Arm CCA Security Model 1.0

Architecture & Technology Group

Document number:	DEN0096
Document quality:	EAC
Document version:	A.a
Confidentiality	Non-confidential
Date of Issue:	02/08/2021

Copyright © 2020-21 Arm Limited or its affiliates. All rights reserved.

Abstract

Summary

The *Arm Confidential Compute Architecture (CCA) Security Model* (SM) defines the security requirements and the essential security properties of the *CCA isolation architecture*. These security properties are required to deploy applications protected by this architecture, with confidence in its underlying implementation.

The CCA SM is concerned with required security properties, expressed as *robustness rules*. Technical requirements for CCA hardware and firmware components identified by the CCA SM are specified in separate technical specifications.

Purpose of this document

An owner of an application deployed for Arm CCA must be able to:

- Establish the trustworthiness of an underlying implementation of Arm CCA
- Protect application assets if that trustworthiness determination can change

This document defines the foundation for establishing that trust by:

- Defining the security guarantee provided by Arm CCA
- Defining the underlying security capabilities that applications protected by Arm CCA can rely upon
- Providing technical input for deploying applications protected by Arm CCA within a wider ecosystem
- Establishing common technical definitions and terminology

Target audience for this document

Security communities in application development, application hosting, firmware development, silicon design and manufacture, and end product design and manufacture, for products and services related to Arm CCA.

Copyright © 2020-21 Arm Limited or its affiliates. All rights reserved.
Non-confidential

Contents

About this document vii

Release Information vii

References vii

Non-Confidential Proprietary Notice viii

Product Status ix

Document outline x

Conventions xi
Cryptographic terminology xi
Typographical conventions xii

Rules-based writing xii
Content item rendering xiii
Content item identifiers xiii

Feedback xiii
Feedback on this book xiii

Open issues xiv

1 Overview 15

1.1 Introduction 15

1.2 Problem statement 15

1.3 CCA security guarantee 16
1.3.1 Security guarantee to Realm Owner 16
1.3.2 Security guarantee to a hosting environment 16

2 CCA ecosystem 18

2.1 Ecosystem roles 18

2.2 Supply chain ecosystem 21

2.3 Attestation model 24

3 CCA overview 27

3.1 Elements of CCA 27

Copyright © 2020-21 Arm Limited or its affiliates. All rights reserved.
Non-confidential

3.2 **CCA platform** 28
3.2.1 CCA system security domain 28
3.2.2 Monitor security domain 28
3.2.3 Realm Management security domain 28

3.3 **Relationship to system platform security services** 29
3.3.1 Security Provisioning 29
3.3.2 Platform attestation 29

4 Hardware enforced security 31

5 CCA identity management 34

5.1 Hardware provisioned parameters 34

5.2 Derived parameters 35

6 CCA system security domain 36

6.1 On-chip memory 36

6.2 Isolated locations 36

6.3 Shielded locations 36

6.4 Immutable initial boot code 36

6.5 Power management 37
6.5.1 System reset 37
6.5.2 Software initiated system reset 37
6.5.3 System reset initiated by a trusted subsystem 37
6.5.4 System hibernation 37
6.5.5 System suspend 38

6.6 Isolation hardware 39

6.7 Trusted subsystems 39

6.8 Invasive subsystems 40

7 Protected memory 41

7.1 General threat model 41

7.2 Possible mitigations 41

7.3 Use of external memory by CCA 42

7.4 External memory initialization 43

Copyright © 2020-21 Arm Limited or its affiliates. All rights reserved.
Non-confidential

7.5 Assets 44

7.6 Baseline memory protection profile 45

7.7 Memory scrubbing 45

7.8 Additional memory protection 46

8 CCA firmware boot 47

8.1 Verified boot 47

8.2 Image formats and signing schemes 49

8.3 Anti-rollback 49

8.4 Off-line boot 50

8.5 CCA HES firmware boot flow 50

8.6 CCA system security domain boot process 51

8.7 Application PE boot process 52

8.8 Robustness 53

9 CCA attestation 54

9.1 Base attestation flow 55

9.2 Token formats and signing schemes 56

9.3 Privacy preserving attestation 56

9.4 Delegated Realm attestation 56

9.5 Local attestation 57

10 CCA firmware updates 58

10.1 Remote update 58

10.2 Local update 60

10.3 Robustness 60

11 CCA security lifecycle management 61

11.1 Firmware enabled debug 61

Copyright © 2020-21 Arm Limited or its affiliates. All rights reserved.
Non-confidential

11.2 CCA system security lifecycle 62

11.3 Reprovisioning 64

11.4 Trusted subsystems and CCA HES 65

12 Cryptographic recommendations 66

12.1 Cryptographic algorithms and key sizes 66

12.2 Post-quantum readiness 66

12.3 Guidance 66
12.3.1 Recommended parameter sizes 66
12.3.2 Recommended algorithms 67
12.3.3 Memory protection 67

13 Appendix A: Rule history 68

13.1 Last used identifier 68

13.2 New identifiers since last major release 68

13.3 Deprecated identifiers since last major release 68

Copyright © 2020-21 Arm Limited or its affiliates. All rights reserved.
Non-confidential

About this document

Release Information

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

Date	Version	Confidentiality	Change
2 August 2021	A.a	Non-confidential	First non-confidential EAC publication

References

Reference	Version	Date	Information
[RME]	A.a	23 June 2021	DDI0615 Arm Architecture Reference Manual Supplement, The Realm Management Extension (RME), for Armv9-A
[RMM]	00alp3	25 June 2021	ARM-AES-0013 Realm Management Monitor specification
[Boot PSG]	1.1 Release 0	July 2020	ARM-DEN-0072 Platform Security Boot Guide
[PSA Token]	1.0.2	19 February 2020	ARM-IHI-0085 PSA Attestation API

Copyright © 2020-21 Arm Limited or its affiliates. All rights reserved.
Non-confidential

Arm CCA Security Model

Copyright © 2020-21 Arm Limited or its affiliates. All rights reserved. The copyright statement reflects the fact that some draft issues of this document have been released, to a limited circulation.

Non-Confidential Proprietary Notice

This document is protected by copyright and other related rights and the practice or implementation of the information contained in this document may be protected by one or more patents or pending patent applications. No part of this document may be reproduced in any form by any means without the express prior written permission of Arm. No license, express or implied, by estoppel or otherwise to any intellectual property rights is granted by this document unless specifically stated.

Your access to the information in this document is conditional upon your acceptance that you will not use or permit others to use the information for the purposes of determining whether implementations infringe any third party patents.

THIS 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.

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, patents, copyrights, trade secrets, or other rights.

This document may include technical inaccuracies or typographical errors.

TO THE EXTENT NOT PROHIBITED BY LAW, IN NO EVENT WILL ARM BE LIABLE FOR ANY DAMAGES, INCLUDING WITHOUT LIMITATION ANY DIRECT, INDIRECT, SPECIAL, INCIDENTAL, PUNITIVE, OR CONSEQUENTIAL DAMAGES, HOWEVER CAUSED AND REGARDLESS OF THE THEORY OF LIABILITY, ARISING OUT OF ANY USE OF THIS DOCUMENT, EVEN IF ARM HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

This document consists solely of commercial items. You shall be responsible for ensuring that any use, duplication or disclosure of this document complies fully with any relevant export laws and regulations to assure that this document or any portion thereof is not exported, directly or indirectly, in violation of such export laws. Use of the word “partner” in reference to Arm’s customers is not intended to create or refer to any partnership relationship with any other company. Arm may make changes to this document at any time and without notice.

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

The Arm corporate logo and words marked with ® or ™ are registered trademarks or trademarks of Arm Limited (or its affiliates) 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. Please follow Arm’s trademark usage guidelines at http://www.arm.com/company/policies/trademarks.

Copyright © 2021 Arm Limited (or its affiliates). All rights reserved.

Arm Limited. Company 02557590 registered in England.

110 Fulbourn Road, Cambridge, England CB1 9NJ.

LES-PRE-20349 version 21.0

Product Status

The information in this document is final, that is for a developed product.

The information in this Manual is at EAC quality, which means that all features of the specification are described in the manual.

Copyright © 2020-21 Arm Limited or its affiliates. All rights reserved.
Non-confidential

Document outline

Section 1: Overview
Sets out context, and the CCA security guarantee.

Section 2: CCA ecosystem
Defines how Arm CCA can be deployed in the context of an ecosystem, and identifies associated roles.

Section 3: CCA overview
Introduces the CCA security architecture, defining central terms and concepts.

Section 4: Hardware enforced security
Defines extensions to the CCA architecture, moving selected CCA functions and secrets off application PE.

Section 5: CCA identity management
Defines CCA hardware provisioned keys, seeds, and identities, their storage, and access rules.

Section 6: CCA system security domain
Architecture and security requirements for CCA hardware elements and subsystems.

Section 7: Protected memory
Security requirements for protecting off-chip memory.

Section 8: CCA firmware boot
Defines a CCA boot architecture and associated security requirements.

Section 9: CCA attestation
Defines a CCA attestation architecture and associated security requirements.

Section 10: CCA firmware updates
Defines a CCA firmware update architecture and associated security requirements.

Section 11: CCA security lifecycle management
A CCA system may be in a fully trusted state or in states of degrees of trustworthiness. This chapter defines the CCA security lifecycle and rules for how it can be managed.

Section 12: Cryptographic recommendations
General recommendations and guidance for cryptography in a CCA enabled system.

Section 13: Appendix A: Rule history
Change history and management of rules defined in this specification.

Section 14: Appendix B: External threat model

Not up to date, will be revised or removed in a later version of this document.

Copyright © 2020-21 Arm Limited or its affiliates. All rights reserved.
Non-confidential

Conventions

Cryptographic terminology

This document assumes general familiarity with cryptographic concepts and methods.

In the context of cryptography, the following English words have special meaning in this document:

Term	Description	Information
Temporal freshness	A contribution that is different on each transaction.	Temporal freshness typically involves a monotonic counter with a scope defined by some session. For example, a monotonic counter reset at boot or at the start of a protocol session. In some cases statistical uniqueness may be used (a random number). For example, when the scope is wider than a boot session, or wider than a system, and a counter might not be feasible.
Boot freshness	A contribution that is at least randomly different following each system reset.	For example, a random boot seed generated on each boot. Or a hash including a monotonic hardware counter incremented on each boot.
Location freshness	A contribution that is different depending on memory location.	Typically a memory address.
Encryption Decryption	Encryption is a cryptographic operation that is used to provide confidentiality for sensitive information, and decryption is the inverse operation. Prevents visibility of data to unauthorized parties.	In this document: 1. Data encryption is done using symmetric encryption algorithms 2. Key wrapping may use either symmetric or asymmetric encryption algorithms, depending on context
Hash function	A hash function is a one-way function that maps an arbitrary length string to a numerical value, such that: • The value returned for a given string is always the same • Any bit change in the input results in a randomly different output value	A hash function may be used to create a “finger print” for a data string. The properties of a hash function are such that it is not possible (in practice highly improbable) to derive the original data from a known hash value. Hash functions are used, for example, for integrity protection or for deriving keys and identities.
Integrity protection	Integrity protection is a cryptographic operation preventing undetected modification of data by unauthorized parties, either in storage or in transit.	Depending on context, integrity protection may be provided by: 1. Locally stored hash values (hash locking, hash chaining, hash trees) 2. A message authentication code (MAC) using symmetric keys 3. A digital signature using asymmetric keys

Copyright © 2020-21 Arm Limited or its affiliates. All rights reserved.
Non-confidential

Term	Description	Information
Replay protection	Provide assurance of data freshness. Prevents replay of older data sets.	Typically requires integrity protection together with freshness. The type of freshness required depends on context and determines the scope of replay protection.
Source authentication	Provide assurance of the source of data. Prevents data being supplied by an unauthorized party.	A digital signature or a MAC can also provide assurance about the origin of data.

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 source code examples.

Also used in the main text for 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

Rules-based writing

This specification consists of a set of individual *content items*. A content item is classified as one of the following:

- Rule
- Information
- Rationale
- Implementation note

Rules are normative statements. An implementation that is compliant with this specification must conform to all Rules in this specification that apply to that implementation.

Copyright © 2020-21 Arm Limited or its affiliates. All rights reserved.
Non-confidential

Rules must not be read in isolation. Where a particular feature is specified by multiple Rules, these are generally grouped into sections and subsections that provide context. Where appropriate, these sections begin with a short introduction.

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

Content items other than Rules are informative statements. These are provided as an aid to understanding this specification.

Content item rendering

Content items in this document are identified by paragraph formatting:

[R0123] This is a rule statement.

This statement is informational, providing rationale, information and context for rules.

This statement is highlighted to explicitly call out a property that is out of scope in the current version of the CCA architecture. These properties are called out because they have only been deemed out of scope of current CCA minimal requirements. They may be considered in future releases of CCA.

This is an editorial note to readers. It will be removed at final release of the document.

Content item identifiers

Each rule may have an associated identifier that is unique within the context of this specification.

Identifiers have the following form:

Rnnnn

Where:

- "nnnn" is a four digit rule identifier unique within this document, allowing referencing individual rules

Example:

[R0123] This is a rule statement.

After this specification reaches beta status, a given content item has the same identifier across subsequent versions of the specification.

Used and deprecated content identifiers are tracked in *Appendix A: Rule history*.

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.newmore-feedback@arm.com. Give:

- The title (Arm CCA Security Model).
- The number and release.
- The page numbers to which your comments apply.
- The rule identifiers to which your comments apply, if applicable.

Copyright © 2020-21 Arm Limited or its affiliates. All rights reserved.
Non-confidential

- A concise explanation of your comments.

Arm also welcomes general suggestions for additions and improvements.

Open issues

Key	Description

Copyright © 2020-21 Arm Limited or its affiliates. All rights reserved.
Non-confidential

1 Overview

1.1 Introduction

Arm *Confidential Compute Architecture (CCA)* is an architecture that provides protected execution environments called *Realms*.

The purpose of a Realm is to provide an environment for confidential computing.

In addition to privilege-based isolation provided by the standard Arm architecture, a Realm receives protections operating in the opposite direction – resources allocated to a Realm are also protected against access by higher privileged agents. This includes software or firmware executing at a higher exception level as well as hardware agents such as DMA agents.

1.2 Problem statement

Software on complex modern systems operates in an environment of mutual distrust. For example:

- Application owners who do not trust each other, hosted on the same system
- Application owners have to trust the hosting environment
- A hosting environment has to trust applications not to attempt to compromise the host, or other applications
- Platform security services do not trust the hosting environment or applications

Existing isolation architectures, such as a hypervisor, protect the hosting environment from applications by preventing these from accessing resources that are private to the hosting environment. However, applications are not protected from the hosting environment.

Similarly, TrustZone can protect secure services from non-secure services, but does not protect applications from secure services.

CCA introduces additional boundaries protecting resources based on ownership, regardless of privilege level. It has been designed to meet the following overall security goals:

- Minimal chain of trust

  An application only needs to trust itself, and the part of the system delivering the CCA security guarantee.

  For example, an application does not need to trust hosting software such as a hypervisor, nor any non-CCA hardware agents, regardless of privilege level.
- Attestable trust

  A user of an application deployed as a Realm on a CCA enabled system can attest the trustworthiness of the application, as well as the trustworthiness of all CCA firmware and all hardware agents involved in delivering the CCA security guarantee.
- Certifiable

  The implementation of the CCA security guarantee is contained within an minimal set of hardware and CCA firmware components that can be developed, certified, and verified.

Copyright © 2020-21 Arm Limited or its affiliates. All rights reserved.

Non-confidential

1.3 CCA security guarantee

Arm CCA has been designed to provide specific security guarantees to both Realms and to the hosting environment.

The guarantees, of course, assume a correct implementation of CCA. The system hardware and firmware features required to provide assurance about the trustworthiness of an implementation of CCA, and the security of Realm assets, are the main topics of this specification.

1.3.1 Security guarantee to Realm Owner

A Realm Owner represents the owner of Realm policy.

In the context of CCA, Realm policy includes the conditions under which a Realm can access assets protected by a Realm on behalf of the Realm Owner itself, or on behalf of users of the Realm. Including, for example, Realm creation policy and attestation policy.

