Learn the architecture - Introducing Arm Confidential Compute Architecture

Version 4.0

Non-Confidential

Issue

den0125_400_en

Learn the architecture - Introducing Arm Confidential Compute Architecture

Release information

Document history

Issue	Date	Confidentiality	Change
400	19 March 2025	Non-Confidential	New update for CCA Arch overview including Planes and use cases
0300-02	14 June 2023	Non-Confidential	Minor update to fix PAS Access NS/NSE table
0300-01	14 June 2023	Non-Confidential	Minor update to fix broken link
300	9 May 2023	Non-Confidential	Update to add DA and MEC
200	24 March 2022	Non-Confidential	markdown migration

Proprietary Notice

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.

Product Status

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

Feedback

Arm welcomes feedback on this product and its documentation. To provide feedback on the product, create a ticket on https://support.developer.arm.com

To provide feedback on the document, fill the following survey: https://developer.arm.com/documentation-feedback-survey.

Inclusive language commitment

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

We believe that this document contains no offensive language. To report offensive language in this document, email terms@arm.com.

Contents

1. Overview

This guide describes:

- Tthe role of confidential computing in modern compute platforms
- The principles of confidential computing.
- How the Arm Confidential Compute Architecture (Arm CCA) enables confidential computing in an Arm compute platform.

After reading 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
- Describe the difference between execution in a Realm and execution of a TrustZone Trusted Application
- Explain how Realms are created, managed, and executed on an implementation of the Arm CCA
- Explain how a Realm owner establishes trust in a Realm
- Understand use cases and usage scenarios for implementing Arm CCA

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 agents.

Any application or Operating System executing in a Confidential Computing environment executes in isolation from any non-trusted agent in the rest of the system. Any data generated or consumed by the isolated execution cannot be observed by other actors executing on that platform, without explicit permission from the owner of the data.

Arm CCA requirements

An application developer wants to deploy workloads securely without having to trust the underlying software infrastructure, for example the hypervisor or code running in the Secure world.

To support application developer workloads, 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 requires the execution environment to have its own Chain of Trust, independent from the Chain of Trust that the parallel untrusted environments in the platform use.

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

The relationship between Arm CCA and Arm RME

Arm Realm Management Extension (RME) is an extension to the Arm A-profile architecture, introduced in version 9.2.The following sections describe the features of the RME. The RME provides the hardware primitives necessary to support Confidential Compute on Arm.

Arm CCA is a complete system architecture, including firmware components which use RME in order to deliver Confidential Compute functionality.

Attestation

Workloads running inside Realms manage confidential data or run confidential algorithms.

For a Realm Owner to be considered confidential, any workload needs to:

- Be sure it is running on a real Arm CCA platform rather than a simulation
- Know that it has been loaded properly and not been tampered with.
- Know that the overall platform, or the realm is not in a debug state that could leak its secrets.

The process of establishing the trust in a platform is called Attestation.

Attestation is broken into key parts:

- Attestation of the platform

- Attestation of the initial state of the Realm

These parts combine to create attestation reports. A Realm can request attestation reports at any time. You can then use reports to authenticate the validity of the platform and the code in the Realm.

Platform Attestation involves establishing and evaluating the identity and configuration of the platform which hosts the Realm. This platform includes all hardware and firmware components which can impact Realm security. This creates requirements on the hardware.

The hardware must be provisioned with an identity. The hardware must 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.

The Realm attestation token describes the initial configuration and contents of the Realm, including its memory contents and register state.

Attestation of a Realm involves actors outside the CCA platform. Readers are encouraged to refer to the Remote Attestation Specification (RATS), which provides a model for reasoning about the roles and responsibilities of these actors.

The local platform, running the CCA workload, uses the RME as this guide and the RME guide (DEN0126) describe.

3. Arm CCA Extensions

As What is Confidential Computing? describes, Arm CCA aenables you to deploy applications 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 a non-CCA platform, a privileged software entity, for example a hypervisor, does have access to the memory of an application or VM.

The Arm CCA enables the hypervisor 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. 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 firmware which manages Realms (the RMM) and the EL3 Monitor both make use of RME. We describe these elements in Arm CCA Hardware Architecture and Arm CCA Software Architecture.

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).

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

The Host can allocate and manage resource allocation. The Host manages 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 described in Arm CCA Software Architecture.

Realm world and Root world

The Armv8-A TrustZone extensions enable secure execution of code and isolation of data by having two separated worlds:

- The Secure world

- 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 (PAS) a PE can access. In the Secure state a PE can access Secure and non-Secure physical address spaces. In the Non-secure state it can only access the Non-secure physical address space. Normal world refers to combination of Non-secure state and Non-secure physical address space.

RME introduces two additional worlds:

- Root world refers to the combination of 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 PAS is separate from the Secure PAS. 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 PAS. The latter is still used by S_EL2/1/0. The Monitor runs in the Root world.
- Realm world comprises of Realm security state and Realm PAS. Realm state code can execute at R_EL2, R_EL1 and R_EL0. The controlling firmware running in the Realm world can access memory in the Normal world to enable shared buffers.

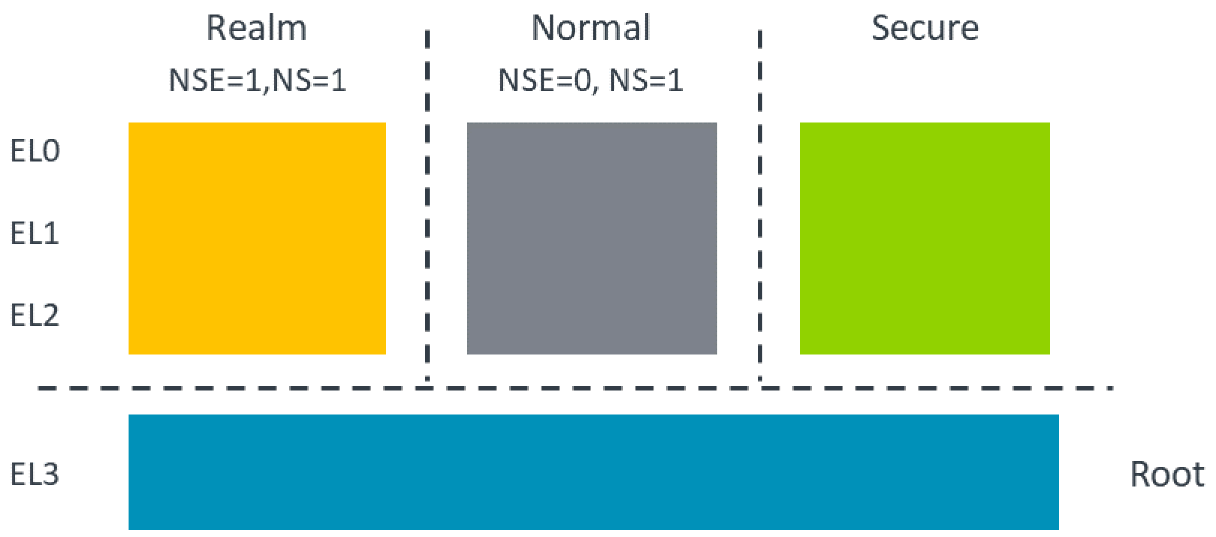
The SCR_EL3.NS bit controls World switching in a non-RME PE. Exception level 3 software sets NS = 0 when switching to the Secure world and sets NS = 1 when switching to the Normal world. World switching in an RME-implemented PE is extended through a new SCR_EL3.NSE bit.

