



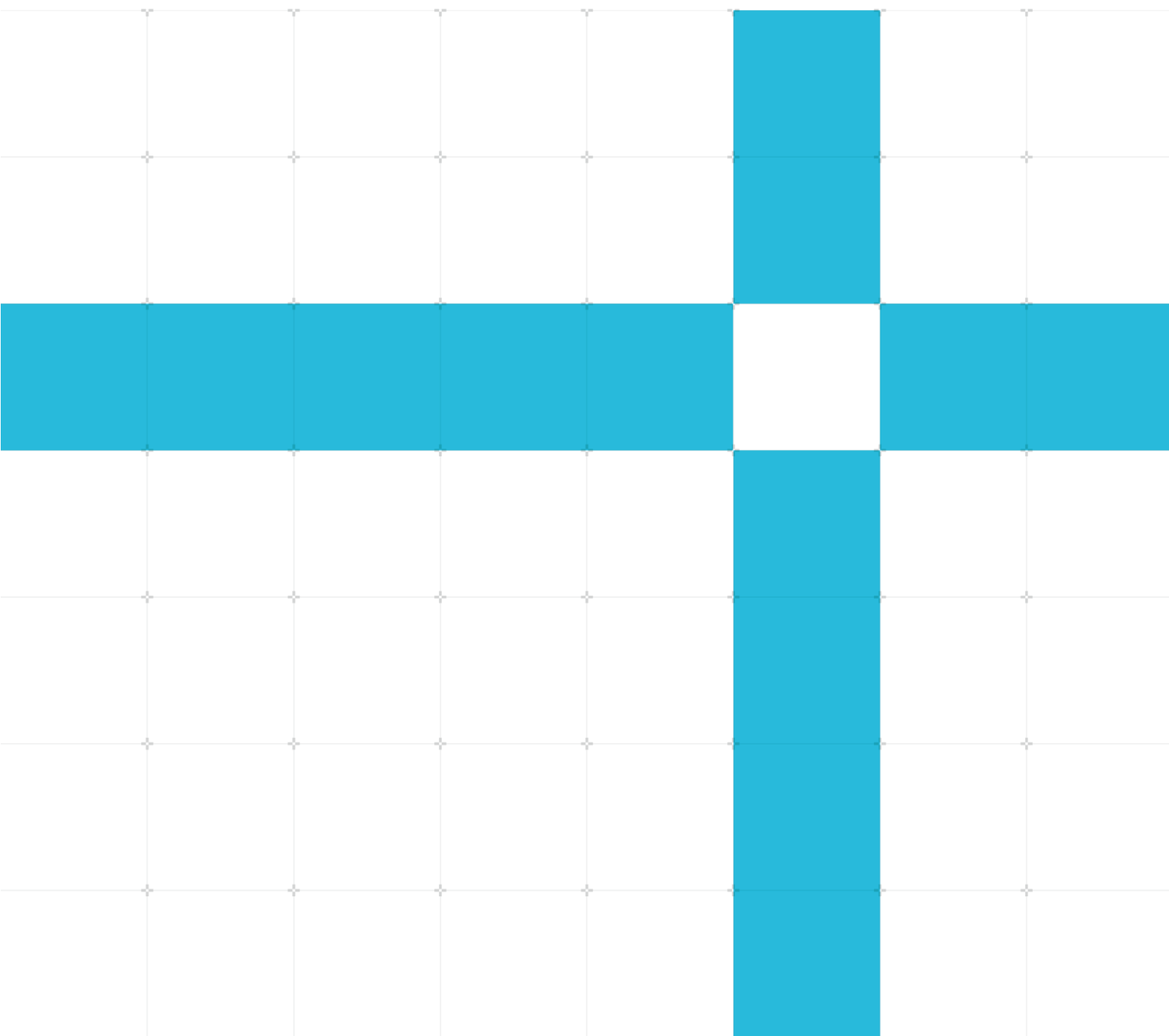
# Introducing Arm Confidential Compute Architecture

**Non-Confidential**

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

**Issue 01**

DEN0125



## Arm Confidential Compute Architecture

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

### Release information

#### Document history

Issue	Date	Confidentiality	Change
01	23-06-2021	Non-confidential	First release

## 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.

If any of the provisions contained in these terms conflict with any of the provisions of any click through or signed written agreement covering this document with Arm, then the click through or signed written agreement prevails over and supersedes the conflicting provisions of these terms. 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)

## Confidentiality Status

This document is Non-Confidential. The right to use, copy and disclose this document may be subject to license restrictions in accordance with the terms of the agreement entered into by Arm and the party that Arm delivered this document to.

Unrestricted Access is an Arm internal classification.

## Web Address

[developer.arm.com](http://developer.arm.com)

## Progressive terminology commitment

Arm values inclusive communities. Arm recognizes that we and our industry have used terms that can be offensive. Arm strives to lead the industry and create change.

# Contents

<b>1 Overview.....</b>	<b>5</b>
1.1 Before you begin.....	5
<b>2 What is Arm CCA?.....</b>	<b>6</b>
2.1 Arm CCA requirements.....	6
<b>3 Arm CCA extensions .....</b>	<b>8</b>
3.1 Realms.....	8
3.2 Realm world and Root world.....	9
3.3 What is the difference between Arm TrustZone extensions and Arm RME? .....	11
<b>4 Arm CCA hardware architecture .....</b>	<b>14</b>
4.1 Realm world requirements.....	14
4.2 Memory management for Arm CCA.....	15
4.3 Attestation .....	19
<b>5 Arm CCA software architecture.....</b>	<b>20</b>
5.1 Software stack overview.....	20
5.2 Realm Management Monitor .....	22
5.3 Realm Management Interface .....	23
5.4 Realm Services Interface .....	24
<b>6 Check your knowledge.....</b>	<b>26</b>
<b>7 Related information .....</b>	<b>27</b>
<b>8 Next steps.....</b>	<b>28</b>

# 1 Overview

In this guide, we look at the role of confidential computing in modern compute platforms and explain the principles of confidential computing. We then describe how the Arm Confidential Compute Architecture (Arm CCA) enables confidential computing in an Arm compute platform.