Guarantee	Description	Notes
Confidentiality and integrity	Realm memory content and execution context cannot be accessed or modified by: 1. Other Realms 2. Non-CCA software, firmware, and hardware	For example, a Realm is protected from: • Other Realms • Any software in the hosting environment, for example a hypervisor • All software in TrustZone running in the Secure state • Secure or non-secure DMA agents
Minimal chain of trust	Both the Realm owner and a user of a Realm only need to trust CCA firmware and hardware.	For example, a Realm does not need to trust: • Other Realms • Any software in the hosting environment, for example a hypervisor • Any software in TrustZone running in the Secure state
Attestable trust	A user of a Realm is able to attest the trustworthiness of the Realm, and of CCA firmware and hardware.	
Protection of persistent Realm assets	Persistent Realm assets can only be accessed by a Realm if the Realm and CCA firmware and hardware are in a trustworthy state.	Trustworthiness is subject to policy controlled by the owner of the Realm.

1.3.2 Security guarantee to a hosting environment

A hosting environment in this context represents non-CCA software, firmware, and hardware on a system hosting a Realm, including policy.

Copyright © 2020-21 Arm Limited or its affiliates. All rights reserved.
Non-confidential

Guarantee	Description	Notes
Management and availability of resources	The hosting environment has full control of resource allocation, including memory allocation, scheduling, and NS devices.	Including resources allocated to Realms.
System policy	For example, the hosting environment power management policy and other system policies.	CCA may enforce that system policies are safe and do not compromise the CCA security guarantee.
Realm creation policy	The hosting environment can create a Realm in any state, including an untrustworthy state such as *debug by host*.	The state of a Realm is reflected in CCA attestation, which cannot be affected by the hosting environment. Realm creation policy and Realm state are out of scope of this document.
System boot state	The system can be booted in any state, including a state that might compromise the delivery of the CCA security guarantee.	

Copyright © 2020-21 Arm Limited or its affiliates. All rights reserved.
Non-confidential

2 CCA ecosystem

2.1 Ecosystem roles

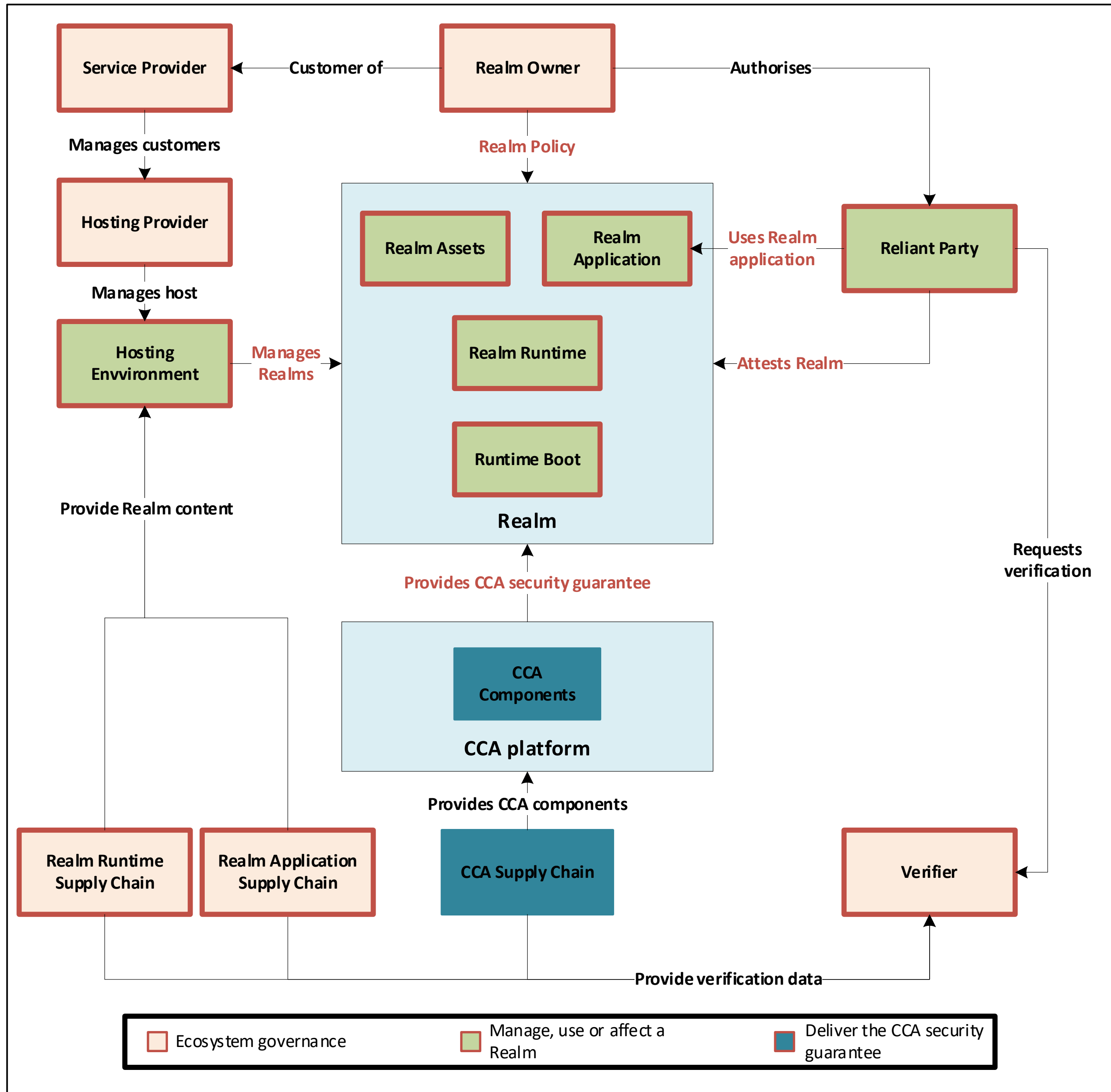

Figure 1. CCA ecosystem roles

A generic CCA ecosystem is outlined in *Figure 1. CCA ecosystem roles*. It is not intended to be specific to any particular ecosystem or any particular application of Arm CCA. Its purpose is to identify generic roles involved in deploying CCA and applications in Realms in any ecosystem.

Copyright © 2020-21 Arm Limited or its affiliates. All rights reserved.
Non-confidential

For example, in a datacenter context:

- The hosting environment might be a physical server
- The Realm might be a protected customer VM or a cloud service instance
- The reliant party might be a remote client connecting to the Realm to perform some task.

In a mobile client example:

- The hosting environment might be a physical mobile device
- The Realm might a protected client app, or an app service
- The reliant party might be a remote service the Realm connects to in order to complete some task on behalf of a user of the mobile device.

When these roles are referred to in this and other CCA specifications then they should be understood in the context defined here.

Role	Description	Notes
Host	Physical device (hardware and software) a Realm is deployed on.	For example, a datacenter server, or a mobile device.
Hosting environment	Non-CCA software, firmware, and hardware on a system hosting a Realm, including policy	For example Hypervisor, and resource allocation policy.
Realm	An isolated region protected by the CCA security guarantee.	
Realm content	Runtime code and data protected by the CCA security guarantee.	
Realm assets	Secrets protected by a Realm	For example, supplied to a Realm by the user of a service provided by the Realm. Or provisioned by a Realm Owner. May include runtime assets, as well as persistent assets. May include data, or code, or both.
Realm Owner	The owner of Realm policy.	
Realm runtime boot	The only code executing in a Realm when it is first launched.	Typically boot loader code.
Realm runtime	General runtime environment in a Realm.	For example, a kernel and user space middleware.
Realm application	Application specific code within a Realm.	

Copyright © 2020-21 Arm Limited or its affiliates. All rights reserved.
Non-confidential

Role	Description	Notes
Reliant party	An end point making use of services provided by a Realm or an endpoint providing a service to a Realm.	For example: • A user client, accessing a cloud service deployed in a Realm • A banking service, interacting with a client application deployed in a Realm • A remote client, accessing a voice recognition service deployed in a Realm
Hosting provider	Provider of hosting services.	For example, a datacenter operator.
Service provider	Operator of services deployed in Realms on behalf of customers.	For example, a cloud service provider, or a mobile communications service provider.
Supply chains	Both Realm content and implementations of CCA components are expected to be provided through ecosystem specific supply chains.	CCA does not define or restrict supply chain models. But to maintain the trustworthiness of a CCA deployment in an ecosystem appropriate governance and certification processes are required.
Verifier	To the determine the trustworthiness of a Realm, a client application needs access to a verification service able to interact with the supply chains to verify both Realm content and the underlying implementation of the CCA security guarantee.	

Copyright © 2020-21 Arm Limited or its affiliates. All rights reserved.
Non-confidential

2.2 Supply chain ecosystem

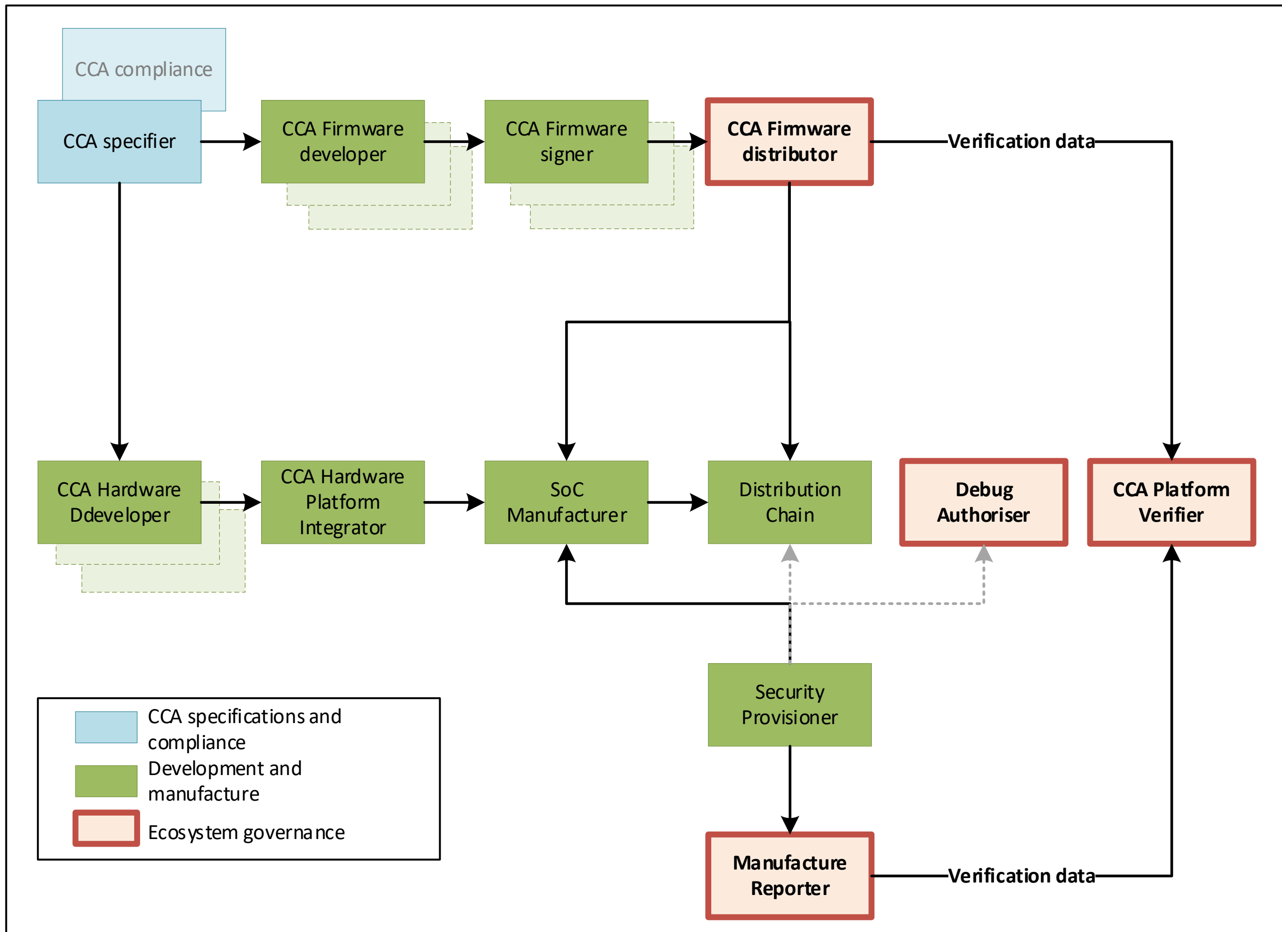

Figure 2. CCA supply chain roles

A generic CCA platform supply chain is outlined in *Figure 2. CCA supply chain roles*. It is not intended to be specific to any particular ecosystem or any particular manufacture or distribution model. Its purpose is to identify generic roles involved in developing and deploying a CCA platform in any ecosystem.

CCA identifies a minimum set of security parameters that must be finalized before an instance of a CCA platform is used. CCA does not define how those parameters are provisioned in a supply chain. For example, at least some initial security provisioning is required at silicon manufacture. However, final security provisioning may take place at some later stage in the distribution chain or in an online activation process at point of use.

In order to use a CCA platform, a verifier needs to establish the security lifecyle status, and revocation status, of the CCA platform hardware and firmware.

The verifier needs to use manufacture status information about the hardware components that make up the CCA platform. For example, hardware may have been manufactured in a fully secured state. Or it may have been manufactured in a non-secured state for test and development purposes. Alternatively, device hardware may have been manufactured to market or customer specific requirements, such as local cryptography standards. Capturing the manufacture state is represented by a manufacture reporter role.

The verifier also needs to use release information about all CCA platform firmware. For example, firmware may be released in a fully secured state, or it may contain revealing firmware level logging or debugging capabilities. Alternatively, a released CCA platform firmware component may have been revoked because of a security vulnerability. All CCA platform firmware is signed by firmware signers. Depending on supply chain requirements,

Copyright © 2020-21 Arm Limited or its affiliates. All rights reserved.
Non-confidential

different CCA platform firmware components may be signed by different firmware signers. Capturing the state of all CCA firmware for a particular CCA platform is represented by a firmware distribution role.

When these roles are referred to in this and other CCA specifications,they should be understood in the context defined here.

Role	Description	Notes
CCA specifier	Specifies required security properties and security features of CCA components. May also define interfaces and formats where interoperability is required.	Some components may be specified by Arm. Some may be ecosystem dependent and specified wholly or in part by ecosystem partners. Compliance is out of scope of this document.
CCA firmware developer	Developer of firmware for one or more CCA components.	It is expected that different components may be developed by different ecosystem partners.
CCA firmware signer	Signer of firmware for one or more CCA components.	All CCA firmware is signed and verified.
CCA hardware developer	Developer of system hardware for one or more CCA components.	It is expected that different components may be developed by different ecosystem partners.
CCA platform integrator	Responsible for integration of components, possibly from multiple suppliers, into a CCA system platform with known security properties.	Security properties here include robustness properties as well as functional security properties.
SoC manufacturer	Manufacturer of a physical SoC containing an integrated implementation of a CCA system platform with known security properties	Security properties here include robustness properties as well as functional security properties.
Distribution chain	Distribution of SoC containing an integrated implementation of a CCA system platform.	Including end product assembly and manufacture, end product distribution, and deployment of an end product by an end user.
Security provisioner	Provisions CCA security parameters on manufactured instances of physical SoC containing an integrated implementation of a CCA system platform with known security properties.	
Debug authorizer	Responsible for implementing governance processes for enabling hardware or firmware debug of CCA components.	Enabling debug features for CCA components can compromise the CCA security guarantee. Authorizing such debug must be controlled with appropriate ecosystem governance.

Copyright © 2020-21 Arm Limited or its affiliates. All rights reserved.
Non-confidential

Role	Description	Notes
Manufacture reporter	Tracks manufactured physical SoC containing an integrated implementation of a CCA system platform with known security properties.	Security properties here include manufacture status. For example, a CCA identity may have been manufactured in a fully secured state. Or it may have been manufactured in a non-secured state for test and development purposes.
CCA firmware distributor	Responsible for release management of CCA firmware with known security properties.	Including version control, release status, and revocation status. Security properties here include robustness properties as well as functional security properties.
CCA platform verifier	A verification service able to interact with the supply chain to verify the CCA platform as part of Realm attestation.	