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

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

The following diagram shows the four worlds, and their relationship to the SCR_EL3 NS and NSE bits:

Figure 3-1: RME-based worlds



The Root world enables 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 enable full control of Realm creation and execution, the Arm CCA system requires the RMM, which is part of the firmware that is required to manage Realm creation and execution under requests from a Normal world Host

For more details, see Arm CCA Hardware Architecture and Arm CCA Software Architecture.

Realm Planes

Realm Planes enable Realms to be subdivided into a set of execution environments which share the same IPA to PA translations, but which can have different memory access permissions. Each Plane has its own register context within the Realm Execution Context (REC). You could consider A REC as a Virtual Processing Element (vPE). Each Plane is an isolated execution context.

Realm Management Monitor specification v1.1 describes the use of Planes, the specification can be found here RMM Specification v1.1.

Realm Planes remove the need to have multiple Realms with connected functionality and all the overhead this entails. All the functionality can reside in a single Realm with the Execution Context isolated in each Plane as required and this only requires one Realm being created and controlled by the RMM.

Planes allow security services required by a Realm to be implemented inside the Realm itself, but isolated from the main guest OS.

Realm Planes are partitioned at Realm creation. A Realm can have several Planes within the REC numbered from P0 to Pn.

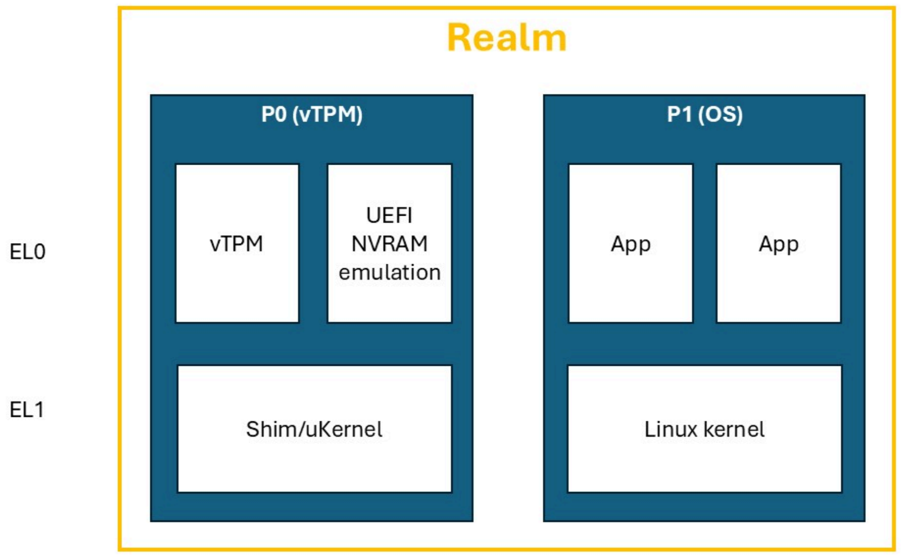
The number of Planes is specified at Realm creation and fixed thereafter. The first Plane (referred to as P0) has additional capabilities which allow it to:

- Control entry into other Planes ("Pn" is used as a shorthand to mean "any Plane other than P0")
- Handle exceptions taken from Pn
- Context switch register state of Pn
- Manage memory access permissions for Pn
- Configure traps taken from Pn
- Either emulate an interrupt controller for Pn, or pass through to Pn interrupts injected by the Host
- Issue RSI commands to interface with the RMM

The following diagram shows a simple use case for Realm Planes. In this example, the Realm contains two Planes:

- P1 contains the main guest OS, in this example based on Linux
- P0 includes a Virtual Trusted Platform Module (vTPM) emulation, which can be used to record the state of P1, including measurements of each of P1's boot stages

Figure 3-2: Realm planes example



A REC executes in a single Plane at a time. Exception handlers control starting and yielding of Plane execution.

There are two options for management of per-Plane memory access permissions:

1. On platforms which implement FEAT_S2POE and FEAT_S2PIE, a single stage 2 translation table can be used by all Planes within a Realm, with per-Plane access permissions being specified indirectly.
2. Alternatively, each Plane can have its own stage 2 translation table, with the RMM ensuring that IPA to PA translations are identical while per-Plane access permissions can be different.

What is the difference between Arm TrustZone Extensions and Arm RME?

All Arm A-Profile processors have can the Arm TrustZone architecture extensions. These extensions enable 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 enable you to manage multiple Secure Partitions in the Secure world. This feature can enable you to apply multiple TOSs 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 is the SPM. This means that the TOS relies on the relationship with the higher privilege firmware developer.

The following methods 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 TOS. Secure type 1 interrupts execute the TOS. A secure type 1 interrupt asserted during Normal world execution calls the TOS through the Monitor.

A Realm VM is different to a TOS or Trusted application because the Realm VM is controlled from the Normal world Host. In areas such as 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 TOS 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.

It is expected that the software entities which make use of each of Realms and TrustZone will differ in terms of the relationships between the software vendor and the platform owner.

A TOS controls execution of Trusted applications that are used for platform-specific services. The Trusted applications are developed during the platform development,by developers such as Silicone Providers (SiPs) and Original Equipment Manufacturers (OEMs). The Trusted applications are very closely tied to the chain of trust for the platform from the platform boot.

Realm execution is intended 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 enables 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.

Information security is often described in terms of the following three principles:

- Confidentiality: data can only be observed by authorized users or processes.
- Integrity: data can only be modified by authorized users or processes.
- Availability: data or a service can be accessed in a timely manner by authorized users or processes.

A TOS can provide a guarantee of Confidentiality, Integrity and Availability. Arm CCA provides Realms with a guarantee of Confidentiality and Integrity, but not Availability.

The four-world environment provided by the Arm CCA system enables the complete separation between the Secure world and Realm world. This allows the entities within each of Secure world and Realm world to be independently deployed and mutually distrusting.

