directly access its data – *partition binding*
- Its data and secrets can only be accessed on the device they were created on – *device binding*
- Debugging agents cannot directly access any of its sensitive data or secrets – *security lifecycle binding*

Partition specific binding keys are always derived on request and never stored persistently.

A typical example might be a secure storage service managing key ladders and integrity trees for encryption, integrity protection, and replay protection for the data it manages. This in turn is rooted in some storage root key, and root metadata. The partition specific binding key may be used to either encrypt or derive such root state in a way that ensures that no other partition can access that state, that the state can only be recovered on the same device it was originally created, and that the state can only be recovered in specific security lifecycle states.

Identifier	Rule	Rationale	Information
	A secure partition shall be able to request a partition specific binding key.	Bind data and protocols to the calling partition, a device instance, and the security lifecycle state of the device.	

Identifier	Rule	Rationale	Information
	Partition specific binding keys must always be derived fresh when first requested following boot, and must never be stored persistently.	Partition specific binding keys depend on lifecycle state, device unique root keys, and other state that may change across device reset events.	A derived partition specific binding key may be cached in volatile memory private to the PSA RoT.
	It must not be possible to export a partition specific binding key.	Partition specific binding keys should only be available for crypto operations.	
	The calling secure partition shall be able to make the following contributions: 1. Usage policy 2. Seed 3. Debug policy	The seed is an application-specific contribution, allowing a calling partition to derive multiple different partition specific binding keys for different use cases. The policies control how and when a partition specific binding key can be used, see below.	
	Usage policy shall be one of: 1. Key derivation 2. Encryption 3. Signing	Prevent abuse of a binding key. For example, preventing a key intended to be used for key derivations (key ladders) from being used for direct encryption/decryption. Or preventing a signing key to be used for encryption/decryption. Either example potentially leading to security issues.	Key derivation: The derived key can only be used to derive other keys. Encryption: The derived keys can only be used for encryption/decryption operations. Signing: The derived keys can only be used for signing/validation operations.
	It shall not be possible to derive the same partition specific binding key for different usages.	Prevent abuse of a binding key from accidentally or maliciously deriving the same root key for different debug policies.	Use different binding root keys for different debug policies, see below.
	Debug policy shall be one of: 1. Secured 2. Non-PSA RoT debug	Secured: The same partition specific binding key can only be derived in the secured security lifecycle state. Non-PSA RoT debug: The same partition specific binding key can be derived in the secured security lifecycle state, and in any debug state which does not compromise the PSA RoT.	If no debug policy is specified then it shall default to Secured.
	It shall not be possible to derive the same partition specific binding key for different debug policies.	Always derive different keys for different debug policies.	Use different binding root keys (see below) depending on debug policy.

Identifier	Rule	Rationale	Information
	The following contributions shall always be included when deriving a partition specific binding key: 1. Calling partition ID 2. An appropriate binding root key (BRK) depending on debug policy	Calling partition ID: Binds the derived key to a secure partition BRK: Binds the derived key to a device instance, and its security lifecycle state.	See **Binding root key**

6.5 Binding root key (mandatory)

The PSA security model defines a generic *Binding Root Keys* (BRK) to be used as a root keys for deriving other binding keys.

Two types of BRK are defined in this specification:

1. Secured BRK – the same BRK can only be derived if the device is in the Secured security lifecycle state
2. Non PSA RoT debug BRK – the same BRK can be derived if the device is in the Secured security lifecycle state, or in any debug state which does not compromise the PSA RoT

In either case, a BRK always depends on:

1. A persistent (immutable) hardware unique identifier (HUK) provisioned at device manufacture – see **Root parameters**

BRK are always derived fresh at boot, and form part of the boot state – see **Boot State**.

Identifier	Rule	Rationale	Information
	Binding root keys (BRK) must be derived fresh by boot code on every boot: 1. Secured BRK 2. Non PSA RoT debug BRK	The BRK depends on the PSA security lifecycle state of the device, and the HUK, and hence may change following device reboot.	
	Derived BRK shall be included in the boot state.	Only directly available to PSA RoT, and derived as part of the boot process.	
	BRK derivation must depend uniquely on the hardware unique key (HUK).	Including the HUK in a BRK derivation makes the BRK unique to a specific instance of the PSA RoT.	
	It must only be possible to derive the same Secured BRK if the device is in the Secured PSA security lifecycle state.	Ensure that anything bound to the Secured BRK is not available in any debug states.	

