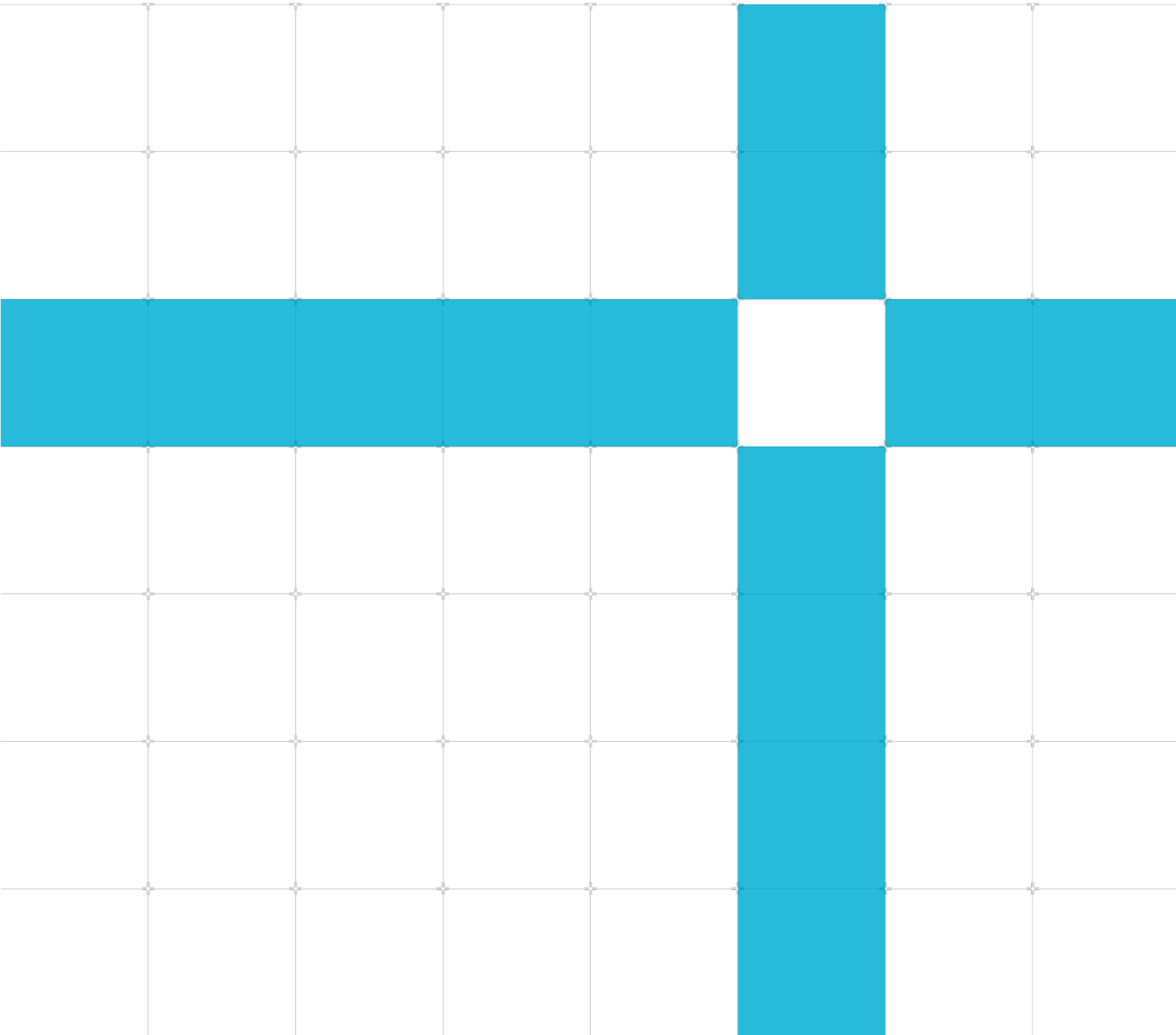
Software stack for Arm Confidential Compute Architecture

Non-Confidential

Issue 1.0
DEN 0127



Software stack for Arm Confidential Compute Architecture

Release information

Document history

Issue	Date	Confidentiality	Change
1.0	23rd June 2021	Non-Confidential	First release

Non-Confidential Proprietary Notice

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.

Product Status

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

Web Address

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.

We believe that this document contains no offensive terms. If you find offensive terms in this document, please email **terms@arm.com**.

