psacertified™

Platform Security Model

1.1

Document number:	JSADEN014
Release Quality:	Beta
Release Number:	0
Confidentiality	Non-Confidential
Date of Issue:	01/12/2021

Abstract

This Platform Security Model motivates the set of platform security specifications that target internet and other forms of connected compute-centric devices. This document underlies the PSA Certified™ framework and certification scheme.

Key goals for designing devices based on essential security properties are described in this security model. These essential properties are used to define a Platform Root of Trust for a platform on which end-products can be built. This Platform Root of Trust covers the set of hardware and firmware necessary to support secure and non-secure processing.

A connected device needs to operate within an ecosystem. This document outlines how such a device and some of the essential security features might fit within an ecosystem. It does this by illustrating typical entities, capabilities and processes required to securely deploy connected services.

Contents

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
February 2020	1.0 Alpha 3	Non-Confidential	Revised for accessibility
October 2020	1.0 Beta 0	Non-Confidential	Clarifications and compatibility with PSA Certified
May 2021	1.0 Rel 0	Non-Confidential	Document is stable.
December 2021	1.1 Beta 0	Non-Confidential	Rewrite of section 2 to reflect existing design patterns and anticipated future patterns, minor changes in sections 3 and 4

Further Reading

Author	Document Number	Title
Arm	DEN 0112	Platform Threat Model and Security Goals
Arm	DEN 0106	Platform Security Requirements
Arm	DEN 0072	Platform Security Boot Guide

Platform Security Model

Arm Non-Confidential Document License ("License")

This Licence is a legal agreement between you and Arm Limited ("**Arm**") for the use of Arm's intellectual property (including, without limitation, any copyright) embodied in the document accompanying this Licence ("**Document**"). Arm licenses its intellectual property in the Document to you on condition that you agree to the terms of this Licence. By using or copying the Document you indicate that you agree to be bound by the terms of this Licence.
"**Subsidiary**" means any company the majority of whose voting shares is now or hereafter owner or controlled, directly or indirectly, by you. A company shall be a Subsidiary only for the period during which such control exists.

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

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

(i) use and copy the Document for the purpose of designing and having designed products that comply with the Document;
(ii) manufacture and have manufactured products which have been created under the license granted in (i) above; and
(iii) sell, supply and distribute products which have been created under the license granted in (i) above.

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

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

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

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

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

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

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

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

The Arm corporate logo and words marked with ® or ™ are registered trademarks or trademarks of Arm Limited (or its subsidiaries) in the US and/or elsewhere. All rights reserved. Other brands and names mentioned in this document may be the trademarks of their respective owners. No licence, express, implied or otherwise, is granted to Licensee under this Licence, to use the Arm trade marks in connection with the Document or any products based thereon. Visit Arm's website at https://www.arm.com/company/policies/trademarks for more information about Arm's trademarks.

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

Arm Limited. Company 02557590 registered in England.

110 Fulbourn Road, Cambridge, England CB1 9NJ.

Arm document reference: LES-PRE-21585 version 4.0

Document Outline

Section 1: Platform Security Model
: Introduces the Platform Security Model and its goals, the relevance of a device within an overall ecosystem, discusses typical device models.

Section 2: Platform Root of Trust
: Introduces the Platform Root of Trust (PRoT), PRoT services, ARoT services. The concepts of partitions, partition management and processing environments are also introduced. The elements in a typical implementation are presented.

Section 3: PRoT Parameters
: Introduces the minimal set of Platform Root of Trust (PRoT) data items to identify and secure the platform.

Section 4: PRoT Security Lifecycle
: Introduces a generic security lifecycle for the Platform Root of Trust.

Section 5: PRoT Binding
: Introduces the minimal set of Platform Root of Trust (PRoT) data items to identify and secure the platform.

Section 6: PRoT Secure Boot and Firmware Update
: Introduces secure boot, firmware update, and system suspend and hibernation.

Section 7: PRoT Services
: Introduces the elements of the Platform Root of Trust, including Internal Trusted Storage, Binding, Cryptography and Initial Attestation service.

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

Current Status and Anticipated Changes

First alpha release, minor changes and clarifications to be expected.

Feedback on this Book

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

- The title (Platform Security Model)
- The number and release (JSADEN014 1.1 Beta 0)

- The page numbers to which your comments apply
- The rule identifiers to which your comments apply, if applicable
- A concise explanation of your comments

We also welcome general suggestions for additions and improvements.

Open Issues

Appendix mapping to PSA Certified requirements
Appendix mapping to TMSA security objectives.

1 Platform Security Model Overview

This Platform Security Model (PSM) is one document in a holistic set that includes threat models and security analyses, security requirement specifications and application programming interfaces, all architecture-agnostic. Together with an open-source reference implementation and test suites, this enables consistent design-in at the right level of security.

This framework builds upon best practice from across the industry and is aimed at different entities throughout the supply chain, from chip designers, software vendors and device developers to cloud and network infrastructure providers. Though the focus is on compute-centric local network or internet connected devices, many aspects are relevant for non-connected devices. No assumptions are made about the solution architecture, only that the properties described are met whether using a resource- and performance-constrained microcontroller or a resource-rich powerful microprocessor-based platform.

This security model is the top-level document for the other platform security specifications and defines a common language, high-level robustness rules, and models.

Products may go through a security evaluation, such as PSA Certified, to provide a measure of the robustness of the implementation.

1.1 The 10 Security Goals

The set of core security goals given below provide a high-level, abstract, way to think about the essential features that are necessary to secure and establish trust. Abstraction allows these goals to be applied as required, for example, to an end user connected device, a hardware component, a software component, or a service. In describing the goals, the term *device* is used to represent any entity at any level that must be secure and trustworthy.

Goal 1: Devices are uniquely identifiable.

In order to interact with a specific device instance, that instance must be uniquely identifiable. The identity must be attestable and that attestation verifiable as a means of proving the device identity, see Goal 3.

Goal 2: Devices support a security lifecycle.

The security state of a device within its security lifecycle depends on software versions, run-time measurements, hardware configuration, status of debug ports, and on the product lifecycle phase. Product lifecycle phases include, for example, development, deployment, returns, and end-of-life. Each security state defines the security properties of the device. The security state must be attestable, see Goal 3, and may impact access to bound data, see Goal 9.

Goal 3: Devices are securely attestable.

The trustworthiness of a device must be established. This requires that its identity, see Goal 1, and security state, see Goal 2, are proven through attestation. To have validity, device identification and attestation data must be part of a device governance regime.

Goal 4: Devices ensure that only authorized software can be executed.

Secure boot and secure loading processes are necessary to ensure that only authorized software can be executed on the device. See also Goal 6. Allowing unauthorized software is acceptable only if such software cannot compromise the security of the device.

Goal 5: Devices support secure update.

Device software, credentials, programmable hardware configuration, must be updateable to resolve security issues or to provide feature updates. Updates must not compromise the device security. Authentication of an update is required. However, execution of any updated software must be authorized in accordance with Goal 4.

Goal 6: Devices prevent unauthorized rollback of updates.

Updates are necessary to resolve known security issues, or provide feature updates, see Goal 5. Preventing rollback, known as anti-rollback, to a previous version with a known (and subsequently fixed) vulnerability is essential. However, authorized rollback for recovery purposes may be allowed.

Goal 7: Devices support isolation.

Isolation of a trustworthy service from less trusted or untrusted services is essential to protect the integrity of that service. More generally, isolation boundaries aim to prevent one service from compromising other services, for example, between any on-device services and between on-device services and the connected world.

Goal 8: Devices support interaction over isolation boundaries.

Interaction over isolation boundaries, see Goal 7, is essential if isolated services are to serve a purpose. Any such interaction must not be able to compromise the interacting services or device. This will require validation of exchanged data. It may also be necessary to ensure the confidentiality and integrity of any data exchanged.

Goal 9: Devices support unique binding of sensitive data to a device.

Sensitive data, for example, user or service credentials, or secret keys, must be bound to a device to prevent disclosure outside of the device. It may also be required to bind such data to prevent disclosure beyond its owner. Inherently secure storage or confidentiality and integrity assured storage may be used. Where binding relies on cryptography and keys, see Goal 10, the keys are sensitive data and so must be bound to the device or the data owner. It may also be necessary to bind the data to the security state, for example, to deny access during debug, see Goal 2.

Goal 10: Devices support a minimal set of trusted services and cryptographic operations that are necessary for the device to support the other goals.

Trusted services may include configuration of the hardware to support security lifecycle (see Goal 2), isolation (see Goal 7), and cryptographic services that may use bound secrets (for example, keys) used to

support attestation (see Goal 3), secure boot and secure loading (see Goal 4), and binding of data (see Goal 9). The trusted services must be kept as small as possible to enable analysis and reduce the likelihood flaws.

These goals inform and are embodied in the platform security specifications that are designed to help in the development and deployment of secure products. It is recommended that all features are implemented. However, the features supported to secure a device are determined by, for example, the intended application domain and cost, by applicable threat models, by applicable national standards, by ecosystem operators, and by any certification scheme. However, implementing the security features to increase the security level and provide product differentiation is a motivation behind this document.

1.2 Connected Devices Within an Ecosystem

Devices are expected to be deployed within the context of an ecosystem. The ecosystem provider is expected to define the requirements for a device based on technical, commercial, and regulatory requirements, and the security processes of the provider. These devices are expected to comply with a variety of functional and security requirements, depending on the use cases of the deployment ecosystem. A generic ecosystem is outlined in Figure 1, and the essential platform security elements are described.

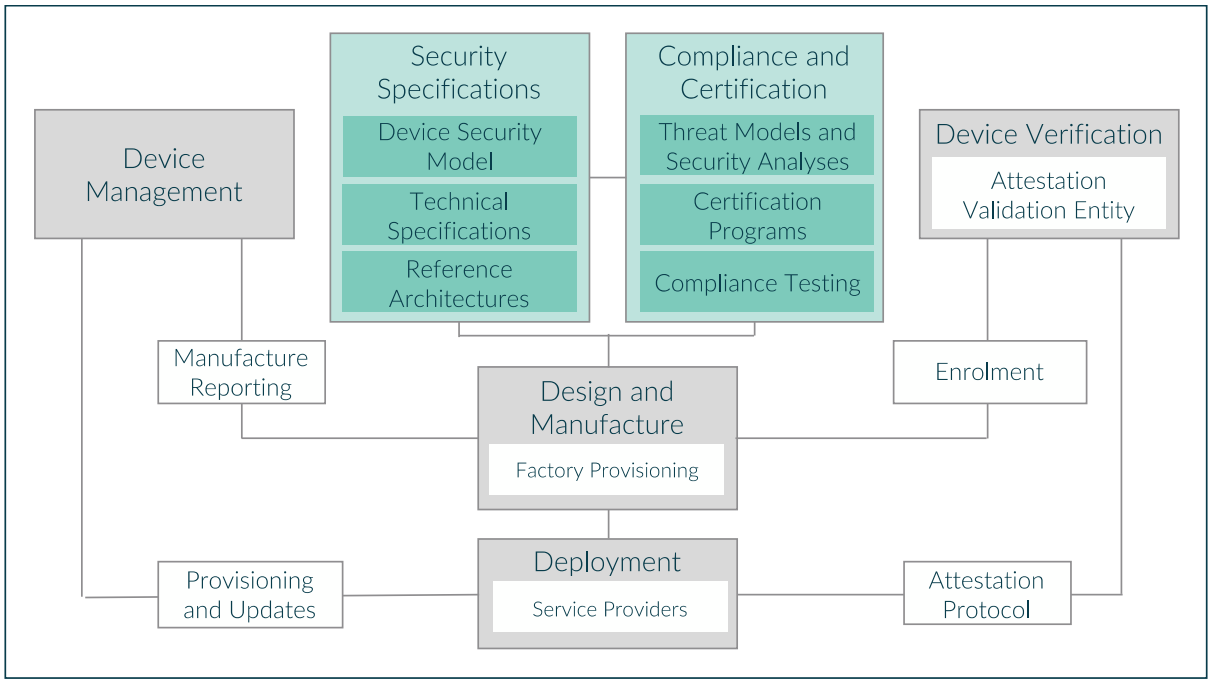

Figure 1: Generic Ecosystem

- Security specifications enable the design and deployment of compliant devices that are compatible with the ecosystem requirements:
    - The Platform Security Model, this document, defines the top-level security concepts, identifies, and defines the Platform Root of Trust and generic Platform Root of Trust services
    - Technical specifications define the security requirements, outline solution architectures, and provide the necessary mitigations identified by threat modelling and security analysis. This includes generic material and references other standards that can be applied to any implementation