Copyright © 2020-21 Arm Limited or its affiliates. All rights reserved.
Non-confidential

2.3 Attestation model

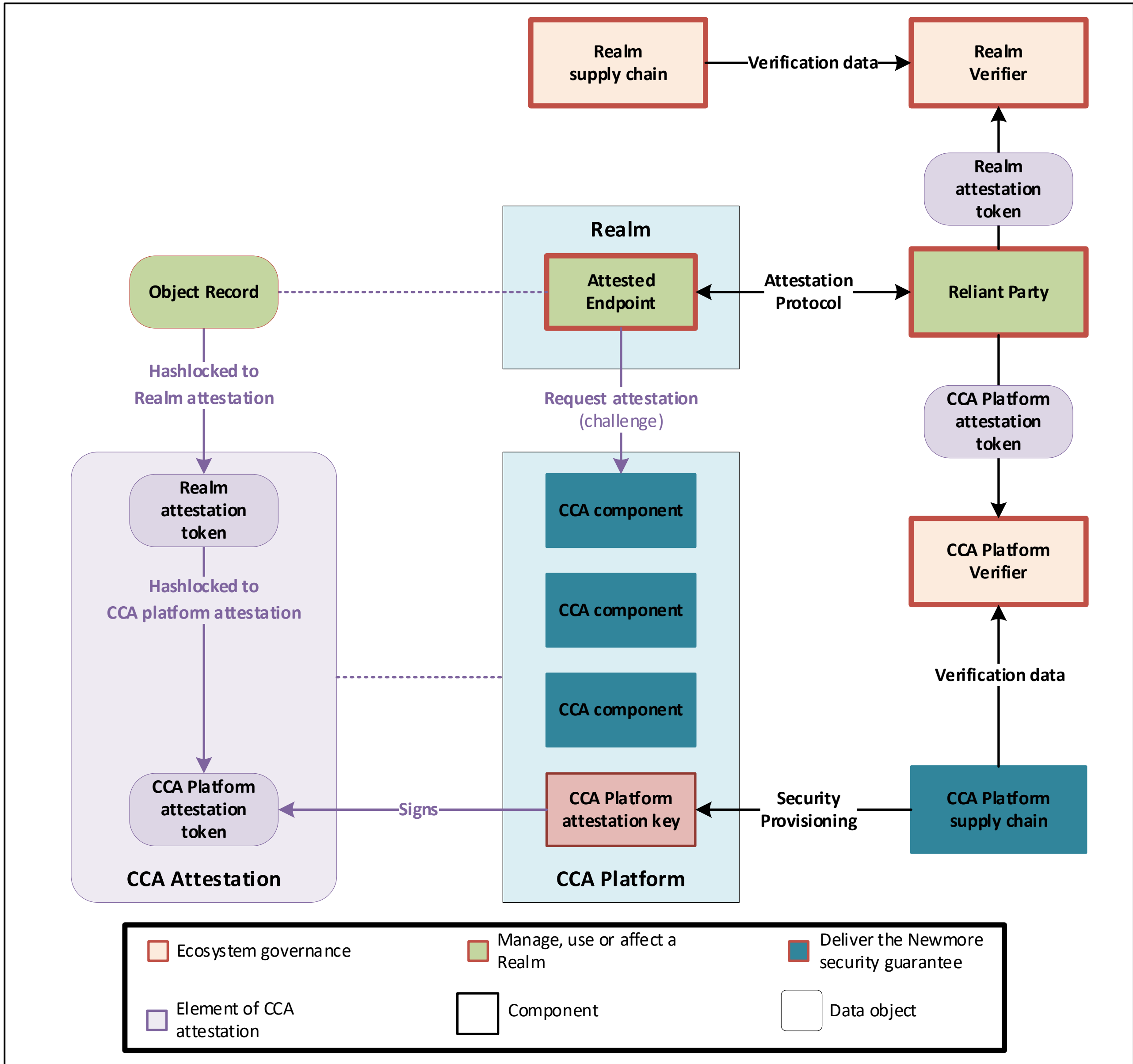

Figure 3. Generic CCA attestation model

The generic CCA attestation model is outlined in *Figure 3. Generic CCA attestation model*.

In this document, a basic group based attestation model is used to describe the principles and security properties of CCA attestation. But the model is general and adaptable to cover a range of possible deployment models. For example, CCA attestation can support individual or group based attestation. It can be deployed using simple identity management at manufacture. It can also be deployed using more complex identity management schemes, including online activation processes at point of use.

CCA attestation is agnostic to choice of attestation protocol. CCA attestation is a token based model, leaving Realms free to implement any attestation protocol and bind that protocol to CCA attestation tokens.

This document describes the underlying principles and common security properties of CCA attestation. It does not aim to describe all possible deployment models.

Copyright © 2020-21 Arm Limited or its affiliates. All rights reserved.
Non-confidential

CCA attestation has been designed with the following goals:

- Attestation protocol and use case independence, leaving Realm designers free to implement any attestation protocol required for their use case
- Independent boot and attestation of Realms, as well as of updateable components of the CCA platform
- Rooted in a hardware provisioned *CCA platform attestation key (CPAK)*, identifying the security properties of immutable[1] CCA system hardware

CCA attestation is a token based model. It is defined in more detail in *CCA attestation*. This introduction section uses a remote attestation use case as an example to outline the general security elements and principles. Other use cases such as local attestation or delegated attestation can also be supported.

The main purpose of attestation is for a Reliant Party to be able to establish a secure connection to an attested end point in a Realm, based on attested security properties of both the Realm itself and of the underlying CCA platform implementing the CCA security guarantee.

An attested end point may represent, for example, a trusted service provided by the Realm, or a credential or key protected by the Realm.

An attested end point is associated with an object record managed by the Realm. This typically includes negotiated security properties for the attestation protocol such as exchanged public keys or Diffie-Helman parameters, protocol freshness parameters, and identifiers for attested objects such as a key ID or a service ID.

The CCA platform manages a Realm boot state and a CCA platform boot state. The Realm boot state represents attested Realm security configurations, and measurements of Realm content. The CCA platform boot state represents attested CCA platform security configurations, and measurements and identification of all CCA platform firmware.

Security configurations include at least any Realm or CCA firmware deployment configurations that may affect the CCA security guarantee.

For the purpose of this introduction, CCA attestation consists of a Realm attestation token capturing the Realm boot state, and a CCA platform attestation token capturing the CCA platform boot state. Tokens are expected to be defined in a standard format, such as the PSA token format [PSA Token].

A Realm can request a CCA attestation from the CCA platform. The Realm supplies a challenge as part of the request, typically a cryptographic hash of the associated object record for the attested end point. This challenge is included in the Realm attestation token, binding security state of the attested end point to a CCA attestation.

The Realm attestation token is in turn hashlocked to the CCA platform attestation token. Finally the CCA platform attestation token is signed by CPAK. CPAK is typically provisioned in hardware by the CCA platform supply chain as part of a security provisioning process, or derived from other hardware provisioned secrets. See *Supply chain ecosystem*.

The tokens get relayed by the attested end point to the reliant party. The reliant party interacts with a CCA platform verifier to verify CPAK and the attested security properties of all hardware and firmware of the CCA platform. To do this, the verifier needs access to hardware manufacture and firmware release management data as discussed in *Supply chain ecosystem*.

The reliant party can then verify the Realm attestation token, hashlocked to the CCA platform attestation, with a Realm verifier with access to verification data from the Realm supply chain.

Finally, the reliant party can now verify the negotiated security state for the connection, as represented by the object record hashlocked to the Realm attestation. For example, negotiated public keys or Diffie-Helman

---

[1] Immutable here means it cannot change on a secured system. See *CCA security lifecycle management*.

Copyright © 2020-21 Arm Limited or its affiliates. All rights reserved.
Non-confidential

parameters, protocol freshness state, and so on. This negotiated security state can then be used to establish a point to point attested secure connection between the reliant party and the attested end point in the Realm.

Copyright © 2020-21 Arm Limited or its affiliates. All rights reserved.
Non-confidential

3 CCA overview

3.1 Elements of CCA

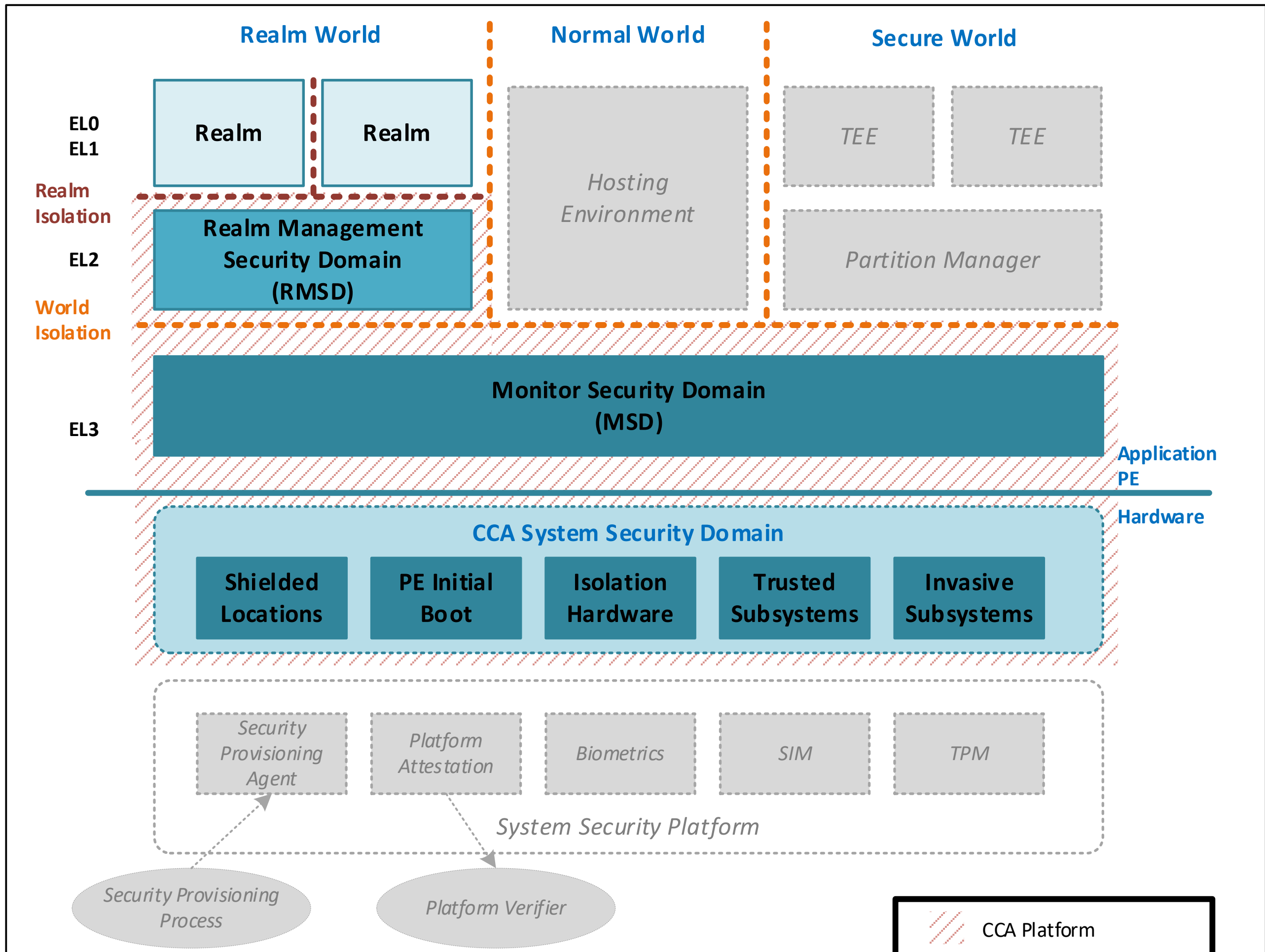

Figure 4. Elements of CCA

The high-level CCA architecture is outlined in *Figure 4. Elements of CCA*.

At the hardware level, the *CCA system security domain* includes all system hardware components that may affect the delivery of the CCA security guarantee.

[R0001] Hardware agents not directly involved in delivering the CCA security guarantee cannot directly affect components in the CCA system security domain.

Firmware and software on application PE are split into four *Worlds* separated by World protection boundaries. A World represents a system security state and associated private resources – such as memory, registers, and devices – which are only accessible when a requester is in that state.

1. Root world – responsible for enforcing the protection boundaries between Worlds
2. Secure world – all TrustZone firmware
3. Realm world – all Realms
4. Normal world – all other hosting environment software and firmware

Copyright © 2020-21 Arm Limited or its affiliates. All rights reserved.
Non-confidential

[R0002] Realm world memory and execution context cannot be affected by Normal world or Secure world.

Secure world memory and execution context cannot be affected by Normal world or Realm world.

Root world memory and execution context cannot be affected by any other world.

Normal world memory and execution context is not afforded any security guarantee by CCA.

Monitor security domain (MSD) represents the highest privilege application PE firmware on a CCA enabled system.

[R0003] Only Monitor firmware can directly access hardware in the CCA system security domain.

Monitor firmware is responsible for enforcing *World isolation* protection boundaries.

Realm Management security domain (RMSD) represents the highest privileged firmware within Realm world, responsible for enforcing the CCA security guarantee between Realms in Realm world.

3.2 CCA platform

The CCA platform is a collective term used to identify all hardware and firmware components involved in delivering the CCA security guarantee. Hence, all hardware and firmware components on a CCA enabled system that a Realm is required to trust.

This includes:

- CCA system security domain
- Monitor security domain
- Realm Management security domain

3.2.1 CCA system security domain

CCA system security domain consists of all elements of the system architecture that may affect the CCA security guarantee. This includes:

- Application PE initial boot code – for example, a boot ROM
- Shielded locations – for example, tamper resistant fuses storing CCA hardware provisioned assets
- Isolation hardware – all system hardware agents involved in enforcing CCA isolation policies, for example MMU or MPU
- Trusted subsystems – isolated and self-contained execution environments with their own firmware, separate from the application PE

  For example, a host for CCA *Hardware Enforced Security (CCA HES)* (see Hardware enforced security) or a power management subsystem.
- Invasive subsystems – for example, debug or diagnostics interfaces

3.2.2 Monitor security domain

Monitor security domain represents all firmware executing in Root world on application PE. It mplements the CCA security guarantee between worlds.

Monitor security domain must be trusted by both Realm world and Secure world.

3.2.3 Realm Management security domain

Realm Management security domain represents all firmware executing in Realm world on application PE. It implements the CCA security guarantee between Realms within Realm world.

Copyright © 2020-21 Arm Limited or its affiliates. All rights reserved.
Non-confidential

Realm Management security domain must be trusted by Realms.

3.3 Relationship to system platform security services

A system may provide platform security services not related to CCA. For example:

- Identity management, such as subscriber identity
- Key storage and key management not related to CCA
- Platform measuring and attestation
- Biometrics and user authentication

CCA does not rely on any such services for its own security. They are not discussed in this document.

Two specific platform security services may intersect with CCA depending on implementation and are mentioned here for context only:

- Security provisioning
- Platform attestation

Platform security and platform attestation is typically implemented in dedicated trusted subsystems, including standards such as [Titan] and [Cerberus].

3.3.1 Security Provisioning

A security provisioning agent represents any features in support of provisioning secrets, for example for provisioning SIM or TPM identities.

CCA also requires hardware provisioned assets for providing the CCA security guarantee, see *CCA identity management*.

Security provisioning processes are expected to be ecosystem specific, and may occur at any stage, or combination of stages, in a device supply chain. For example:

- At silicon manufacture
- As part of device activation at the end of device manufacture
- As part of an online activation process by a device owner

CCA does not define or restrict what security provisioning processes are used or how they are designed.

3.3.2 Platform attestation

Platform attestation is a common feature of modern complex devices with multiple platform security subsystems. Each individual platform security subsystem may implement its own secure boot process, but does not in itself provide an overall security boot state of the platform as a whole.

In general, platform attestation consists of a dedicated trusted subsystem capable of independently measuring the critical boot stages of all platform security components, and provides an overall platform security attestation.

Platform attestation is transparent to the platform security subsystems it measures.

The result of a platform attestation may be used in several ways. For example:

- Enforce local device policy
- Provide independent proof of the overall security state of a system to a remote platform verifier

  For example, enable a Cloud hosting provider to enforce policy in a datacenter.

Copyright © 2020-21 Arm Limited or its affiliates. All rights reserved.
Non-confidential

Measurements taken as part of the platform boot process may also be used in key derivations. For example, it may be used to protect user data against unauthorized firmware.

Platform attestation does not replace local security processes within individual platform security subsystems. Instead it attempts to enforce global policy across multiple mutually distrusting platform security subsystems.

CCA does not define or restrict platform attestation processes, and does not rely on platform attestation for its own security processes.