Contents

1 Overview

This guide describes key software features that software architecture for Arm Confidential Compute introduces or changes to provide an environment for confidential computing. The guide also gives a brief overview of the expected software architecture, and the Realm construction flow.

The Realm Management Monitor (RMM) specification describes the software architecture for Arm Confidential Compute.

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

- Understand the whole software stack architecture for Arm Confidential Compute
- Understand the role of RMM and how to use the Realm Management Interface (RMI) and Realm Services Interface (RSI)
- Describe how a Realm is constructed, including dynamic memory allocation and delegation

Some software might use the hardware architecture features introduced by the Realm Management Extension (RME), for purposes other than hosting Realms. For example, software could make use of the Granule Protection Check (GPC) that is introduced by the RME for media content protection. This type of scenario is beyond the scope of this document. To learn more about this type of scenario, you can read Introducing Arm's dynamic TrustZone technology.

1.1 Before you begin

We assume that readers of this guide are familiar with the AArch64 Exception model, AArch64 memory management, AArch64 virtualization, and virtualization software like Linux Kernel-based Virtual Machine (KVM). If you are not familiar with these topics, you can read our guides:

- AArch64 exception model: This guide introduces the exception and privilege model in AArch64
- AArch64 memory management: This guide introduces the MMU, which is used to control virtual to physical address translation
- AArch64 virtualization: This guide describes the virtualization support in Armv8-A AArch64. Topics include stage 2 translation, virtual exceptions, and trapping
- Introducing Arm Confidential Compute Architecture: This guide describes confidential computing in modern compute platforms, explains what confidential computing involves, and then describes how Arm Confidential Compute Architecture supports this in an Arm compute platform

The Arm Confidential Compute architecture is described in more detail in a set of documents that you can find in Related information.

2 Concepts

This section of the guide introduces some concepts to help you understand the RMM architecture and software stack for Arm Confidential Compute architecture.

2.1 Realm attributes

A Realm is an execution environment which is protected from agents in the Non-Secure and Secure states, and from other Realms.

Some key Realm attributes include:

- The Realm lifecycle state. While in the New state, the contents of the Realm can be modified by untrusted Host software, and the RMM ensures that the Realm cannot be scheduled. Once transitioned to the Active state, the Realm contents are protected, and the Realm can be scheduled by the Host.
- A measurement of the initial contents and configuration of the Realm.
- The Protected Address Range (PAR), which is the region of the Realm IPA space where confidentiality and integrity guarantees apply.

2.2 Realm Descriptor

A Realm Descriptor (RD) is a data structure which stores the attributes of a Realm. The size of an RD is one Granule. The memory that is used to store an RD is allocated by the Host, but is only directly accessible by the RMM. The Host can manipulate RD contents only indirectly through RMI commands.

2.3 Realm Execution Context

A Realm Execution Context (REC) is a data structure which stores the saved context of a thread of execution within a Realm. There is one REC instance per Virtual CPU (VCPU). Like with an RD, the memory that is used to store a REC is allocated by the Host, but is only directly accessible by the RMM.

2.4 Realm Translation Table

A Realm Translation Table (RTT) is the Stage 2 translation table which describes the IPA space of a Realm. Like with the other Realm management data structures, the memory that is used to store an RTT is allocated by the Host, but is only directly accessible by the RMM.

2.5 Granule

A Granule is a unit of physical memory. The size of a Granule is the smallest unit, usually 4KB, of memory which can be delegated by the Host for use by the RMM.

A Granule can be used to store one of the following:

- Code or data used by the Host
- Code or data used by a Realm
- Data used by the RMM to manage a Realm, for example an RD, REC or RTT
- Code or data used by software in the Secure state

Permitted transitions between these usages of a Granule are performed on request from the Host, using the RMI interface.

3 Arm Confidential Compute Architecture software stack

This section of the guide describes the software components of the Arm Confidential Compute Architecture.