At the end of this guide, you will be able to:

- Define confidential computing
- Describe a complex system Chain of Trust
- Understand that a Realm is a protected execution environment introduced by the Arm CCA
- Explain how Realms are created, managed, and executed on an implementation of the Arm CCA
- Define the difference between a Trusted Execution Environment (TEE) and a Realm
- Explain how a Realm owner establishes trust in a Realm

## 1.1 Before you begin

This guide assumes that you are familiar with the Arm exception model and memory management. If you are not familiar with these subjects, read our [AArch64 Exception model](#) and [AArch64 Memory management](#) guides.

Some of the operation of the Arm CCA refers to virtual machines and virtualization. If you are not familiar with these concepts, refer to [AArch64 virtualization](#).

If you are not familiar with Arm security concepts, see [Introduction to security](#).

## 2 What is Confidential Computing?

Confidential Computing is the **protection of data in use**, by performing computation within a trustworthy hardware-backed secure environment. This protection shields code and data from observation or modification by privileged software and hardware agent.

Any application or Operating System executing in a Confidential Computing environment can expect to execute in isolation from the rest of the system. Any data generated or consumed by the isolated execution cannot be observed by any other actor executing on that platform without explicit permission.

### 2.1 Arm CCA requirements

Code executing in an Arm CCA system does not have to trust large and complex software stacks executing on the environment or any peripherals that might affect it, for example DMA capable devices. Arm CCA removes the need for many of the relationships with the developers of the software stack or hardware.

A security architect might consider Arm CCA if they are deploying loads on a cloud server system where they may not know who the developer of the hypervisor is for that system. Because the hypervisor is unknown, this can lead to a lack of trust in execution on the platform without Arm CCA.

Arm CCA allows the application developer to deploy workloads securely without having to trust the underlying software infrastructure, for example the hypervisor, kernel, or core running in the Secure world.

To allow Arm CCA, a platform must provide the following:

- An execution environment which provides isolation from all untrusted agents
- A mechanism to establish that the execution environment has been initialized into a trustworthy state. Initialization will require the execution environment to have its own Chain of Trust, independent from the Chain of Trust that is used by the parallel untrusted environments in the platform.

In this guide, we explain how the Arm CCA fulfills these requirements through hardware implementation and use of software.

## 3 Arm CCA extensions

As described in [What is Confidential Computing](#), Arm CCA allows you to deploy application or Virtual Machines (VMs), while preventing access by more privileged software entities such as a hypervisor. However, it is these privileged software entities that typically manage resources like memory. In this case a privileged software entity, for example a hypervisor, does have access to the memory of an application or VM.

The Arm CCA allows you to control the VM, but removes the right for access to the code, register state, or data that is used by that VM.

The separation is enabled by creating [protected VM execution spaces called Realms](#). A Realm has complete isolation from the normal world in terms of code execution and data access. The Arm CCA achieves this separation through a [combination of architectural hardware extensions and firmware](#).

Within the Arm CCA, the hardware extensions on an Arm application PE are called the Realm Management Extension (RME). The RME interacts with the specialist firmware for Realm control, called the Realm Management Monitor (RMM), and the Monitor code in Exception level 3. We describe these elements in [Arm CCA hardware architecture](#) and [Arm CCA software architecture](#).

### 3.1 Realms

A Realm is an Arm CCA environment that can be dynamically allocated by the Normal world Host. The Host is the supervisory software that manages an application or Virtual Machine (VM).

The initial state of a Realm, and of the platform on which it executes, can be attested. Attestation allows the Realm owner to establish trust in the Realm, before provisioning any secrets to it. The Realm does not have to inherit the trust from the Non-secure hypervisor which controls it.

The Host can allocate and manage resource allocation. The Host can manage the scheduling of the Realm VM operation. However, the Host cannot observe or modify the instructions executed by the Realm.

Realms can be created and destroyed under Host control. Pages can be added or removed through Host requests in a way that is similar to a hypervisor managing any other non-confidential VM.



To run a CCA system, a Host needs to be modified. The Host continues to control the non-confidential VMs but needs to communicate with the Arm CCA firmware, in particular the Realm Management Monitor (RMM). The operation of the RMM is discussed in [Arm CCA software extensions](#).

## 3.2 Realm world and Root world

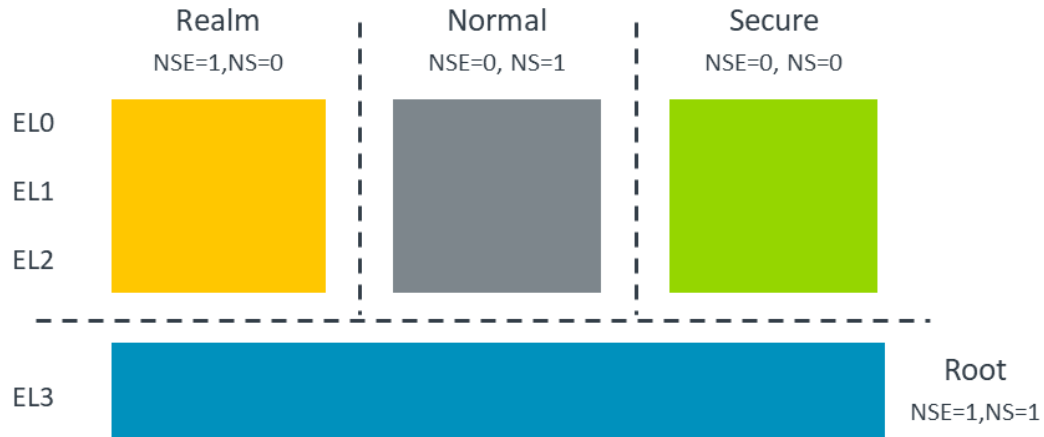
The Armv8-A TrustZone extensions allow secure execution of code and isolation of data by having two separated worlds, the Secure world and the Normal world.

