Arm® Realm Management Extension (RME) System Architecture

Document number	DEN0129
Document quality	EAC
Document version	A.a
Document confidentiality	Non-confidential

Arm® Realm Management Extension (RME) System Architecture

Release information

Date	Version	Changes
2021/Jun/23	A.a	• First EAC publication.

Product Status

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

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

Contents

Arm® Realm Management Extension (RME) System Architecture

Preface

About this book

This book is the Arm® Realm Management Extension (RME) System Architecture. This book describes the changes and additions to the Armv9-A system architecture that are introduced by RME, and therefore must be read in conjunction with the *Arm® Architecture Reference Manual Supplement, The Realm Management Extension (RME), for Armv9-A*.

It is assumed that the reader is familiar with the Armv8-A and Armv9-A architecture.

Conventions

Typographical conventions

The typographical conventions are:

italic

Introduces special terminology, and denotes citations.

bold

Denotes signal names, and is used for terms in descriptive lists, where appropriate.

`monospace`

Used for assembler syntax descriptions, pseudocode, and source code examples.

Also used in the main text for instruction mnemonics and for references to other items appearing in assembler syntax descriptions, pseudocode, and source code examples.

SMALL CAPITALS

Used for some common terms like IMPLEMENTATION DEFINED.

Used for a few terms that have specific technical meanings, and are included in the Glossary.

Red text

Indicates an open issue.

Blue text

Indicates a link. This can be

- A cross-reference to another location within the document
- A URL, for example http://developer.arm.com

Numbers

Numbers are normally written in decimal. Binary numbers are preceded by `0b`, and hexadecimal numbers by `0x`. In both cases, the prefix and the associated value are written in a monospace font, for example `0xFFFF0000`. To improve readability, long numbers can be written with an underscore separator between every four characters, for example `0xFFFF_0000_0000_0000`. Ignore any underscores when interpreting the value of a number.

Pseudocode descriptions

This book uses a form of pseudocode to provide precise descriptions of the specified functionality. This pseudocode is written in a monospace font. The pseudocode language is described in the Arm Architecture Reference Manual.

Assembler syntax descriptions

This book contains numerous syntax descriptions for assembler instructions and for components of assembler instructions. These are shown in a `monospace` font.

Rules-based writing

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

- Declaration
- Rule
- Goal
- Information
- Rationale
- Implementation note
- Software usage

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

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

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

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

Content item identifiers

A content item may have an associated identifier which is unique among content items in this specification.

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

Content item rendering

In this document, a content item is rendered with a token of the following format in the left margin: L_{iiiii}

- *L* is a label that indicates the content class of the content item.
- *iiiii* is the identifier of the content item.

Content item classes

Declaration

A Declaration is a statement that does one or more of the following:

- Introduces a concept
- Introduces a term
- Describes the structure of data
- Describes the encoding of data

A Declaration does not describe behavior.

A Declaration is rendered with the label *D*.

Rule

A Rule is a statement that describes the behavior of a compliant implementation.

A Rule explains what happens in a particular situation.

A Rule does not define concepts or terminology.

A Rule is rendered with the label *R*.

Goal

A Goal is a statement about the purpose of a set of rules.

A Goal explains why a particular feature has been included in the specification.

A Goal is comparable to a “business requirement” or an “emergent property.”

A Goal is intended to be upheld by the logical conjunction of a set of rules.

A Goal is rendered with the label *G*.

Information

An Information statement provides information and guidance as an aid to understanding the specification.

An Information statement is rendered with the label *I*.

Rationale

A Rationale statement explains why the specification was specified in the way it was.

A Rationale statement is rendered with the label *X*.

Implementation note

An Implementation note provides guidance on implementation of the specification.

An Implementation note is rendered with the label *U*.

Software usage

A Software usage statement provides guidance on how software can make use of the features defined by the specification.

A Software usage statement is rendered with the label *S*.

Additional reading

This section lists publications by Arm and by third parties.

See Arm Developer (http://developer.arm.com) for access to Arm documentation.

[1] *Arm® Architecture Reference Manual Supplement, The Realm Management Extension (RME), for Armv9-A*. (ARM DDI 0615) Arm Ltd.