Copyright © 2020-21 Arm Limited or its affiliates. All rights reserved.
Non-confidential

4 Hardware enforced security

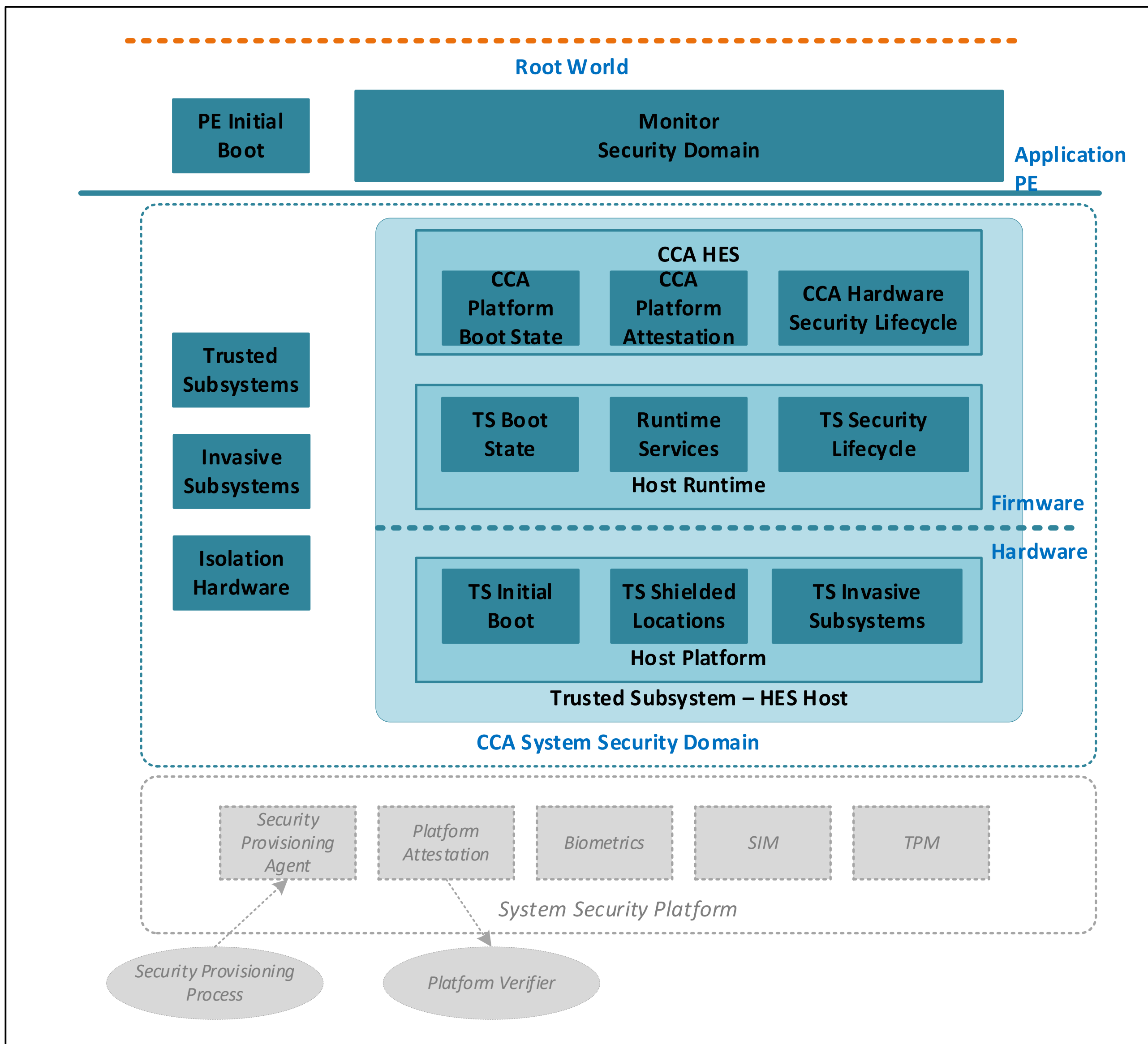

Figure 5. CCA hardware enforced security

[R0004] Arm strongly recommends that all implementations of CCA utilize *hardware enforced security (CCA HES)*.

The remainder of this document assumes a CCA HES enabled system.

CCA HES is a tenant of a trusted subsystem – a *CCA HES Host*. See *Figure 5. CCA hardware enforced security*. It re-locates the following Monitor security domain services away from application PE. These services involve all

Copyright © 2020-21 Arm Limited or its affiliates. All rights reserved.
Non-confidential

operations directly affecting initial measurements for CCA, and all operations directly affecting CCA hardware provisioned parameters. In particular:

- CCA platform attestation
- Tracking of CCA boot state
- CCA security lifecycle management
- CCA enforced key derivations (future CCA use cases)

In addition, the following hardware elements of the CCA platform domain are moved to the CCA HES Host:

- Shielded locations

The main motivations include:

- Reduce exposure of root secrets to application PE side-channels

  Especially on complex systems with a large number of application PE sharing a complex memory subsystem.

- Reduce exposure to mis-reporting of attestation state

  In particular the boot measurements and boot state of CCA in case of a vulnerability compromising Root world and the Monitor security domain.

[R0005] CCA HES can only be deployed on a dedicated trusted subsystem, or as a tenant on a multi-tenant trusted subsystem.

A trusted subsystem provides a hardware isolated execution environment for sensitive security services, implementing its own boot processes and security lifecycle management. See *Trusted subsystems*.

[R0006] A trusted subsystem hosting CCA HES services also hosts all shielded locations used by CCA.

[R0007] Only CCA HES can have direct access to those locations.

CCA HES provides the following services and interfaces to application PE:

- CCA platform attestation service
- Interface to capture Monitor firmware measurements as measured by PE initial boot code, and associated Monitor firmware identity metadata
- Interface to capture Realm Management firmware measurements as measured by Monitor firmware and associated Realm Management firmware identity metadata

[R0008] Only Monitor security domain, and PE initial boot, can directly access CCA HES from an application PE.

CCA HES provides the following services and interfaces to other trusted subsystems in the CCA system security domain

- Interface to capture individual Trusted subsystem firmware measurements as measured by each trusted subsystem, and associated Trusted subsystem firmware identity metadata

[R0009] Interfaces to trusted subsystems in the CCA system security domain can only be directly accessed from CCA system security domain.

CCA HES, and the CCA HES host, implement the following private services:

- Cryptographic services
- Interfaces to capture CCA HES host firmware measurements taken by the CCA HES host itself as part of its own boot process and associated CCA HES host firmware identity metadata

Copyright © 2020-21 Arm Limited or its affiliates. All rights reserved.
Non-confidential

[R0010] Measurements taken by the CCA HES host cover at least all firmware, executing in the context of the CCA HES host, which may affect the CCA security guarantee.

Interfaces to capture measurements take the form of registers with hash extend and lock semantics.

[R0011] Measurement registers are reset to a known state following system reset.

[R0012] The value of a measurement register is a hash value, which can only be extended. It is not possible to modify the value of a measurement register other than by extending it.

[R0013] Once locked, a measurement register cannot be updated again until following system reset. System reset is the only way to reset a measurement register.

The primary purpose of CCA HES is to build up a boot state of all parts of a system that may affect the CCA security guarantee, enabling CCA platform attestation. CCA HES does not itself calculate those measurements. For example, CCA HES trusts immutable PE initial boot to measure Monitor firmware as part of application PE boot on a secured system. Likewise, CCA HES trusts the immutable initial boot processes of other trusted subsystems in the CCA system security domain.

To guarantee correct reporting of measurements CCA HES enforces the security lifecycle state of the CCA system security domain.

[R0014] No part of the CCA system security domain can boot in a non-secured state without authorization from CCA HES. Including trusted subsystems and application PE.

Copyright © 2020-21 Arm Limited or its affiliates. All rights reserved.
Non-confidential

5 CCA identity management

This specification defines a set of secrets and public identifiers that may be used in determining the trustworthiness of a CCA deployment, and to protect CCA assets.

They are listed here for reference.

This document uses the term *parameter* to mean any CCA specific key, seed, identifier or version information either provisioned in CCA system security domain, or derived from hardware provisioned parameters.

5.1 Hardware provisioned parameters

An *immutable parameter* is a parameter that cannot change.

A CCA hardware provisioned parameter is an immutable parameter provisioned in CCA system security domain.

A *shielded location* is an on-chip NV storage location with a degree of tamper resistance.

A *private parameter* holds a secret value, such as a secret key.

A *public parameter* holds a value that is not secret, such as a public key or identifier.

[R0018] CCA hardware provisioned parameters are only stored in shielded locations.

[R0019] CCA hardware provisioned parameters are immutable on a secured device.

[R0020] Private CCA hardware provisioned parameters can only be directly accessed by CCA HES (see *Hardware enforced security*).

Minimum CCA hardware provisioned parameters include:

Parameter	Description	Secret
HUK	Hardware unique symmetric key. It represents a randomly unique seed for each manufactured instance of a CCA enabled system.	Yes
GUK	Group unique symmetric key. It represents a randomly unique seed that may be shared with some group of manufactured CCA enabled systems with the same immutable hardware security properties.	Yes

[R0021] HUK is randomly generated at manufacture.

[R0022] HUK can only be accessed by CCA HES at any point in the security lifecycle of the system.

For example, HUK must not be collected at manufacture, or stored outside the system. HUK may be used, for example, as a seed for deriving root keys for sealing or secure storage use cases.

[R0023] GUK is at least randomly unique for some group of manufactured instances of a CCA enabled system, all with the same CCA security properties.

[R0024] GUK can only be accessed by CCA HES, other than temporarily at security provisioning.

For example, GUK can be stored temporarily outside the system for the purpose of manufacture provisioning, but must not be collected or stored persistently outside the system.

GUK may be used, for example, as a root seed for deriving anonymised keys.

Copyright © 2020-21 Arm Limited or its affiliates. All rights reserved.
Non-confidential

5.2 Derived parameters

Derived parameter is not stored persistently, but derived as part of the boot process. This table summarises the most important derived parameters for CCA.

[R0155] Private derived CCA parameters owned by CCA HES can only be directly accessed by CCA HES (see *Hardware enforced security*).

[R0156] Private derived CCA parameters owned by another CCA security domain cannot be directly accessed by any less trusted security domain.

Parameter	Description	Private	Ownership	Source
Derivation root keys for Monitor security domain	Virtual HUK/GUK, derived uniquely for Monitor security domain at boot.	Yes	Monitor security domain	PE initial boot, or CCA HES
Derivation root keys for Realm Management security domain	Virtual HUK/GUK, derived uniquely for Realm Management security domain at boot.	Yes	Realm Management security domain	Monitor security domain
CCA platform attestation key (CPAK)	Asymmetric attestation key for the CCA platform. Identifies the immutable security properties of the CCA system security domain.	Yes, private portion	CCA HES	Typically derived from GUK, or equivalent[1].
CPAK ID	Public identifier for CPAK	No	CCA HES	Hash of public part of CPAK Uniquely identifies the immutable hardware security properties of a CCA platform.

[1] Direct derivation of IAK is used in this document as a simple generic workflow for deriving a CCA initial attestation key, and as such it represents a minimal implementation.
More complex workflows including, for example, online activation flows, one time keys, and so on, can be supported if required. Such flows are ecosystem specific and not discussed in this document.

Copyright © 2020-21 Arm Limited or its affiliates. All rights reserved.
Non-confidential

6 CCA system security domain

The CCA system security domain represents all system hardware components that may directly affect the CCA security guarantee.

[R0026] CCA hardware in CCA system security domain can only be directly accessed by Monitor security domain.

6.1 On-chip memory

On a secured system, on-chip memory is any volatile memory location not accessible outside the system boundary.

[R0027] Ownership of on-chip memory is enforced at least along world isolation protection boundaries.

World ownership of on-chip memory may be, for example, statically assigned at boot, or dynamically assigned at runtime by Monitor security domain.

6.2 Isolated locations

An isolated location is a non-volatile storage location that on a secured system is not accessible outside the system boundary.

For example:

- An inseparable *secure element (SE)*
- An on-chip *one-time programmable (OTP)* memory device
- On-chip flash

6.3 Shielded locations

A shielded location is an isolated location designed and used for the purpose of storing secrets.

[R0028] A shielded location provides tamper resistant storage.

Examples of attacks to consider include:

- Passive and active probing
- Glitching
- Hardware sidechannel attacks, including power or frequency analysis
- Software sidechannel attacks
- Environmental attacks, including energy, radiation, and temperature

The degree of tamper resistance required is deployment and ecosystem specific and out of scope of this specification. Individual ecosystems may define more specific certification profiles.

6.4 Immutable initial boot code

The CCA boot process relies on a verified boot process, which starts with immutable initial boot code. See *CCA firmware boot*.

Immutable initial boot code is stored within the CCA system security domain using, for example, on-chip ROM, on-chip OTP, or locked on-chip flash.

Copyright © 2020-21 Arm Limited or its affiliates. All rights reserved.
Non-confidential

Combination storage can also be used, such as an on-chip ROM in the CCA system security domain combined with a hashlocked image from external storage.

6.5 Power management

6.5.1 System reset

The term *system reset* is used to describe a complete system reset, including any trusted subsystems.

[R0029] All CCA system security domain components are reset following a system reset and start in a fresh state.

This includes, for example, all trusted subsystems and memory protection hardware.

[R0030] Following system reset, all non-persistent CCA runtime state or data is either reset or rendered unusable.

Arm recommends that this is achieved by resetting root keys for protected memory, ensuring the system is in a fresh state and that any volatile data from before the reset cannot be retrieved by unauthorized agents. See *Protected memory*.

6.5.2 Software initiated system reset

Software initiated system reset is any action taken by an application PE to reset CCA firmware running in Monitor security domain.

[R0031] A software initiated system reset results in the same actions being taken as following a hardware initiated system reset.

Failing to reset all CCA system security domain may compromise the CCA security guarantee. For example, failure to reset memory encryption may expose the contents of memory previously allocated to a Realm. Restarting execution from PE initial Boot is not sufficient.

[R0032] Any software initiated system reset can only be initiated by Root world.

Realm world, Secure world, or Normal world cannot directly initiate a software initiated system reset. They must go through root world to initiate a system reset.

Realm world, secure world, or Normal world may perform a software initiated reset within their own domains.

6.5.3 System reset initiated by a trusted subsystem

[R0033] CCA HES host is the only trusted subsystem able to directly initiate a system reset.

6.5.4 System hibernation

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

System hibernation is not supported by the current release of CCA, and is only included here for reference.

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, boot code is then able to detect and restore the previously saved state as part of the boot process.

Copyright © 2020-21 Arm Limited or its affiliates. All rights reserved.
Non-confidential

[R0034] The runtime state of a system that has been hibernated and then restored is indistinguishable from if the hibernation event never took place.

[R0035] Any stored CCA and Realm state is protected, including privacy, integrity, and replay protection.

[R0036] The security logic for entering and exiting a system hibernation state can only be implemented within the Monitor domain.

Ephemeral state might not survive hibernation. For example, network connections may have been lost.

System hibernation typically requires on-chip trusted storage, at least for a small amount of root metadata.

General power management code, including the decision whether to hibernate or not, may be implemented outside the Monitor security domain. For example by Normal world or Secure world.

An example implementation of system hibernation may be based on the following conceptual flow:

1. Normal world system software pages out all memory allocated to Realms in Realm world using CCA protected paging (not defined in this document)
2. Normal world system software pages out all remaining memory allocated to RMM using CCA protected paging, leaving only a small hibernation state stored by RMM in a shielded location
3. Normal world system software pages out Monitor data structures using Monitor protected paging, leaving only a small hibernation state stored by Monitor in a shielded location
4. Monitor initiates power off

System restore follows in reverse order following system power-on and reset.

6.5.5 System suspend

The term *system suspend* is used to describe any low-power state in which the system has not been fully reset or power-cycled, but in which most resources have been suspended. In particular, protected external memory is maintained in system suspend.

System suspend is an optional feature. If supported, suspend typically involves:

- Suspend code ensuring orderly suspend of the current execution state
- 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.

[R0038] The runtime state of a system that has been suspended and then restored is indistinguishable from if the hibernation event never took place.

Ephemeral state might not survive system suspend. For example, network connections may have been lost.

[R0039] The suspend state is protected, as far as possible, including privacy, integrity, and replay protection.

System suspend typically relies on external memory protection to protect its suspend state and will typically only offer the same level of protection as that implemented by the system for external memory at runtime. See *Protected memory*.