A world is combination of a Security state of a PE and physical address space. The Security state a PE is executing in determines which physical address spaces a PE can access. In the Secure state a PE can access Secure and non-Secure physical address spaces, whereas in the Non-secure state it can only access the Non-secure physical address spaces. Normal world is generally used to refer to combination of Non-secure state and Non-secure physical address space.

The Arm CCA provided as part of Armv9-A introduces the Realm Management Extension or RME. This extension introduces **two added additional worlds**, the Realm world and the Root world:

- Root world introduces the **Root security state and Root physical address space**. A PE is in the Root security state when it is running in **Exception level 3**. The Root PA is separate from the Secure PA. This is a key difference to Armv8-A TrustZone, where Exception level 3 code did not have a private address space and instead used the Secure PA. The latter is still used by S\_EL2/1/0. **The Monitor runs in the Root world**.
- **Realm world is similar to the TrustZone Secure world**. Realm world comprises of **Realm security state and Realm PA**. Realm state code can execute at R\_EL2, R\_EL1 and R\_EL0 and the controlling firmware running in the Realm world can access memory in the Normal world to allow shared buffers.

The following diagram shows the four RME-based worlds, and their relationship to the SCR\_EL3 NS and NSE bits:



**Figure 1: Arm CCA four world locations**

The Root world allows trusted boot execution and switching between the different worlds. The PE resets into the Root world.

The Realm world provides an execution environment for VMs that is isolated from the Normal and Secure worlds. VMs require control from the Host in the Normal world. To allow for full control of Realm creation and execution, the Arm CCA system provides:

- Realm Management Extension, which are the hardware extensions that are required by the architecture to allow isolated Realm VM execution
- Realm Management Monitor, which is part of the firmware that is required to manage Realm creation and execution under requests from a Normal world Host

We describe these components in more detail in [Arm CCA hardware architecture](#) and [Arm CCA software architecture](#).

World switching in a non-RME PE is controlled by the SCR\_EL3.NS bit. Exception level 3 software sets NS = 0 when switching to the Secure world and sets NS = 1 when switching to the Normal world. World switching in an RME-implemented PE is extended through a new SCR\_EL3.NSE bit added to the SCR\_EL3 register.

The following table shows how the bits control execution and access between the four worlds:

SCR_EL3.NS	SCR_EL3.NSE	World	EL0	EL1	EL2	EL3
0	0	Normal	EL0	EL1	EL2	-
1	0	Secure	S-EL0	S-EL1	S-EL2	-

SCR_EL3.NS	SCR_EL3.NSE	World	EL0	EL1	EL2	EL3
0	1	Realm	R_EL0	R-EL1	R-EL2	-
1	1	Root	-	-	-	EL3

**Table 1: World control and status bits**

### 3.3 What is the difference between Arm TrustZone extensions and Arm RME?

All Arm A-Profile processors have the option to implement the Arm TrustZone architecture extensions. These extensions allow development of an isolated execution and data environment. Elements like a Trusted Operating System (TOS) can service Trusted applications, which execute in isolation, to service Secure requests from the Rich OS that is running in the Normal world.

The addition of virtualization to the Secure world in Armv8.4-A allows you to manage multiple Secure Partitions in the Secure world. This feature can allow multiple TOSs to be applied to a system. The Secure Partition Manager (SPM), executing at S\_EL2, is the manager for the Secure Partitions. The SPM has a similar functionality to the hypervisor in the Normal world.

In operation, the Trusted OS is often part of a chain of trust, it is verified by higher privilege firmware, in some systems this may be the SPM. This means the TOS relies on the relationship with the higher privilege firmware developer.

There are two methods that will initiate the execution of the TOS:

- Rich OS yielding, where the Rich OS enters an idle loop and executes an SMC instruction to call the TOS through the Monitor
- Interrupt targeted at the Trusted OS. Secure type 1 interrupts are used for the execution of the TOS. A secure type 1 interrupt asserted during Normal world execution calls the TOS through the Monitor.

A Realm Virtual Machine is different to a Trusted OS or Trusted application because the Realm VM is controlled from the Normal world Host. In areas like creation and memory allocation, the Realm VM acts like any other VM being controlled from the Host.

A difference between the Realm VM execution and the Trusted OS execution is that the Realm does not have any physical interrupts

enabled. All interrupts for the Realm are virtualized by the hypervisor and then signaled to the Realm through commands passed to the RMM. This means that a compromised hypervisor might prevent execution of the Realm VM, so there is no guarantee of Realm execution.

The Realm execution and memory access are initialized by the controlling Host software, for example the hypervisor. The Realm does not have to be verified by the Host. The Realm can bypass any Chain of Trust because it can use the RME initialization attestation. For more information, see [Attestation](#). The Realm is also completely isolated from the controlling software. If a Realm is initialized by a Host, the Host has no ability to see the data or data memory of the Realm.

The main difference between the use of Realms and TOS is in the design intent between Secure execution and Realm execution.

Trusted applications are used for platform specific services owned by actors close to the system development, such as Silicone Providers (SiPs) and Original Equipment Manufacturers (OEMs).

The intention for Realm execution is to allow general developers to execute code on a system without being involved in complex business relationships with the developers in the compute system.