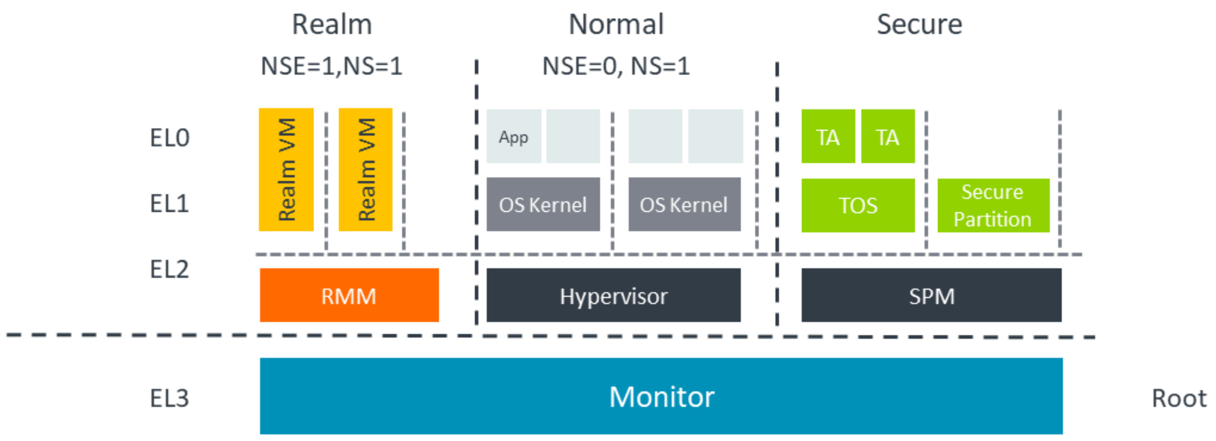
4. Arm CCA Hardware Architecture

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

Realm world requirements

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

Figure 4-1: Realm world software execution



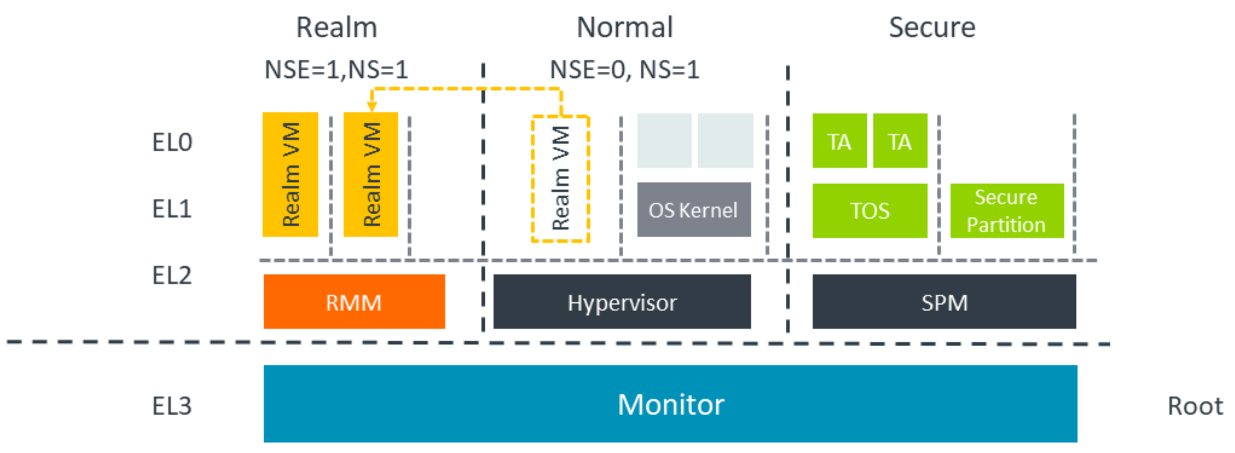
The Realm world must be able to execute code, and access memory and trusted devices, in complete isolation from all other non-Root worlds and devices.

Similar to 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 [Arm CCA Software Stack] describes the RMM.

Isolation between worlds is enforced by hardware through faulting exceptions and by physical memory encryption.

The following diagram illustrates that a Realm VM executes in the Realm world, but its resources are managed by the hypervisor executing in Normal world. This management is performed via requests sent by the hypervisor to the RMM.

Figure 4-2: Realm VM execution



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. The Monitor allows communication and control where needed.

Memory management for Arm CCA

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

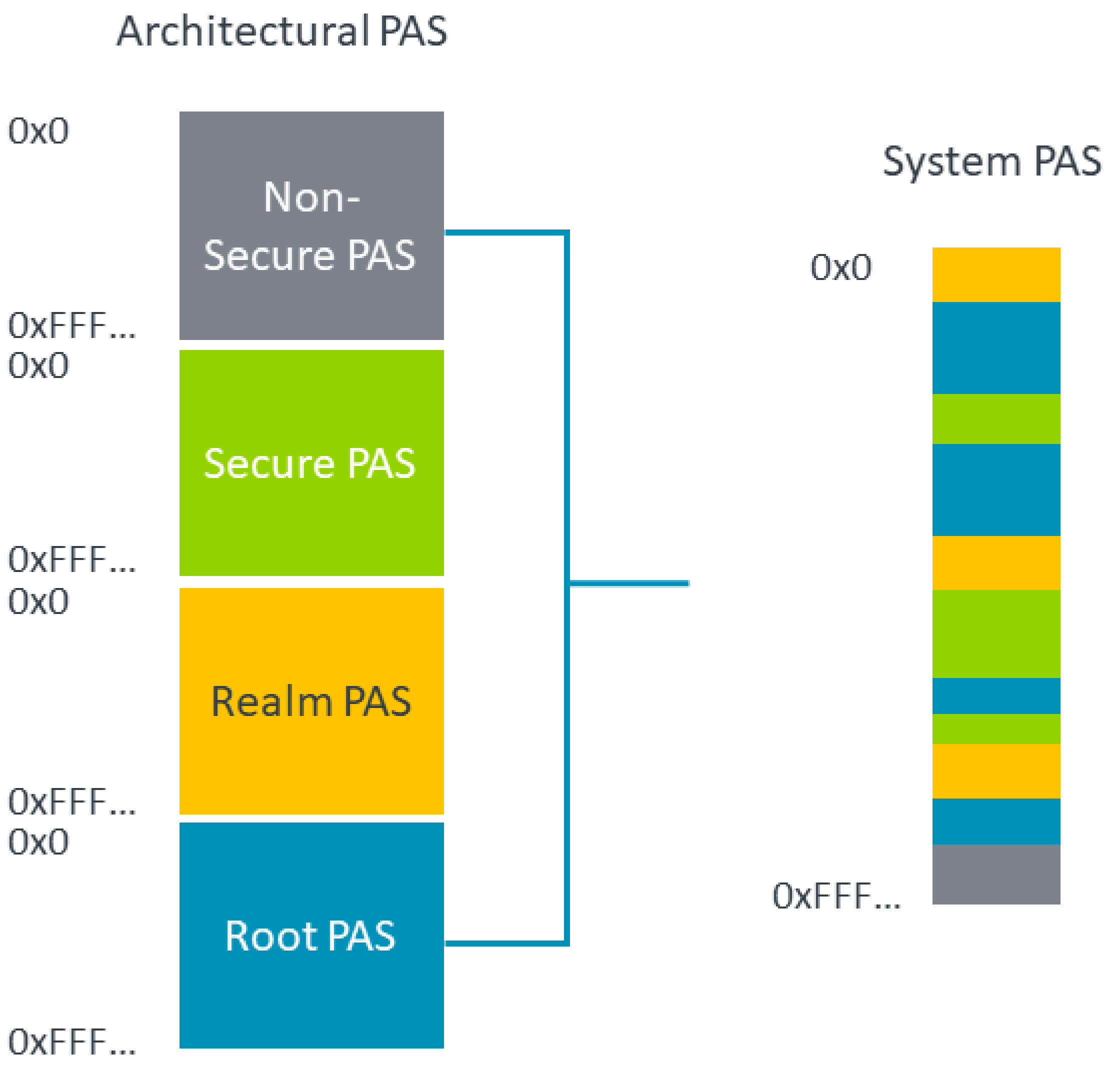
- Non-secure physical address space
- Secure physical address space

The Realm Management Extension add another two PAS:

- Realm physical address space
- Root physical address space

Each location within physical memory can be made accessible via a single PAS. This is illustrated in the following diagram.

Figure 4-3: Physical address spaces



TDepending on the Security state in which a PE is executing, a subset of PAS are accessible. The subset is shown in the following table.

Security state	Non-secure PAS	Secure PAS	Realm PAS	Root PAS
Non-secure	Yes	No	No	No
Secure	Yes	Yes	No	No
Realm	Yes	No	Yes	No
Root	Yes	Yes	Yes	Yes

The PAS via which each physical location is accessible is controlled by Root security state. This means that firmware executing in Root security state is effectively able to transfer ownership of physical memory between each of Non-secure, Secure and Realm worlds.

To ensure that the isolation rules for all worlds are enforced, the physical memory access controls in the preceding table are enforced by the Memory Management Unit (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). Root state controls the enablement of the GPC, the contents of the GPT and the routing of a GPF.

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.