[R0040] The security logic for entering and exiting a system suspend state can only be implemented within the Monitor domain.

General power management code, including the decision whether to suspend the system or not, may be implemented outside the Monitor security domain. For example, by Normal world or Secure world.

Copyright © 2020-21 Arm Limited or its affiliates. All rights reserved.
Non-confidential

6.6 Isolation hardware

Isolation hardware includes all hardware agents enforcing CCA protection boundaries. For example, MMU or MPU.

6.7 Trusted subsystems

A trusted subsystem is an isolated hardware subsystem with:

- A private execution environment not accessible to any other system agent
- Private resources not accessible to any other hardware or firmware agent on the system
- Its own Trusted subsystem firmware
- Its own boot process

[R0041] On a secured system, a trusted subsystem is isolated from all other hardware agents, including all invasive subsystems.

A trusted subsystem is responsible for its own boot process. A trusted subsystem running unauthorized firmware may compromise the CCA security guarantee.

[R0042] All trusted subsystems support a verified boot process, see *CCA firmware boot*.

The trustworthiness of a Realm depends on the trustworthiness of the CCA platform on the system the Realm is deployed on. This includes the trustworthiness of all trusted subsystems in the CCA system security domain.

[R0043] All trusted subsystems are attestable, see *CCA attestation*.

A trusted subsystem is responsible for its own security lifecycle management. As such a trusted subsystem may represent a root of trust in its own right.

[R0141] A trusted subsystem may delegate its security lifecycle management to another trusted subsystem.

For example, a power management subsystem may delegate its security lifecycle management to a CCA HES host.

[R0044] On a secured system, Arm recommends that all debug capabilities are permanently disabled on all trusted subsystems.

A trusted subsystem may be single tenant or multi-tenant. On a multi-tenant system more than one security service may be present.

[R0045] In the case of multi-tenancy, each tenant is isolated from all other tenants and is guaranteed its own private resources.

Examples of trusted subsystems may include:

- A dedicated security processor
- A power management controller, such as a *system control processor (SCP)*
- A *Secure Enclave (SEn)*

Copyright © 2020-21 Arm Limited or its affiliates. All rights reserved.
Non-confidential

6.8 Invasive subsystems

Invasive subsystems include any system feature or interface that might compromise the CCA security guarantee, such as:

- External debug interface
- Boundary scan interface
- RAS and other fault detection and recovery technologies
- Power management, including entering and exiting system suspend and system hibernation states

[R0046] CCA HES moderates all invasive subsystems and ensures the CCA security guarantee is not compromised.

For example, enabling external debug can only be done following a system reset and must be reflected in attestation. Similarly, a RAS error detection event must not compromise CCA protection boundaries.

Copyright © 2020-21 Arm Limited or its affiliates. All rights reserved.
Non-confidential

7 Protected memory

Many Realm and CCA assets are held in external memory.

This section discusses mitigations against memory based attacks.

7.1 General threat model

The following is a summary of general threats to be considered in relation to external memory.

Direct memory access: An unauthorized agent on the same system attempts to directly access the contents of memory allocated to a different world, or to a Realm.

Probing: An attacker attempts to physically access the contents of memory allocated to a world, or to a Realm. For example using hardware probes, or recording devices such as NVDIMM.

Replay: An attacker attempts to physically replay previously captured memory content. For example, to revert a Realm to an early known state, or to replay content captured from one Realm into memory allocated to a different Realm.

Leakage: An attacker gains access to the contents of memory allocated to a world or a Realm, for example through misconfigurations or errors in the implementation of the CCA platform.

Software based sidechannels: For example timing attacks, or row-hammer style indirect memory corruption attacks

7.2 Possible mitigations

The following table enumerates possible mitigations that may be deployed to address some or all of the main classes of memory attacks and their associated essential properties.

Mitigation	Properties	Mitigates against	Information
Boot freshness	Memory encryption is randomly re-seeded on boot.	Content allocated to Realms being available following a system reset.	
Location freshness	Memory encryption is unique per memory location,	The same content plaintext resulting in the same ciphertext for all memory locations.	For example, including location as part of an encryption tweak or IV.
CCA isolation	Enforces access control boundaries for memory allocated to worlds, and to Realms within Realm world.	**Direct memory access** between worlds, and between Realms in Realm world.	Assuming correct configuration.
Physical protection	External memory has a degree of tamper resistance.	**Probing** – read, modify **Replay**	For example, in-package memory.

Copyright © 2020-21 Arm Limited or its affiliates. All rights reserved.
Non-confidential

Mitigation	Properties	Mitigates against	Information
World unique encryption	Memory is uniquely encrypted per world and location.	**Probing** – read **Leakage** across world boundaries **Replay** of contents captured from one world into a different world	Memory contents remain protected even if unauthorized access is gained across a world boundary.
Realm unique encryption	Memory is uniquely encrypted per Realm and per location.	**Probing** – read **Leakage** across Realm boundaries **Replay** of contents captured from one Realm into a different Realm	May affect cache performance.
Realm controlled encryption	Memory allocated to a Realm is encrypted using keys supplied by the Realm.	Same as Realm unique encryption.	Can support memory mapped storage, such as storage class memory. Including across Realm restarts.
Integrity protection	Memory contents are integrity checked.	**Probing** – modify **Software based side-channels** – modify	Memory contents cannot be modified undetected. Affects system performance. Introduces a memory overhead for storing metadata.
Integrity protection with temporal freshness	Adds temporal freshness to integrity protection.	**Probing** – modify **Replay** – any **Software based side-channels** – modify	Requires integrity protection. Increases memory overhead.
Encryption with temporal freshness	Adds temporal freshness to world or Realm unique encryption.	**Probing** – read, modify **Leakage** – no additional protection **Replay** - any	Each memory write adds unique freshness to the encryption, such as a nonce counter value. Requires similar overhead as integrity and replay protection.

7.3 Use of external memory by CCA

A trusted subsystem is a private on-chip execution environment. It must be able to execute independently from other CCA components. Trusted subsystems provide essential security services and need the most protection for both their code and data. They typically also need to be fully operational early in the boot process when only on-chip memory is available.

[R0146] Trusted subsystems, including CCA HES host, can only use private on-chip memory and do not use external memory.

Copyright © 2020-21 Arm Limited or its affiliates. All rights reserved.
Non-confidential

Root world firmware, including Monitor, is the most trusted CCA component on application PE. It enforces CCA security guarantees for not just Realm world, but also for Secure world and for itself.

It is expected to be small enough to feasibly fit in on-chip memory, and typically needs to be available early in the boot process when only on-chip memory is available.

Monitor manages a *granule protection table (GPT)* data structure to enforce world isolation security guarantees (see [RME]). This data structure is too large to fit in on-chip memory, so use of external memory is required. Because this structure is critical to CCA, additional hardware enforced error detection protection is provided regardless of external memory encryption (see [RME]).

[R0147] Monitor code executes entirely from on-chip memory.

[R0148] External memory used for GPT has hardware enforced error detection (see [RME]) in addition to external memory encryption.

It is expected that Monitor data structures that affect the CCA security guarantee, other than GPT, can feasibly use on-chip memory.

Some large or unbounded EL3 structures, such as *scalable vector extension (SVE)* structures, may require external memory. For any such structures that may affect the CCA security guarantee Monitor should implement additional integrity protection. For example, storing a hash in on-chip memory.

[R0149] Any monitor data that may affect the CCA security guarantee, other than GPT, is either held in on-chip memory, or in external memory but with additional integrity protection.

CCA firmware in Realm world – including *Realm Management Monitor (RMM)* – is expected to use external memory for both code and data.

[R0150] External memory used by RMM is protected to at least the same level as the maximum protection offered to Realms.

Implementations may choose to go further. However, from a CCA isolation threat model point of view, RMM only needs to be protected at least as well as the most protected Realm.

[R0151] RMM firmware is expected to employ appropriate defensive programming techniques to minimise exposure to side-channel and memory corruption attacks.

Without integrity protected external memory, both RMM and Realms may be exposed to memory corruption attacks. An example is Rowhammer.

Additional sidechannel attacks to consider for all CCA firmware include, for example, timing attacks.

Where feasible, appropriate defensive programming techniques should be employed as mitigations.

7.4 External memory initialization

External memory initialization is hardware dependent and in general out of scope of this specification and of CCA.

For example, external memory may be initialized by a power management trusted subsystem early in the boot process. Or it can be initialized by secure world firmware later in the boot process. Or in some cases, external memory may be discovered and initialized dynamically at runtime.

[R0152] Monitor can claim a section of external memory at boot for holding GPT.

[R0153] External memory cannot be allocated to Realm world or Secure world until the GPT has been initialized by Monitor.

Copyright © 2020-21 Arm Limited or its affiliates. All rights reserved.
Non-confidential

The exact mechanism for claiming memory for holding GPT is implementation defined, but once claimed the memory is locked by monitor to Root world and so protected by CCA isolation from then on.

External memory protection, and other hardware configurations such as clock and power, should be kept within stable bounds. Where any such configurations are accessible to non-CCA firmware, protections should be put in place to prevent the system to be configured in an unstable state that might be exploited to bypass CCA security guarantees.

[R0154] It is only possible to configure hardware to a stable state.

For example, such configurations may be made available to non-CCA firmware via a trusted subsystem that can enforce policy on chosen values. Or such configurations can be restricted to Monitor (EL3) only, and then exposed through Monitor mediated interfaces to non-CCA firmware.

7.5 Assets

The following table enumerates CCA assets, and details that require external memory protection.

Component	Asset	Description	Used from	Protected by
Trusted subsystem	Initial boot code	First code executing following reset.	Private on-chip ROM or Locked on-chip memory	Hardware isolation. Physical protection.
	Trusted subsystem firmware	Trusted subsystem firmware loaded at boot.	Private on-chip memory	
	CCA hardware provisioned parameters	CCA hardware provisioned seeds, keys and identities.	Private storage or Private on-chip memory	
	Runtime assets	Examples: CCA root seeds and secrets. CCA initial boot state.	Private on-chip memory	
Application PE initial boot	Initial boot code and data	Examples: Verification or decryption keys.	Private on-chip ROM or locked on-chip memory	Hardware isolation. Physical protection
Monitor	Runtime code and data		On-chip memory	Hardware isolation. Physical protection.
	Granule protection table (GPT)	See [RME].	External memory	GPT error detection (enforced by MMU). Memory protection.
Realm Management	Runtime code and data		External memory	Memory protection.
	RMM translation tables	See [RMM].	External memory	
	Realm boot state	Not defined in this document.	External memory	

Copyright © 2020-21 Arm Limited or its affiliates. All rights reserved.
Non-confidential

Component	Asset	Description	Used from	Protected by
	Runtime code and data		External memory	
Realm	Runtime code and data.		External memory	Memory protection
	Persistent Realm assets		External storage	

7.6 Baseline memory protection profile

This section defines the minimum required memory protection for the current version of CCA. It is targeted at deployments where physical attack on host hardware is deemed out of scope of the deployment threat model.

[R0047] World unique encryption with boot freshness and location freshness is provided.

This meets the threat model as follows:

Threat	Tamper resistant external memory	External memory without tamper resistance
Direct memory access	CCA isolation	CCA isolation
Probing	Yes	Yes
Replay	Yes	No
Leakage	Across World boundaries	Across World boundaries
Software-based sidechannels	No	No

7.7 Memory scrubbing

In addition to encryption, memory scrubbing is also required to ensure confidentiality of Realm data.

[R0125] Memory re-allocated from one Realm to a different Realm is always be scrubbed.

Without either Realm unique encryption, or encryption with temporal freshness, data can leak from one Realm to another if the memory content is not scrubbed.

[R0126] Arm recommends that memory is scrubbed when it is re-allocated from Normal world to Realm world.

Scrubbing in this case prevents normal world from choosing values used to populate memory locations allocated to a Realm.

In the absence of either Realm unique encryption, or encryption with temporal freshness, the hosting environment can, for a given location, choose pairs of {[value written before re-allocation], [value seen by a Realm after allocation]}. It can do this regardless of world unique encryption.

[R0127] Arm recommends that memory is scrubbed when it is re-allocated from Realm world to Normal world.

In this case, direct data leak is prevented by world unique encryption. Normal world can only see scrambled data.

Copyright © 2020-21 Arm Limited or its affiliates. All rights reserved.
Non-confidential

But without temporal freshness in the encryption then, for a given location, the same plaintext will always result in the same ciphertext.

[R0128] Arm recommends that scrubbing is implemented by either writing random data, or writing unique data sourced from a monotonic counter, or equivalent.

Scrubbing can be lazy. For example, only performed immediately before a page is first accessed by a recipient of memory.

7.8 Additional memory protection

Implementations may implement stronger memory protection, if required.

Arm may define additional memory protection profiles in later releases of CCA covering other deployment threat models and use cases.

For example, Realm unique encryption adds protection against leakage across Realm boundaries. It also protects against replay of content captured from one Realm into a different Realm.

Realm controlled encryption, a variation on Realm unique encryption, can support memory mapped storage use cases, such as storage class memory. It can also support sharing memory mapped storage between multiple instances of a Realm to Realm Owner policy.

Integrity protection with temporal freshness prevents all forms of modification and replay of memory content, including capture and replay of Realm Management memory content, as well memory corruption.

All of the above methods directly impact system performance. Integrity protection and temporal freshness also introduces memory overhead for storing memory protection state.

Copyright © 2020-21 Arm Limited or its affiliates. All rights reserved.

Non-confidential

8 CCA firmware boot

This section defines requirements and a generic flow for booting CCA firmware into an attestable state.

Requirements for booting secure world firmware, and normal world firmware and software, are out of scope of this specification.

CCA firmware includes:

- Application PE firmware for Monitor security domain and Realm Management security domain
- CCA HES firmware, and trusted subsystem firmware for the CCA HES host
- Trusted subsystem firmware for all other trusted subsystems within CCA system security domain

Implementations of the CCA platform should comply with [Boot PSG].

8.1 Verified boot

Loading unauthorized CCA firmware can compromise the CCA security guarantee.

[R0048] A secured system can only load authorized CCA firmware.

Verified boot is rooted in initial boot code. This applies to each trusted subsystem in the CCA system security domain, as well as to application PE boot.

[R0049] On application PE, and on each trusted subsystem, the only code running immediately after release from reset is boot code.

[R0157] At least CCA HES host initial boot code is immutable on a secured system.

[R0158] Arm recommends that all initial boot code is immutable on a secured system.

[R0159] Initial boot code that is not immutable is always measured by CCA HES host, and included in CCA platform attestation

Immutable here means that, on a secured system, executing initial boot code always results in the same actions taken, and always leaves the system in a defined state.

Immutable initial boot code forms the start of the trust chain for the CCA platform. It is trusted implicitly and as such is not measured and attested directly. Instead it is considered part of the CCA system security domain, and the security properties of immutable initial boot code can be identified completely by the CCA platform attestation identity. See *CCA attestation*.

Making all initial boot code immutable maintains separation of boot paths and supply chains for different components of the CCA platform, such as trusted subsystems and application PE, and can simplify the verification process.

Depending on ecosystem requirements, updates of some initial boot code may be required. For example, to allow migration of a system to future security algorithms. In this case, those initial boot code components are not immutable and must instead be measured and verified at boot, and declared as part of CCA platform attestation. This model provides more flexibility for future system updates, but may create more dependencies and complexity in the supply chain and in the attestation verification chain.

Regardless, initial boot code for at least the CCA HES host must always be immutable on a secured system. It is the root for either guaranteeing the immutability of, or measuring and attesting, initial boot code for other components of the CCA platform.

Initial boot code can be stored and instantiated in different ways. For example, initial boot code may be held in on-chip ROM or locked on-chip storage. Or it may be held partially in ROM, and partially in a hash locked image in external storage.

Copyright © 2020-21 Arm Limited or its affiliates. All rights reserved.
Non-confidential

Another implementation example may see the CCA HES host boot from its own private ROM. It may then instantiate initial boot code for other trusted subsystems, as well as for application PE. For example, by loading hash locked (or measured, verified and attested) images into on-chip memory before releasing trusted subsystems and application PE from reset.

[R0050] If all or part of initial boot code is instantiated in on-chip memory then other trusted subsystems or application PE cannot modify that code before it has been executed.

For example, on-chip memory holding initial boot code for a trusted subsystem, or for application PE, may be write protected until it has been executed.