Arm CCA allows Realms to be created and destroyed on demand under the control of the Normal world host. Resources can be added or retrieved from Realms dynamically. This brings the benefits of isolation available to services tied to the device to many more applications and users.

Trust is often defined in terms of Confidentiality, Integrity, and Authenticity, and is explained in the following list:

- With confidentiality, code data or state of an Arm CCA environment cannot be observed by other software running on the same device, even if that software is more privileged
- With integrity, code data or state of an Arm CCA environment cannot be modified by other software running on the same device, even if that software is more privileged
- With authenticity, code or data can be modified by other software running on the same device, but any changes can be identified

Trusted applications and TOS can provide a system with Confidentiality, Integrity, and Authenticity. Realm execution can provide a system with Confidentiality and Integrity.

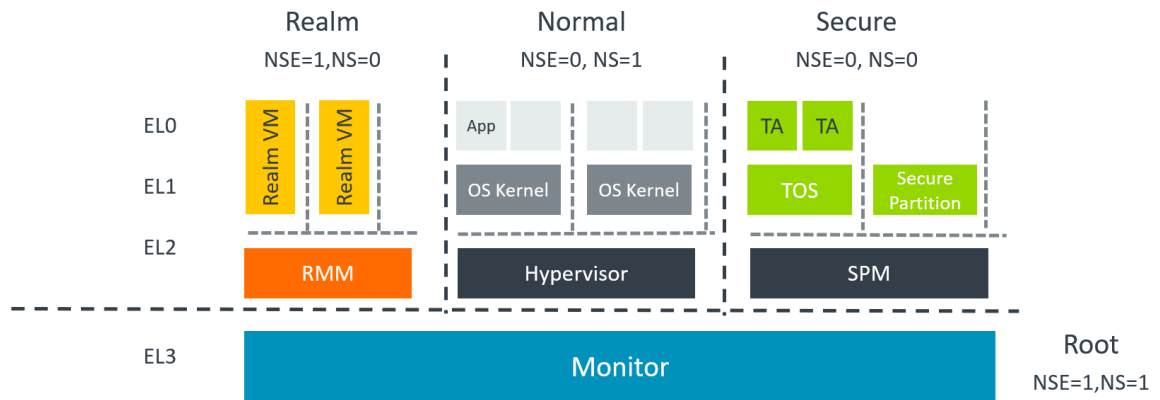
The four world environment provided by the Arm CCA system allows the complete separation between the Secure world and Realm world. This means that Trusted applications do not have to be concerned about the execution of any Realm VM and the Realm VM does not have concerns about any trusted application that is executed.

# 4 Arm CCA hardware architecture

This section describes the Realm Management Extensions, which are the changes to PE architecture that enable PEs to run Realms.

## 4.1 Realm world requirements

The following diagram shows a complete view of how Realms fit within an Arm CCA system:



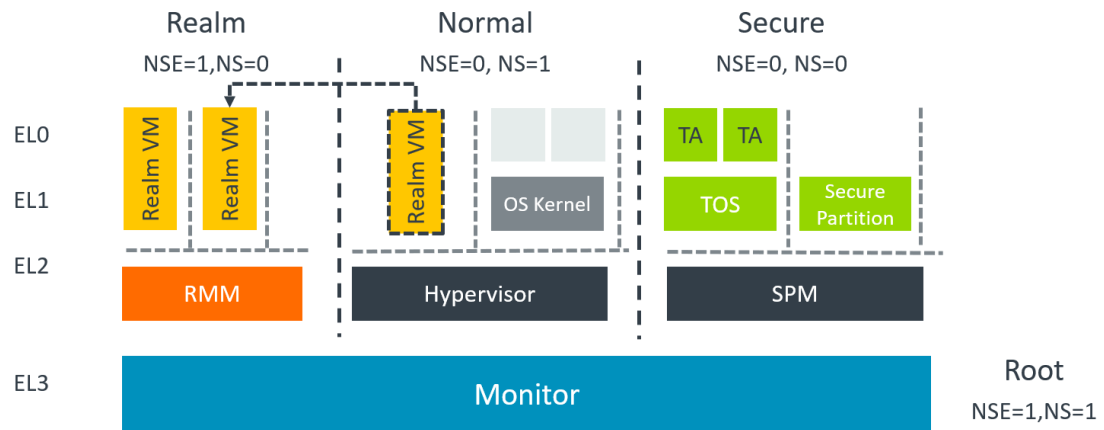
**Figure 2: Realm world software execution**

The Realm world must be able to execute code, and access memory and trusted devices, in complete isolation from all other non-Root worlds and devices.

Just like the other worlds, or security states, the Realm world has three exception levels, R\_EL0, R\_EL1, and R\_EL2. Realm VMs run in R\_EL1 and R\_EL0. The Realm Management Monitor (RMM) runs in R\_EL2. The RMM is described in [Arm CCA software stack](#).

The isolation is hardware enforced through the architecture Realm Management Extension, which allow control over memory management, execution, and the isolation of the Realm for context and data. Isolation means that access is prevented through faulting exceptions by the PE or by encryption or the Realm and Root worlds.

In the following diagram, the isolated Realm VM is generated and controlled in the Normal world by the hypervisor, but physical execution is in the Realm world:



**Figure 3: Realm VM execution**

The execution of the Realm VM is initialized through hypervisor commands that are passed to the Monitor, and then pushed through the Monitor to the RMM.

The **Monitor is the gatekeeper between the separate worlds.** It controls any passage between Normal world, Secure world, and Realm world to ensure that the isolation between the worlds is maintained whilst allowing communication and control where needed.