Identifier	Rule	Rationale	Information
	The same Non PSA RoT debug BRK should be derived both on the Secured PSA RoT security lifecycle state, and in any debug state which does not compromise the PSA RoT.	Allow the same BRK to be derived as long as the PSA RoT has not been compromised. Should only be used where it is acceptable that data protected by the BRK is available to debug agents.	
	The BRK derivation must be a cryptographic key derivation function.	The BRK is used for cryptographic binding and BRK derivation must result in a cryptographically safe key.	See **PSA key management**.
	A BRK must only be used for deriving other keys.	Ensure key diversity by not using root keys, for example BRK, directly.	

6.6 Use cases (Informative)

6.6.1 Simple data store

6.6.2 Simple key store

6.6.3 General data store

6.6.4 General key store

7 Cryptographic services

7.1 General (mandatory)

The PSA RoT provides basic cryptographic services to other code on the device. Implementations of PSA cryptographic services must be designed with the following essential security properties:

1. Isolation: Provide the ability to manage actual values of keys and other secret cryptographic materials within the PSA RoT crypto service when appropriate
2. Access control: Manage access to keys and other cryptographic materials at secure partition granularity
3. Policy: Provide the ability to control usage policy for secrets managed by the PSA cryptographic services such that they can only be use for specific purposes

Isolation allows device code to be designed such that keys and other secrets are not exposed to less trusted software.

Access control ensures that a partition can neither access values of nor perform operations using keys and other secrets belonging to a different partition, allowing code to be designed such that keys used by one secure partition (service) cannot be accessed by other partitions.

Finally, policies allow designers of code running on a device to restrict how individual keys and secrets owned by a particular partition can be used by that partition, preventing misuse whether deliberate or unintentional.

Identifier	Rule	Rationale	Information
	The PSA RoT cryptographic services must be able to manage keys and other secrets on behalf of other less trusted code, such that less trusted code cannot directly access values of such keys and secrets.	Allow code to be designed, where appropriate, such that less trusted code does not need to have direct access to keys and secrets.	
	The PSA cryptographic services must be able to manage persistent keys and secrets stored in isolated locations or shielded locations.	For example, support ARoT or NS level provisioned root keys.	
	All keys and secrets managed by the PSA RoT cryptographic services must always be assigned a unique owning secure partition.	A secure partition should not be able to access or use secrets owned by a different partition unless explicitly authorized to do so.	This includes any keys or secrets originating from isolated locations or shielded locations, for example provisioned root keys.
	It must be possible for an owning partition to delegate the usage of a key or secret it owns to another partition.	Allow keys or secrets to be generated or derived by one (more trusted) partition, and used by a different (less trusted) partition. Allow general application-specific secure key management services to be built on top of PSA RoT crypto services.	Subject to policy.
	All keys and secrets managed by the PSA RoT cryptographic services must always be assigned a unique usage policy.	It must be possible for designers of code using PSA cryptographic services to restrict how keys and secrets can be used.	
	The following minimum set of policies must always be supported: 1. Usage: encryption, signing, key derivation 2. Export: No export, clear export, export wrapped 3. Delegation: Usage and export policies for keys delegated for use by other partitions		Usage and export policies should include allowing or restricting the use of specific algorithms with a particular key or secret. Note that specific algorithms and features supported by a particular implementation may vary.

7.2 Cryptographic algorithms and key sizes (informative)

Required cryptographical algorithms and key sizes will vary depending on use case, and by market and geographic region.

As a general recommendation, in the absence of specific requirements by application, certification profile, or regulation, PSA-compliant devices should be designed to comply with NIST recommendations, or local equivalents depending on target region:

https://csrc.nist.gov/Projects/Cryptographic-Standards-and-Guidelines

8 Appendices

8.1 Mapping to TMSA security objectives

Threat Models and Security Analyses (TMSA) represent a suite of documents produced by Arm providing use case specific threat model analysis for a number of target PSA applications.

Each TMSA contains a set of security objectives. A security objective mitigates one or more identified threats within the TMSA.

The PSA Security Model (this document) defines a security architecture designed to address a generic set of threats identified in TMSA, providing a security foundation covering all anticipated PSA applications.

This section shows the corresponding generic PSA Security Model goals for each identified TMSA security objective.