Each measured and attested CCA firmware image must be associated with firmware identity metadata, and a firmware payload.

Signed identity metadata constitutes an authorization to load and execute the firmware on a CCA system, and must be verified as issued by a trusted source. For example, immutable initial boot code can verify a signer against a hardware provisioned boot verification key. Later stage firmware can verify a signer against embedded verification keys.

[R0051] CCA firmware identity metadata is cryptographically bound to an associated CCA firmware payload.

Typically a hashlock based on a payload measurement.

[R0052] CCA firmware identity metadata is signed by a *firmware signer*

[R0053] Signed firmware identity metadata includes at least expected measurement of the CCA firmware payload, firmware version information, and a signer identity.

[R0054] A CCA firmware payload can only be executed if the firmware identity metadata can be verified as authorized by a trusted source.

Firmware identity metadata may be delivered as part of a single firmware image together with the firmware payload, or it may be delivered as a separate firmware authorization as part of a firmware update protocol.

Signature verification can take place on every boot. It can also take place when a firmware update is first installed and executed, if the image is then hashlocked locally for subsequent boot events.

The expected measurement is used to verify an actual firmware payload measurement, calculated at boot.

[R0055] An image can only be loaded if its actual measurement, as calculated at boot, is the same as its expected measurement as defined in the signed firmware identity metadata.

Version information is included in attestation. It can also be used for firmware anti-rollback.

The signer identity is included in attestation, and may be used by the boot process to identify an appropriate verification key for an image.

Version information and signer identity may also be used in CCA key derivations. For example, to support local secure storage implementations within the CCA platform itself, or to support local secure storage within Realms. CCA derived keys can also be used for sharing Realm assets across multiple Realm instances, to Realm Owner policy. Such use cases are out of scope of the current version of this document but may be considered in future versions.

Copyright © 2020-21 Arm Limited or its affiliates. All rights reserved.
Non-confidential

CCA firmware images and signed identity metadata loaded from off-chip storage require additional verification. Verifying and executing an image in protected memory rather than external storage prevents substitution attacks. Verifying size and copying to a known size buffer prevents overflow attacks.

[R0056] CCA firmware is never executed from off-chip storage, only from on-chip memory or protected external memory (see *Protected memory*).

[R0057] Signed CCA firmware identity metadata is not verified in off-chip storage, only in on-chip memory or protected external memory (see *Protected memory*).

[R0058] The amount of memory required for loading signed CCA firmware identity metadata must be known and verified before the identity metadata is copied into on-chip memory or protected external memory (see *Protected memory*).

[R0059] The amount of memory required for the image payload must be known and verified before it is copied into on-chip memory or protected external memory (see *Protected memory*).

[R0060] The required memory is allocated and available before data is copied from external storage.

For example, the signed firmware identity data and the firmware payload may be combined into a single image of a fixed maximum size. An implementation can then ensure that a fixed buffer of at least that size is always available before copying.

Alternatively, the signed firmware identity metadata may be fixed size and include the size of the payload. The identity metadata can then be copied into a fixed buffer and verified there, and then a second dynamic buffer can be allocated based on a verified payload size.

[R0131] Data from external storage cannot be copied beyond the bounds of a receiving buffer.

This prevents buffer overflow attacks.

[R0132] It is not possible to copy data that is never verified from external storage to a receiving buffer.

This prevents malicious insertion of “gadgets” into trusted memory, which might be exploited by later code.

For example, all copied firmware identity metadata should always be verified against a signature. All copied firmware payload data should always be included in the payload measurement calculation and verified against signed firmware identity metadata.

[R0133] If any part of the copy process or the verification process fails then all copied data is erased from receiving buffers.

This too prevents malicious insertion of “gadgets” into trusted memory, which might be exploited by later code.

8.2 Image formats and signing schemes

Recommended algorithms and key sizes for image signing can be found in *Cryptographic recommendations*.

Recommended signing schemes and image formats can be found in [Boot PSG], and in Arm *Trusted Firmware (TF-A)* reference code.

8.3 Anti-rollback

Rolling back CCA firmware to some earlier version with known vulnerabilities may compromise the CCA security guarantee.

[R0061] Arm recommends that anti-rollback is implemented in the CCA boot process.

In general, anti-rollback consists of:

Copyright © 2020-21 Arm Limited or its affiliates. All rights reserved.
Non-confidential

- One or more monotonic counters tracking approved versions of CCA firmware
- A policy for when to update counters following a CCA firmware update
- A recovery policy

For example, an implementation may choose to update counters following a certain number of successful boot events, based on check points and watchdog timers in the boot process. Alternatively, an implementation may only update counters following explicit signalling from an update service.

In either case, the implementation may allow recovery by fallback to a last known good version before then. Alternatively, an implementation may only allow recovery by fallback to a previously installed version following an explicit roll-back authorization.

Policies are out of scope of this specification.

8.4 Off-line boot

[R0062] CCA can be booted on a system without communicating with remote systems.

For example, much of the early CCA boot process takes place before network communications may be possible. Alternatively, on devices where network connections cannot be guaranteed, such as mobile clients.

[R0063] Implementations of CCA cache firmware identity metadata following firmware updates.

[R0064] Caching firmware identity metadata cannot compromise boot guarantees, such as anti-rollback protection.

[R0065] Cached firmware identity metadata can only be stored in isolated locations.

Cached firmware identity metadata allows a system to boot without a network connection.

8.5 CCA HES firmware boot flow

CCA HES represents a set of core CCA services hosted on a trusted subsystem, a CCA HES host, off application PE.

A trusted subsystem is an isolated execution environment, which must implement its own verified boot process, independent of application PE and any other hardware agents.

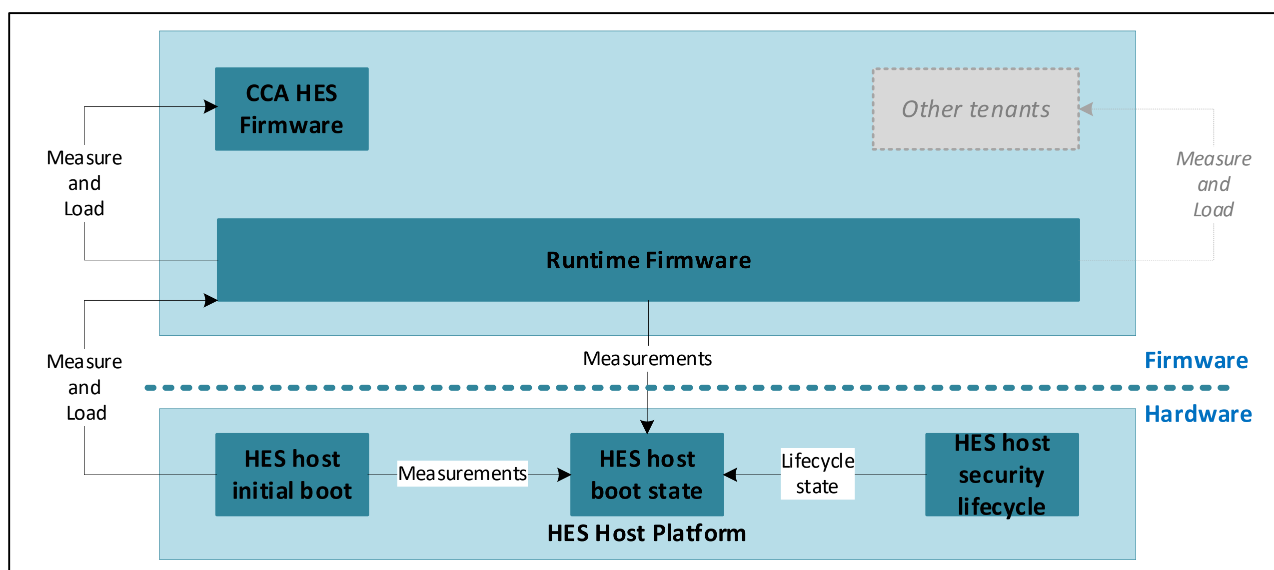

Figure 6. Conceptual CCA HES firmware boot flow

Copyright © 2020-21 Arm Limited or its affiliates. All rights reserved.
Non-confidential

A conceptual boot flow for a trusted subsystem hosting CCA HES services is outlined in *Figure 6. Conceptual CCA HES firmware boot* flow. It is intended to illustrate the general security properties, rather than any particular implementation of such a flow.

The CCA HES host is a trusted subsystem.

The CCA HES platform represents all hardware elements private to the CCA HES host.

Runtime firmware represents local generic services such as crypto, scheduling, storage, and local isolation.

CCA HES firmware represents all CCA specific firmware, providing CCA HES services.

Other tenants represent any non-CCA tenants co-hosted with CCA HES on a multi-tenant subsystem.

[R0066] On a secured system, only verified Trusted subsystem firmware can be loaded on a trusted subsystem.

[R0067] All Trusted subsystem firmware loaded is measured and verified as outlined in *CCA firmware boot*.

[R0068] Measurements are tracked by hash registers with lock and extend semantics, enforced by the CCA HES host.

[R0069] In the case of a multitenant host, Arm recommends that CCA measurements are tracked and stored separately from measurements of other tenants.

[R0070] The result of the boot process is a Trusted subsystem boot state, including all actual measurements taken during the boot process, as well as associated firmware metadata as outlined in *CCA firmware boot*.

Arm recommends that hardware hash extend registers are used to track measurements taken during the boot process of the CCA HES host and CCA HES firmware. Firmware registers implemented by CCA HES host firmware can be used to track other measurements.

8.6 CCA system security domain boot process

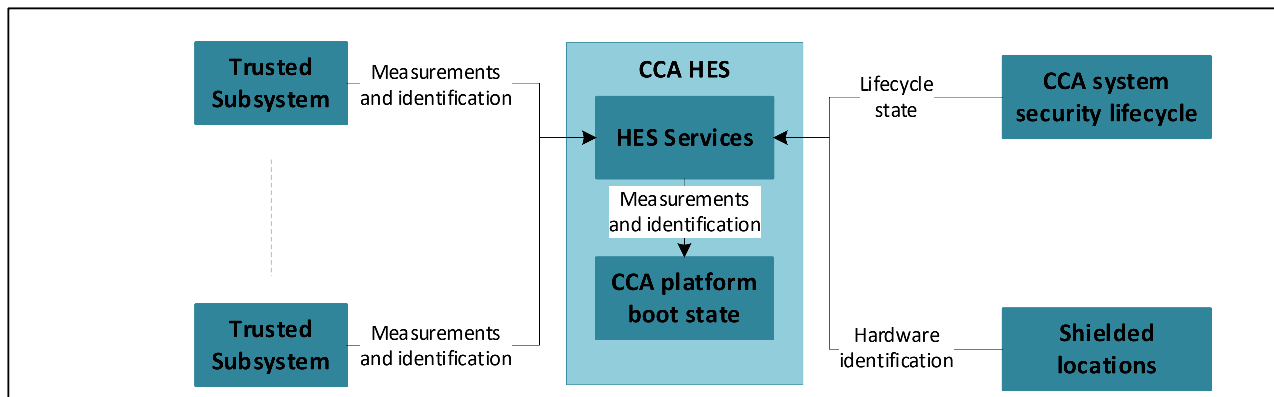

Figure 7. Conceptual CCA system security domain boot process

Following system reset, the CCA system security domain will be measured as outlined in *Figure 7. Conceptual CCA system security domain boot process*. This is a conceptual flow to illustrate how the security properties of the CCA system security domain are established, rather than any actual implementation or actual boot sequence.

CCA HES is responsible for collating a CCA hardware boot state.

[R0071] The result of the CCA system boot process is a *CCA System Boot State* including: measurements of trusted subsystem firmware for each trusted subsystem, including CCA HES host firmware, and associated firmware identity metadata as outlined in *CCA firmware boot*.

Copyright © 2020-21 Arm Limited or its affiliates. All rights reserved.
Non-confidential

[R0072] Each trusted subsystem implements its own verified boot flow following the general principles outlined in *CCA firmware boot*.

[R0130] CCA HES provides a service to track and store measurements taken by other trusted subsystems following system reset.

[R0073] Measurements are tracked by hash registers with lock and extend semantics, enforced by CCA HES.

For measurements taken by other trusted subsystems, CCA HES may implement software managed hash extend registers.

8.7 Application PE boot process

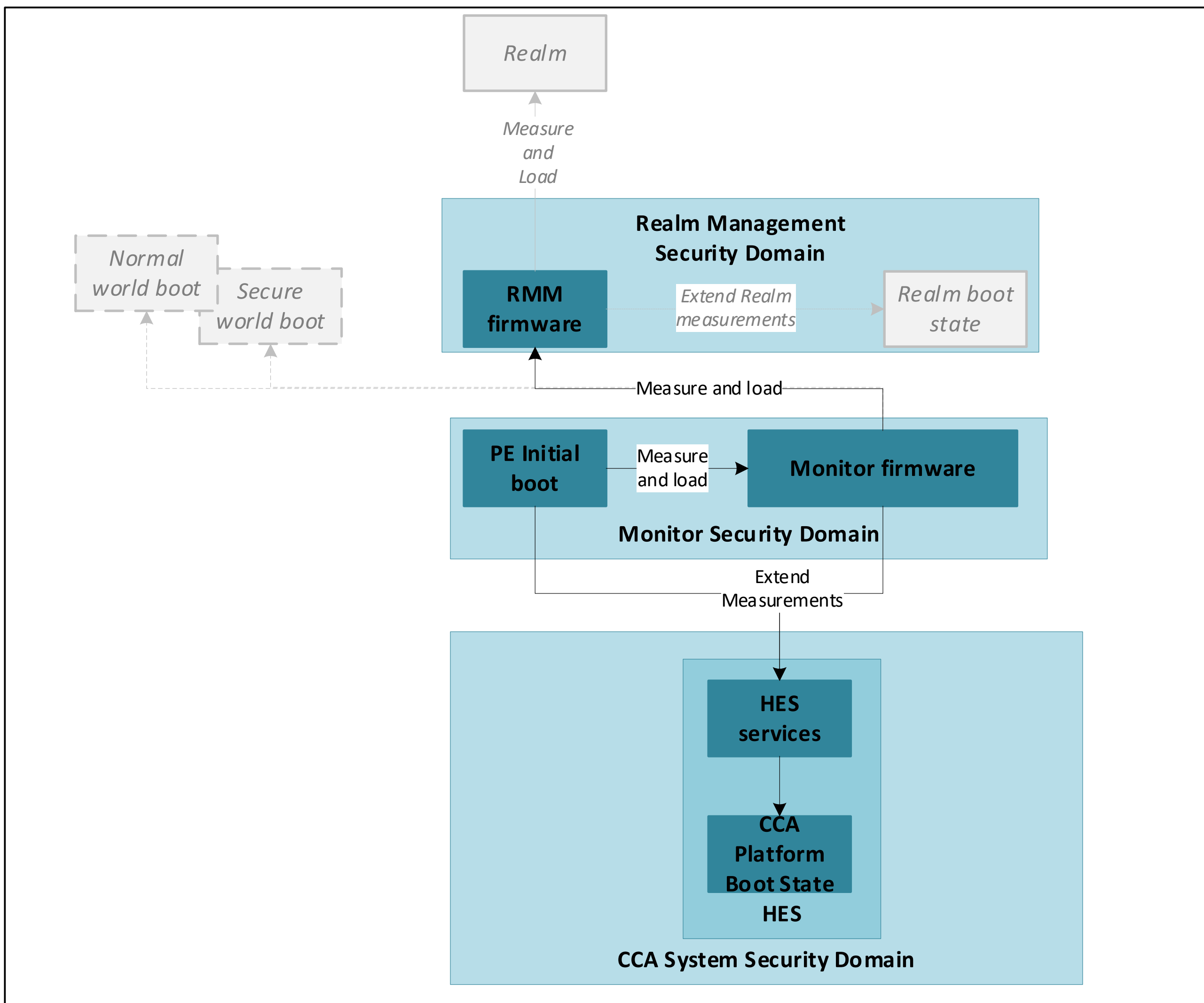

Figure 8. Application PE conceptual boot process

A conceptual application PE boot process is outlined in *Figure 8. Application PE conceptual boot process.* It is intended to illustrate the general security properties of the application PE boot process, rather than any particular implementation.

[R0074] On a secured system, only verified CCA firmware can be loaded on an application PE.

Copyright © 2020-21 Arm Limited or its affiliates. All rights reserved.
Non-confidential

The CCA system security domain includes immutable PE initial boot code.