## 4.2 Memory management for Arm CCA

Arm A-profile processors that implement the TrustZone Security Extensions present two Physical Address Spaces (PAS):

- Non-secure physical address space
- Secure physical address space

The Realm Management Extension add another two PAS:

- Realm physical address space
- Root physical address space

The following diagram shows the physical address spaces and how to implement the spaces in a working system:

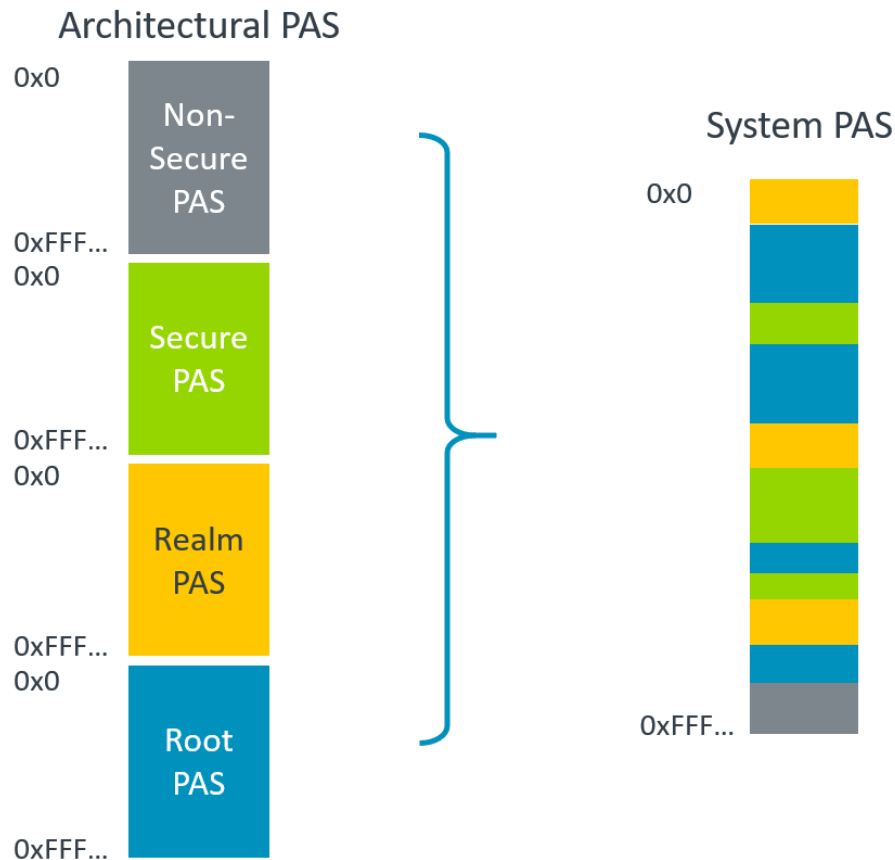


Figure 4: Physical address spaces

PAS access between the different worlds is hardware enforced, as defined in the following table:

Security state	Non-secure PAS	Secure PAS	Realm PAS	Root PAS
Non-secure	✓	✗	✗	✗
Secure	✓	✓	✗	✗
Realm	✓	✗	✓	✗
Root	✓	✓	✓	✓

Table 2: Physical address access permissions

As the table shows, the Root state can access all physical address spaces. The Root state enables memory transitioning between Non-secure PAS and Secure or Realm PAS where required.



To ensure that the isolation rules for all worlds are enforced, the physical memory access controls in the preceding table are enforced by the MMU, downstream of any address translation. This process is called **Granule Protection Check (GPC)**.

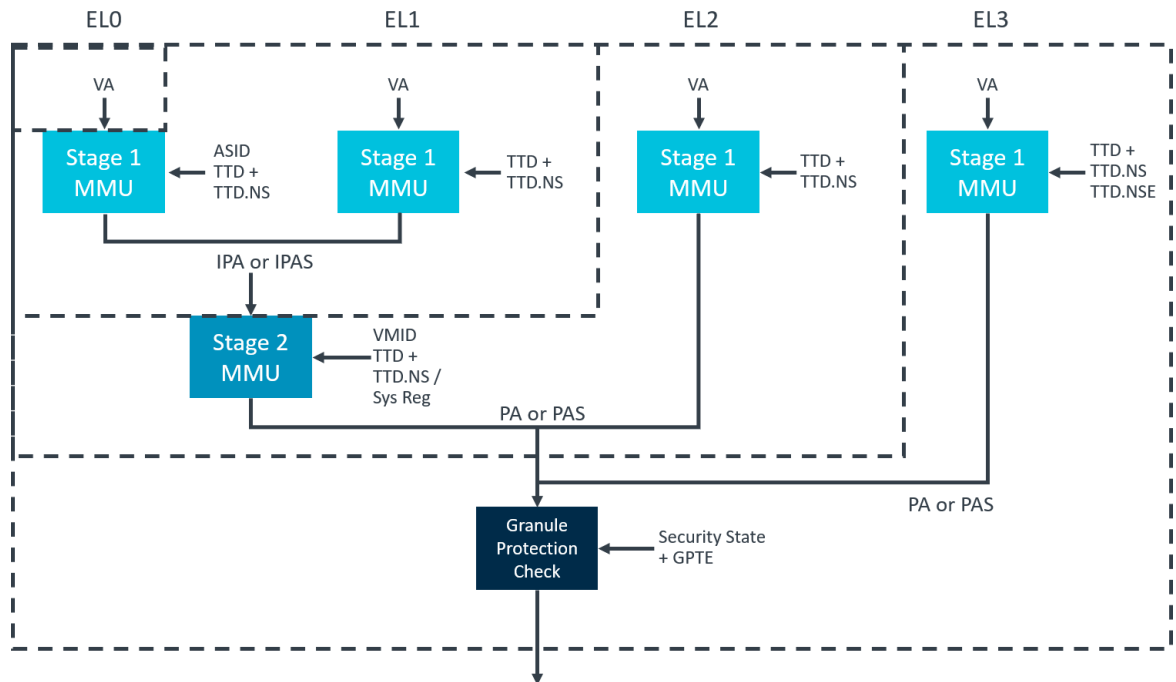
The PAS assignment of every granule of physical memory is described in the **Granule Protection Table (GPT)**. The Monitor in Exception level 3 can dynamically update the GPT which allows the physical memory to be moved between the worlds.