[2] *Arm® Confidential Compute Architecture (CCA) Security Model*. (ARM DEN 0096) Arm Ltd.

[3] *Arm® System Memory Management Unit Architecture supplement - The Realm Management Extension (RME), for SMMUv3*. (ARM IHI 0094) Arm Ltd.

[4] *Arm® Architecture Reference Manual Armv8, for Armv8-A architecture profile*. (ARM DDI 0487) Arm Ltd.

[5] *Arm® Server Base System Architecture Platform Design Document*. (ARM DEN 0029) Arm Ltd.

[6] *Arm® Reliability, Availability, and Serviceability (RAS) Specification Armv8, for the Armv8-A architecture profile*. (ARM DDI 0587) Arm Ltd.

[7] *Arm® Architecture Reference Manual Supplement, Memory System Resource Partitioning and Monitoring (MPAM), for A-profile architecture.* (ARM DDI 0598) Arm Ltd.

[8] *Arm® CoreSight™ Performance Monitoring Unit Architecture*. (ARM IHI 0091) Arm Ltd.

[9] *Arm® Power State Coordination Interface*. (ARM DEN 0022) Arm Ltd.

Feedback

Arm welcomes feedback on its documentation.

Feedback on this book

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

- The title (Arm® Realm Management Extension (RME) System Architecture).
- The number (DEN0129 A.a).
- The page numbers to which your comments apply.
- The rule identifiers to which your comments apply, if applicable.
- A concise explanation of your comments.

Arm also welcomes general suggestions for additions and improvements.

Note

Arm tests PDFs only in Adobe Acrobat and Acrobat Reader, and cannot guarantee the appearance or behavior of any document when viewed with any other PDF reader.

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 contact terms@arm.com.

Part A
Overview

Chapter A1
Introduction

A1.1 Overview

A1.1.1 Context

This chapter introduces the *Realm Management Extension* (RME) system architecture. The RME architecture is specified in [1] and is defined as an extension to the A-profile of the Armv9 architecture.

The RME architecture defines the set of hardware features and properties that are required to comply with the Arm CCA architecture.

The Arm Confidential Compute Architecture (Arm CCA) enables the construction of protected execution environments called Realms. Realms allow lower-privileged software, such as application or a Virtual Machine to protect its content and execution from attacks by higher-privileged software, such as an OS or a hypervisor.

Higher-privileged software retains the responsibility for allocating and managing the resources that are utilized by a Realm, but cannot access its contents, nor affect its execution flow.

This document describes the required system properties for implementing the RME functionality.

This includes definitions of:

- Concepts and terms of the RME system architecture.
- System resources, capabilities, and components required by the architecture.
- System flows and identifiers.
- Security properties of an RME system.

IP that supports RME complies with the System Architecture defined by this specification.

The RME system architecture is applicable to multiple topologies and platform use-cases, for example Cloud, Mobile, and IoT.

Figure A1.1 provides an example illustration of the RME impact on a representative single-socket system topology.