[R0075] PE initial boot code is immutable code executing on application PE immediately following system reset.

[R0076] Only PE initial boot code can execute on application PE immediately following system reset.

The PE initial boot code is responsible for initial system configuration, and for loading Monitor firmware in Root world.

[R0077] All Monitor firmware in Root world is loaded in on-chip memory only.

[R0078] Once monitor firmware has been initialized then firmware for other worlds may be loaded.

Loading Secure world and Normal world is out of scope of this specification.

It is expected that some system initialization may be performed by Secure world or Normal world, such as external memory initialization. This may require checkpoints or interaction between Monitor and either Secure world or Normal world. Such interactions are out of scope of this specification.

[R0079] All Monitor firmware loaded by PE initial boot is measured and verified as outlined in *Verified boot*.

[R0080] CCA HES provides a service to track and store measurements taken by PE initial boot.

Monitor security domain is responsible for loading and measuring Realm management firmware.

[R0081] Realm management firmware can be loaded in protected external memory.

[R0082] All Realm management firmware is measured and verified as outlined in *Verified boot*.

[R0083] CCA HES provides a service to track and store Realm management firmware measurements taken by Monitor.

[R0084] Measurements are tracked by hash registers with lock and extend semantics, enforced by CCA HES.

[R0085] The result of the boot process is a CCA platform boot state, including all actual measurements affecting the CCA security guarantee, taken during the application PE boot process by PE initial boot and Monitor, as well as associated CCA firmware identity metadata as outlined in *Verified boot*.

For measurements taken by application PE firmware, CCA HES may implement software managed hash extend registers.

8.8 Robustness

[R0086] All aspects of the CCA boot process must be robust against hardware and software attacks designed to circumvent the boot process, which might allow an attacker to run unauthorized code or to modify established boot states.

Examples include resistance against power and timing attacks (“glitching” or clock attacks), and resistance against attacks designed to reveal secrets such as image decryption keys (if used). The latter may include, for example, active or passive probing, or sidechannels such as power and timing analysis.

The degree of robustness required is deployment and ecosystem specific and out of scope of this specification. Individual ecosystems may define more specific protection profiles.

Copyright © 2020-21 Arm Limited or its affiliates. All rights reserved.
Non-confidential

9 CCA attestation

CCA attestation allows a user of a service provided by a Realm – a *reliant party* – to determine the trustworthiness of the Realm, and of the implementation of the CCA platform .

A reliant party can be

- a remote client – *remote attestation*
- a local client on the same system – *local attestation*

Most of CCA attestation is defined in terms of remote attestation.

Local attestation follows the same basic process and flow but differs in how verification is performed, and in what security guarantees can be provided.

The desired result of successful attestation is typically a secure point to point connection between an attested end point in the Realm, and the reliant party. A Realm can implement more than one attested end point.

An attestation protocol can be, for example, an SSL-style negotiated session or a Diffie-Helman key negotiation.

Attestation protocols are out of scope of CCA. CCA attestation is designed to enable Realms to implement any attestation protocol.

CCA attestation is a token based model that binds metadata specific to the attested end point to the attested boot state of a Realm, and the attested boot state of the underlying implementation of the CCA platform.

Copyright © 2020-21 Arm Limited or its affiliates. All rights reserved.
Non-confidential

9.1 Base attestation flow

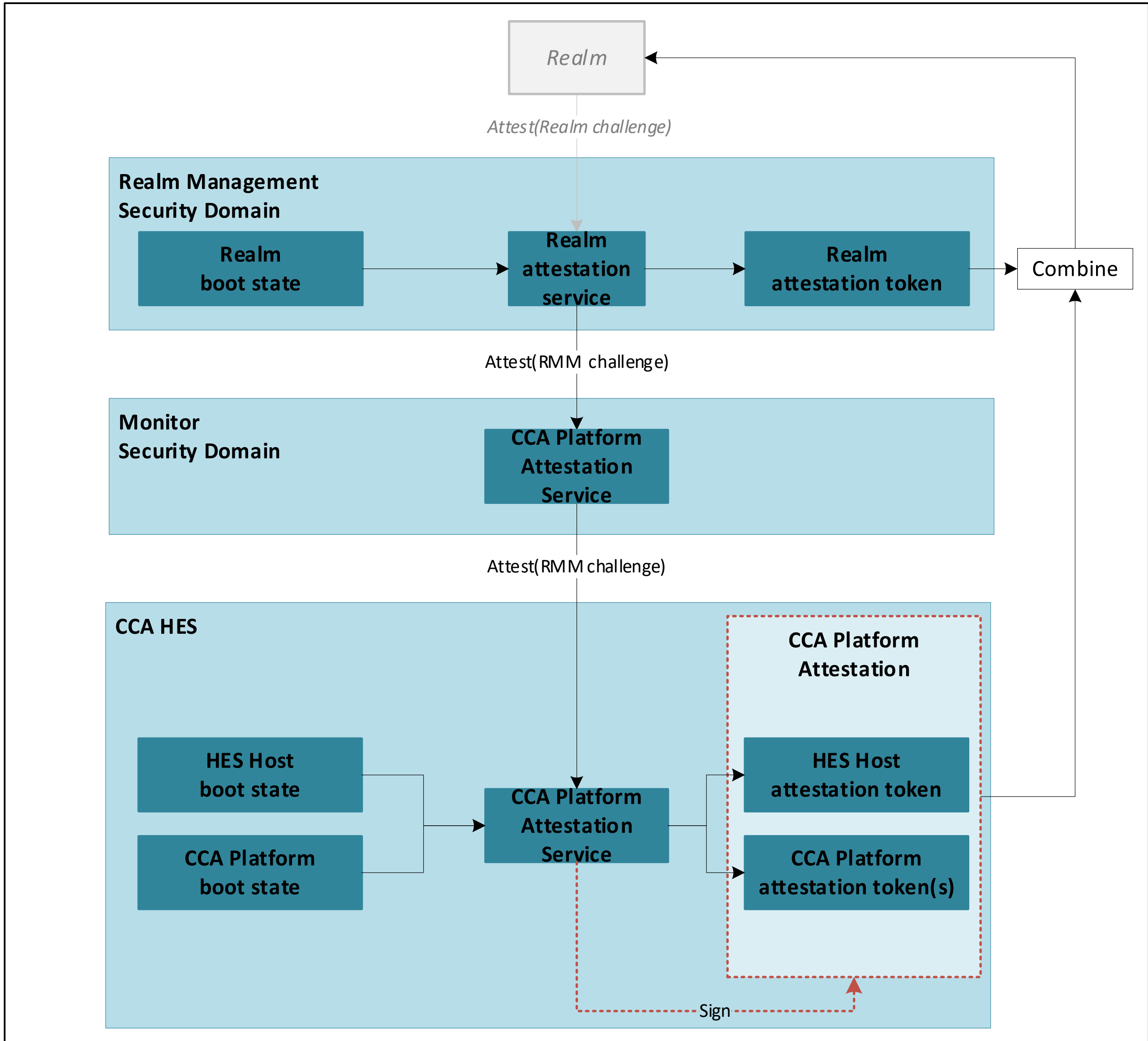

Figure 9. Basic CCA attestation flow

The basic CCA attestation flow is outlined in *Figure 9. Basic CCA attestation flow*.

A Realm requests a Realm attestation from Realm Managed security domain. The request includes a Realm challenge. For example, a hash of protocol metadata.

Realm Management security domain is responsible for maintaining a Realm boot state for each Realm. A Realm boot state defines content measurements and security configurations for a Realm. It is not defined in this document.

[R0087] The Realm boot state is combined with the Realm challenge into a Realm attestation token.

This creates a cryptographic binding (hash lock) of the service specific metadata and the Realm attestation.

Realm Management security domain then requests a CCA platform attestation. The request includes an RMM challenge.

Copyright © 2020-21 Arm Limited or its affiliates. All rights reserved.
Non-confidential

[R0088] The RMM challenge is calculated as a hash of the Realm attestation token.

In a CCA HES enabled implementation of CCA, the attestation request is passed through Monitor security domain to CCA HES.

CCA HES holds boot state for the CCA HES host itself, for the CCA system security domain (including all trusted subsystems), and for CCA application PE firmware.

[R0089] The CCA platform boot state is combined with the RMM token challenge into a CCA platform attestation.

[R0090] The CCA platform attestation is a collection of attestation tokens for the CCA HES host, for each trusted subsystem in the CCA system security domain, for the immutable implementation of CCA system security domain, and for CCA application PE firmware for Monitor and Realm Management security domains respectively.

[R0129] Each token carries its own set of measurements as established during the boot process.

[R0091] The CCA platform attestation is signed by the *CCA platform attestation key (CPAK)* (see *CCA identity management*)

This creates a cryptographic binding (hash lock) of the Realm attestation token to the CCA platform attestation.

9.2 Token formats and signing schemes

[R0092] All attestation tokens are instances of a PSA attestation token.

A PSA attestation token is a standard compact format based on IETF *Entity Attestation Token (EAT)* and *Remote Attestation Procedures (RATS)*. See [PSA Token].

[R0093] Remote attestation requires an asymmetric signing scheme.

For general recommendations about algorithms and key sizes, see *Cryptographic recommendations*.

9.3 Privacy preserving attestation

Privacy preserving attestation in this context relates to preventing tracking or inference, such as:

- Use a Realm attestation to link a Realm (or a device user) to a device unique identifier
- Use Realm attestation to infer that two Realm instances are on the same physical device

[R0025] Arm recommends that as a minimum IAK is derived or provisioned uniquely for a group of implementations with the same security properties, rather than for unique instance.

Other more complex schemes may be deployed, such as provisioned or derived one-time CPAK.

Such schemes are ecosystem specific and out of scope of this specification.

9.4 Delegated Realm attestation

Delegated attestation is out of scope of the current version of CCA and only mentioned here for completeness.

The basic attestation model described above requires a full roundtrip from Realm to CCA HES on every Realm attestation requests.

In deployments with a high frequency of Realm attestation requests, such as a datacenter deployment with frequent incoming client connections, this model may not scale.

Copyright © 2020-21 Arm Limited or its affiliates. All rights reserved.
Non-confidential

As an alternative, the Realm Management security domain can derive one or more *Realm attestation key (RAK)* at boot. RAK can also be attested at boot – key attestation – binding a hash of the RAK public key to the CCA platform attestation.

This allows the CCA platform attestation to be cached at the Realm Management security domain level. Subsequent Realm attestation requests can now be signed directly at the Realm Management security domain level without requiring a roundtrip to CCA HES.

To maintain privacy preservation properties, RAK derivation should follow a similar pattern to CPAK derivation. See *Privacy preserving attestation*.

9.5 Local attestation

Local attestation is out of scope of the current version of CCA and only mentioned here for completeness.

Local attestation is one element of supporting off-line use cases where the reliant party is located on the same system as the attested end point, and the reliant party might not itself have access to a remote verification service.

An example pattern may be as follows:

1. An interactive application is deployed in a Realm
2. The application makes use of third party media processing IP, itself deployed as a local service in a different Realm
3. When the application is first installed it goes through an activation process, including remote attestation of both the application Realm and the Realm providing the third party service
4. If successful, the application is provisioned with metadata, allowing it to operate locally without external interaction (off-line use) as long as the attested state of itself, the third party service, and the CCA platform do not change

To completely support this kind of use case requires:

- Local attestation
- CCA derived Realm binding keys for local protection of persistent Realm assets – not defined in this document

CCA derived binding keys guarantee that a Realm can only access persistent data if both the Realm itself and the underlying implementation of the CCA platform are in a trustworthy state.

Local attestation itself provides a reduced attestation guarantee. It proves that a pair or a group of Realms were instantiated on the same system, and in the same security state – i.e. it guarantees the original pairing or grouping of Realms from the activation process. But local attestation in itself provides no guarantee about the trustworthiness of the underlying implementation of the CCA platform.

Local attestation follows the same basic flow as remote attestation, but with the following extensions:

- The attestation tokens are signed with a local attestation key which can only be verified locally on the same system, for example a symmetric key derived at boot
- A CCA service is provided for verification of local attestations

A local attestation typically uses symmetric signing rather than asymmetric.

Copyright © 2020-21 Arm Limited or its affiliates. All rights reserved.
Non-confidential

10 CCA firmware updates

10.1 Remote update

Arm welcomes feedback on firmware update requirements for CCA.

In general, a CCA firmware update process can be described as:

1. A CCA firmware update client communicates with a remote update service using a firmware update protocol.
2. The CCA firmware update client triggers a copy of downloaded CCA firmware to a temporary staging location
3. A reset occurs, and the update is used as part of the boot process

[R0134] Arm recommends that firmware update protocol metadata is signed and verified, independently of firmware signing and firmware verification.

Choice of firmware update protocol is out of scope of CCA. For example, a firmware update protocol may include security features such as targeting of updates, and anti-rollback counter controls. Those features may include protocol level firmware update control messages, in addition to a firmware payload.

A firmware update client should verify that a CCA firmware update, or a CCA firmware update control message, comes from an authorized update source. Source in this case is typically different from an image signer. For example, an image may be signed by a CCA firmware distributor, whereas the update process may be controlled separately by a service provider or a hosting provider.

An update is only effected after reset. The scope of the reset may vary depending on the scope of an update.

Example CCA firmware update scopes include:

Root update: Affects Monitor security domain, or a trusted subsystem in the CCA system security domain

Realm world update: Only affects Realm world

RMM update: Only affects a CCA component within Realm world

In this document, the term *performing an update* is used to describe the process of verifying, installing and executing an update after it has been made available to CCA firmware by a firmware update client. Updates affect the trustworthiness of CCA, and it is the responsibility of CCA firmware to ensure that only authorized updates are performed.

Firmware updates can only be performed by a more trusted security domain, or by the most trusted component within a security domain.

[R0142] A Root update can only be performed by the CCA HES host.

[R0143] A Realm world update can be performed by Monitor.

[R0144] An RMM update can be performed by RMM.

A Root update requires a full system reset. A Realm world update only requires a reset of Realm world.

An RMM update may only require a reset of RMM and not a full reset of all Realms. Such live updates of Realm world are out of scope of the current CCA release but may be addressed in later releases.

[R0094] At least the validity of an update is always verified as part of performing an update.

Depending on ecosystem requirements, a valid update may be any update that is signed correctly and meets anti-rollback policy. Or a system may require explicit update authorization through a secure firmware update protocol before an update can be performed.

Copyright © 2020-21 Arm Limited or its affiliates. All rights reserved.
Non-confidential

A staging location is typically general purpose external storage. Depending on ecosystem requirements, additional integrity controls may be required to prevent unauthorized substitution. For example, a requirement might be using a local hashlocking scheme, or implementing an anti-replay mechanism in any explicit update authorization messages.

[R0098] The amount of memory required at least for loading signed CCA firmware identity metadata must be known and verified before the identity metadata is copied to the staging area.

[R0135] The amount of memory required for the image payload must be known and verified before it is copied into on-chip memory or protected external memory.

[R0136] The required memory is allocated and available before data is copied from external storage.

For example, the signed firmware identity metadata and the firmware payload may be combined into a single image of a fixed maximum size. An implementation can then ensure that a fixed buffer of at least that size is always available before copying.

Alternatively, the signed firmware identity metadata may be fixed size, and include the size of the payload. The identity metadata can then be copied into a fixed buffer and verified there, and then a second dynamic buffer can be allocated based on a verified payload size.

[R0137] Data cannot be copied beyond the bounds of a staging area.

This prevents buffer overflow attacks.

[R0099] Following reset, the relevant part of the boot process detects the available update and attempts to load it in the normal way.

The system must remain attestable following any update.

A system is always attestable following a Root update as it requires a full system reset. It does not alter the boot state of the system, as attested to a reliant party, at runtime.

Following a Realm world update, Realm world is reset including all Realms, ensuring that Realms launched after the update are attested against the new Realm world state.

In the case of a live RMM update, the boot state of the system has changed at runtime, compared to that attested to a reliant party up to the point the update is effected. The capability of RMM to perform a live update must be reflected in the original attestation to a reliant party. For example, included as part of a service level agreement with a hosting provider against firmware versions and measurements, or against signed firmware identity metadata.

In the case of a live RMM update, there is no reliable way to update the state of any already attested Realms. Hence why the capability of performing a live update must form part of the contract of the original attestation.

However, any attestation request following a live RMM update must reflect the new state of the CCA platform.

Likewise in the case of a live RMM update, the changed state following the update may affect CCA derived Realm keys.