3.1 Software overview of Arm Confidential Compute

RMM runs under EL2 of Realm security state, which is based on AArch64 virtualization technology. To better understand the Arm Confidential Compute Architecture, let's review an example of type-2 virtualization software on AArch64, as shown in the following diagram:

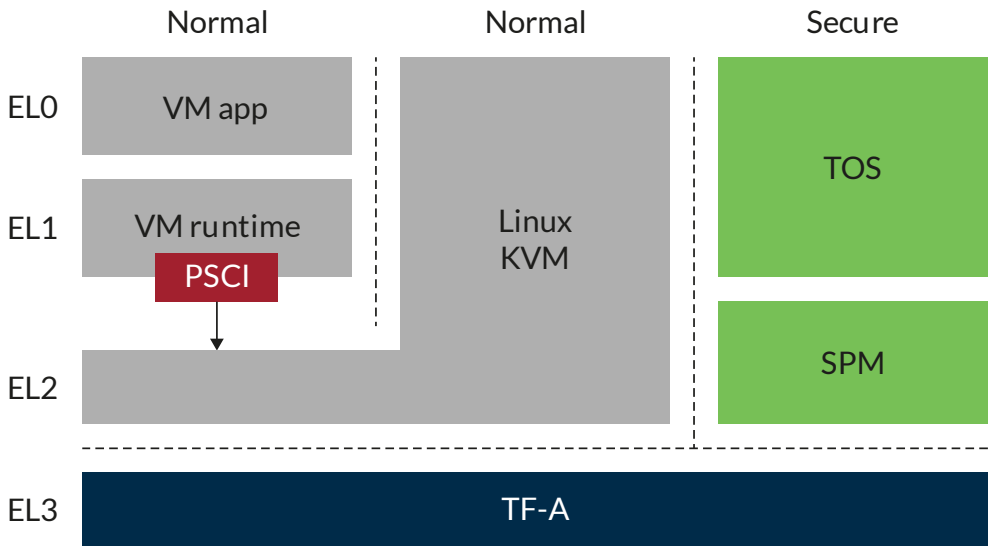


Figure 1 – Virtualization software on AArch64

Note

For a type-1 standalone hypervisor, the NS side might look a little different.

With AArch64 virtualization, a Non-Secure hypervisor manages Virtual Machines (VMs). The responsibilities of the hypervisor include scheduling and management of physical memory. The hypervisor can read and modify both the memory contents of the VM, and the register state of a VCPU within the VM.

The primary goal of the Arm Confidential Compute Architecture is to remove these capabilities from the hypervisor, while still allowing it to manage resources for a Realm in the same way as for a normal VM.

Under the new architecture, an additional Realm world is introduced to provide a protected computing environment. In the Realm world, the software that is executed can include a number of Realms. Each Realm is isolated from other Realms by a Realm Management Monitor (RMM) through stage 2 translation, and isolated for address space from other worlds through GPC. The following diagram shows the whole software stack:

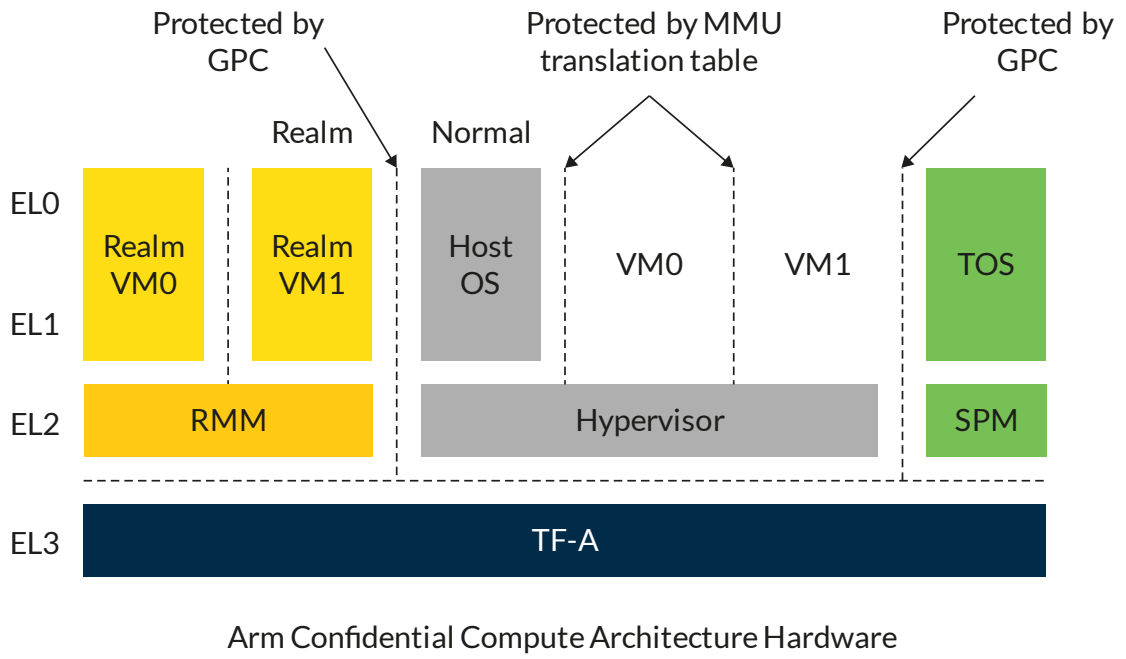


Figure 2 – Software stack for Confidential Compute Architecture

3.2 Realm Management Monitor

The RMM is a newly introduced software component. Its responsibilities include:

- Managing the life cycle of Realms
- Measuring the initial state of a Realm, and allowing the Realm to request this measurement for attestation
- Protecting the confidentiality and integrity of a Realm during operation, allowing for the paging system in the host OS to manage pages into and out of the Realm
- Protecting the confidentiality of a Realm during and following destruction of the Realm and reclaiming the resources that were previously allocated to the Realm

Arm plans to publish a reference implementation of the RMM.

The RMM exposes four interfaces, as shown in the following diagram:

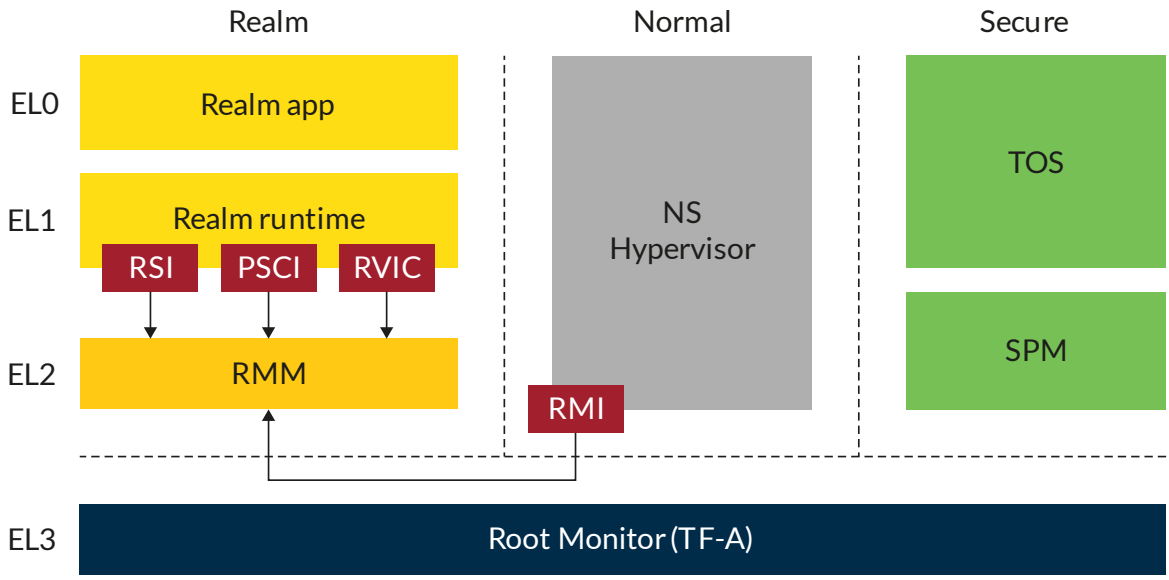


Figure 1 – Software Architecture for Confidential Compute ABIs

The Realm Management Interface (RMI) provides the following services to host software:

- Create or destroy Realms
- Schedule Realm execution
- Manage Realm memory including adding or removing, and mapping or unmapping pages

The Realm Services Interface (RSI) provides the following services to a Realm:

- Obtaining attestation reports
- Interacting with the Host for memory management
- Managing resources allocated to Realm and performing cryptographic operations

Both interfaces are implemented as Secure Monitor Call (SMC) calls. RSI calls that are executed in the Realm are trapped into RMM. RMI calls are executed by the Non-Secure hypervisor and routed by EL3 Root Monitor software to the RMM in R-EL2. The RMM specification details all RMI and RSI Interfaces.