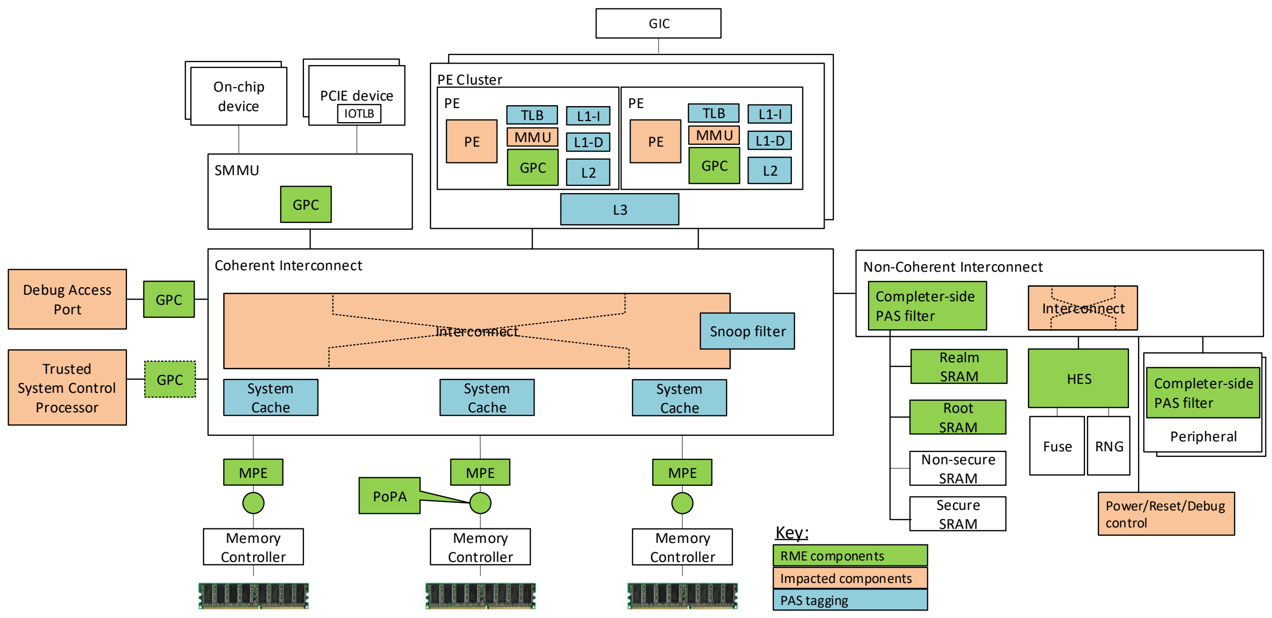

Figure A1.1: RME System Architecture

A1.2 Scope and intended audience

This specification forms part of the Arm guide to implementing RME.

The intended audiences for this specification include:

- SoC architects and micro-architects.
- System IP and CPU micro-architects.
- System Firmware developers.

Part B
Architecture

Chapter B1
Identifiers

This chapter specifies the RME-specific identifiers that are sent in transactions across the system fabric.

B1.1 Physical Address Space tag

I_{TVRTM} An RME memory system supports multiple physical address spaces.

A Physical Address of any memory-mapped Resource in the system (*Hardware Physical Address*) is associated with an architectural *Physical Address Space* (PAS).

I_{RZSTK} The following architectural physical address spaces are defined by RME:

- Non-secure PAS.
- Secure PAS.
- Realm PAS.
- Root PAS.

Figure B1.1 illustrates the concept of architectural physical address spaces.

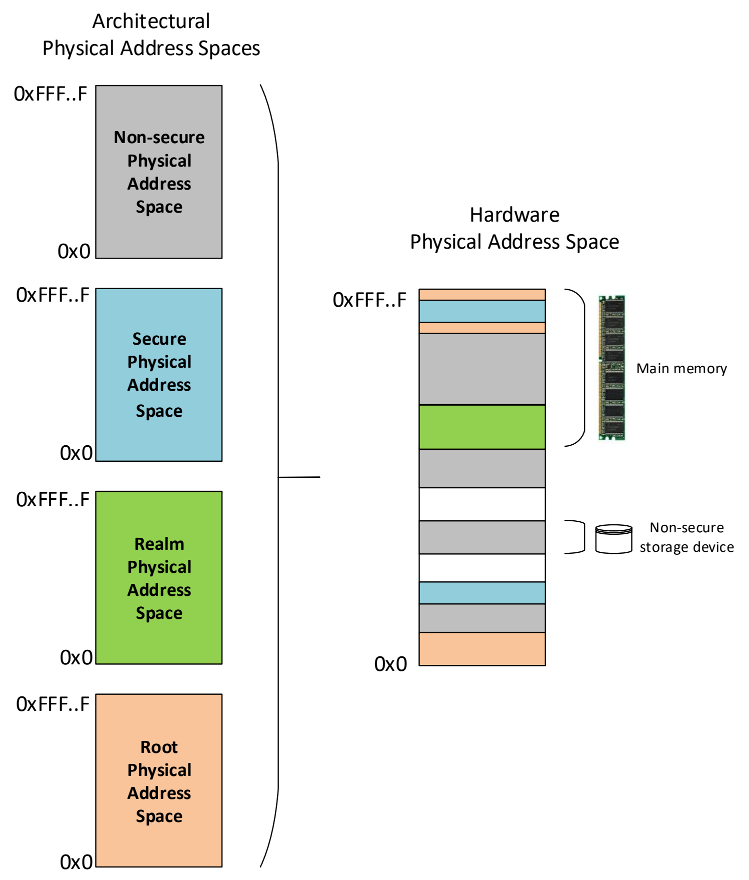
Architectural
Physical Address Spaces
0xFFF..F
Non-secure Physical Address Space
0x0
0xFFF..F
Secure Physical Address Space
0x0
0xFFF..F
Realm Physical Address Space
0x0
0xFFF..F
Root Physical Address Space
0x0
Hardware
Physical Address Space
0xFFF..F
0x0
Main memory
Non-secure storage device