    - Reference architectures are any other applicable specifications that define, for example, hardware and software functional and robustness requirements, and standardized functional interfaces
- Certification and compliance show that a deployed device is compliant and compatible with the ecosystem requirements
    - Threat Models and Security Analyses (TMSA) identify use case-specific security threats and motivate use case-specific security functional requirements
    - Compliance testing, for example PSA Functional API Certification, deals with interfaces, functional behavior, and interoperability
    - Certification programs, for example PSA Certified, assess the implementation of the security functional requirements of a compliant device for vulnerability against the identified threats. The robustness that is required is based on use case, cost and security trade-off and the assessment scope. Certification schemes are an assessment and not a guarantee that a device is free from vulnerabilities

Figure 1 also illustrates the following elements that are typical in an ecosystem, but not defined by in this security model:

- Design and manufacture: Secure-by-design devices should be designed and then manufactured based on security specifications with the aim of achieving robustness certification and functional compliance. Device manufacture includes the provisioning of root secrets and other sensitive information. See section 4.
- Deployment: Service providers manage and support deployed devices through:
    - Device management: Device manufacturers provide device manufacture data, provisioning, firmware update services, and other support functions. Device management considers devices throughout their lifetime, from factory provisioning through deployment, any re-deployment, in-field analysis, repair, to end-of-life. Data specific to the device management system must be defined by that system. The storage of such data may make use of the Platform Root of Trust services
    - Device verification: Devices are enrolled with a device verification system, including attestation verification. Depending on ecosystem requirements, device and attestation verification services are expected to be deployed by manufacturers, service providers, or by industry consortia

1.3 Device Models

The device models defined by PSM covers three fundamental parts:

1. The overall trust anchor for the system. This ensures the platform is securely booted, configured, establishes the secure environments required to support the protection of security sensitive operations, and contains the root parameters. This is called the Platform Root of Trust (PRoT).
2. The secure environment for the PSM-defined generic security services, referred to as PRoT services (section 7). These services facilitate the PRoT and are intended to be used by application specific security services and functionality.
3. The secure environment for any Application Root of Trust (ARoT) services. An ARoT service can be any application defined security service, which may make use of the PRoT services. Note that some systems may need nothing more than the PRoT and PRoT services.

There are many ways to build a PSM compliant device. For this reason, the following generalized concepts are used.

- A partition is defined as some processing with a defined logical boundary. Typically, this means software but may also apply to hardware, including hardware controlled by software. Interaction with a partition and its data, both in- and out-bound, must be only via defined interfaces with defined behavior. Abuse of the interface must not be able to compromise the partitions on either side of the boundary. Partitions are managed by a partition management function, see section 2. Partitions may be nested within other partitions, see section 1.3.1.
- A partition manager is used to allocate resources to the partitions that it manages, for example, communication channels, memory, interrupts, peripherals, and processing time. A partition manager will have a defined logical boundary with defined interfaces and interactions (in this sense it is also a partition). Typically, the partition management function comprises software that dynamically manages partitions. In the case where there is only one partition, the partition management function may not exist. Note that there may be some elements of partition management that are static and set when the partition manager is initialized.
- A processing environment hosts partitions, including any partition manager (and the partitions that it manages), and attributes the partitions with specific security properties, though these might be trivial for any processing not considered security sensitive. For example, a processing environment might, through hardware design, have sole access to system resources that enforce isolation. A sub-system with hardware-based countermeasures against physical attacks is another example. Processing environments can be necessary to mitigate specific threats and attacks, or to comply with different certification schemes and assurance levels. Applying the most demanding of the attributes to the entirety of a system is often not practical, and non-security motivations may also apply, for example, real-time characteristics, system architecture considerations, re-use of existing designs, separation of responsibilities and ownership. Processing environments may be established at run-time, typically, but not necessarily, during system boot, or they may be inherent from the design of the system. This can lead to devices with more than one trust anchor if the isolation between them is fixed, for example, in the hardware design.

A minimal PSM compliant device is illustrated in Figure 2. There are two processing environments. One, labelled Secure Processing Environment (SPE), hosts partitions, called secure partitions, for PSM-defined security processing. The other, labelled Non-Secure Processing Environment (NSPE), hosts partitions for all other processing. The SPE shown in Figure 2 hosts the following PSM elements:

- The Platform Root of Trust, which is discussed in section 2, comprises:
    - The Immutable Platform Root of Trust, which is inherently trusted. This largely refers to the hardware, see section 2.1, including any firmware that cannot be updated on a production device.
    - The Updateable Platform Root of Trust, such as firmware and configuration that is trusted by verification through a chain of trust tied to the trust anchor and can be updated on a production device. This largely refers to software, see section 2.2, but may include configurable hardware that can be changed.
    - The Platform Root of Trust Services, which includes the secure storage, cryptographic, and attestation services that are covered in section 7. Typically, these are part of the updateable Platform Root of Trust.
- Any Application Root of Trust (ARoT) service(s), which are not covered in this document. They are expected to use interfaces provided by the PRoT services but must have no direct access to the immutable PRoT. Typically, ARoT services are updateable.

A secure partition must only host the PRoT, the PRoT services, or any ARoT services. All other processing must be completely outside any secure partition and secure processing environment.

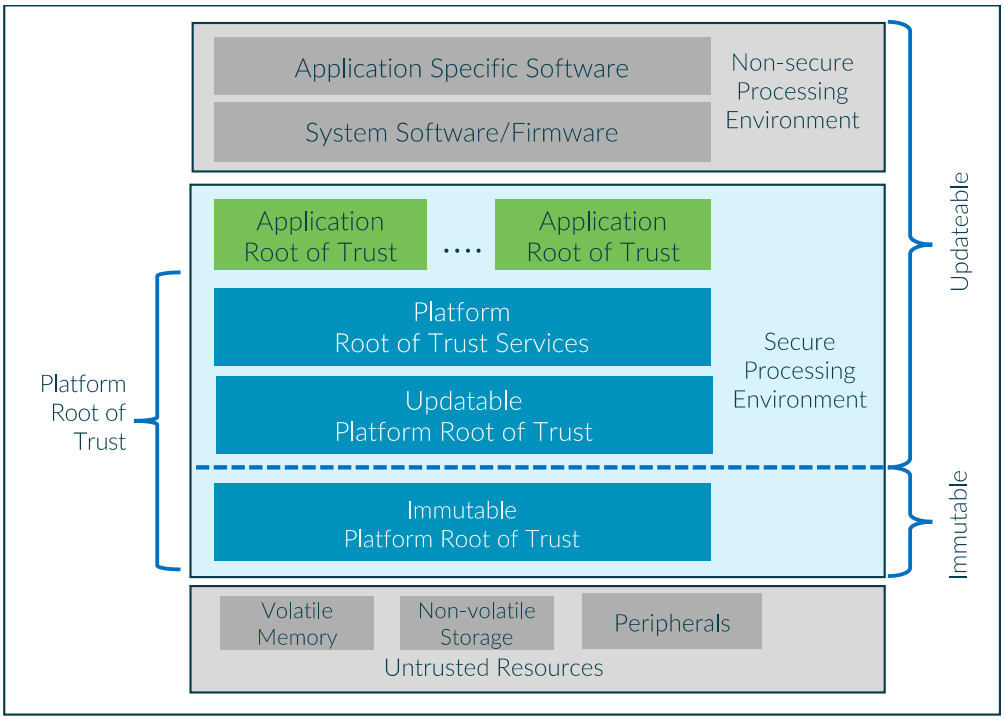

Figure 2: Minimal PSM device

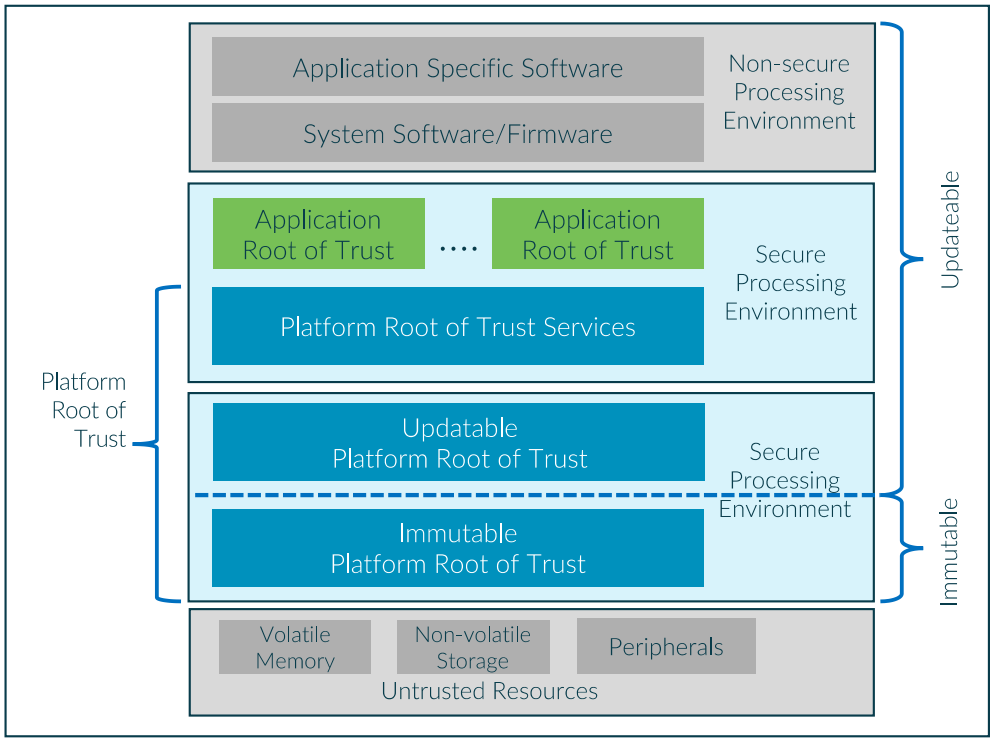

Figure 3: PSM device with two Secure Processing Environments

PSM allows for systems where there is the need for more than one secure processing environment (SPE). For example, Figure 3 illustrates a device with two SPEs, one for the PRoT and the other for the PRoT services and any ARoT services. See also section 1.3.3.