Security objective	Explanation	PSA security requirements chapters(mandatory)	Optional PSA requirements	Notes
OT.ACCESS_CONTROL	The TOE shall authenticate Remote and Local Admin entities before granting access the water meter configuration and logs and before performing firmware update.	**PSA Root of Trust** **Initial attestation** **PSA security lifecycle**		The PSA Root of Trust provides protection of immutable secrets and enforces isolation between components.
OT.SECURE_STORAGE	The TOE shall protect integrity and confidentiality of Credentials when stored, and protect integrity of Firmware Certificate, Configuration and Logs when stored.	**Storage** **Cryptographic services** **PSA security lifecycle**		The PSA Root of Trust binding API and cryptography API can be used to build a secure storage service within the Application RoT.
OT.FIRMWARE_ AUTHENTICITY	The TOE shall authenticate and verify integrity of firmware image during boot and of new firmware versions prior upgrade.	**Boot** **PSA root of trust**		

	The TOE shall also reject attempts of firmware downgrade.			
OT.COMMUNICATION	The TOE shall only accept remote connections from configured back-end servers and be able to authenticate these servers. The TOE shall also provide authenticity, confidentiality and replay protection for export outside of the TOE.	**Cryptographic services** **PSA security lifecycle** **Initial attestation** **Storage**		PSA does not specify protocols, but provides the security building blocks for storing secretes and for binding protocols to a device and its state.
OT.AUDIT	The TOE shall maintain log of all significant events and allow access and analysis of these logs to authorized users only.	**Storage** **Cryptographic services** **Initial attestation**		PSA does not specify protocols, but provides the security building blocks for storing secretes and for binding protocols to a device and its state.
OT.SECURE_STATE	The TOE shall maintain a secure state even in case of failures, for instance failure of verification of firmware integrity.	**Boot** **PSA security lifecycle** **PSA root of trust**		
OT.TAMPER		PSA does not specify protocols, but provides the security building blocks for storing secretes and for binding protocols to a device and its state. Please see Arm's Trusted Base System Architecture (TBSA-M)		Detailed tamper resistance requirements are expected to vary depending on robustness level and ecosystem requirements, and are not discussed further in this document.

8.2 Hardware example: Implementation based on Armv8-M with CryptoCell based MCU

The following reference design is based on a simplified Arm Musca-B1 test chip. A high-level diagram of the architecture is shown below in Figure 1.