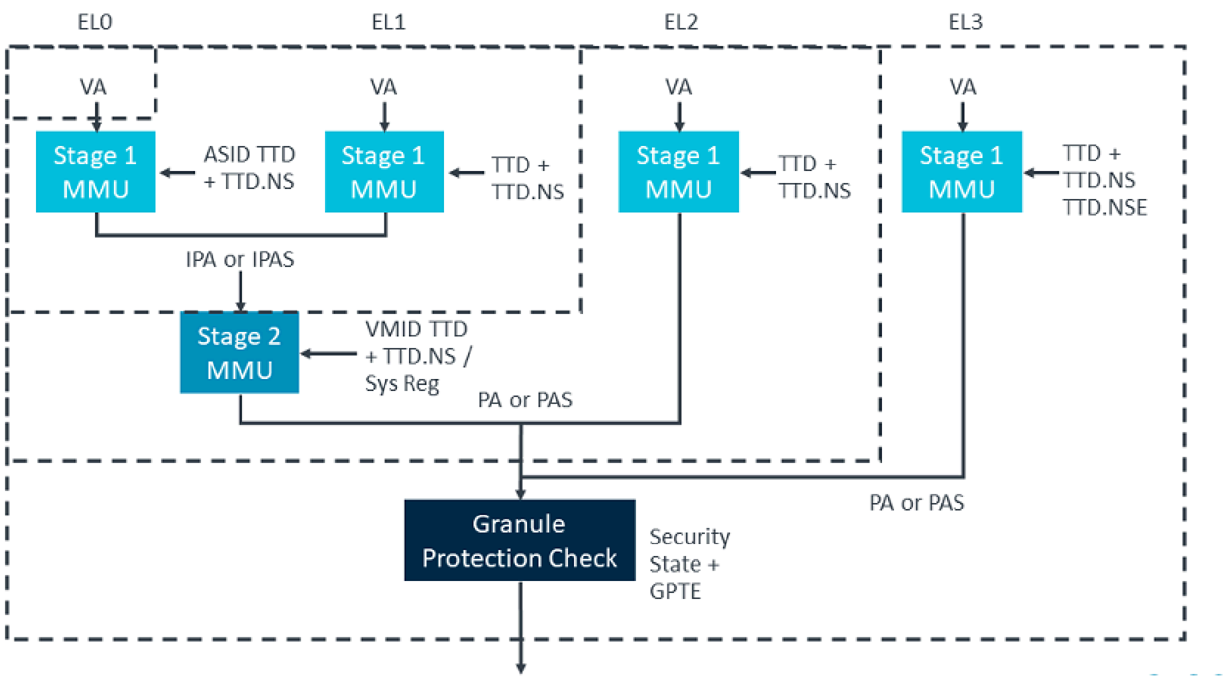
Within the EL3 translation regime, RME introduces an additional bit in the translation table descriptor, called NSE. Using NSE together with the existing NS bit, EL3 software can map any of Root, Non-Secure, Secure and Realm PAS.

Within the Realm EL1&0 translation regime and the Realm EL2&0 translation regime, the NS bit allows software at R-EL2 to map either Realm PAS or Non-Secure 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 position of the GPC in the logical sequence of steps which occur during translation of a Virtual Address (VA) to a Physical Address (PA). In this diagram, TTD is Translation Table Descriptor and GPTD is Granule Protection Table Descriptor:

Figure 4-4: 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.

RME adds the 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 can also be included in this check if they are connected to a Requester side filter like a System MMU (SMMU).

5. Arm CCA Software Architecture

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

Software Stack

The actions performed by software to manage a Realm can be divided into two parts:

- Policy: decisions regarding which resources (memory; processor cycles; devices) to allocate to a Realm. This is the responsibility of the hypervisor.
- Mechanism: enacting a policy decision requires changing architectural state, such as modifying a translation table, or context-switching the state held in a processor register.

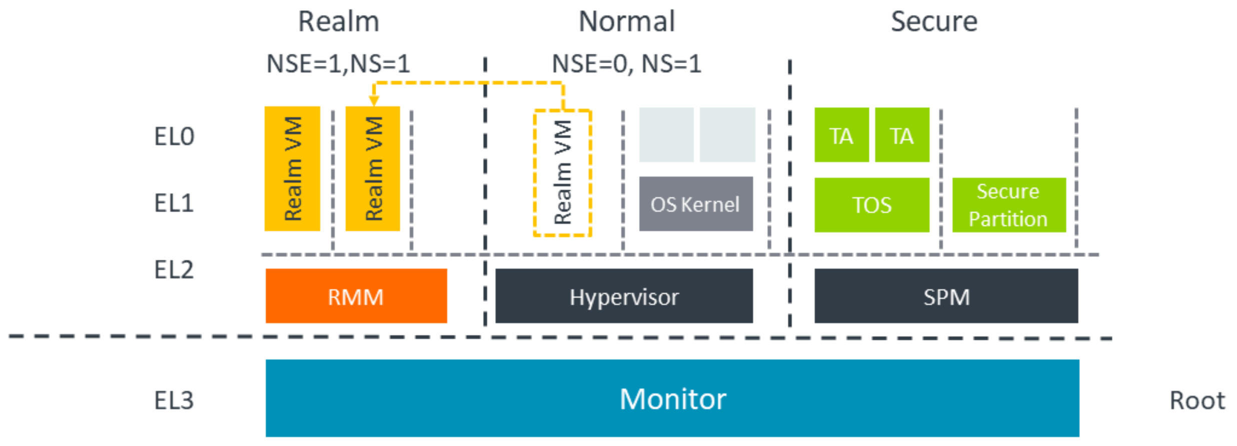
When this action relates to a Realm and the hypervisor cannot be trusted to perform it, the RMM performs the action for the hypervisor.

The RMM isolates 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 has fully defined functionality to manage the requests from the Host and Realms. Because this interface is 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 5-1: 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 enable 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.

Realm Management Monitor

The RMM is the Realm world firmware that manages 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.

In the Arm CCA system, the RMM provides services to the Host, to enable 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 Host owns all the policy decisions, including 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:

- 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.

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

TThe RMM specification defines two interfaces:

- The Realm Management Interface (RMI) used by the Host
- The Realm Services Interface (RSI) used by the Realm.

Each of these interfaces comprises a set of SMCCC-compliant commands.

Arm CCA additionally defines an interface called the Realm Host Interface (RHI) used for communications between the guest and the hypervisor. The RMM is not used for the RHI communication.

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

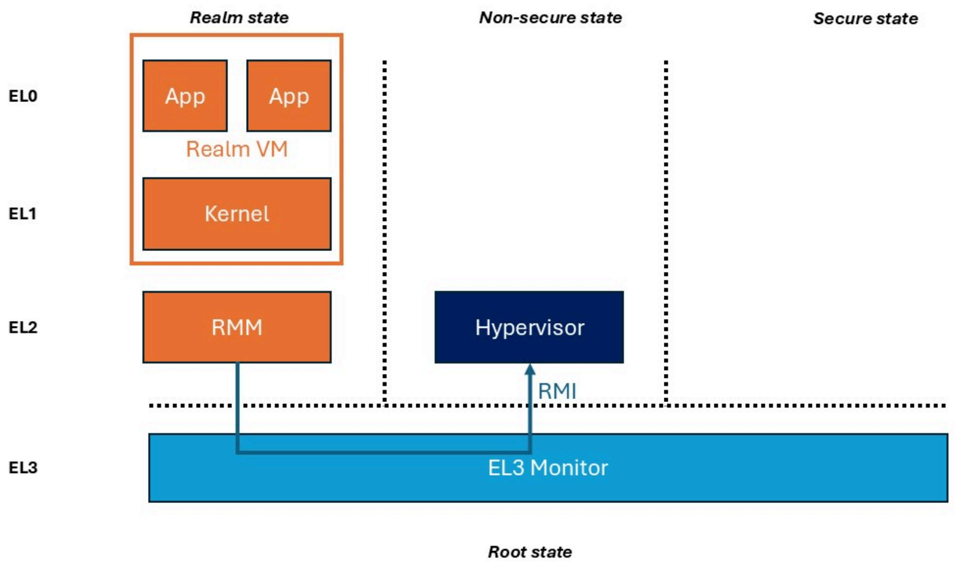
Realm Management Interface

The RMI is the interface between the RMM and the Normal world Host.