PSM is concerned with the protection of the PRoT, the PRoT services and any ARoT services. Provided the PSM security requirements are not violated, non-PSM security processing environments can co-exist, as shown in the example on Figure 4. As the trust anchor for the device, the PRoT may need to include functionality specific to any non-PSM security service, and the PRoT SPE will need to meet any specific non-PSM processing environment security properties.

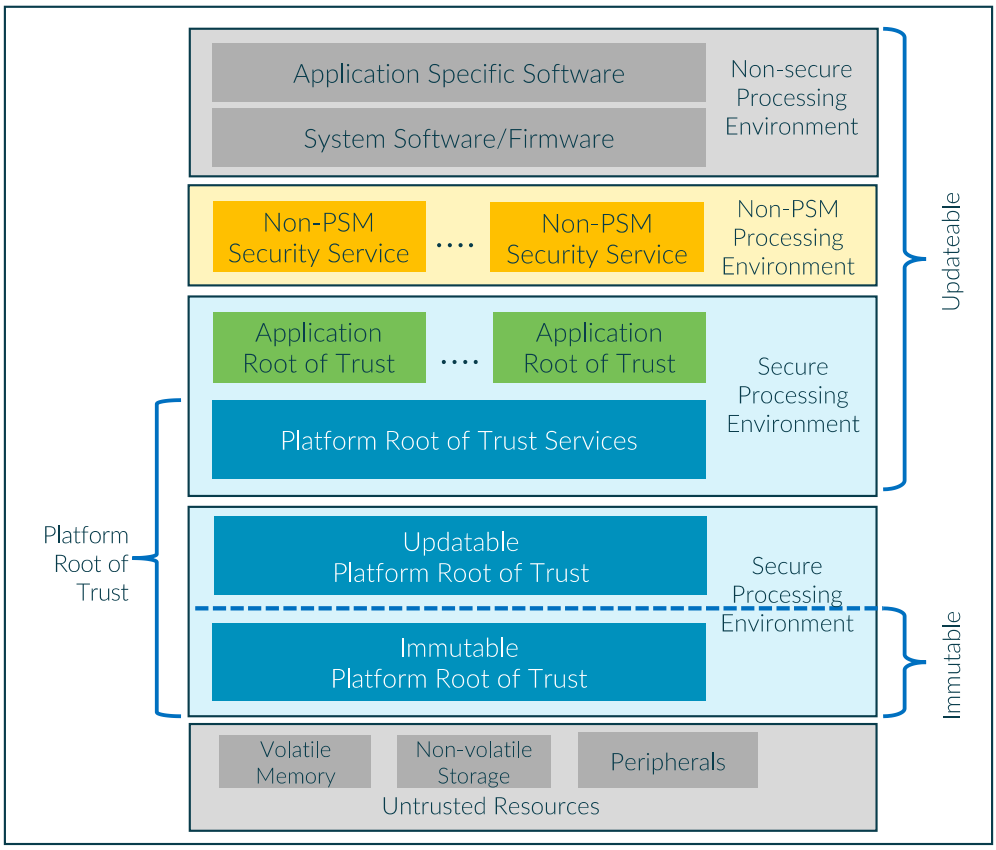

Figure 4: PSM device with non-PSM processing environment

In practice it may be necessary to distribute the PRoT and PRoT services functionality over multiple partitions and, possibly, multiple secure processing environments. Such distribution leads to the need to either restrict access to, or protect, the transactions between the defined interfaces of the distributed partitions.

Access to transactions between partitions should be restricted only to those interacting partitions or agents that are trusted, for example:

- Access can be permitted, for example when on the same die, through static design topology or though secure configuration that is set and protected by the PRoT
- Where access cannot be denied, cryptography can be used to ensure that the transactions are protected in at least confidentiality, but where necessary are protected also in integrity and against replay

It is also necessary to handle any unauthorized substitution of any one of those interacting partitions in a safe and secure way, for example:

- When on the same die, the hardware components are physically inseparable
- Where on more than one die, more generally where physical inseparability cannot be guaranteed, then separation must result in safe and secure failure. For example, using cryptography to ensure that the transactions are protected in integrity, confidentiality, and against replay

The mechanisms deployed to meet the accessibility and inseparability objectives will depend on the security requirements derived from threat modelling, any applicable certification scheme and related protection profiles.

1.3.1 Isolation

Isolation ensures that processing in one partition cannot compromise any code, run-time state including hardware, and secrets, of any other partition either directly or via misuse of software-controlled hardware elements.

PSM requires each secure processing environment (SPE) to be isolated by hardware means from all other processing environments. Figure 5 illustrates SPE isolation, shown by the red border, from the NSPE in the minimal PSM system of Figure 2. Any run-time hardware required to enforce this isolation must be configurable only by the processing environment management function fulfilled by the PRoT, see section 2.2. Non-PSM processing environment isolation requirements, for example, isolation from the SPEs, though very advisable, is not defined by PSM.

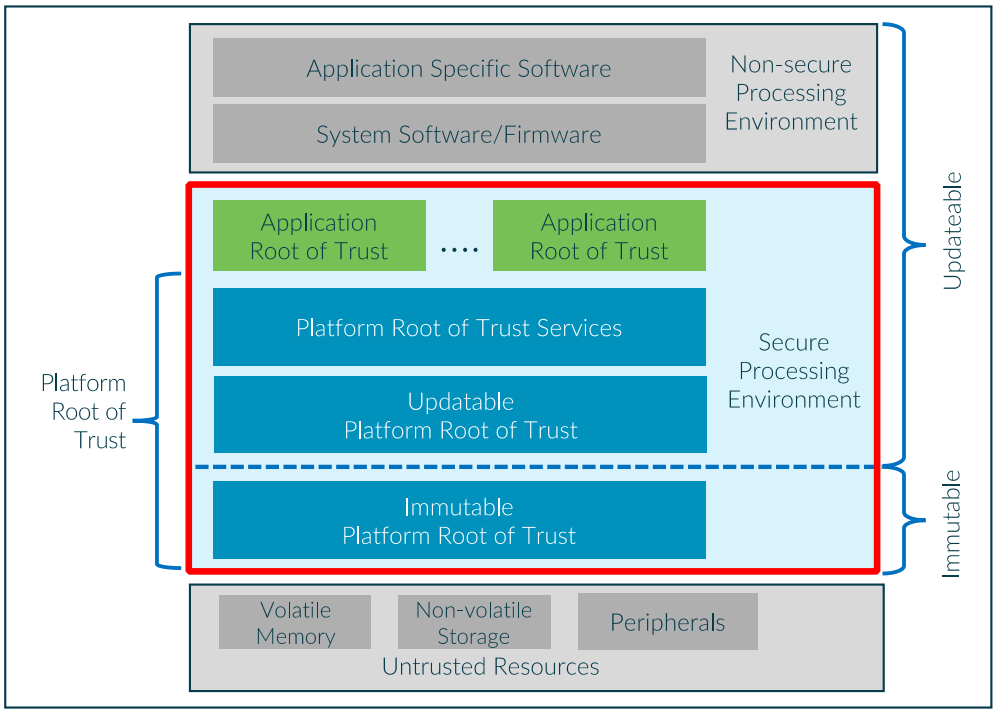

Figure 5: Minimal SPE isolation from NSPE

Isolation between partitions is not defined by PSM but will depend upon the security requirements and any applicable certification scheme. Figure 6 illustrates, with an orange border, the isolation of secure partitions that form the PRoT and PRoT services and the isolation of secure partitions that form the ARoT services. It is recommended that all processing, especially software, is designed assuming that the isolation boundaries are

enforced. This encourages the development of robust solutions that will continue to work when, for example, the isolation is activated during later stages of the development process or where the solution is re-used on devices that do enforce the isolation.

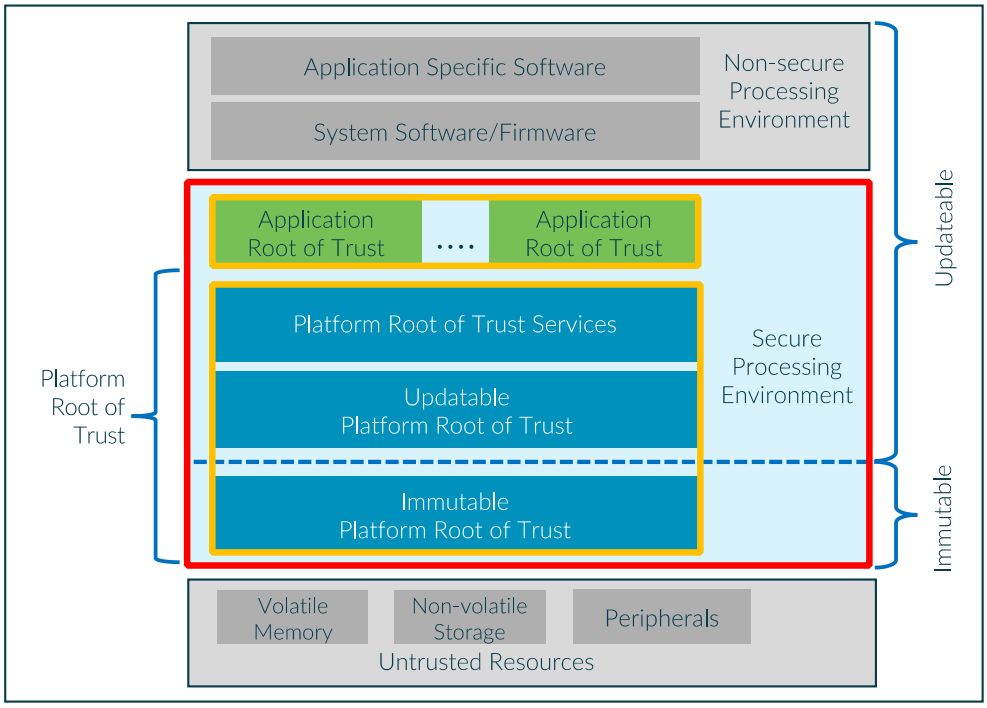

Figure 6: ARoT/PRoT secure partition isolation

Isolation of partitions is handled by a partition management function (see section 2.2). This function may be fulfilled, for example, by an Operating System, which may, itself, be a partition isolated from other virtual machines by a hypervisor that also fulfils a partition management function. In other words, a partition may be nested in another partition. Each outer most partition manager is hosted in a single processing environment. Note that a nested secure partition can only be nested within another secure partition, and hosted in a secure processing environment.

Figure 7 illustrates an example where a processing environment includes a partition management function. One of the managed partitions contains another partition management function that manages two partitions. The orange border illustrates where isolation has been enforced by the applicable partition manager. Multiple processing environments, as in Figure 3, will require a partition management function for every processing environment that supports more than one partition.

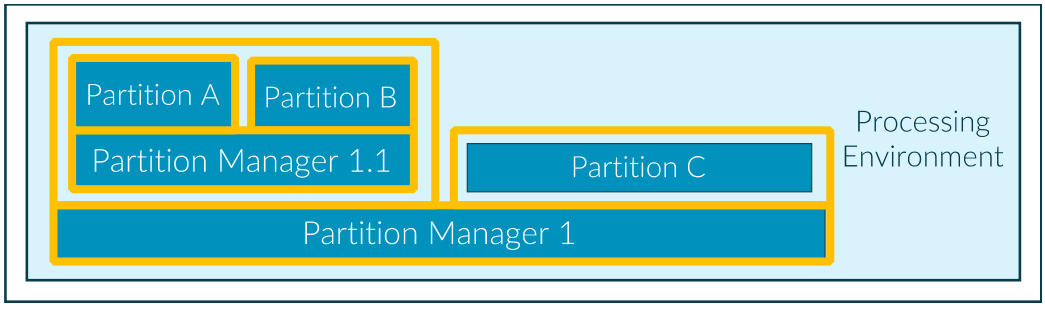

Figure 7: Partitions, nested partitions, and isolation

1.3.2 Immutable Platform Root-of-Trust Protection

Together, the immutable PRoT Boot ROM (see section 2) and the PRoT Root Parameters (see section 3) form the inherently trusted anchor for the system. Any compromise means that the anchor may no longer be trusted. Consequently, the PRoT and PRoT services may no longer be trusted. Any ARoT services may not be trusted. Trust in the entire device may be lost.