Figure B1.1: Physical Address Spaces

I_{HTPJJ} Associating memory-mapped Resources with architectural physical address spaces can be done in several ways:

- The association of a Resource with a PA in one physical address space can make it inaccessible through other physical address spaces.
  - For example, main memory in a RME-enabled system is associated this way using the Granule Protection Table.

- Two distinct Resources can be each assigned to a different PAS while each Resource is accessed through the same PA within that PAS.
  - This banked assignment method might be used for peripheral registers.
- In specific cases, a single Resource can be simultaneously accessible through the same PA in multiple physical address spaces.
  - For example, peripheral registers in the Secure PAS can be accessible also in the Root PAS.
  - Arm strongly recommends that this approach is only used with peripherals, and only with peripherals that are PAS tag-aware.

I_{CWKBT} The Physical Address Space tag (PAS tag) is an Address Space Identifier which permits the forming of multiple physical address spaces in the system. A Physical Address (PA) is associated with a physical address space by qualifying it with a PAS tag.

All accesses are associated with a PAS, which is checked by PAS filters assigned to protect memory resources. Depending on system implementation, requester type, and memory type, this can be the requester-side PAS filter implemented within PEs and System MMUs, referred to as the Granule Protection Check, or a completer-side PAS filter, or a combination of both.

Accesses to cacheable memory retain their associated PAS until reaching the Point of Physical Aliasing (PoPA).

Accesses to non-cacheable memory retain their associated PAS at least until reaching the PAS filter assigned to protect that memory. Whether the associated PAS is retained beyond that filter for non-cacheable accesses depends on system implementation.

See also:

- B2.2 *Memory isolation and protection*

Chapter B2
System capabilities

This chapter specifies the system capabilities required by RME for guaranteeing Arm CCA security and isolation properties for Realms.

B2.1 Execution isolation

B2.1.1 Security states

I$_{QPTSX}$ An RME system supports the following Security states:

- Non-secure.
- Secure.
- Realm.
- Root.

I$_{CLTDC}$ The term *requester* refers to a hardware agent that is capable of initiating accesses. A requester can be a PE or a non-PE agent.

R$_{DFYXL}$ In an RME system, any access by a requester and any instruction executed by a PE is associated with a single Security state.

I$_{LJDDC}$ The Realm Management Extension capability defined in [1] specifies how PE execution context is mapped to Security states.

I$_{VHCHD}$ RME provides hardware-based isolation that allows execution contexts to run in different Security states and share resources in the system while ensuring that:

- Execution in the Realm Security state cannot be observed or modified by an agent associated with either the Non-secure Security state or the Secure Security state.
- Execution in the Secure Security state cannot be observed or modified by an agent associated with either the Non-secure Security state or the Realm Security state.
- Execution in the Root Security state cannot be observed or modified by an agent associated with any other Security state.
- Memory assigned to the Realm Security state cannot be read or modified by an agent associated with either the Non-secure Security state or the Secure Security state.
- Memory assigned to the Secure Security state cannot be read or modified by an agent associated with either the Non-secure Security state or the Realm Security state.
- Memory assigned to the Root Security state cannot be read or modified by an agent associated with any other Security state.