The RMI enables control of 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 5-2: Realm management interface route



Realm Services Interface

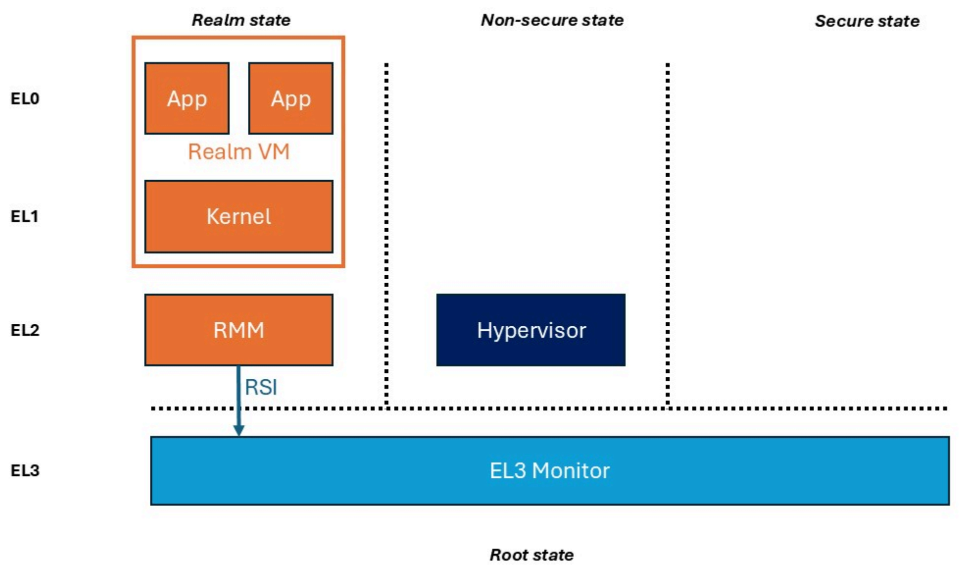
The Realm Service Interface (RSI) is the interface between the Realm VM and the RMM.

The RSI allows the Realm to request services including:

- Requesting an attestation report describing the CCA platform and the Realm’s initial state
- Managing properties of the Realm’s address space, including sharing memory with the Host
- Attesting and accepting devices which are assigned to the Realm

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

Figure 5-3: Realm service interface



Realm Host Interface

The Realm Host Interface (RHI) is the interface between the Realm and the untrusted host hypervisor in the non-secure world.

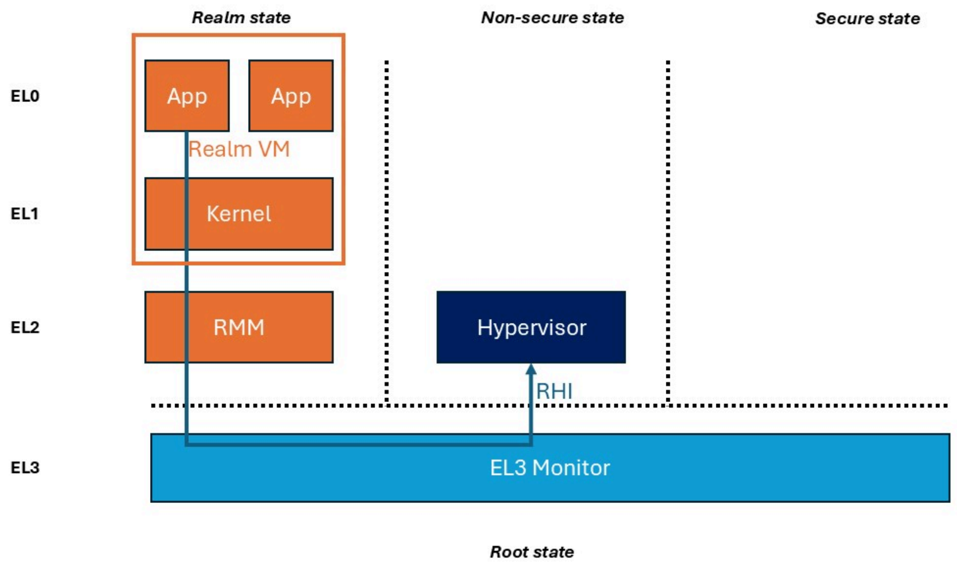
The RHI enables the host to request Realm management functions through a trusted interface.

The RHI provides a standard interface for a Realm to request services from the Host, where those services do not require participation of the CCA firmware (RMM and EL3 Monitor). Examples of such services include:

- Early Provisioning of secrets to a Realm during guest boot.
- Discovery of Host-imposed constraints on the granularity that memory can be shared between Realm and Host.
- Retrieval of attestation evidence related to Realm-assigned devices. The attestation token is stored by the Host and secured via hashes held by the RMM.

The following diagram shows how RHI is used by the guest to request services from the Host. Although the RMM and EL3 Monitor are not participants in this communication, they are part of the transport for RHI commands.

Figure 5-4: Realm Host Interface routing



6. Device Assignment and Memory Encryption Contexts

Previous sections of this guide showed how Realms provide an execution environment that is completely isolated from the Normal world Rich OS, Hypervisor, and TrustZone. A Realm can be fully attested at initialization to guarantee its initial content and that it is running on an RME-based platform.

Many Realm workloads need to make use of devices - including storage, networking and acceleration - in order to deliver the required performance. By default, all devices in the system are blocked from accessing the contents of all Realms. If a Realm needs to make use of a device, it must first attest the device, and then provide its consent to the RMM for that device to be granted access to the Realm’s memory.

This device assignment process upholds the security guarantees provided by the Confidential Compute Architecture. Using standard RME capabilities, isolation can be guaranteed for on-SoC peripherals that are not DMA-capable and have fixed address ranges. Fixed address reanges are set using RMM page tables and either a GPT or a completer-side PAS filter to permit or deny access to that memory region.

However, some device systems, such as a PCIe Root Port in a system with PCIe, have multiple devices controlled and accessed through a single memory region. This means that memory access control alone is not enough. Also, the device will have its own caches which need to be managed.

Device Assignment

Hardware devices often support multiple interfaces that can be assigned to individual virtual machines. The host hypervisor controls the assignment process, resulting in a hardware interface for the device that can be directly managed by a specific virtual machine. The typical properties of a device assignment architecture are as follows:

- It relies on standardization. This enables the host to assign the device to a VM without needing a device-specific driver, for example a GPU driver.
- The MMIO for the device interface is accessible to the VM it is assigned to. This means that the VM can directly interact with the device interface. The host programs the PE MMU to enable this.
- Direct Memory Access (DMA) from the device interface can access the VM’s memory. The host programs the SMMU to enable this.
- The VM can handle interrupts from the device interface . The host ensures that these interrupts are forwarded.