Live RMM updates, and CCA derived Realm keys, are out of scope of the current release of CCA. They may be addressed in later CCA releases.

[R0101] A reliant party is able to determine if an implementation of the CCA platform is capable of world updates, or Realm world incremental updates.

Copyright © 2020-21 Arm Limited or its affiliates. All rights reserved.
Non-confidential

10.2 Local update

A local update is one that requires physical access to the system. For example, a USB update, an update over a serial link, or a rescue loader feature in the boot process.

The general process, and its security properties, should be the same as in the case of a remote update.

For example, a USB or serial link update can be viewed as a different transport for a firmware update protocol.

Or a rescue loader can be viewed as a special case of a CCA firmware update client.

10.3 Robustness

[R0100] Any CCA firmware update mechanism must be robust against update failures.

Anti-rollback and recovery should be managed as defined in *CCA firmware boot*.

Copyright © 2020-21 Arm Limited or its affiliates. All rights reserved.
Non-confidential

11 CCA security lifecycle management

In this document, a *secured* CCA enabled system is one where the implementation of the CCA platform is in a trustworthy state.

A CCA enabled system may be in a non-secured state for a number of reasons, such as:

- During manufacture assembly, test and provisioning processes
- During supply chain provisioning processes
- During debug, diagnostic and repair stages

The security lifecycles of Secure world and Normal world are out of scope of CCA, and of this document.

It is expected that Normal world or Secure world debug and diagnostics can be enabled without affecting the CCA security guarantee.

11.1 Firmware enabled debug

CCA firmware is all firmware that may affect the CCA security guarantee, including application PE firmware and trusted subsystem firmware.

[R0103] The trustworthiness of CCA firmware is always attestable.

On a secured system, CCA will only load authorized CCA firmware at boot.

Capabilities of CCA firmware, including capabilities that may compromise the CCA security guarantee, are always reflected in signed firmware identity metadata, captured in boot states and included in CCA platform attestation.

[R0104] Capabilities that may compromise the CCA security guarantee include any capabilities that may reveal CCA assets or affect the execution flow of CCA firmware or Realms.

Examples of a capabilities which may compromise the CCA security guarantee include debug capabilities, such as self-hosted debug and remote debug, or revealing diagnostics.

Only the capabilities of CCA firmware are attested. Whether such capabilities are actually used or not at runtime is not attested.

For example, debug capable CCA firmware may implement additional authorization steps, such as additional authorization processes for accessing remote debug or revealing diagnostics. But the attested property is that such capabilities exist, not whether they are actually used or not.

[R0105] A reliant party is always able to determine the full capabilities of CCA firmware from boot state reflected in CCA platform attestation.

A reliant party can achieve this through directly verifying CCA firmware measurements. Alternatively, it can be achieved through verifying signed CCA firmware identity metadata and trusted firmware signers.

The current release of CCA does not support CCA derived Realm keys, nor any CCA key derivations for use by CCA firmware outside of CCA HES.

Such key derivations may be added in future releases. At that point additional rules for CCA firmware enabled debug will be required to ensure that debug enabled CCA firmware cannot derive the same keys as secured CCA firmware. Such rules may mandate Signer ID and security versioning in firmware identity metadata for use in CCA key derivations.

Copyright © 2020-21 Arm Limited or its affiliates. All rights reserved.

Non-confidential

11.2 CCA system security lifecycle

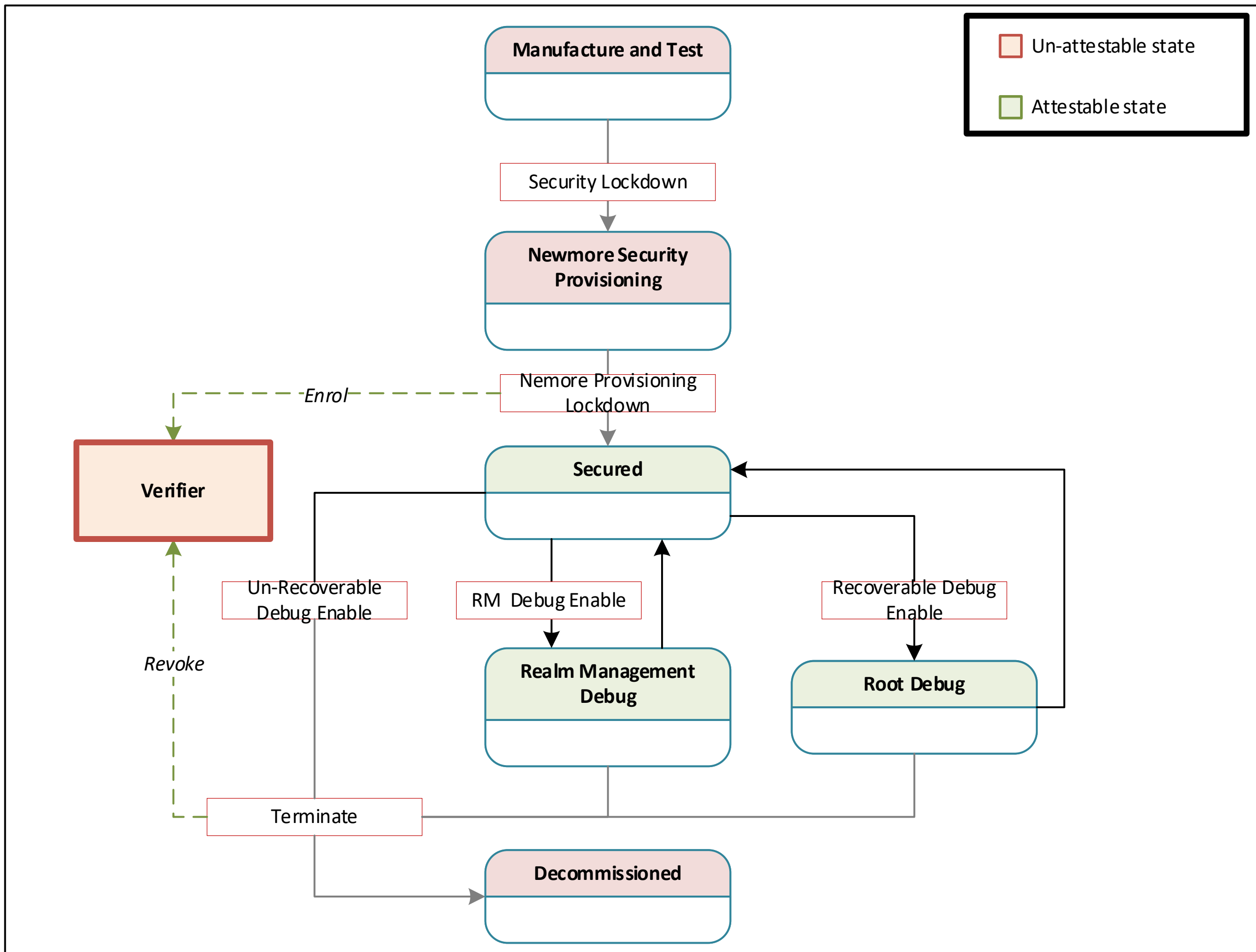

Figure 10. Conceptual CCA system security lifecycle

The CCA system security domain will go through a security lifecycle as outlined in *Figure 10. Conceptual CCA system security lifecycle*.

Actual implementations may have additional sub-states, in particular in the provisioning process, in support of more complex supply chain and deployment models.

CCA does not restrict or limit actual provisioning processes other than that the final end result should have the attestable properties described here.

For the purpose of defining the attestable properties of a CCA enabled system, the simpler model is used in this document.

The lifecycle starts with manufacture and testing of a CCA enabled system. At this stage the system has no attestation identity and any security features of the system may be disabled and bypassed.

The next step is provisioning of CCA hardware provisioned parameters.

[R0106] The system goes through security lock-down before security provisioning, such that it can only run authorized CCA firmware, and that external diagnostics and debug interfaces have been disabled or locked such that CCA hardware provisioned parameters are not exposed.

Copyright © 2020-21 Arm Limited or its affiliates. All rights reserved.
Non-confidential

For example, on a CCA HES enabled system it may be sufficient to only disable CCA HES host debug in order to protect CCA hardware provisioned parameters. Locking the whole system is recommended, and may be required to protect non-CCA security features.

Once provisioning is complete, the lifecycle can move to a secured state. Once a system is in a secured state it can also be enrolled with an implementation verifier, allowing a reliant party to verify Realms deployed on the system.

[R0107] The provisioning process is only complete once the system has gone through provisioning lockdown for all CCA hardware parameters, such that they have been successfully provisioned and that their values are immutable.

Immutable here means hardware provisioned CCA parameters do not change for the full duration of the CCA security lifecycle.

Some implementations may support hardware recovery schemes, for example allowing a system to assume a new identity following a re-provisioning process at a repair center. For the purpose of the CCA platform security lifecycle, this equates to decommissioning the previous identity and then starting the new identity in a security provisioning state.

The system can in principle be attested in the CCA security provisioning state as soon as it is able to derive a valid CPAK. But a system in this state should not be trusted as it may not be fully provisioned or fully locked down yet. For the same reason, the system should not be enrolled with a verifier until it is in the secured state.

From secured state, the system can move to a state that enables diagnostics or debug that may break the CCA security guarantee. This can be done either by loading debug capable CCA firmware or by enabling an external debug or diagnostic interface.

For a system to remain attestable, its attested state cannot change at runtime.

[R0108] Both CCA firmware enabled debug and external debug can only be enabled following system reset, before initial boot code has been completed.

[R0109] On a secured system only CCA HES can enable an external debug or diagnostic interface.

[R0110] External debug or diagnostic interfaces can only be enabled following explicit authorization.

[R0111] The authorization can only be issued by a trusted source, verified by CCA HES.

CCA does not mandate any particular hardware level debug authorization protocol. But any such protocol should be cryptographically at least as secure as CCA verified boot.

Debug states can be recoverable, or irreversible.

A recoverable debug state is one where the system can be returned to the secured state. Irreversible debug is one where it cannot.

[R0112] A system can only be returned to the secured state following system reset.

[R0113] CCA HES ensures all debug interfaces have been closed, and that only authorized CCA firmware can be loaded, before returning a system to a secured state.

[R0114] Realm Management debug state represents any external or CCA firmware debug capability that only affects Realm world.

Realm Management debug is both attestable and recoverable.

[R0115] Root debug state represents any external or CCA firmware debug capability that affects Monitor security domain and CCA system security domain.

Copyright © 2020-21 Arm Limited or its affiliates. All rights reserved.
Non-confidential

On a CCA HES enabled CCA system, this state remains both attestable and recoverable.

[R0116] Decommissioned state represents any external or CCA firmware debug capability that affects CCA HES.

[R0117] Arm recommends that all debug capabilities affecting CCA HES are permanently disabled on a CCA secured system.

[R0118] Before entering decommissioned state, at least private CCA hardware provisioned parameters are put permanently beyond use.

The decommissioned state is un-attestable, and irreversible.

[R0124] For a decommissioned system, the CCA attestation identity must be revoked for the purpose of attestation, and cannot be recovered or reused.

The current release of CCA does not support CCA derived Realm keys, nor any CCA key derivations for use by CCA firmware outside of CCA HES.

Such key derivations may be added in future releases. At that point additional rules for enabling external debug will be required to ensure that the same keys cannot be derived in a debug state as in the secured state. For example, by including the CCA platform security lifecycle state in CCA key derivations.

11.3 Reprovisioning

Strictly, the CCA system security lifecycle defines the lifecycle of immutable CCA system security domain properties. This includes the security properties of immutable initial boot code, hardware provisioned parameters, as well as the security properties of immutable hardware elements of the CCA system security domain.

The immutable properties of the CCA system security domain constitute the most fundamental root of trust for a CCA enabled system.

[R0138] Immutable properties are attested indirectly by the CCA platform attestation key and identity.

This means those properties, and the associated CCA platform attestation key, must remain immutable in any attestable state of the CCA system security lifecycle.

Entering any un-attestable state may revoke or void any immutable security claims, and the trustworthiness of the CCA platform cannot be determined by a verifier.

[R0139] Immutable CCA system security domain properties can only be modified in either a provisioning state, or in an implementation specific recovery process following decommissioning.

Any change to immutable properties constitutes a reprovisioning event as the CCA platform attestation key and identity must be changed to reflect the new immutable security properties.

This includes the case where the change is the CCA platform attestation key and identity themselves (renewal). Typically following an upgrade to security algorithms, or following recovery of a compromised CCA platform key.

[R0140] Any reprovisioning event always revokes any previous CCA platform attestation key and identity, and a new CCA platform attestation key and identity is issued.

Arm CCA does not define provisioning, reprovisioning, or recovery processes. These are implementation and ecosystem specific and out of scope of this specification.

Copyright © 2020-21 Arm Limited or its affiliates. All rights reserved.
Non-confidential

11.4 Trusted subsystems and CCA HES

It is expected that trusted subsystems will follow the same pattern and rules as the main CCA system security lifecycle.

[R0145] A trusted subsystem may delegate its security lifecycle management to another trusted subsystem.

For example, a power management subsystem may delegate its security lifecycle management to a CCA HES host.

[R0119] No CCA trusted subsystem is affected by Root debug.

[R0120] Any external or CCA firmware enabled debug capability affecting any CCA trusted subsystem can only be enabled following system reset.

[R0121] Any external or CCA firmware enabled debug capability affecting any CCA trusted subsystem can only be enabled by CCA HES.

[R0122] Arm recommends that all debug capabilities affecting CCA HES are permanently disabled on a CCA secured system.

Copyright © 2020-21 Arm Limited or its affiliates. All rights reserved.
Non-confidential

12 Cryptographic recommendations

12.1 Cryptographic algorithms and key sizes

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

As a general recommendation, in the absence of specific requirements by application, protection profile, or regulation, CCA devices should be designed to comply with NIST recommendations, or CNSS Advisory Memorandum Information Assurance 02-15, or local equivalents depending on target region.

See:

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

http://www.cnss.gov/CNSS/issuances/Memoranda.cfm

12.2 Post-quantum readiness

In line with current general guidance, such as https://www.ncsc.gov.uk/whitepaper/preparing-for-quantum-safe-cryptography, post-quantum safe cryptography is not currently included in recommendations for CCA implementations at this point in time.

Implementations may choose to adopt such algorithms if required for a market.

All use of cryptography in CCA can be updated with CCA firmware updates, except immutable initial boot code.

[R0123] Arm recommends that, wherever possible, implementations of CCA implement an upgrade path to post-quantum safe cryptography.

12.3 Guidance

The following tables outline general recommendations for algorithms and key sizes in a CCA implementation.

12.3.1 Recommended parameter sizes

Parameter type	Sizes	Information
Public identifier	512 bits	Typically a hash of a public key or a certificate. Arm does not recommend using truncated hashes for public identifiers.
Counter values used as unique identifiers	64 bits	
Seeds used for symmetric key derivations	256 bits	
Symmetric keys	256 bits	AES-256
Asymmetric keys	RSA: 3072 bits ECC: 384 bits	RSA 3072 ECC Curve-P384

Copyright © 2020-21 Arm Limited or its affiliates. All rights reserved.
Non-confidential

12.3.2 Recommended algorithms

Operation	Algorithm	Information
General encryption	AES-256	
Hash function	SHA-512	NIST SP 800-107
Signing – RSA option	RSA-PKCS#1 v2.2 RFC 8017, FIPS PUB 186-4	RSASSA-PSS 3072 EMSA-PSS SHA-512
Signing – ECC option	ECDSA RFC 6979, FIPS PUB 186-4	Curve P-384 SHA-512
Signing- Symmetric	HMAC-SHA512 RFC 2104, NIST SP 800-107	512 bits Arm does not recommend truncating HMAC.
Key derivations	NIST SP 800-108 Counter mode HMAC-SHA512 RFC 2104, NIST SP 800-107	

12.3.3 Memory protection

Operation	Algorithm	Information
Encryption	QARMA-128 with a 256-bit key AES-128-XEX with two independent 128-bit keys	

Copyright © 2020-21 Arm Limited or its affiliates. All rights reserved.
Non-confidential

13 Appendix A: Rule history

13.1 Last used identifier

Unique identifier: R0159

13.2 New identifiers since last major release

13.3 Deprecated identifiers since last major release

R0037, R0096, R0097, R0102.

Copyright © 2020-21 Arm Limited or its affiliates. All rights reserved.
Non-confidential