To minimize this risk, all access to the root parameters and the Boot ROM should be denied as soon as possible after they have fulfilled their purpose, which typically means at the point where control passes to the updateable PRoT. This is termed Temporal Isolation. All processing before a temporal isolation boundary must complete its task and handover to the stage after the boundary; the only way back requiring a reset leading to a system boot. This should be enforced in hardware.

Figure 8 illustrates temporal isolation in the secure boot flow of section 6.

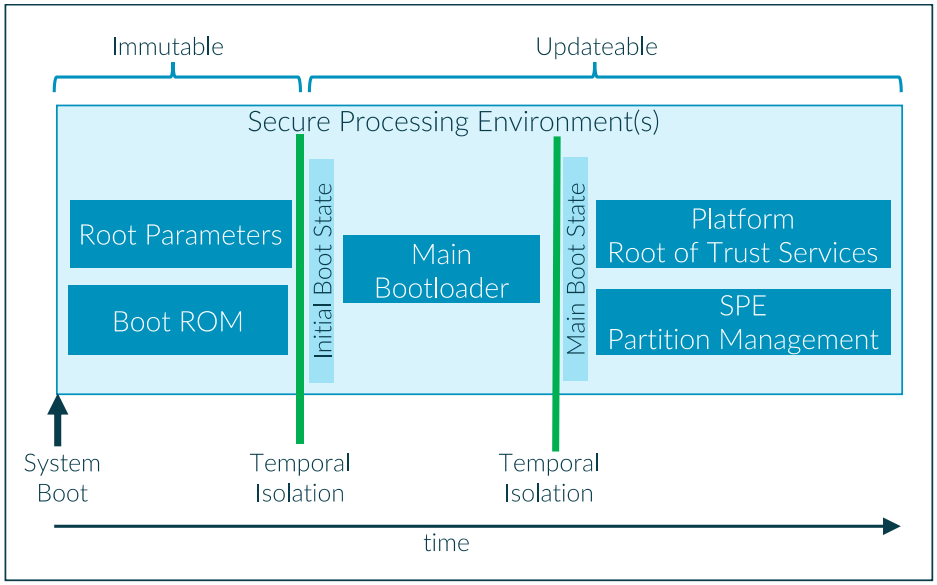

Figure 8: Temporal isolation during secure boot

Only state explicitly provided for use by the processing to be performed after the temporal boundary must be accessible after the boundary. For secure boot, that state will likely be derived from sensitive assets in the root parameters and must be stored on-chip because external storage exposes the state to attack. Note that re-use of code from before a temporal boundary to reduce the overall footprint may be acceptable provided it does not expose any sensitive data that the temporal boundary is there to protect.

The immutable PRoT must also be protected from compromise throughout the lifetime of the product, see section 4.

1.3.3 Trusted Subsystems

A trusted subsystem is any configurable or updateable security functionality, possibly containing a processor capable of code execution, that the PRoT or PRoT services rely on for correct operation. Examples include on-chip security co-processors, and off-chip secure elements, Trusted Processing Modules or Subscriber Identity Modules. The example in Figure 9 illustrates a device with a trusted subsystem and identifies the isolation boundaries as described in section 1.3.1.

Typically, a trusted subsystem is used because of its security properties and, therefore, is a secure processing environment. The result is a PRoT, or PRoT services, that is distributed across multiple secure processing

environments. There must be no reliance on a trusted subsystem until it is established that there has been no unauthorized substitution, see section 1.3.

A trusted subsystem may have its own root of trust and its own security lifecycle. However, all configuration and updates must be performed by the PRoT, and that configuration and state must always be attestable by the PRoT, see section 7.3.

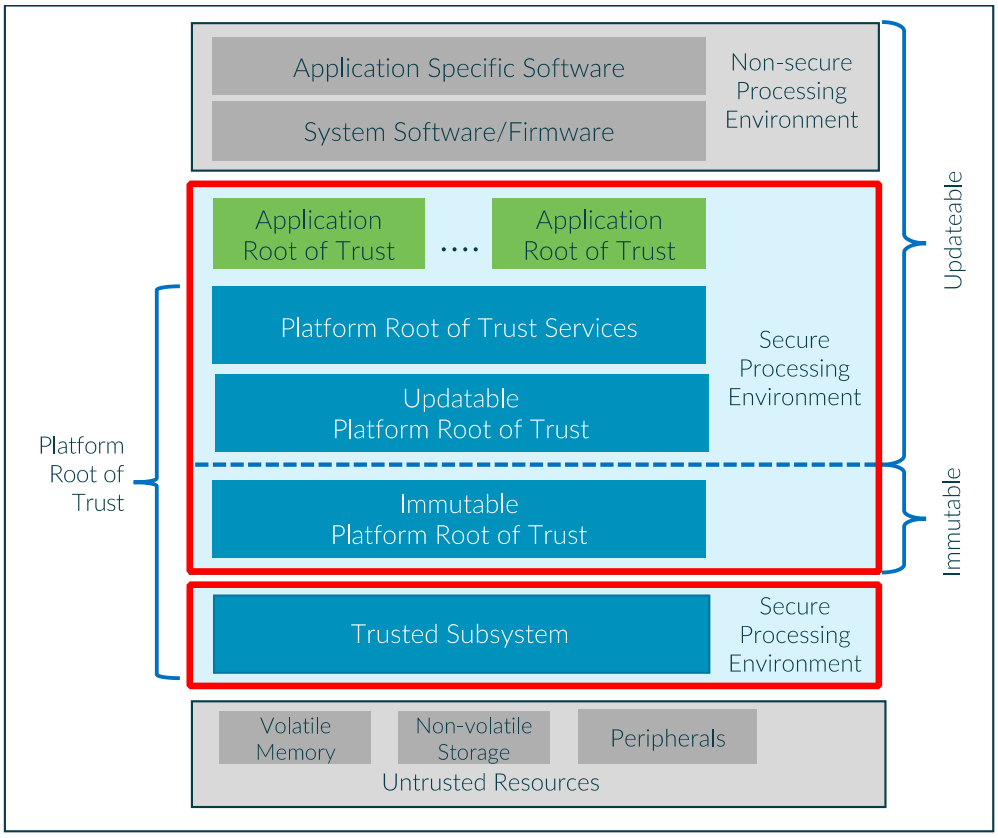

Figure 9: PSM Device with a Trusted Subsystem

2 Platform Root of Trust

The Platform Root of Trust (PRoT) includes the hardware and software components that are responsible for system level security configuration, anchoring the secure boot process to establish a chain of trust for the platform, establishing any partitions for its own functionality and establishing the isolated secure processing environments. The PRoT services provide functionality available to the system beyond the initialization phase, for these, PSA Certified define some interfaces in the form of APIs and ABIs.

The set of functionalities illustrated in Figure 10 is indicative of the minimal PSM device model shown in Figure 2. Figure 10 also illustrates a typical split between the hardware and software[1] parts of an implementation. Mapping of the functions to specific partitions and secure processing environments will be specific to each platform architecture. For example, a system like that shown in Figure 4 will likely have a secure partition management function for each of the SPEs, and possibly partition management functions for the NSPE and any non-PSM processing environments.

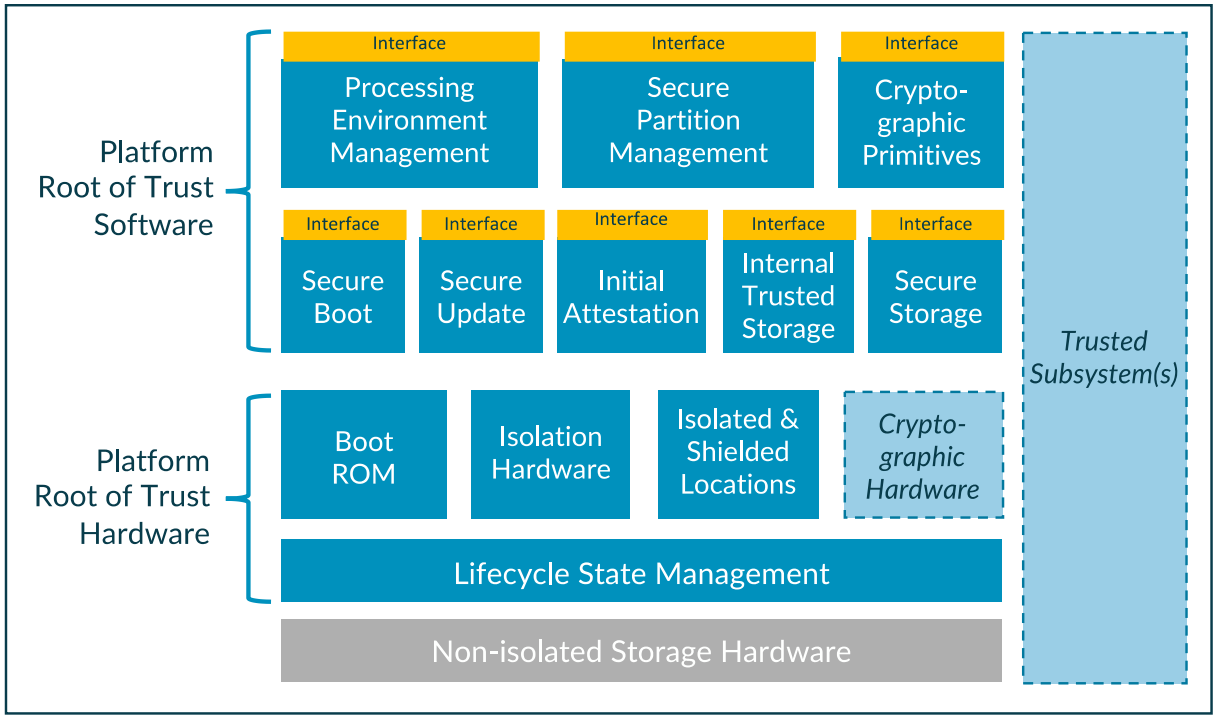

Figure 10: Minimal set of platform security services

2.1 Hardware Elements

The Platform Root of Trust includes the following hardware elements. Access to these hardware elements should be possible only via the PRoT and PRoT services.

- The Boot ROM contains the first code to execute after release from reset and must be on-chip. The Boot ROM code and the configuration that it performs establishes the trust anchor for the chain of trust for the platform and device. The term ROM is used to indicate that it must not be modifiable after manufacture. The Boot ROM is typically implemented using Mask ROM, permanently locked

[1] In this document no distinction is made between software and firmware.

one-time-programmable memory (OTP), or locked on-chip flash, or some combination of these. See section 6.