PCI SR-IOV is the most used device assignment architecture

Device Assignment under CCA

Device Assignment (DA) enables a device to be uniquely assigned to an individual Realm, and to allow the Realm to attest the device before granting it access to the Realm’s contents. DA prevents agents which are untrusted by the Realm, including other Realms and the hypervisor, from accessing or controlling an assigned device. This includes protection of device MMIO interfaces and device caches. DA is a standard mechanism for attesting all device types without the RMM having any special knowledge of the devices being attested, whether static memory map or behind a PCIe Root Port. This is RME Device Assignment (RME-DA).

Before to the introduction of RME-DA, DMA from a device to Realm memory was not allowed by the RME architecture. In this situation, both the device and the memory which it is accessing are untrusted by the Realm. Therefore, the Realm must apply appropriate security measures such as encryption and authentication to any data which it exchanges with the device. RME-DA is an addition to the RME architecture that overcomes this limitation and describes how device interfaces are assigned to Realms. At a functional level, RME-DA, has the same properties as assignment to VMs:

- The methodology for assignment relies on standardization that allows it to be device agnostic
- The Realm can access the MMIO of the device interface
- Device interfaces can perform DMA to the assigned Realm's memory
- Interrupts from the device interface can be handled by the Realm. Depending on the interrupt controller hardware architecture. This may require actions to be taken by the Host, in which case correct interrupt delivery cannot be relied upon by the Realm.

Although functionally RME-DA is similar to device assignment to VMs, RME-DA also upholds the Realm security model. This results in the following security model and security guarantees:

- A Realm can decide based on an attestation process whether it accepts a device interface into its Trusted Computing Base (TCB). A device interface that is accepted by a Realm can access the Realm's memory. However, any device interface not accepted by a Realm must not have any access to the memory regions used by that Realm.
- All Realm data is protected from non-Secure state.
- For devices that support multiple interfaces, a Realm must trust the device to isolate the device context it holds for each interface.
- Where a device is discrete and not on the same SoC as the host, the physical link between them, used to communicate Realm transitions, must be protected against physical attacks. The link protection scheme must provide confidentiality, integrity, and replay protection.

To achieve this security model, the host architecture must provide the necessary protections and mechanisms. RME-DA describes this architecture and is composed of two parts:

- Arm system architecture specifications:
    - SMMU Architecture for RME-DA
    - RME System architecture
- PCI specifications:
    - TEE Device Interface Security Protocol (TDISP) specification
    - Integrity and Data Encryption (IDE)

The RMM manages the life cycle of device assignment and is implemented using TDISP (TEE Device Interface Security Protocol). The host platform protects the RMM and Realms from devices which have not been accepted through attestation.

Arm system architecture specifications

RME-DA extends the SMMU specification to enable support for DMA into Realm memory, by adding a Realm world programming interface. This allows the RMM to manage Realm streams that are aimed at device DMA to Realm memory. This includes support for the MEC feature described below.

Support is included for devices that make use of translated transactions. These devices cache address translation locally within the device, and present physical addresses to the memory system. A malicious device interface can potentially present physical addresses for memory that is outside the boundary of a Realm to which it is assigned. So that the host can prevent this, a new structure, the Device Permission Table, is included in the SMMU architecture. This structure associates a device interface with a specific Realm and checks that transactions from that device interface are within the memory footprint of its associated Realm. The DPT is only required on devices that support translated transactions, for example PCI-ATS.

Also, RME-DA extends the RME system architecture to provide requirements for PCI Root Ports and interconnects. These requirements standardize RP features to enable a platform-independent RMM, resolving gaps left by PCI TDISP and associated specifications. See PCI TDISP and device attestation and assignment for more information. RME-DA system architecture also describes the architectural mapping between Arm memory system and PCI transaction attributes, and defines requirements for RCiEP devices.

PCI TDISP and device attestation and assignment

The TEE Device Interface Security Protocol (TDISP) provides a standardized way to manage the life cycle of assignment of a device interface. This lifecycle includes the ability for a Realm to attest the device. Based on the data obtained a Realm can decide whether to accept a device into its Trusted Computing Base (TCB). The TDISP reference architecture defines a set of components, their roles and responsibilities, and standard interfaces through which they can communicate. The following table summarizes the TDISP components, and the corresponding component in CCA.

TDISP component	Description	CCA component
Trusted Virtual Machine (TVM)	The TEE-hosted VM which makes use of an assigned device.	Realm
Device Security Manager (DSM)	A logical entity in the device which enforces device security policy, including ensuring isolation between (potentially multiple) interfaces exposed by the device.	Device firmware
Trusted Device Interface (TDI)	Device interface which can be assigned to a Realm.	Interface exposed by device firmware
TEE Security Manager (TSM)	Manages assignment of a TDI to a TVM, by communicating with the DSM.	RMM, executing at R-EL2
Virtual Machine Monitor (VMM)	Untrusted agent which orchestrates TVM creation and resource management.	VMM, executing in Non-secure state

TDISP relies on a set of associated protocols. The following table summarizes.

Protocol	Description
Data Object Exchange (DOE)	Transport layer for communication between TSM and DSM. Implemented in PCI config space.
Security Protocol and Data Model (SPDM)	Establishes a secure channel for communication between TSM and DSM. This can be implemented over DOE.
Integrity and Data Encryption (IDE)	Provides protection of the stream of data between PCI Root Port and device.
IDE Key Management (IDE-KM)	Establishes shared keys used to provide IDE. Programmed over SPDM.

Device assignment flow

The process of assigning a TDI to a Realm is as follows:

1. The TDI starts off in an unlocked state. In this state VMM can access and modify the TDI's configuration space. The VMM configures the TDI ready for assignment to a Realm.
2. The VMM passes control to the RMM (TSM) and the RMM establishes secure communication channel using SPDM with the DSM
3. Using this communication channel, the RMM transitions the TDI to a locked state. At this point the VMM can no longer change the TDI's configuration. At this stage, IDE is configured.
4. The RMM queries the DSM to obtain TDI configuration and device authentication information. This information is stored by the Non-secure Host, with integrity protected via a digest stored by the RMM.
5. The Realm retrieves the device attestation evidence, and establishes its integrity. The Realm decides on the basis of this data whether to accept the device into its Trusted Computing Base (TCB). The Realm communicates its decision to the RMM. The IDE configuration is checked.
6. If the TDI is accepted, the RMM does the following:
   a. Where the device is integrated into the SoC, the RMM sets up an IDE stream between the RP and the device
   b. Programs a Realm stream in the SMMU and associates it with stage 2 page tables of the Realm
   c. If the device is using ATS then the DPT is also programmed to associate granules in the Realm that can be accessed by the TDI with the VMID of the Realm
7. The RMM transfers the device into a running state and enables the SMMU stream. At this point the device can perform DMA into the Realm's address space

The DSM monitors the configuration and state of the TDI components and can transition the TDI into an error state if threats or security violations are detected while in the locked configuration or running states. In the error state you will not receive any transactions from the TDI, you can only reset the TDI back to an unlocked state. As part of that reset transition any confidential information associated with the TDI that is held in the device is cleared.

IDE key programming

The IDE keys, used to protect the physical link between the SoC and the device, are generated by the CCA platform. Key programming at the device end of the IDE stream is performed via the Secure SPDM stream which was negotiated between the RMM and the DSM. Key programming at the Root Port is performed via the EL3 Monitor using an IMPDEF interface.

Disconnecting the Realm

In order to reclaim a TDI which has been assigned to a Realm, the Host sends a request to the RMM, which in turn sends a request to the DSM for the TDI to exit the running state. In response, the DSM ensures that the TDI does the following:

- Aborts any existing accepted operations
- Completes or aborts any existing transactions
- Stops presenting interrupts
- Stops presenting Address Translation Service (ATS) and Page Request Interface (PRI) requests, if either are being used

Also, the device scrubs any internal state associated with the TDI, and invalidates translations cached for that TDI if ATS is in use. When these operations are competed the TDI is transferred back to the unlocked state.