Any access control violation results in a new type of fault, which is called a Granule Protection Fault (GPF). **Enablement of the GPC, the contents of the GPT and the routing of a GPF are controlled by Root state.**

Resources belonging to a Realm must be in Realm owned memory, meaning part of the Realm PAS, to ensure isolation. However, a Realm might need to access some resources held in Non-secure memory, for example to enable message passing. This means that a Realm needs to be able to access physical addresses in both the Realm and Non-secure PASs.

Any Realm VM is executing in the Realm world but is executing under control of a Normal world Host. The Realm VM will need to be able to access memory in the Normal world PAS and the Realm world PAS. Access to the different PAS is controlled by the state of the NS bit in the Realm stage 2 translation table.

The following diagram shows the full stages and the location of the GPC in the Virtual Address (VA) to Physical Address (PA) chain. In this diagram, TTD is Translation Table Descriptor and GPTD is Granule Protection Table Descriptor:



**Figure 5: Granule protection check**

The diagram shows the translation stages within an RME-based platform for the virtual address to physical address translation.

For more information on stage 1 and stage 2 translation see [AArch64 Memory Management](#). However, in this example, an extra bit is defined in the Exception level 3 stage 2 page table entries to allow for access to the four PAS by the Monitor. This is the Non-secure Extension (NSE) bit in the translation table definition.

Normal world, Realm world, and Secure world all have translation available at Exception level 1 and Exception level 2 through stage 1 translation and, where necessary, stage 2 translation.

RME adds the Granule Protection Check (GPC) after the translation process for stage 1 and stage 1 and 2 translations. The GPC checks all physical addresses and PAS against the GPT to allow memory access or create a fault. The **GPT is held in Root memory** to ensure that it is isolated from all other worlds. The GPT can only be created and modified by code running in the Root world, from the Monitor code or Trusted Firmware.

Non-PE Requesters will also be included in this check if they are connected to a Requester side filter like a System MMU (SMMU).

## 4.3 Attestation

Code running inside Realms will manage confidential data or run confidential algorithms. Therefore, that code needs to be sure it's running a real Arm CCA platform rather than some simulation. The code also needs to know that it has been loaded properly and not been tampered with. Finally, the code also needs to know that the overall platform, or the realm are not in a debug state that could leak its secrets. The process of establishing this trust is called Attestation.

Attestation is broken into two key parts:

- Attestation of the platform
- Attestation of the initial state of the Realm

These two parts combine to create attestation reports, that a realm code can request at any time. The reports can then be used to authenticate the validity of the platform and the code in the realm.

Platform Attestation involves proving that silicon, and firmware, that underlies the Realm is genuine. This creates requirements on the hardware. The hardware needs to be provisioned with an identity. Equally the hardware needs to support measurements of key firmware images such as the Monitor, the RMM and firmware for any other controller in the platform that can materially impact security such as a power controller.

# 5 Arm CCA software architecture

Arm CCA platforms come together through a mix of hardware additions, such a RME in the PEs, and firmware components, in particular the Monitor and Realm Management Monitor. This section introduces the software stack for an Arm CCA platform.

## 5.1 Software stack overview

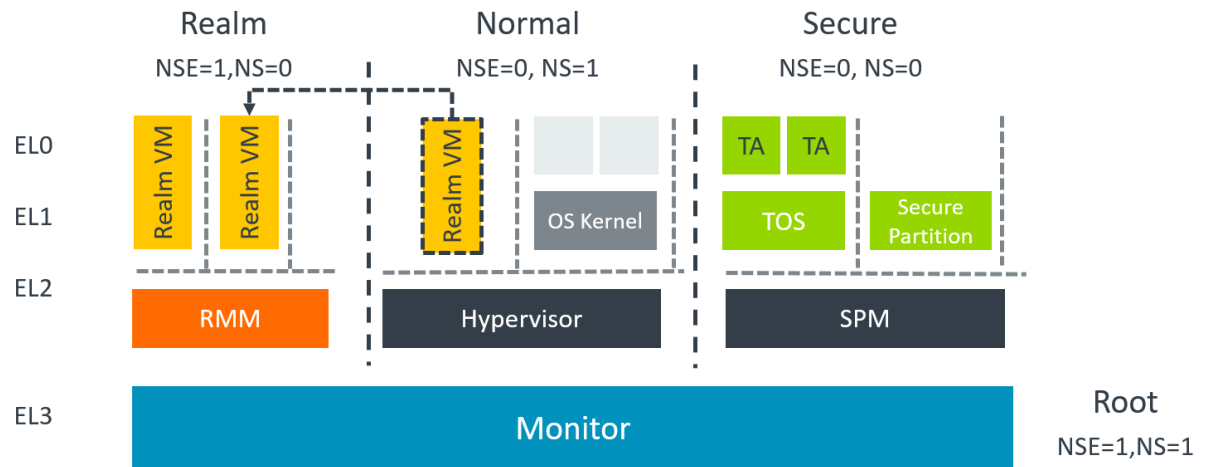
The execution of a Realm VM is intended to be isolated from the Normal world, the Realm VM is initiated and controlled by the Normal world Host. To allow the isolated execution of the Realm VM, a communication stack is implemented between the Host Hypervisor executing in Normal world Exception level 2 and the Realm VM executing in Realm world Exception level 2, or R\_EL2.