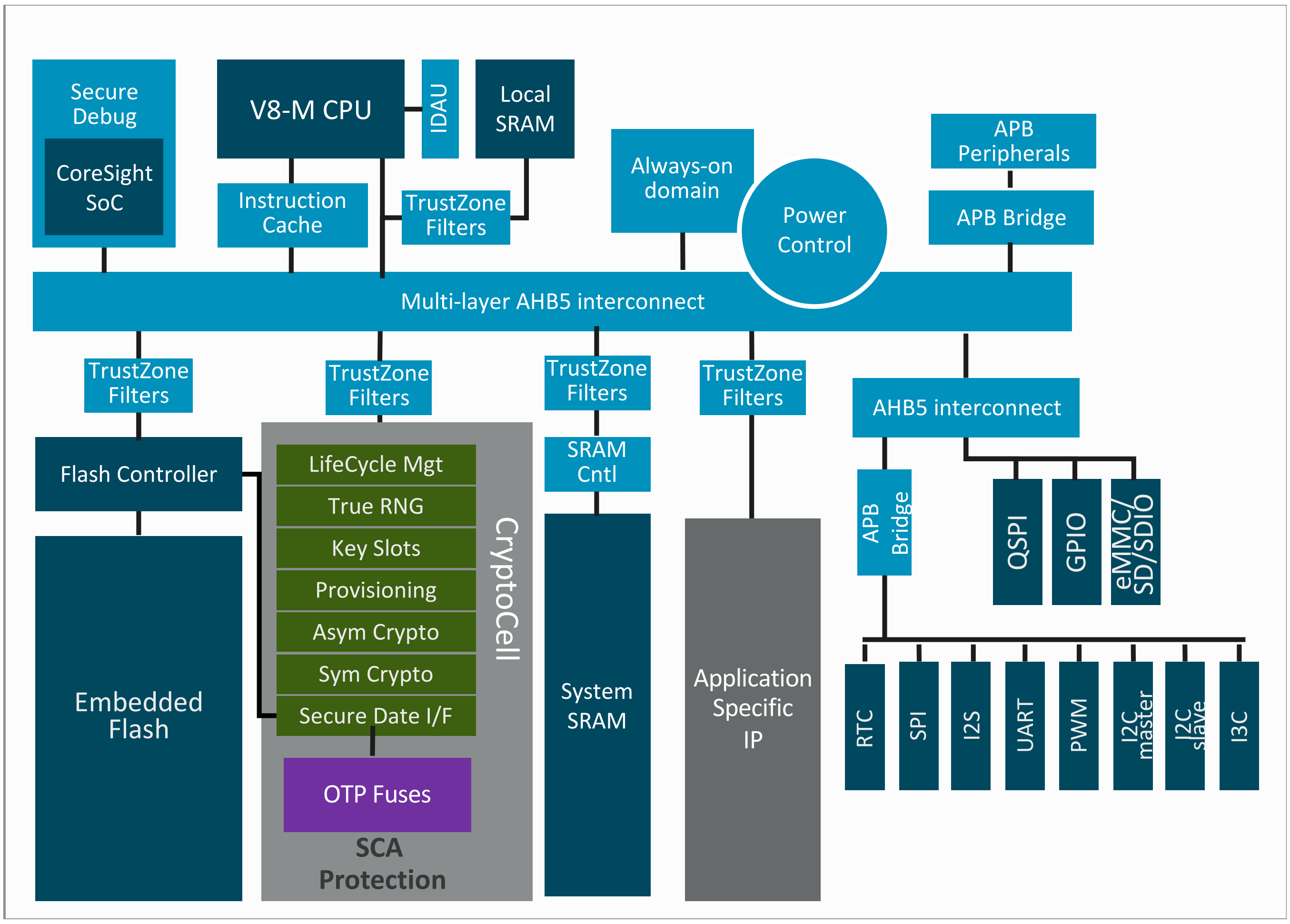


Figure 1 Reference Architecture

8.2.1 Architecture Description

CPU

The design uses an Armv8-M CPU with the security package (TrustZone Support) implemented. It implements two MPUs – one dedicated for the non-secure execution state and the other dedicated to the secure state (MPU_S). It also implements a Secure Attribution Unit (SAU) with 8 entries. The CPU subsystem includes a 2KB instruction cache, an Implementation Defined Attribution Unit (IDAU), and a block of Tightly coupled RAM dedicated to the CPU.

Interconnect

The interconnect uses Arm Corelink SSE-200 interconnect centred on AMBA AHB5 Bus Matrix.

The interconnect is supplemented by TrustZone filters at each slave port which can accept or deny transactions dependent on their security attribution.

Non-Volatile Memory

The system on chip integrates several partitions of embedded Flash – the partitions are lockable by fuse enabling their contents to become immutable. Boot ROM is implemented in this manner. The ROM is written at manufacture and a permanent fuse is set so that subsequent update is not possible.

In addition there is several kilobits of One-Time-Programmable (OTP) efuses. These are used to store IDs, device keys and other secrets, and non-volatile device flags.

CryptoCell

The TrustZone CryptoCell is a trusted subsystem providing platform security services and a set of cryptographic services. It supports the following functions:

- Cryptographic acceleration hardware for the protection of data-in-transit (communication protocols) and data-at-rest
- Protection of various assets belonging to different (optional) stakeholders (IC vendor or device manufacturer or service operator or user). These asset protection features include:
    - Image verification at Boot or during Runtime
    - Authenticated Debug
    - Random Number Generation
    - Lifecycle Management
    - Provisioning of assets

System RAM

This reference architecture integrates a Block of system RAM to support the application. The RAM is situated behind a TrustZone filter allowing it to be partitioned into secure and non-secure regions. The secure regions are exclusively accessible to software running in the secure execution state of the processor.

8.2.2 Mapping PSA Isolation

PSA isolation enforces isolation boundaries to separate the device firmware into partitions.

One such mapping onto this reference architecture is shown in Figure 2.

Application

Most of the application runs in the unprivileged mode of the non-secure execution state of the processor. The RTOS and device drivers are commonly also mapped to the non-secure execution state. The application software together with its operating system and drivers is mapped into PSA non-secure processing environment.

Updateable Root of Trust

There are two types of updateable roots of trust in a PSA system – the PSA Root of Trust and the Application Root of Trust. See **PSA Root of Trust**. Both are mapped into the Secure Processing Environment which executes in the secure execution state of the CPU.

- The Application Root of Trust implements functions specific to the application – for example TLS primitives and application level secure storage

- The PSA Root of Trust implements functions common to all PSA platforms and forms the most trusted firmware on the device

  The PSA Root of Trust has exclusive access to the hardware security systems. Interacting with the CryptoCell and other root of trust components, for example the TrustZone filters, can only be carried out by PSA Root of Trust functions.