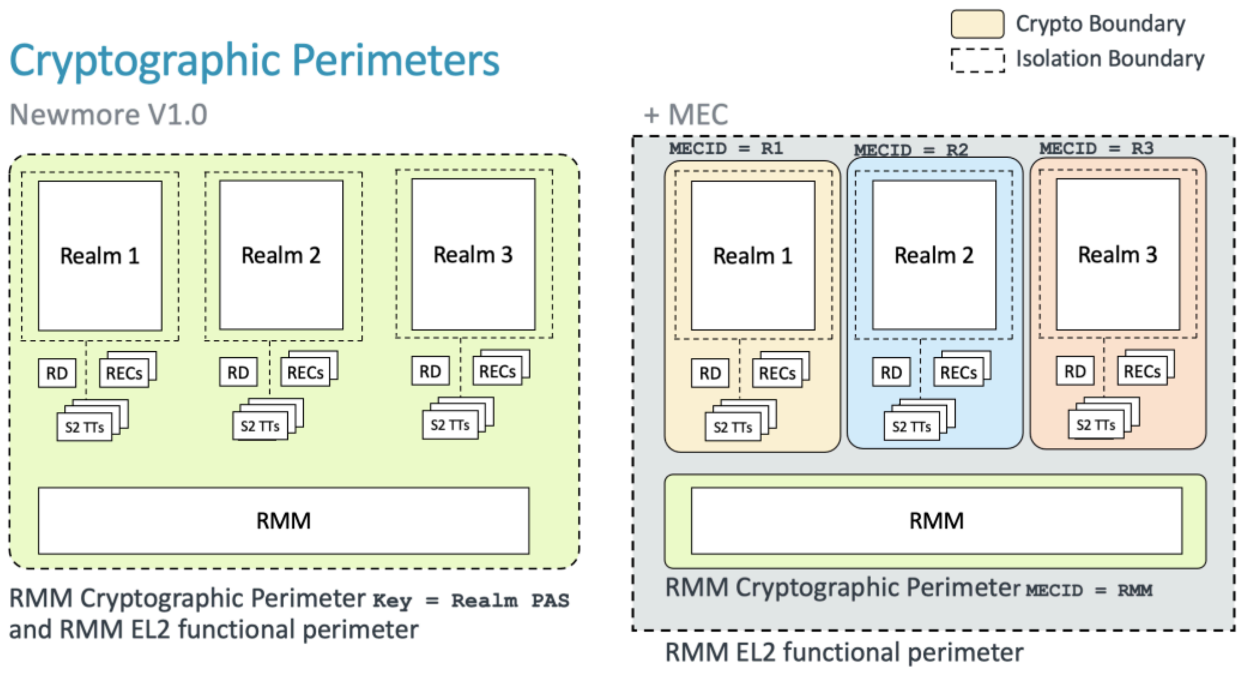
Memory Encryption Contexts

Memory Encryption Contexts (MEC) are configurations of encryption that are associated with areas of memory, assigned by the MMU.

MEC is an extension to the Arm RME. The RME system architecture requires that the Realm, Secure, and Root PASes are encrypted. The encryption key or tweak, or encryption context, used with each of these PASes is global within that PAS. So, for example, for the Realm PAS, all Realm memory uses the same encryption context. MEC allows more fine-grained control of memory encryption, and for the Realm PAS specifically, each Realm has a unique encryption context. This provides additional defense in depth to the isolation already provided in RME. Realms and the RMM can all have separate encryption.

MECIDs are identifying tags that are associated with different Memory Encryption Contexts. MECIDs are assigned to different software entities in the system, for example, Realms or the RMM. The following figure shows where each Realm has an individual MECID:

Figure 6-1: Comparison between non-MEC and MEC


Cryptographic Perimeters
Crypto Boundary
Isolation Boundary
Newmore V1.0
Realm 1
Realm 2
Realm 3
RD
RECs
S2 TTs
RMM
RMM Cryptographic Perimeter Key = Realm PAS and RMM EL2 functional perimeter
+ MEC
MECID = R1
MECID = R2
MECID = R3
RMM Cryptographic Perimeter MECID = RMM
RMM EL2 functional perimeter

The Memory Protection Engine (MPE) maps each MECID value to a corresponding encryption context, which typically consists of an encryption key or tweak. Encryption contexts are initialized at system boot and can be updated, on request from Root world, when a MECID is assigned to a different software entity.

The MECID is assigned by software (the RMM for Realm PAS), using a combination of system register values and page table bits. MEC allows bodies of software, Realms, and the RMM to use more than one MECID register, with page table bits used to select which of the MECID values applies to a specific memory region mapped to that body of SW. This, for example, enables the RMM to use one MECID register to manage its own data structures and another to manage those of the Realm it is currently managing, such as the Realm's stage 2 page tables. In addition, every Advanced Microcontroller Bus Architecture (AMBA) transaction coming from any entity with a MECID is annotated with that entity's MECID attribute.

When MEC is enabled, the MECID associated with a transaction is used to retrieve a tweak or encryption key. At system boot, a set of encryption contexts is generated and stored by the Memory Protection Engine (MPE).

The MECID size is between 1 and 16 bits and is architecturally discoverable. Because the size of the MECID is limited, this means that during the lifetime of the running system MECIDs will be reused between different entities in the system. Therefore, every assignment of a MECID must happen only after the MECID's associated encryption context is invalidated and then it can be regenerated.

7. Confidential Compute in use

The need for Confidentiality in Compute systems spans many market segments. Confidential Compute enables shorter trust chain relationships for application and OS developers and will ease the certification load.

The confidentiality afforded within a Realm VM enables different services to be isolated from the Normal world execution, while being controlled from the Normal world for setup and execution.

7.1 Use cases

The following sections describe some of the use cases and how they are set up in an Arm Confidential Compute platform, with pointers to any examples, where available.

The use cases will describe what is being enabled and what it allows an end user to achieve while showing which CCA features the use case exercises.

Boot and attest a Realm

Booting and attesting a Realm is the very basic task of initiating and booting a Realm on a platform. For CCA this should be from an attested and trusted platform.

Boot and attest use case enables an end user to attest that a Realm has been created correctly, on a trustworthy CCA platform

The CCA features that are exercised by this are:

- Realm creation
- Realm execution
- Remote attestation of CCA platform and Realm

The Boot and Attest use case is responsible for creating a Realm on an attested platform and then executing code within that Realm in isolation from the Non-secure and Secure worlds. The Realm is created under control from the Normal world hypervisor using the RMI interface to create the isolated VM for execution.

This example use case should be used as a starting point for other CCA use cases. Although the Realm is a protected execution environment, its contents are, by definition, entirely non-confidential due to being input from a non-confidential source media. Therefore, this should be used as a starting point for any further CCA use cases.

The following link shows the Arm Learning Path for how to create a Realm in the Arm Software stack:

Create a Virtual Machine in a Realm

Boot a Realm from an encrypted disk image -

Cloud service developers will want to provision each Realm with minimal firmware to enable standard boot of a Linux based encrypted Root FileSystem (FS). Owners of Cloud workloads must ensure their data and workloads are provisioned after the Root FS has been attested.

Booting a Realm from an encrypted disc builds on the basic Realm creation described in the section above and ensures the Realm is provisioned with confidential information, from the encrypted contents of a disk image.