The RMM is responsible for managing communication and context switching. The RMM does not make policy decisions, such as which Realm to run or what memory to allocate to a Realm. Those decisions remain with the Host Hypervisor.

The RMM isolates the Realms from each other through the stage 2 page tables in the Realm world.

The RMM interfaces directly to the Monitor which also interfaces with the Secure world and the Normal world. The Monitor, running in Exception level 3, has the platform-specific code that must service all the Trusted functionality of the system. The RMM responds to a specific interface and will have fully defined functionality to manage the requests from the Host and the Monitor. Because this interface will be well-defined, the RMM can be generic code for all Arm CCA systems.

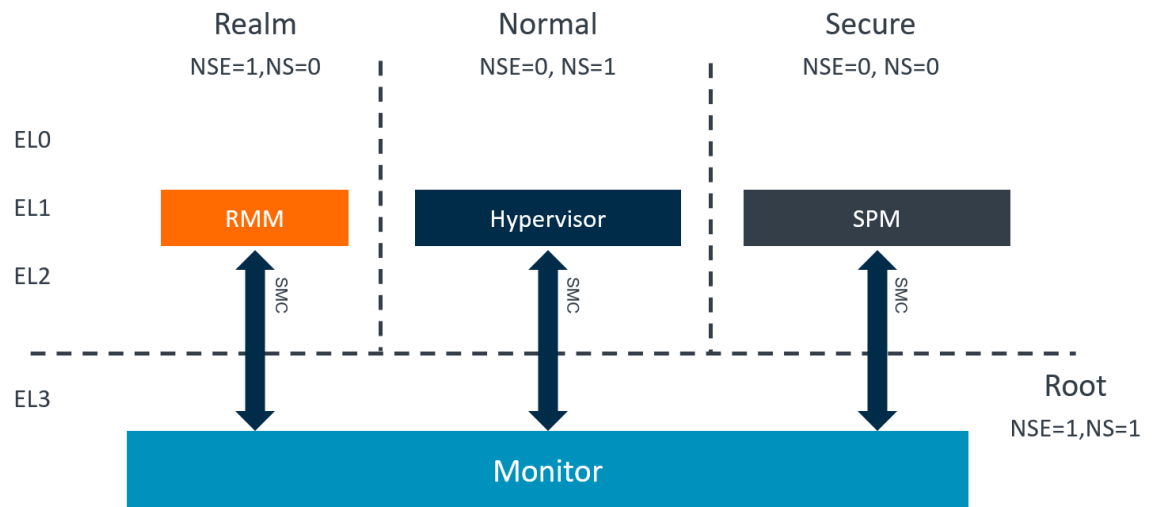
The following diagram shows the complete Arm CCA platform running a confidential Realm VM in the Realm world:



**Figure 6: Realm VM execution**

The RMM is the controlling software in the Realm world that reacts to requests from the hypervisor in the Normal world to allow the management of the Realm VM execution. The RMM communicates through the Monitor in Root world to control Realm memory management functions for memory transition between NS PAS and Realm PAS.

The SMC instruction allows the RMM, hypervisor, and SPM to return control to the Monitor, and allows a communication channel to be implemented between all Exception level 2 software and the Monitor. The following diagram shows the communication channels between the Monitor and the different controlling software in each world:



**Figure 7: World communication through SMC channels**

In each world, Exception level 2 execution can make a call into the Exception level 3 Monitor by executing an SMC instruction. The use of the SMC instruction can form the basis of a communication channel between the individual Exception level 2 controlling hosts and the Monitor in Exception level 3. This is the method that the Host in Normal world Exception level 2 can communicate through the Monitor to the RMM in Realm Exception level 2.

## 5.2 Realm Management Monitor

The Realm Management Monitor (RMM) is the Realm world firmware that is used to manage the execution of the Realm VMs and their interaction with the hypervisor in Normal world. The RMM operates in Exception level 2 in the Realm world, known as R\_EL2.

The RMM has two sets of responsibility in the Arm CCA system, the RMM provides services to the Host, to allow the Host to manage the Realms, and the RME also supplies services directly to the Realms.

The Host services can be split into areas of Policy and Mechanics.

For the Policy functionality, the RMM owns all the policy decisions on the following:

- When to create or destroy a Realm
- When to add or remove memory from a Realm
- When to schedule a Realm in or out

The RMM supports the host Policies by providing the following functionality:

- Providing **services to manipulate Realm page tables**, which are used in creation or destruction and the addition or removal of Realm memory
- **Management of Realm context**. This is context save and restore used in scheduling.
- **Interrupt support**
- **PSCI call interception**. This is power management requests. The RMM also provides services to Realms, primarily **attestation and cryptographic services**.

Finally, the RMM also upholds the following security primitives for the Realms:

- The RMM validates hosts requests for correctness
- The RMM isolates Realms from each other

The RMM specification defines two communication channels to allow all functionality to be requested and controlled between the Normal world Host and the Realm VM. The communication channel from the Host to the RMM is called the Realm Management Interface (RMI). A second channel defined between the RMM and the Realm VM is called the Realm Service Interface (RSI). The RSI is the channel for requesting services from the RMM.

In the following sections, we look at each of these interfaces.