This specification uses the term *RME security guarantee* to describe the preceding properties.

B2.1.2 Security model

I$_{LYDGX}$ The Arm CCA System Security Domain (SSD) includes all hardware agents capable of affecting the Arm CCA and RME security guarantees. Examples include isolation hardware and Trusted subsystems.

I$_{PPRLG}$ A *Trusted subsystem* is a system function with private resources, configuration, and firmware that are attestable, for example a Trusted SCP.

I$_{WPQRT}$ The Monitor Security Domain (MSD) is updatable PE firmware executing in the Root security state at EL3 that is responsible for enforcing the Arm CCA and RME security guarantees.

I$_{YRMHM}$ The Realm Management Security Domain (RMSD) is updatable PE firmware executing in the Realm Security state at EL2 that is responsible for enforcing the Arm CCA security guarantee for Realms.

The terms in this section are formally defined in [2]. Figure B2.1 provides an illustration of the security model.

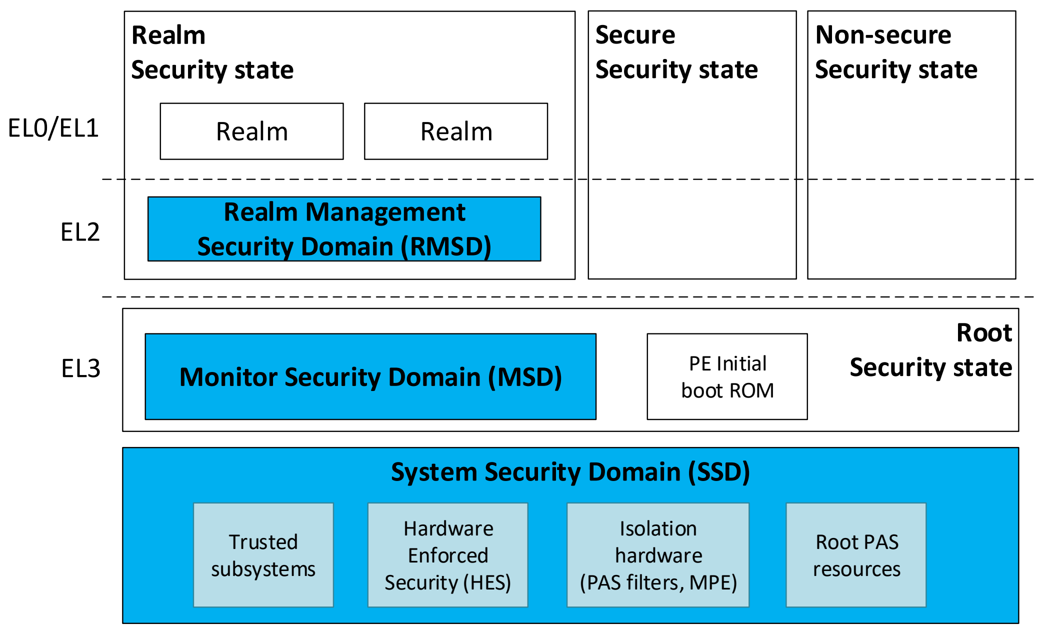
Realm Security state
Secure Security state
Non-secure Security state
EL0/EL1
Realm
Realm
EL2
Realm Management Security Domain (RMSD)
EL3
Monitor Security Domain (MSD)
PE Initial boot ROM
Root Security state
System Security Domain (SSD)
Trusted subsystems
Hardware Enforced Security (HES)
Isolation hardware (PAS filters, MPE)
Root PAS resources

Figure B2.1: Arm CCA security model

B2.2 Memory isolation and protection

The concept of architecturally-separate physical address spaces enables the formation of robust isolation boundaries for memory protection.

This chapter defines rules for guaranteeing memory isolation using physical address spaces and methods for mapping resources to Execution states through these spaces.