The CCA features that are exercised in this use case are:

- The functionality described in Boot and attest a Realm
- A Boot Injection Protocol

Arm provids an example of a boot injection reference software with the Boot Synchronisation RHI (RHI-BS) interface. Implementors have the option to:

- use the Arm interface
- develop their own interface code.

The reference software is developed from the Realm Host Interface specification and requirements.

The protocol describes the trusted injection of the disk encryption key following Realm attestation to allow secrets to be exchanged between the Realm application and the Realm owner.

Confidential Compute is not a single point solution. The operation of Secure boot of a Realm on a platform will require extensive cloud infrastructure to enable the platform to attest and execute the boot process.

Live CCA Firmware update

Cloud vendors require the ability to update CCA firmware, such as the Realm Management Monitor (RMM) or the Root Monitor in EL3, in a live CCA system, without having to reset the platform to reboot. This operation is required for all initial CCA platforms.

Arm has developed reference software for this operation which is depends on the Firmware Activity Log from the RHI 1.0 specification and Live Firmware Activation specification 1.0.

All platforms will have their own Firmware update solutions so Arm reference software demonstrating this operation considers the update process optional. Again, the deployment of the firmware update is dependant upon a wider cloud infrastructure.

Per-Realm encryption

A cloud workspace owner will require confidentiality in their data when running in a Realm and would prefer the reassurance of having isolation from all other Realms running on that platform. Use the Memory Encryption Contexts (MEC) to enforce separate encryption keys being used for each Realm. For more information on MEC see Memory Encryption Contexts

Per Realm encryption will be mandatory in early CCA deployments to ensure the confidentiality of separate workload memory.

Implementation is dependent upon the underlying processors supporting the MEC feature (FEAT_MEC) corresponding to the architecture version v9.3. Any platform implementing MEC must include an SMMU which is compliant with the SMMU architecture specification defined in release F.a or later.

The software development must be compliant with the RMM specification v1.1 with the MEC sections applied.

Non-Realm workload Device Assignment (DA) on KVM Host

When running a workload on a cloud there is a requirement to be able to use a PCIe-based DMA engine to access memory from a non-Realm source. The user must be able to assign a PCIe Trusted Device Interface to a workload that is running a Realm with at least one Plane as a Virtual Machine.

The Realm must fully attest the TDI to confirm the identity and configuration of the device functionality before allowing the DMA to access the Realm (workload) memory.

DA use cases require significant support and development to the whole Cloud infrastructure and to the underlying platform with the integration of the PCIe elements before the use cases are able to guarantee full CCA operation. The CCA support for PCIe should include Root Complex Integrated Endpoint (RCiEP) devices and any off-chip devices. Support for Security Protocol and Data Model (SPDM) is not mandatory.

7.2 CCA usage scenarios

The above use cases are effectively building blocks to enable Arm-based platforms to provide the CCA Guarantee for workloads running on the Arm-based cloud platforms. The following sections describe some potential areas in which the use cases can be implemented. This is not an exhaustive list but just a few examples where the RME and use cases can be used in wider system operations.

Operating system vendor services

The use of Arm CCA allows for the execution of individual services in isolation from the Rich OS environment that has been validated (and potentially attested) by the OEM Chain of Trust. This means that an OS vendor can develop services to be run as Realm VMs to ensure they can be executed within strict security requirements.

The basic use of the Arm RME will ensure that all VMs are fully attested on a trusted platform that can also be regularly attested to ensure CCA Guarantees are maintained.

Sensitive data services

Many OS services collect and store sensitive or personal data during their execution. This data can be the target for malicious agents who mine and sell data. Arm CCA allows these services to be executed on any Arm CCA system in which the RME enables the data to be completely isolated from any Normal world interference. This can apply to areas such as:

- Address book data
- Healthcare information
- Browser history and cookies
- Auto-complete systems
- Calendar and email
- Clipboard data

Arm CCA also allows isolation of personal data in AI models which can then allow cloud access to that data and more confidence in off-system analysis of personal data.

User data confidentiality

As OS vendor information in Sensitive data services shows, you can also ensure isolation of user data from the Normal world without needing the many relationships with the OEMs and Trusted OS developers. For users, there is no ability for the platform developer and OS or hypervisor developer to access user data through privilege escalation or hierarchical privilege.

This means that the user can trust their data on a large server system and be assured that their data cannot be accessed by the platform owner. For example, photos can never be seen, and personal works like music files can be stored and executed without the controlling OS or hypervisor, in the Normal world, being able to access the files.

This lack of access can both reassure the user and aid the platform developer. An example is a personally created music file that is stored and executed on a server using a service like Soundcloud, If any copyright issues for the music led to a legal challenge, the platform provider is not responsible for a data leak, and can claim zero liability.

Arm CCA also allows for any service or application developer to run their code and data in an isolated Realm VM. Executing as a Realm VM allows the developer to have a higher level of trust in the execution of his code on any platform supporting Arm CCA.

The following sections show some examples:

Machine Learning models

Machine Learning (ML) and Computer Vision (CV) models for learning and inference will contain IP that requires isolating to stop theft. With the additional requirement stop any chance of replacement when used for security functions like face or other biometric recognition. Arm CCA allows the service to be managed, updated, and executed in any system without the need to trust the Rich OS or hypervisor that is being used.

This also allows service developers to ensure that their own ML service is used for the biometric recognition without having to rely on an OS-based service that they may not have a trust relationship with.

As Memory Encryption Contexts shows, the confidentiality enabled by Arm CCA enables data for AI models to be secure on a personal device which can be accessed and analyzed by cloud services. The user can be confident that the data is secure on their platform and the access from the cloud is fully secured and attested.

Large Trusted services and applications

In many systems the development of the TrustZone enabled Secure world requires on-SoC memory which is required to be smaller in size than the Normal world DRAM memory. Smaller memory makes it harder for trusted applications running in the Secure World to manage large applications - due to memory limitations, lack of dynamic memory allocation and potential certification constraints.

The mechanisms in place for excution of applications in Realm world means large applications can run in a Realm VM with full confidentiality using the RME including DA and MEC.

Bring Your Own Device

With Bring Your Own Device (BYOD), an organization allows users to run services and applications on their personal device. This means that users can access sensitive company data such as email and work calendars.

BYOD use is often allowed with the understanding that, if the device is lost or stolen, the device OS and all data are completely erased. The device is returned to a factory initialized state, so that malicious actors cannot force access to the company sensitive data. This can be an issue for the user, whose personal data must also be erased.

Arm CCA can allow Secure services to be executed on a BYOD device, without the need to completely reset the device and erase all user data if the device is lost or stolen. Realms can be managed externally, and only the Realm world needs to be reset. This configuration gives the BYOD device user more assurance when using organization services.

8. Check your knowledge

Q: What does attestation mean in the context of Arm CCA?

There are two parts to an Arm CCA 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.

Q: 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.

Q: What are the two interfaces that the Realm Management Monitor presents?

The three interfaces that the RMM presents are:

- The Realm Management Interface for communication with the controlling Host
- The Realm Service Interface, which allows the Realm Management Monitor to take requests for services from any Realm that it is controlling
- The Realm Host interface, which enables the host to request Realm management functions through a trusted interface

Q: 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.

Q: What is the difference between the operation of Plane P0 and the operation of Plane P1

Plane P0, in multi-plane execution, is the control Plane for access to all other Planes in the same Realm. Plane P0 acts similar to a Hypervisor for the other planes in the Realm, control of execution for all other Planes is started through Plane P0 and communication from the Realm to the RMM is through Plane P0.