The RMM handles Power State Coordination Interface (PSCI) calls in the following way:

- PSCI calls are trapped to RMM.
- RMM interposes on a subset of PSCI commands executed by a Realm.
- No OS impacts

The RMM implements PSCI v1.1 through RSI calls to the RMM using the standard PSCI function identifiers. Some of these calls are handled by both the RMM itself and the NS hypervisor. For example, PSCI_CPU_ON sets the entry point PC of the target REC, which can only be done by the RMM. However, the NS hypervisor must also be notified that a PSCI_CPU_ON call was made. To implement this shared PSCI handling, the RMM will assume that the Non-secure hypervisor also implements PSCI v1.1 when running Realms.

The RMM also has context management for Realms:

- Realm entry occurs as a result of the Host executing the RMI.REC.Run command. During Realm entry, the RMM restores VCPU context from the REC.
- Realm exit occurs as a result of an exception which is routed to EL2 or higher, during Realm execution. During Realm exit, the RMM saves vCPU context into the REC.

The RMM software stack is loaded during system boot, according to the Trusted boot specification.

3.3 NS hypervisor

The actions performed by the Non-Secure (NS) hypervisor to manage a Realm are similar to those required to manage a normal VM. Specifically, the hypervisor is required to:

- Allocate memory that is used by the Realm
- Allocate memory for data structures that are used to manage the Realm in the RMM, like translation tables and storage for the register state of a VCPU
- Decide when to schedule a VCPU in the Realm
- Handle exceptions which cause exit from the Realm
- Deliver virtual interrupts to the Realm
- Handle firmware calls from the Realm, for example PSCI requests

The difference is that most of those actions must be performed indirectly through the RMM, to uphold the security guarantees of the Realm. In other words, compared to management of a normal VM, the policy for managing a Realm remains in the hypervisor, but the mechanism is moved from the hypervisor into the RMM.

In the RMM specification, the NS hypervisor software is called the Host.

3.4 Realm software

A Realm is a protected virtual machine, consisting of an EL1 kernel and EL0 applications running within a Realm state under stage 2 translation. The software could be based on a kernel written for execution in Non-secure state. However, there are some additional requirements on Realm software:

- Hardening of the GIC driver, to defend against a malicious GIC emulation that is provided by the Host
- Handling timer interrupts
- Managing NS memory that is mapped outside the PAR for virtual devices

To support measurement and attestation, some RSI interfaces are called.

3.5 Root Monitor

Root Monitor is runtime firmware executing in EL3 Root state, which manages Security state switching and assigning resources between Security states. Root Monitor is the most privileged software component. Arm plans to implement the Root Monitor in Trusted Firmware for A profile (TF-A).

Additional context management besides Secure and Non-secure context in EL3 monitor should be added for Realm security state during Security state transitions between Non-secure, Secure, and Realm states.

Root Monitor also handles SMC calls from RMM to configure Protected Address Space (PAS) for Granule Protection Table (GPT) management.

Arm Confidential Compute Architecture hardware is described in a separate guide, Introducing Arm Confidential Compute Architecture.

4 Reference software implementation

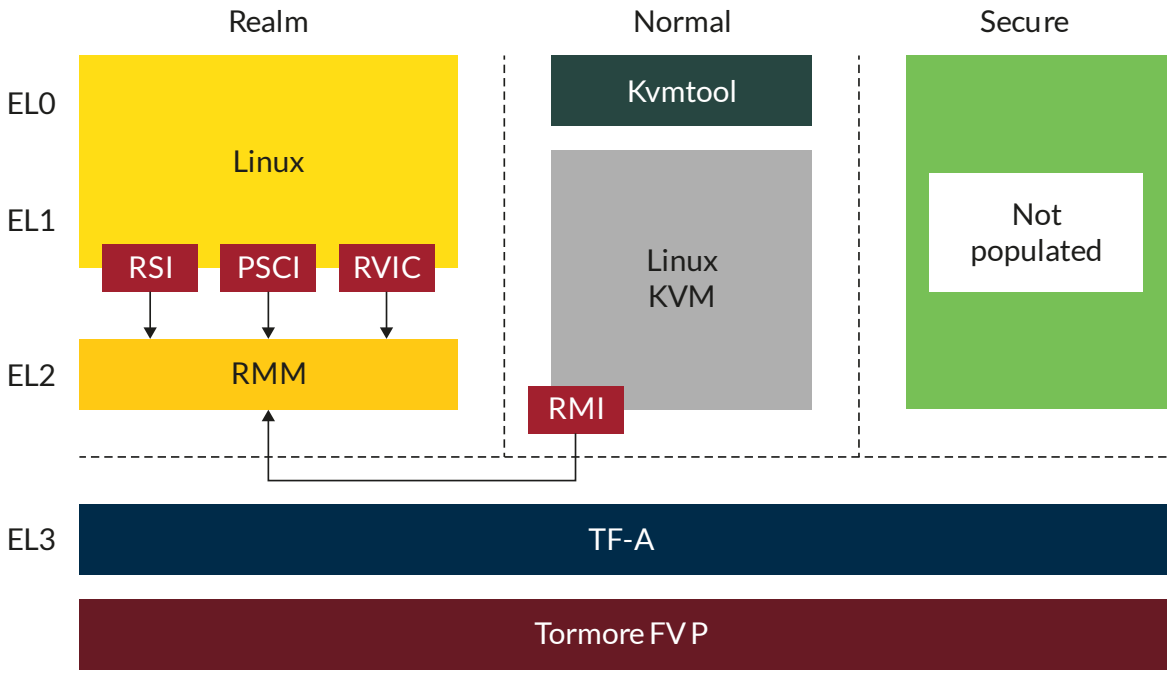
The Realm prototype software code is at draft stage. However, the code demonstrates the feasibility of adding Realm support to a contemporary High Level Operating System (HLOS), and the suitability of the modeled RME architecture for hosting RMM and Realms.