- Isolated locations contain sensitive data stored in non-volatile memory. There must be some locations on-chip, typically OTP, or on-chip flash (if practical). Access should only be by the PRoT or PRoT services. Some on-chip isolated locations may be used by the Boot ROM during system configuration, and some may, via a hardware path, directly control cryptographic hardware or other security configuration hardware. Off-chip storage hardware can be classed as a trusted subsystem, see section 1.3.3.
- Shielded locations are tamper-resistant isolated locations intended to hold provisioned secrets, including the PRoT root parameters, see section 3. Tamper resistance may include mechanisms to make active probing difficult, for example, physical disassembly for access to internal buses and interfaces, may include countermeasures against side-channel attacks, for example, power and timing analysis, or may include protection against power and clock glitching. However, the extent of tamper-resistance and the choice of data stored will depend on the threats identified during threat analysis that are applicable to the deployment requirements, or to any certification requirements. Off-chip tamper resistant storage hardware can be classed as a trusted subsystem, see section 1.3.3.
- Isolation hardware is the on-chip hardware that is necessary to implement the isolation requirements covered in sections 1.3.1 and 1.3.2. The extent of the isolation depends on any certification scheme, deployment requirements or to mitigate threats identified during threat analysis. Control and configuration of the hardware that isolates the processing environments must be by the PRoT.
- Cryptographic on-chip hardware may be available. On-chip hardware may provide performance benefits and make direct use of keys and configuration stored in on-chip isolated locations or shielded locations. Some countermeasures against side-channel attacks and resistance to fault injection may be possible, however, off-chip cryptographic hardware may provide stronger physical protection should that be needed. See below and section 1.3.3 on trusted sub-systems.
- A trusted subsystem may provide functionality that the PRoT or PRoT services rely on for correct operation. For example, a trusted subsystem may provide cryptographic operations, such as encryption, decryption, and signature generation, while denying all access to the key material stored and used for the operation. See section 1.3.3.
- Lifecycle state management represents any hardware used to indicate or control operations that are state dependent. See section 4.

2.2 Software Elements

The Platform Root of Trust and services typically includes the following software elements, as shown in Figure 10.

- Secure boot extends the chain of trust anchored in the Boot ROM to the system. Secure boot is a mandatory requirement for all PSM secure processing environments. Secure boot of any other processing environments is a common requirement, though will depend on the specific security requirements.
- Secure update enables the immutable PRoT, the PRoT services and any ARoT services to be updated to mitigate any vulnerabilities identified. Secure update cannot apply to the immutable PRoT. Secure update of other processing is a common requirement and is recommended.
- Processing environment management enforces any required isolation between the managed processing environments and enforces any association of system-level resources to those processing environments. All PSM secure processing environments must be isolated from any other processing

environments. This function may also be responsible for allocating and scheduling the use of shared processing resources between the managed secure processing environments.

- Secure partition management enforces any required isolation between the managed secure partitions and controls access to resources for each secure partition, for example, to access to Internal Trusted Storage and operations performed by the Cryptographic service, see below and section 5. This function may also be responsible for allocating and scheduling the use of shared processing resources between the managed secure partitions.
- Internal Trusted Storage (ITS) is a PRoT service that provides partition-based access control to isolated and shielded locations for all PSM secure processing environments. That means that ITS is available to any secure partition that implements PRoT or ARoT service functionality. See section 7.1.
- Secure Storage is a PRoT service that provides secure storage in non-isolated storage for any ARoT service, see sections 2.3 and 7.4.
- Cryptographic operations is a PRoT service that provides partition-based access control to the use of keys and cryptographic processing for all secure partitions that implement PRoT or ARoT functionality. See section 7.2.
- Initial Attestation is a PRoT service that a allows a validation entity, as shown in Figure 1, to validate the trustworthiness of the Platform Root of Trust, its implementation, and updateable components loaded during secure boot. See section 7.3.

Provided that all processing environment isolation requirements are upheld, the PRoT services that support internal trusted storage, the cryptographic operations and initial attestation may be available to non-PSM processing environments, for example, to the NSPE shown in Figure 2, or the Non-PSM Processing Environment shown in Figure 4.

2.3 Non-Isolated Storage

Non-isolated storage, as shown in Figure 10, is storage where access control either logically or physically (if off-chip or removable) cannot be enforced by the PRoT. Non-isolated storage is useful for application-specific data, which may include cryptographically protected data where the required keys are protected by the PRoT service. Section 7.4 outlines how to protect any sensitive data through the use of binding, see section 5 and the cryptographic service, see section 7.2.

3 PRoT Parameters

A device platform requires at least the parameters, or equivalent, listed below and in Table 1.

- Implementation ID; public data that uniquely identifies the implementation. Typically, this identifies the device manufacturer, the device model and version number, and any other data necessary to uniquely identify the device and the Immutable PRoT
- Instance ID: a public value that identifies the specific instance of the device identified by the Implementation ID. This also allows identification of the instance Initial Attestation Key, and, possibly, the Hardware Unique Key.
- Hardware Unique Key: a secret key unique to each device instance that is used to derive other device unique secrets and to bind data to that instance (see section 5)
- Boot Validation Key: the public part of an asymmetric key pair used for authentication during secure boot, however, see section 6
- Initial Attestation Key: a secret private key from an asymmetric[2] key-pair accessible only to the Initial Attestation service, see section 7.3. To prevent cloning or spoofing, this key must be unique to each device instance or to a collection of identical devices. They should not be shared between different versions of a device, between different devices, or between manufacturers.
- Boot Decryption Key: a secret symmetric key used where the boot images authenticated by the Boot ROM are encrypted. Note that the device requirements or any applicable certification scheme may not require the images to be encrypted.

Table 1: Platform parameters

Parameter	Initial param	Updateable param	Security class	Recommended Storage Type
Implementation ID	Yes	Yes	Public	Isolated Location
Instance ID	No	Yes	Public	Isolated Location or derived
Hardware Unique Key	Yes	No	Secret	Shielded Location
Boot Validation Key	Yes	No	see section 4	Isolated Location or Boot ROM
Initial Attestation Key	Yes	Yes	Secret	Shielded Location or derived
Boot Decryption Key	Yes	No	see section 4	Isolated Location or Boot ROM

As a consequence of temporal isolation boundaries for secure boot, see sections 6 and 6.4, the parameters are categorized as follows, based on whether each is secret or public. Secret means that the data that must not be accessible to unauthorized agents, and public means that the data may be shared inside and outside the device.

- Initial parameters are those used by the immutable PRoT. Depending on the certification profile, initial parameters are stored in isolated locations or shielded locations. They should only be directly

[2] The ecosystem may permit symmetric cryptography if the device cannot support asymmetric operations.

accessible or useable by the immutable PRoT, see section 6, but can be copied to, or used as seeds for derivation of Initial Boot State data, see section 6.4.1.

- Updateable parameters are those that are used by the PRoT services. They should only be directly accessible by the updateable PRoT code and can be copied to or derived and stored in the Main Boot State, as required, see section 6.4.2.

Direct use of a Shielded Location or Isolated Location, or using derivation, is implementation specific. In general, derivation can be more flexible and future proof. Derivation can support additional strategies for protecting secret parameters and may reduce the risk of exposure of secret parameters during device manufacture. However, derivation may require additional computational resources at boot.

4 PRoT Security Lifecycle

A lifecycle defines the states of an object through its lifetime. Each security state in the security lifecycle of a device defines the security properties in that state. Security state can depend on:

- Software versions
- Run-time status such as data measurements, hardware configuration, and status of debug ports
- Product lifecycle phase, for example, development, deployment, returned to manufacturer, or end-of-life

This section is concerned with how the Platform Root-of-Trust and its data and secrets may be compromised when debug is enabled after the platform has been secured, and how to ensure valid attestation only in attestable states. The generic security lifecycle shown in Figure 11 is intended to capture only the minimum set of notional lifecycle states and transitions. The described characteristics of each state will need to be related to the actual states that arise in an implementation and deployment.

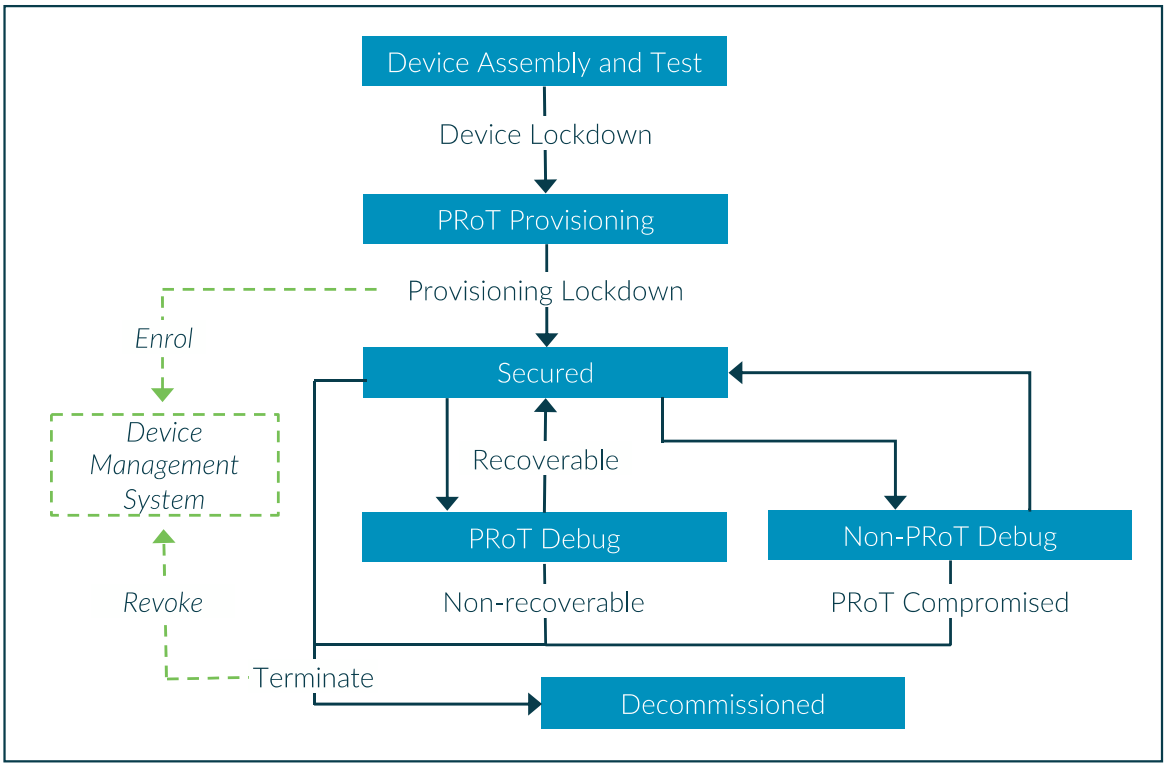

Figure 11 : Generic Platform Root of Trust security lifecycle

- Device assembly and test: The device will be running manufacture and diagnostic software. This is not a secure state and, therefore, is not an attestable state
    - Hardware debug interfaces may be open, secure boot may not be enabled
    - If secure boot is already enabled, manufacture and diagnostic software should not be signed by the same authority as the final production software. There must be the ability to revoke any manufacture and diagnostic software certificates.
    - Test secrets and identities may be present. Production platform security parameters must not be present