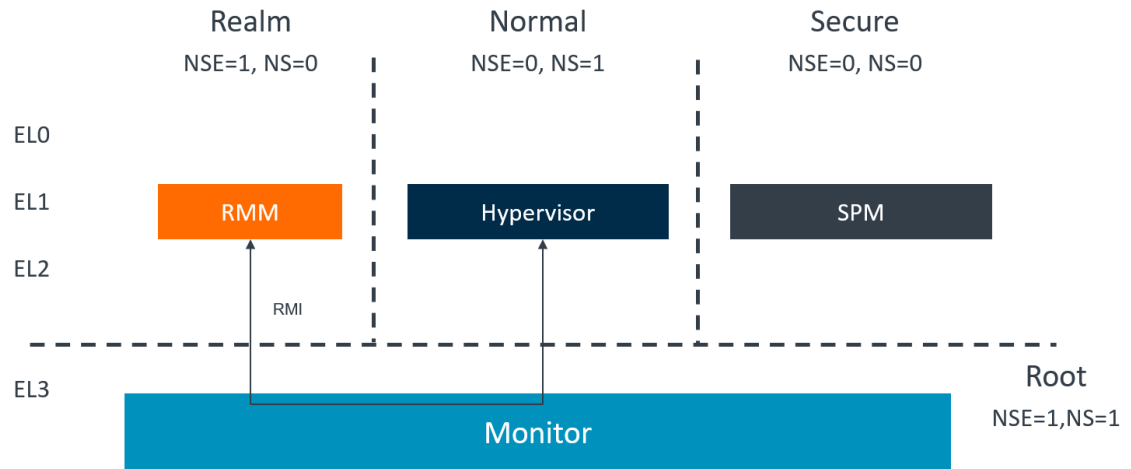
## 5.3 Realm Management Interface

The Realm Management Interface (RMI) is the interface between the RMM and the Normal world Host.

The RMI allows the Normal world hypervisor to issue instructions to the RMM that will manage the realm. The RMI uses SMC calls from the host hypervisor to request management control from the RMM.

The RMI enables control of the Realm management which includes creation, population, execution, and destruction of the Realms.

The following diagram shows where the RMI is implemented between the Normal world Host, Monitor, and the RMM:



**Figure 8: Realm Management Interface route**

## 5.4 Realm Services Interface

The Realm Service Interface (RSI) is the interface between the Realm VM and the RMM.

The RSI allows a channel for external services that some Realm Management operations need to pass to the Realm from the RMM. These services can include cryptographic services and attestation. The RSI is also the channel for memory management requests from the Realm VM to the RMM.

The following diagram shows the position of the RSI between the RMM and each individual Realm VM:



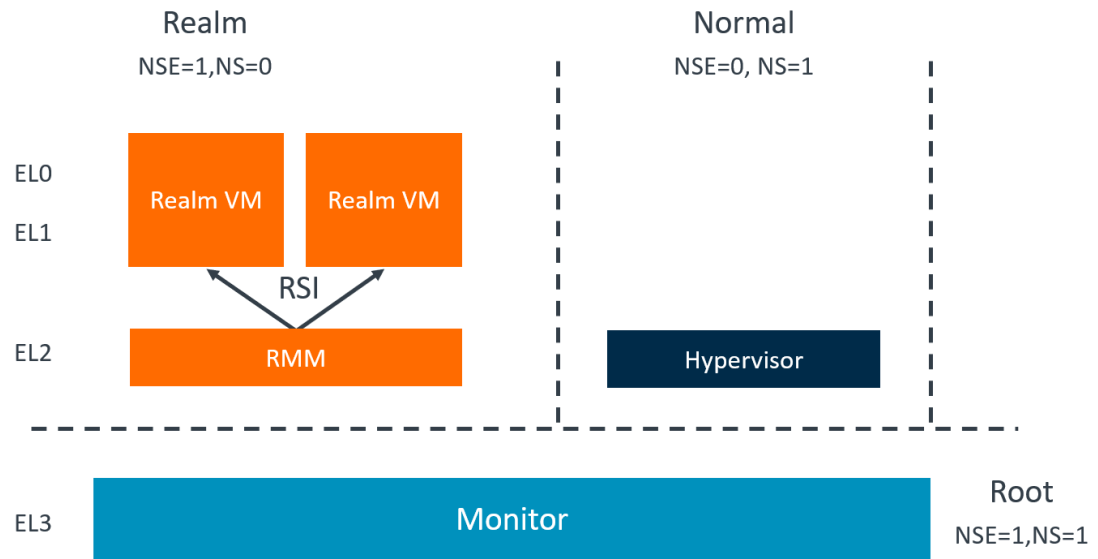


Figure 9: Realm Service Interface

## 6 Check your knowledge

### **What does attestation mean in the context of Arm CCA?**

There are two parts to an Arm CCA Realm attestation, platform attestation and Realm attestation. Platform attestation proves the status of the underlying firmware and silicone through a hardware-based entity. Realm attestation is a check on the initial state of the Realm.

### **What is the final check for allowed access to Physical Address Spaces in an RME based system?**

An RME based system adds the Granule Protection Check after the VA to PA translations have all been completed. These are managed by the Monitor Firmware against the Granule Protection Table created by the Monitor firmware.

### **What are the two interfaces that the Realm Management Monitor presents?**

The first interface is the Realm Management Interface for communication with the controlling Host.

The second interface is the Realm Service Interface, which allows the Realm Management Monitor to take requests for services from any Realm that it is controlling.

### **What is the final stage of the translation process to generate the Physical Address Space?**

A Granule Protection Check, to ensure that the access to the Physical Address Space does not contravene the Arm CCA requirement for Realm PAS or Secure PAS access.

## 7 Related information

Here are some resources that are related to the material in this guide:

- [Arm community](#)
- [Confidential computing](#)

## 8 Next steps

This guide provides an overview of the Arm Confidential Computing Architecture extensions and the hardware and software implementation required to develop an Arm CCA platform.

To learn more about the Arm A-profile architecture, see the [Arm Architecture Reference Manual Supplement Armv9, for Armv9-A architecture profile](#).