The Realm prototype software includes the following:

- TF-A, which implements Root Monitor
- Linux KVM, in which host and guest kernels use the same image
- kvmtool
- A root filesystem image

Arm plans to release the prototype software to partners.

The following diagram shows a high-level overview of the reference software implementation:



4.1 KVM host kernel and hypervisor

KVM is part of Linux kernel, so the hypervisor could reuse kernel system components to run VMs. A host kernel exports one device node interface for KVM hosting. A host kernel also exports another interface that can be used to manage Realms as provided by the RME.

The Host kernel provides three-level ioctls:

- System ioctls, `/dev/kvm` and `/dev/rme`:
  - Applied to the whole KVM subsystem
  - Used to create virtual machines using KVM_CREATE_VM
  - The `rme` device driver provides mmap and manages the virtual space of the Realm that is used by Kvmtool
- VM ioctls:
  - Applied to one entire virtual machine
  - Used to create VCPUs
  - Only run VM ioctls from the same process and address space that is used to create the VM
- VCPU ioctls:
  - Control the operation of a single VCPU
  - Only run VCPU ioctls from the same thread that was used to create the VCPU

A host kernel with RME support introduces a new KVM device, KVM_DEV_TYPE_ARM_REALM, which allows setting attributes to control the Realm. To allow control creating Realms, add additional control that is handed to the user space Kvmtool.

4.2 Kvmtool for KVM hosting

Kvmtool is a lightweight tool for hosting KVM guests, to run Linux guest VMs. Like with KVM, every VM is implemented as a regular Linux process, scheduled by the standard Linux scheduler. Kvmtool also provides network and block devices, using Virtio.

Kvmtool implements:

- One software thread for each VCPU. Realm Execution Context is scheduled in the host as one POSIX thread.
- Device models run concurrently in VCPU thread.
- Long running operations run in additional device threads.

The following diagram shows the run-time Kvmtool process for KVM guest hosting:

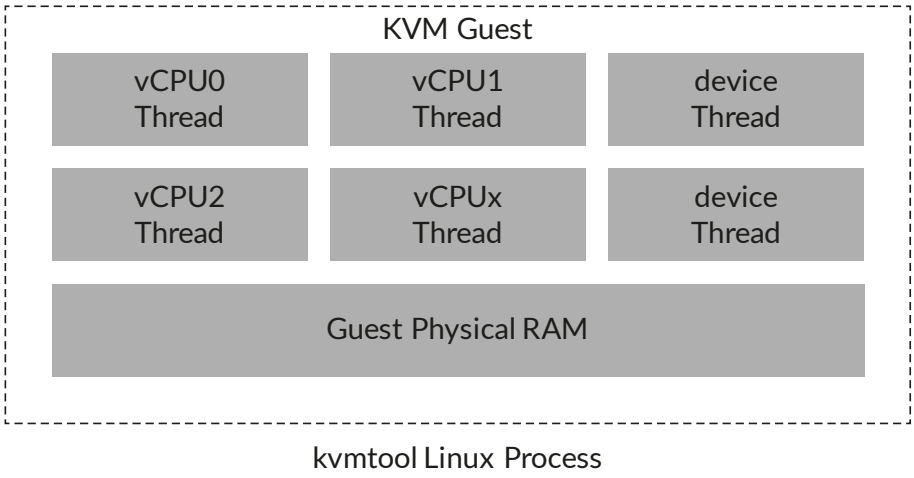


Figure 4 – Kvmtool for KVM guest hosting

4.3 Realm construction flow

The KVM hypervisor creates and manages the VCPU for guests. At the host level, a VCPU is a thread that executes in the guest mode. The VCPU is scheduled on a physical CPU just like any other host-level processes, as follows:

- A VCPU thread transitions between execution states: running, ready, or blocked.
- The host OS scheduler makes scheduling decisions considering all VCPU threads from all of the VMs that are being hosted.

When a VCPU thread is scheduled on a physical CPU to run the Realm Guest VCPU, it calls ioctl and runs guest:

- If the VCPU thread is a Realm guest OS, it will run RMI REC Run command to enter Realm guest mode from the hypervisor. The Run command is executed by RMM in Realm EL2.
- The EL3 monitor firmware switches to Realm state first. The RMM runs the REC run command for the Realm guest VCPU.
- The realm guest VCPU context is saved in the REC, which can only be accessed by the RMM to restore VCPU context. The KVM hypervisor cannot access the realm guest VCPU context.

The interaction among the Realm, RMM, Root Monitor, NS hypervisor, and NS EL0 Kvmtool are performed by the AArch64 exception model, using exception entry and exit. The following diagram shows the running flow of a Realm:

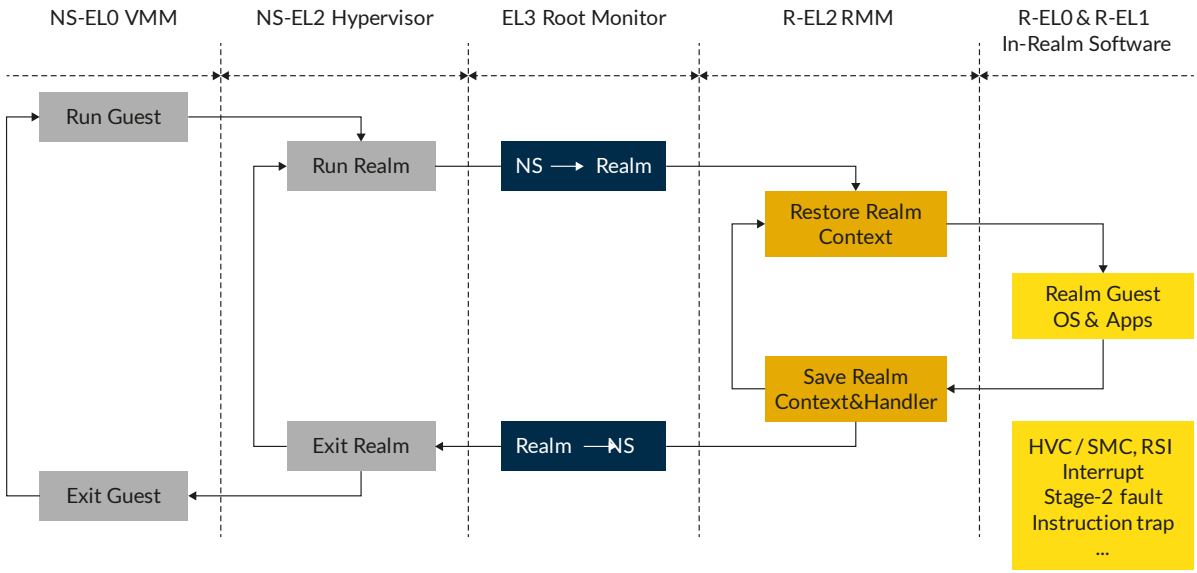


Figure 5 – Realm execution flow

5 Related information

Here are resources related to material in this guide:

- Introducing Arm Confidential Compute Architecture
- Arm System Memory Management Unit Architecture supplement - The Realm Management Extension (RME), for SMMUv3
- Arm Architecture Reference Manual Supplement, The Realm Management Extension (RME), for Armv9-A
- Arm Realm Management Extension (RME) System Architecture

6 Next steps

This guide has introduced you to the key software features relating to the Arm Confidential Compute Architecture.

Further guides in this series provide more information about the Arm Confidential Compute Architecture, as follows:

- Introducing Arm Confidential Compute Architecture guide
- Arm Realm Management Extension guide