- PRoT Provisioning: The device is programmed with production platform security parameters, see Table 1[3], and is configured so it can enter the Secured state. This is not a secure state and, therefore, not an attestable state
    - Production platform security parameters are provisioned and locked to prevent modification. Random values that are generated on the device should use a random source with sufficient entropy. Where parameters are provisioned separately, for example on a secure element, then it should not be possible for any device software to access or perform any operations with them until the device has reached this state
    - Hardware debug ports should be disabled, or an access control mechanism activated
    - Secure boot should be enabled
    - Only signed production software should be present. Manufacture and diagnostic software must not be present after any use in the PRoT provisioning process.
    - Once the device has been provisioned it may be enrolled with a device management system. This will declare the existence of the device to the ecosystem
- Secured: The PRoT is fully operational and secured. This is the operational security state for most of the life of the device. Only in this state do the platform security parameters become available to any PRoT software. Provisioning of any application-specific data that is to be protected using PRoT services can now be performed. This is expected to require manufacture reporting for tracking and identification of fully secured devices. This is a secure and attestable state
- Debug: Figure 11 shows non-PRoT debug and PRoT debug states. If entering either debug state is supported once a device is in the Secured state, then a system reset is required. This is because the device includes production secrets and the act of reset requires the removal of, or denial of access to, any previous data and derivation of Secured state sensitive data. Only if the device can still be considered trustworthy after debug can it be returned to the Secured state; this is called recoverable debug. Otherwise, debug is considered as non-recoverable, see section 4.1
- Decommissioned: Entering this state requires a system reset. This is because the device includes production secrets and the act of reset requires the removal of, or permanent denial of access to, any previous data and derivation of Secured state sensitive data. This is not a secure state and so not an attestable state
    - Production platform security parameters secrets should be permanently inaccessible, denying attestation and access to any bound data. The overall product lifecycle may require non-secret platform identities to remain accessible
    - It is not possible for the device to leave the Decommissioned state unless a full factory reset is possible, and the device becomes indistinguishable from a new device. This means that the device can be securely re-provisioned with a new set of platform security parameters stored in Isolated Locations or Shielded Locations, or in the Boot ROM, effectively resulting in a new device instance
    - Revocation of the device by the device management system can be used to ensure that the device is no longer operational. This is necessary even if the device can be factory reset

Additional states and sub-states may exist in any real device. Any sub-state must retain the properties of its parent. Both parent and sub-states must always be attestable once a device enters an attestable state. The security state is included in the PRoT attestation claims, see section 7.3. Therefore, it is not permitted for the security state to change without a device reset if either the current state or the new state is attestable.

[3] This may include the Boot ROM firmware if not a design-time mask ROM.

4.1 Debug

It may be necessary to debug a device that has reached the Secured state in its lifecycle, see Figure 11. Debug during product development is likely to be on non-secure parts or with secure parts provisioned with development credentials.

When the device is in Secured state, logging of events or reporting of diagnostics in a way that does not expose sensitive data and does not affect the trustworthy operation of the device is permitted. The ecosystem may require that such logs and reports are only accessible by a device management system, and are integrity and confidentiality protected. This is called non-intrusive debug.

Debug that is more intrusive than event logging depends on the access level that is granted to the debugging agent. For example, it is possible that device-sensitive data can be compromised or that secure boot may be circumvented, meaning that the device is no longer trustworthy.

This PSM is concerned with protection of the PRoT and PRoT services when the device is subject to intrusive debug. Table 2 identifies two notional debug states. One, called PRoT Debug, covers explicit debug of the PRoT and PRoT Services. The other, called Non-PRoT Debug, covers the security requirements for the PRoT, PRoT services and any ARoT services that may arise when debugging any other part of the device. An implementation where debug in a Non-PRoT debug state cannot guarantee that the PRoT and PRoT services are free from compromise is indistinguishable from the PRoT Debug. Note that the actual debug states and options that are available to the manufacturer will depend on the specific hardware implementation on the device.

The act of entering a debug mode should be by a secure mechanism, typically some form of authentication, even if protection of the sensitive assets is not implemented.

Table 2: Intrusive debug

Debug State	Secure and Attestable	Security Implications/Requirements
PRoT Debug	No	Non-recoverable Compromises PRoT operation and platform security parameters, any PRoT services and any ARoT services.
		Recoverable May compromise the PRoT operation while in this state but cannot compromise the platform security parameters.
Non-PRoT Debug	Yes	PRoT, PRoT services and any ARoT services remain intact and trustworthy.
		Compromises only ARoT services and data. PRoT operation and platform security parameters, and PRoT services remain intact and trustworthy.

Intrusive forms of debug should be restricted to authorized debug agents and require a secure authorization process, with the debug rights granted by an appropriate device management service. This process is beyond the scope of this document. However, other platform security specifications provide guidance. Non-PRoT debug is beyond the scope of this document.

Debug of the PRoT is the most intrusive. When the PRoT debug state is entered it must not be possible to issue an attestation token that is signed using the Initial Attestation Key. This is because the device is no longer in a trustworthy state. Also, it must not be possible to derive production binding keys in order to prevent access to securely stored data, see section 5.

Recoverable PRoT debug applies to devices that can deny access to production platform security secret parameters when in PRoT debug state and can be restored to a fully trustworthy and attestable state on exit, provided the debug activity has not compromised the Secured state.

Non-recoverable PRoT debug applies to devices that are not able to protect the platform security parameters and are not able to ensure that the debug has not compromised the Secured state. This type of device must make the platform security parameters permanently inaccessible on entry to debug. The only valid next state is Decommissioned.

4.2 Lifecycle of other components

Other components in a system, for example, trusted subsystems, the system software, and application software, may have their own lifecycles The overall product built from the components may also have its own lifecycle. Such lifecycles, and any associated data, are implementation or application specific and out of scope of this document. Note that such lifecycles must never be in a state that conflicts with or compromises the security requirements of the Platform Root of Trust security lifecycle.

5 PRoT Binding

The following types of binding are applicable when sensitive data owned by secure partitions are stored in Non-isolated Storage, see section 2.3. Binding may also be used to protect Internal Trusted Storage.

- Device binding binds the data to the owning device instance. No other device instances can directly access the data.
- Partition binding binds the data to the owning secure partition. No other partition in any processing environment can directly access the data.
- Lifecycle State Binding binds the data to specific security lifecycle state, see section 4.

Binding relies on secret keys and cryptography. The generation and storage of the required keys depends on the lifecycle security state, which includes the supported debug modes. It is recommended to use run-time-derived keys to support lifecycle binding, see section 4. Derivation is essential if recoverable debug is supported because the return to the secured state requires the generation of the secured state keys.

Section 5.2 outlines a run-time key derivation scheme to support partition binding.

5.1 Device-binding Root Keys

Two binding root keys are defined. Both are security lifecycle-state dependent, and so should be derived after a system reset. Derivation for device binding is essential if recoverable debug is supported.

- Secure BRK (SBRK): The same key can only be derived when the device is in a secure security lifecycle state, see section 4. This ensures that the data that is bound to this key cannot be accessed when the device is in any debug mode.
- Non-PRoT Debug BRK (DBRK): The same key can only be derived when the device is in a secure security lifecycle state, or in any debug mode that does not compromise the PRoT, see section 4). This key should only be used when it is acceptable for any data that is bound to it can be accessed when the device is in a debug mode.

This means that any PRoT data that is cryptographically secured should be bound only to SBRK.

Derivation of the binding root keys must be through a cryptographic one-way derivation from a hardware unique secret key, for example, the Hardware Unique Key, see Table 1. Derivation ensures that any derived BRK is unique to the device. The derivation must result in SBRK and DBRK being different.

The derived SBRK or DBRK should be used only to derive a Partition Specific Binding Key (PSBK), see section 5.2.

5.2 Secure Partition-specific Binding Keys

Partition Specific Binding Keys (PBSKs) are required where binding to a secure partition is required. The derived PSBK will be unique to that secure partition.

Each PSBK is a cryptographic one-way derivation either from one of the binding root keys, see section 5.1, or from an existing derived PSBK that is owned by that secure partition. This means that all PSBKs are ultimately derived from either SBRK or DBRK. When a PSBK is derived, the following data are used.

- Explicit parameters provided by the calling secure partition:
    - Seed: Allows the derivation of different PSBKs for different secure partition specific use cases
    - Use Policy for the derived PSBK: One and only one of:
        - Encryption/decryption

        - Signing/verification
        - PSBK derivation: The derived PSBK can only be used to derive other PSBKs
    - Debug Policy, with reference to section 5.1, one of:
        - Secure: The PSBK derivation must be anchored at SBRK
        - Non-PRoT Debug: The PSBK must be anchored at DBRK
- The identity of the calling secure partition, which is called the Partition ID, must be an implicit parameter that is provided on behalf of the calling secure partition. This binds the derived PSBK to that secure partition and guarantees the uniqueness of the derived PBSK

The Use Policy does not include export in the clear of the derived PSBK. Thus, a PSBK cannot be used where shared knowledge of the key is required.

6 PRoT Secure Boot and Firmware Update

All devices must support a secure boot flow to ensure only authorized software can be executed on the platform. Secure boot, sometimes called verified boot, uses cryptography to verify the next stage code and any metadata. Execution of the next stage proceeds only if any validation checks on the verified metadata pass, for example, version comparison, see section 6.3.

Secure boot is required for the firmware and software in the SPE. It should also apply to the first NSPE image loaded. Note that in some contexts, the term Measured Boot is used, however, this involves measuring the code, typically for attestation, but without any verification and validity checks.

The secure boot flow must start with an inherently trusted Boot ROM in the Immutable PRoT; this is the trust anchor for the boot validation chain. Both the following are recommended, as shown in Figure 12:

- The Boot ROM is small, simple, and verifiable. This minimizes the risk of a vulnerability that cannot be corrected once on the chip, and
- The complex steps are handled by a main bootloader, which is subject to validity check by the Boot ROM, because it can be corrected through a secure firmware update process, see section 6.5

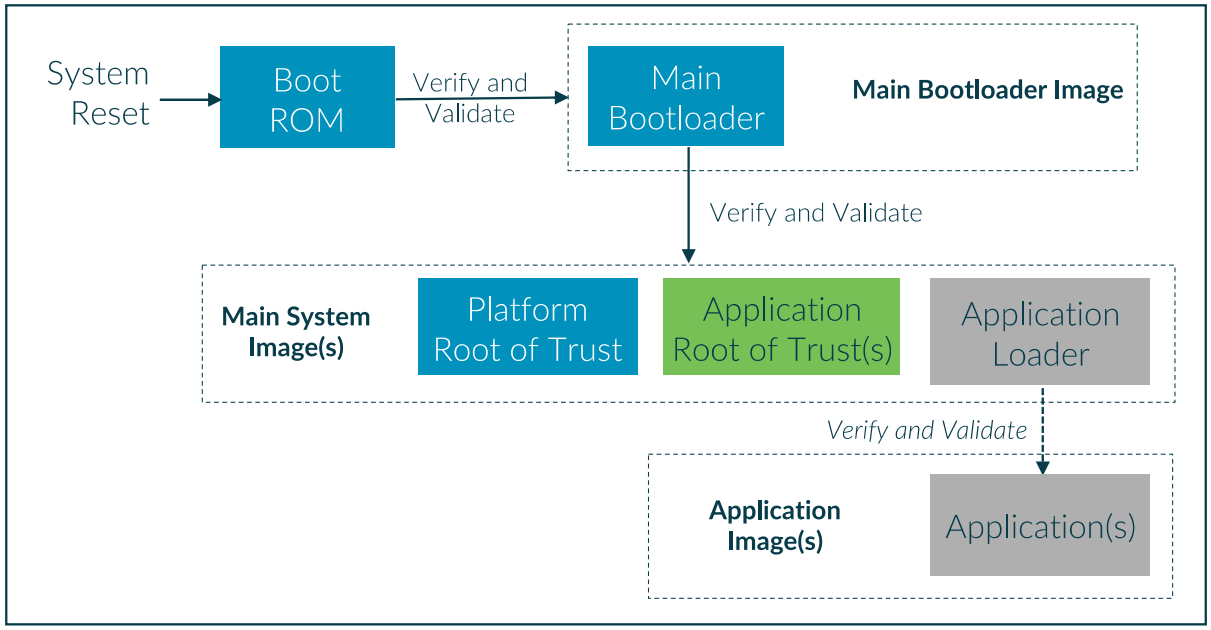

Figure 12: Example Secure Boot Flow

The example in Figure 12 shows the Main Bootloader verifying and validating the Platform and Application Root of Trust codes. Figure 12 also shows that the NSPE First Stage Loader is also verified and validated, and that it verifies and validates each of the NSPE images. Actual implementations may include more boot stages, or support multiple images to allow for incremental updates, or to support supply chains with different signing entities. Any such variation must meet the following security properties to establish a secure boot chain:

1. A device must always boot from a fixed address in the Boot ROM following system reset. This is because it acts as the trust anchor for the secure boot chain. The term system reset is used to describe a complete system reset, including any trusted subsystems, see section 1.3.3. No run-time state from before the reset should be retained or used, except where necessary if suspend or hibernate are supported, see section 1.3.3.