Immutable Root of Trust

Some of the PSA Root of Trust is defined to be immutable and cannot be updated after manufacture.

In this design, this comprises the Boot ROM, which is provisioned into embedded flash and locked at manufacture, and also the OTP fuses, which are commonly provisioned in a secure environment with a set of assets, namely keys and identifiers required by both PSA and the application. The PSA assets are described in the security model and also TBSA-M for Armv8-M. In this design, the CryptoCell manages OTP fuses in terms of access and their life-time guarantee. The CryptoCell also uses the OTP fuses to support management of the device lifecycle, including the PSA security life cycle for the PSA Root of Trust.

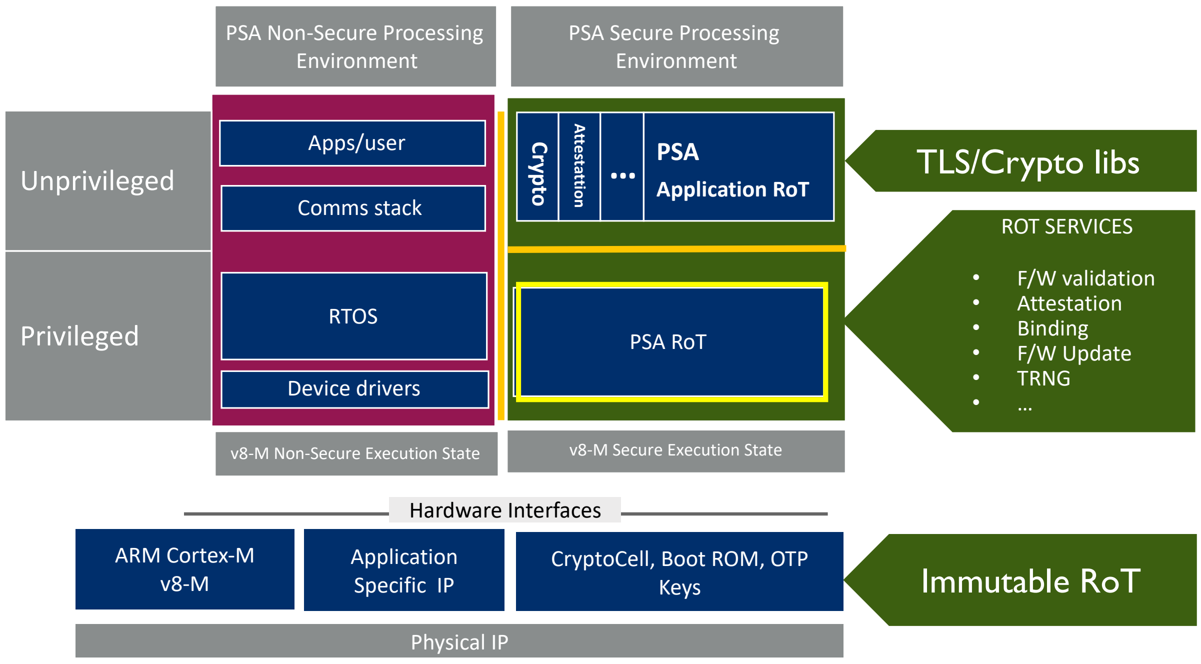


Figure 2 Mapping PSA Firmware Framework

Some of the Root of Trust firmware is updateable. The chain of trust anchored in the Immutable RoT measures and validates all new firmware images. In this reference architecture all RoT firmware is stored in the embedded flash macro.

8.2.3 Mapping PSA isolation boundaries

A key aspect of all PSA implementation is the hardware support provided for the isolation boundaries. In this example a level 2 isolation system is realized but higher levels of isolation can also be supported in this hardware architecture.

Level 1 Boundary

The level 1 boundary between the NSPE and the SPE is implemented with TrustZone. NSPE corresponds to the Non-Secure state and SPE corresponds to the Secure state. Via use of the NS-bit carried by all interconnect transactions this isolation boundary is implemented through the device. The SAU, IDAU and TrustZone filters must be configured appropriately to implement this boundary.

Level 2 Boundary

The level 2 boundary sits between the application RoT and the PSA Root of Trust.

For software running on the Armv8-M processor the boundary is implemented by MPU_S, the memory protection unit dedicated to processes executing in the secure state.