2. The Boot ROM verifies and validates all images that are associated with the next stage before executing any next stage code. The immutable Boot Validation Key (BVK), see Table 1, is used. The BVK is the public part of an asymmetric key pair. Despite being a public key, it is good practice to consider a BVK as secret. The Boot ROM may use a stored hash value for the second and subsequent verification of the same image, see section 6.1.
3. The next stage verifies and validates all data and images that are associated with the following stage before executing any of the following stage code. This process is repeated until all the chained images have been verified and validated. The keys required and any management of anti-rollback, see section 6.3, are not defined by the Platform Root of Trust. However, following the principles that are outlined in this document is recommended.

6.1 Image Signing, Validation and Encryption

All images should be signed using asymmetric cryptography and must be verified before use. Each signature must cover at least the image content, for example code and data, any critical parameters, for example, entry point for the code, and the version number, see section 6.2.

Asymmetric cryptography during boot can be too time-consuming for some applications. Securely storing the computed cryptographic hash value of the images verified during an initial secure boot asymmetric check allows the stored value to be used to reduce the time on subsequent checks. Using a saved hash value in this way is called Hash Locking. For the hash to be secure, the stored value must be accessible only by the Boot ROM. To prevent the execution of a different image, the stored hash value should be integrity protected. To prevent the use of an old image, the integrity-protected stored hash should also be replay protected. To support update, the stored hash value must be updateable.

The private counterpart of the on-device Boot Validation Key, see Table 1, must be held securely by the operator or device manufacturer, and should never be stored on the device. This applies also to the private counterparts of any other on-device keys that are used in the validation of the boot chain.

Symmetric signing is not recommended. This is because any disclosure of the key from a compromised device allows any image to be signed. If symmetric signing is unavoidable then the key must be unique per device, thus compromising one device does not allow images for another to be signed.

Boot images should be encrypted if they contain sensitive data or code. This requires a secret Boot Decryption Key that should be available only to the Immutable PRoT. That key may be stored in an Isolated or Shielded Location or derived from another Isolated or Shielded Location. It is normal for the image to be signed then encrypted, thus the device decrypts the image before verification and validation. However, if the content is considered confidential and should not be visible to the signing entity then encryption followed by signing is necessary.

6.2 Verified and Version Validated Components

Regarding the example in Figure 12, the verified components column of Table 3 shows the minimal set of verified components. If a component comprises multiple sub-images, each image should be verified separately. For the version checked components columns, see section 6.3.

Table 3: Verification and version check requirements

Component	Verified components		Version checked components	
	Secure boot verification	Hash recorded for attestation	Anti-rollback check	Version recorded for attestation
Boot ROM	N/A	N/A	N/A	Yes
Main Bootloader	Yes	Yes	Yes	Yes
Platform Root of Trust				
Trusted Subsystems				
Application Root of Trust				
Application Loader	Yes	Yes	Yes	Yes

6.3 Anti-rollback

Anti-rollback is used to reject earlier versions of the firmware, software or data that may contain known errors or vulnerabilities. Secure boot must only allow components that have the same or newer (typically higher) version number than the reference version number for that component to be executed. To ensure that the component version number is valid, it must be included in the signature of the component and verified before use. The verified version number of each software component must be compared against a reference version number as part of a secure boot process. To ensure the integrity of the reference version number, it is, typically, stored in an updateable Isolated Location.

The Boot ROM must include an anti-rollback check on all images that it verifies on devices in the Secured lifecycle state, see section 4. Policies for updating the reference number used by the Boot ROM are covered in section 6.3.1 and section 6.3.2. Subsequent stages in the secure boot chain should include an anti-rollback check on the images that are validated by that stage. The principles that are discussed in section 6.3.1 and section 6.3.2 can be applied.

Regarding the example in Figure 12, the minimal set of version-checked components is given in the Version Checked Components columns of Table 3.

For the purposes of attestation, see section 7.3, the version of the Boot ROM must also be recorded. If components are not individually versioned, they should be recorded with the version of the overall image.

6.3.1 Anti-rollback Reference Version Update

The reference version number must be updated only within a secure boot process. Operational requirements will determine when this is done. For the version checks that are performed by the Boot ROM, the following methods are recognized:

- Automatic update on reset: The Boot ROM updates the anti-rollback reference version when it has successfully loaded a newer version[4]. Success means that the image has passed all secure boot checks.

[4] Where the Boot ROM can differentiate an intentional reset from a failure-induced reset, it may choose not to update the reference counter in the case of a failure reset and so fall back to the last known good version.

- Update on command: The anti-rollback reference version is updated in response to a secure message from an external device management service. This means that the existing version is revoked only after the device management service signals that the newer version has no identified faults.

The first option does not support a legitimate rollback if the new image boots correctly, but some aspect of its function is later found to be broken and the service provider decides that it is necessary to rollback to an earlier version.

The second option means that the service provider decides when to revoke the previous version after rolling out a newer version to the device. This option may also signal which version, that is newer than the current anti-rollback reference counter, to load if more than one version is available. A secure messaging service is required to ensure that the messages are from a trusted command issuer and are replay protected. This ability to legitimately rollback leaves devices susceptible to illegitimate rollback for longer. This ability also makes devices susceptible to denial-of-service attacks that block the update message.

To ensure that a device remains bootable, the device should never set the reference version counter to a value that is higher than the value of the newest image that is available.

6.3.2 Anti-rollback Reference Number Reset

Where the chip uses a technology that allows the reference version number to be reset, the device can support a factory reset mode or can be signaled from a device management service to reset the reference number. This requires a secure messaging service[5] to ensure that messages are from a trusted command issuer and are replay protected.

6.4 Boot State

Boot state refers to the data intentionally left at a temporal isolation boundary (section 1.3.2) in the secure boot flow. There are two isolation boundaries defined, as shown in Figure 12, leading to an Initial Boot State and a Main Boot State.

6.4.1 Initial Boot State

The initial boot state is the set of data that is provided by the Boot ROM for use by the Main Bootloader. Initial boot state includes the following:

- A random boot seed that is generated on each system reset. This can be used by later services, for example, to allow an attestation validation entity to ensure that attestation reports for different Attestation End Points, see section 7.3, were generated in the same boot session
- For each component that is validated by the Boot ROM, see Table 3, at least the following component information must be captured:
    - Version
    - Signer Identity[6]
    - The measurement that is made by the Boot ROM

[5] It must not be easier to subvert the messaging protocol than to subvert the secure boot. This means that authentication of the command issuer should have at least the same cryptographic properties as the ones that are used for image signing.
[6] The Signer Identity, if present, would typically be in image metadata.

- Any Initial Parameters, see Table 1, that are required by later stages must be copied from Isolated Locations or Shielded locations, or be derived as appropriate. This is to comply with the parameter visibility rules. See section 3.

Initial boot state must be only directly accessible to the Main Bootloader, and should not be modified once set by the Boot ROM.

6.4.2 Main Boot State

The main boot state is the set of data provided by the Main Bootloader for use by the Platform Root of Trust Services. The main boot state includes the following:

- The initial boot state data, see section 6.4.1
- All platform security parameters, see Table 1, must be copied to the main boot state, to comply with the parameter visibility rules. see section 3
- For each component that is validated by the Main Bootloader, see Table 3, at least the following component information must be captured: version and signer ID[6] and the measurement that is made by the Main Bootloader code

Main Boot State must be only directly accessible to the Platform Root of Trust services, and to ensure that the data are trusted, should not be modified once set by the Main Bootloader.

6.5 Firmware Update

Update of firmware is crucial for fixing security vulnerabilities and enhancing the features of devices that are already deployed. It is essential that the update mechanism cannot be used to compromise the device with unauthorized software.

The selection and download of updates depend on the specific ecosystem and is not defined here. Ideally:

- Downloads are over a secure connection from an authorized repository. This helps to prevent networked attackers from sending arbitrary images, though this may make it impractical for future operators to use a different repository. The download may be encrypted
- Downloads are authenticated and integrity checked at the time of download, though this may be impractical in some constrained systems. The checks should follow the concepts that are covered in the other parts of section 6

Where resources permit, a banking scheme is recommended. This ensures that a bootable image is always available until that image is revoked through increment of the anti-rollback reference counter, see section 6.3.1. This requires enough non-volatile memory to store at least two copies of the images and any dependencies.

In all cases, a new image can only be executed after it has been validated by the secure boot process as covered in this section 6.

6.6 System Suspend and Hibernation

Suspension and hibernation are low power modes that allow a device to resume from a known point more rapidly than a full system start (a cold boot). Support for either is optional.

Suspension and hibernation both require the saving of enough state before going into the low power mode to allow resumption when the power is re-applied. Suspension typically means that some of that state is held in an always-on-power domain on the chip, and any dynamic RAM is placed in self-refresh mode. Hibernation typically means that all the state is written to some persistent storage before the power is removed. Some of this storage must be on-chip to deny substitution and ensure integrity and freshness.

The state that is necessary for resumption may be accessible when the device is suspended or hibernated. This introduces the risk that it can be modified, breaching the principle that a system that has been suspended or hibernated should be indistinguishable from one that has not. Therefore, that state must be subject to integrity and replay checks on resumption. It may also need to be subject to privacy protection too, for example, secrets may need to be erased or encrypted.

In general, deciding whether to suspend, hibernate, or shut down can be performed by application code. For suspension or hibernation, the creation and signing of the required resume state, and the process of entering the low power mode, must be performed by trusted code. On resume, validation of the saved resume state must be performed by trusted code anchored in the Boot ROM.

If the integrity of any of the saved resume state is invalid, or replay is detected, then the Boot ROM must proceed with a full system restart.

7 PRoT Services

This section covers the Platform Root of Trust Services that are made available to any secure partitions, see section 2 and Figure 10. They may also be made available to other partitions provided that the isolation properties are met. PSM does not require the use of any specific API. However, APIs are available and are referenced in this section as examples, and PSA Certified has an API functional compliance programme.

7.1 Internal Trusted Storage

Internal trusted storage provides secure partition access to data stored in isolated locations and shielded locations, see sections 2.1 and 2.2. Individual data objects have an owning secure partition. Only the owning secure partition can access or modification its data.

Access to sensitive data may depend on the security lifecycle state, see section 4. For example, it may be necessary to deny access to debugging agents when in any debug state. Access for other lifecycle states is application dependent.

Implementation may rely on the physical access properties of isolated locations or shielded locations, or by binding based on the security lifecycle state of the device, see section 5.

Internal Trusted Storage is supported by the PSA Storage API.

7.2 Cryptographic Operations

The cryptographic operations service performs cryptographic processing using input, derived, or persistent keys that are stored in isolated locations or shielded locations. Implementations should meet the following essential security properties:

1. Isolation: Keys and other sensitive data is isolated within the cryptographic service, so that this type of material is not exposed to less trusted software[7]
2. Access control: Keys and other sensitive data must have an owning secure partition and can be used only by that partition. However, see the next point: Policy
3. Policy: This property restricts how individual keys and secrets that are owned by a specific secure partition can be used by that partition, preventing misuse

The minimum set of cryptographic policy options is as follows:

- Usage, including allowing or denying the use of specific algorithms or modes to a specific key for encryption, decryption, derivation, signing and verification
- Export: none, clear, wrapped or delegate. Delegation allows a secure partition to make a key that it owns available, with specified usage and export policies, to another specified secure partition or any non-PSM partition, for example, the Non-secure Processing Environment in Figure 2. The usage and export policies of a delegate key may be more restrictive than the source key, but the policies cannot be relaxed

Cryptographic algorithm support, modes and key sizes will depend on the use cases, any certification profile, or applicable regional or application. It is recommended to use algorithms with at least 128-bits of security.

[7] This does not prevent the use of hardware solutions where the cryptographic service can only request the hardware to use a key.

A source of random data is required, typically for establishing a secure communication channel. A True Random Number Generator (TRNG) or a suitably seeded Deterministic Random Bit Generator (DRBG) should be used to provide such random data.

The PSA Crypto API provides an interface to the cryptographic operations. Meeting the properties list above is dependent upon the implementation.

7.3 Initial Attestation Service

Evidence provided in an attestation report makes a statement about the state of the entity being attested and is the Attestation End Point (AEP) in Figure 13. An attestation report is intended to be checked by an Attestation Validation Entity (VE). Deployment of attestation services is eco-system dependent. The scope of the AEP and how the attested AEP report is used are application specific and not defined by PSM.

The Initial Attestation Service (IAS) provides a signed Initial Attestation Token (IAT). The IAT includes the state of the Platform Root-of-Trust, including whether a debug state has been entered, and any claims made by AEP. The IAT is bound to the specific device instance through use of the Initial Attestation Key (IAK), see section 3. The format of the IAT is not defined by PSM, however, the PSA Attestation API supports CBOR encoded IETF Entity Attestation Token.

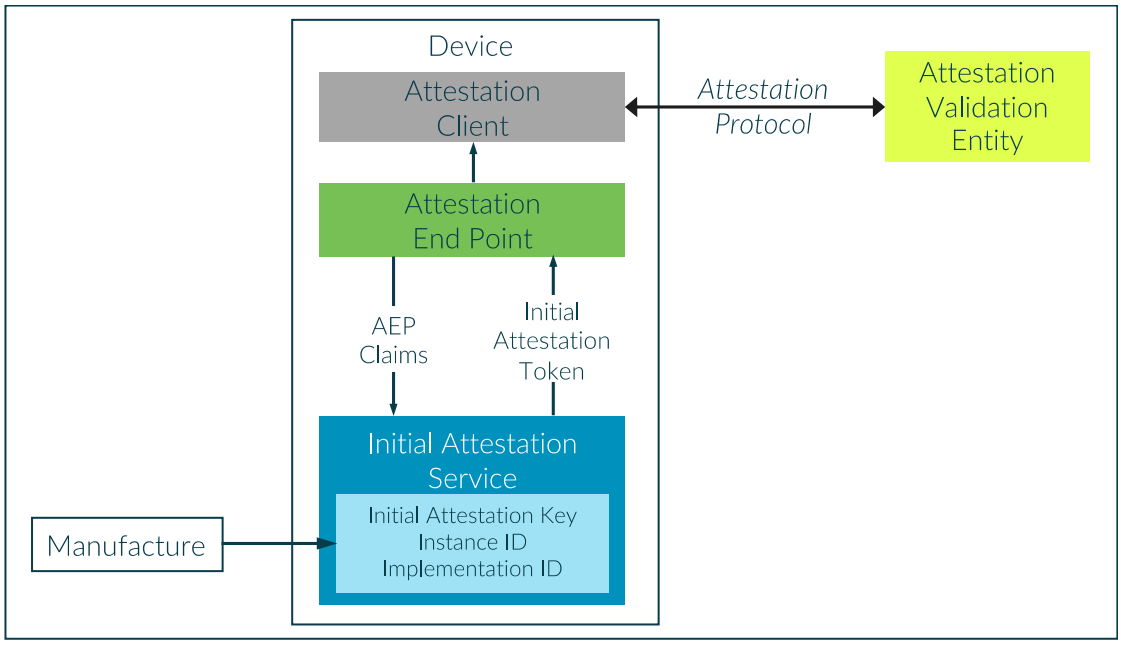

Figure 13: Attestation context

Figure 14 illustrates a typical initial attestation sequence. The validation entity (VE) challenges the Attestation End Point with the metadata in the object record VE_{OR}, the content and use depend on the attestation scheme. Use of a nonce in each challenge to ensure freshness of the data is recommended.

The AEP requests an Initial Attestation Token (IAT) from the Initial Attestation Service, providing at least the metadata that must be validated by the VE for the attestation scheme. Typically, this will be a cryptographic hash of the AEP-specific data in the object record AEP_{OR}, and VE_{OR}.

The Initial Attestation Service constructs the object record, IAS_{OR}, which typically includes:

- The boot seed from the Initial Boot State and the boot state of every updateable component loaded at secure boot. See section 6.4

- Current security lifecycle state of the system. See section 4
- Instance ID and Implementation ID, see section 3, and calling Partition ID

The Initial Attestation Key is used to sign a cryptographic hash of the data from the AEP and IASOR. Once signed using the IAK, it is returned to the AEP. This is the Initial Attestation Token.

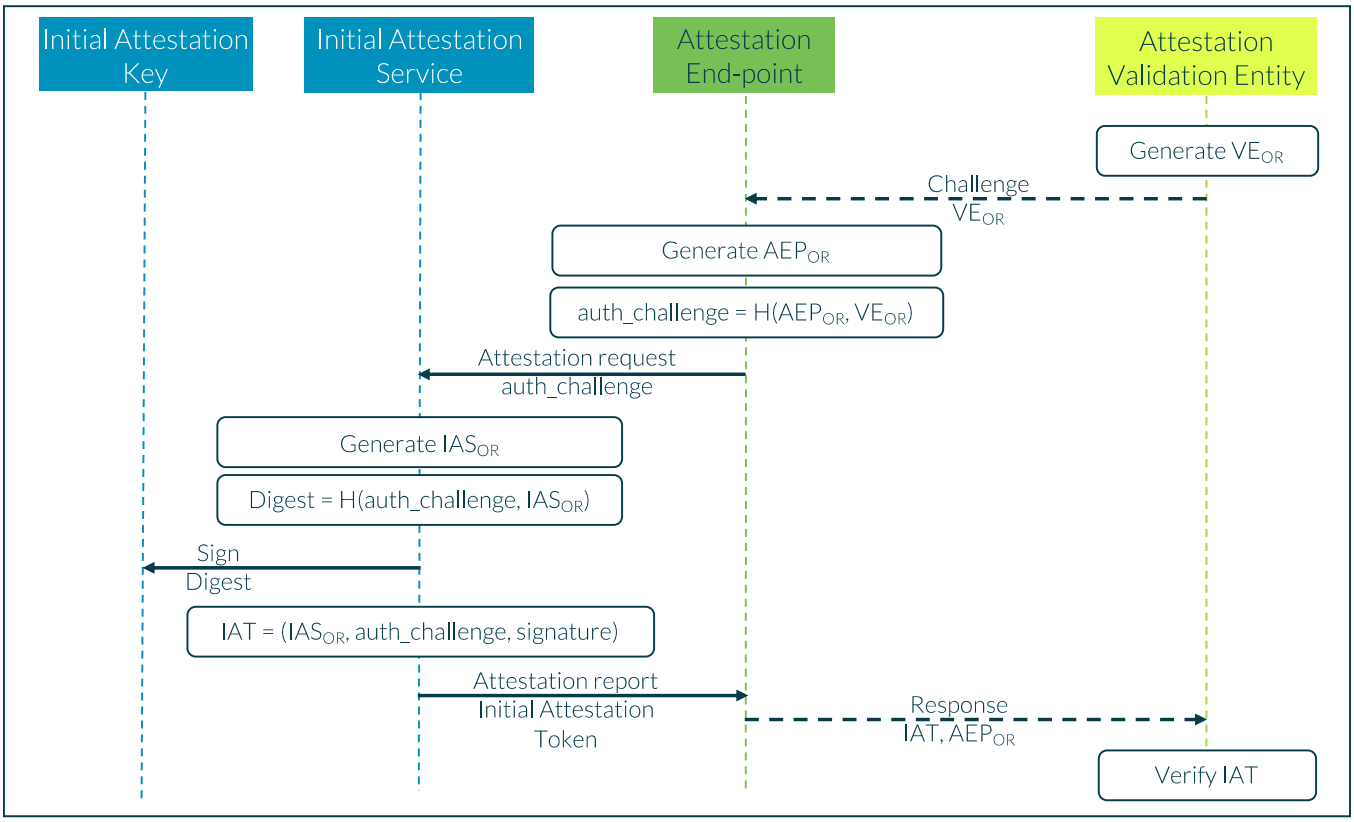

Figure 14: Example initial attestation challenge and response sequence

The challenge is then completed by the AEP returning the signed IAT and its object record AEP_{OR} to the validation entity. The VE can use the returned data along with knowledge of a valid set of updateable components for the corresponding implementation to validate:

- The trustworthiness of the Platform Root of Trust and its implementation
- The authenticity of the Object Record AEP_{OR}
- The context of the original validating entity challenge data VE_{OR}

The Initial Attestation Service can form the basis of delegated attestation. For example, a Delegated Attestation Service generates its own attestation key, which is included in the Initial Attestation Token by invoking the Initial Attestation Service. This provides proof of possession of the Initial Attestation Key as the root of the delegated attestation service.

7.4 Secure Storage

Many devices require secure persistent storage to hold sensitive data. That data could come from the manufacturer, or could be application-generated, service-generated, or user-generated. Such storage is

called non-isolated storage, see section 2.3, and is typically implemented with flash memory either on- or off-chip.

Secure storage is expected to be available to Application RoT services, but may be available to other partitions provided that that the following properties are met:

- Defined ownership of all stored data, regardless of the management of data on the storage device
- Privacy and integrity protection to prevent the data from being accessed or modified by an unauthorized agent, including when the device is in a non-secure state, such as during debug
- Replay protection to prevent a stored set of data from being replaced by a previously stored version of the data set

Depending on implementation requirements and certification profiles, these properties may be enforced by isolation, or by cryptography, or a combination of both, see section 5.

Secure storage is supported by the PSA Storage API.