$\mathrm{I}_{\mathrm{BHZXC}}$ The physical address spaces that can be reached from each Security state are defined in [1] and captured in Table B2.1, the PAS Access Table:

Table B2.1: PAS Access Table

	Secure state	Non-secure state	Root state	Realm state
PAS[1:0]				
0b00 - Secure	Yes	No	Yes	No
0b01 - Non-secure	Yes	Yes	Yes	Yes
0b10 - Root	No	No	Yes	No
0b11 - Realm	No	No	Yes	Yes

$\mathrm{I}_{\mathrm{YLXPX}}$ A Resource is an addressable physical entity that is formally defined in [1]. A Resource in an RME system is accessible only when it is associated with a *Resource PAS*.

$\mathrm{R}_{\mathrm{QDWVC}}$ Association of a Resource with a Resource PAS is controlled by either SSD or MSD.

$\mathrm{I}_{\mathrm{JVLCJ}}$ Association of a Resource with a Resource PAS can be set statically by isolation hardware or can change during runtime, for example by MSD firmware.

$\mathrm{R}_{\mathrm{BJVZS}}$ An access to a Resource is associated with an *Access PAS* in accordance with the PAS Access Table.

$\mathrm{I}_{\mathrm{TYMYY}}$ For requesters that access Resources through a stage 1 or stage 2 MMU, assigning the Access PAS in accordance with the PAS Access Table is enforced by MMU SSD hardware.

For requesters that do not access Resources through a stage 1 or stage 2 MMU, such as the Generic Interrupt Controller (GIC) or a Debug Access Port, assigning the Access PAS in accordance with the PAS Access Table is enforced at the requester side by SSD hardware.

SSD hardware is hardware that is either immutable or exclusively controlled by an SSD component, for example a Trusted subsystem.

The PAS tag attached to a request conveys its associated Access PAS.

$\mathrm{I}_{\mathrm{TFRNC}}$ [1] and [3] define the programming model for setting the Access PAS from each Security state.

$\mathrm{R}_{\mathrm{SCDLL}}$ Once assigned, the value of an Access PAS cannot be altered.

$\mathrm{I}_{\mathrm{JFTJL}}$ The system must not expose any registers or debug mechanisms that permit the value of an Access PAS to be overridden.

$\mathrm{R}_{\mathrm{YKVJK}}$ A *PAS filter* enforces the PAS protection check by permitting access to a Resource only if the Access PAS matches a Resource PAS with which that the Resource is associated.

$\mathrm{R}_{\mathrm{MZJXC}}$ Every requester in the system is subjected to the PAS protection check.

I$_{CDDPM}$ A requester in this context includes any of the following:

- Processing elements (PEs) used by the host operating system or hypervisor to execute user applications or kernel threads.
  - The term *application PEs* is also used in this specification to make a distinction between processing elements visible to host software and processing elements embedded in system devices.
- Non-PE requesters that might be fully coherent, IO-coherent, or Non-coherent.
  - This includes any device that supports initiation of a memory access, such as cache prefetchers, the Generic Interrupt Controller (GIC), or a Debug Access Port.
  - Other examples are peripheral devices and control processors that might contain non-application PEs.

I$_{TJRFZ}$ Certain Trusted requesters comply with the PAS protection check without going through a PAS filter.

- A PAS filter can directly access resources in the Root PAS such as protection tables stored in DRAM.
- A Memory Protection Engine (MPE) can directly access resources in the Root PAS, such as integrity tags stored in DRAM.
- Trusted subsystems can directly access peripheral registers or SMEM using a hardwired map that associates resources with the Root PAS.

I$_{NDZHH}$ The term *completer* refers to a component that contains Resources and responds to accesses.

I$_{HQDWZ}$ For certain Resources such as peripherals or SMEM, the PAS filter can be located at the completer-side. For other Resources, such as DRAM, a PAS filter must be attached to all requesters that access that Resource. RME system architecture rules guarantee that isolation is maintained in either construction.

I$_{VPYFG}$ MSD Resources are in the Root PAS and are managed by software running at EL3.

R$_{WRGTF}$ Access to the Root PAS is only permitted for Trusted requesters.

I$_{CNYKS}$ Trusted requesters are:

- A PE executing in the Root Security state (EL3).
- Trusted subsystems. For example, a Trusted SCP, or a subsystem hosting HES.
- Memory Protection Engines.
- PAS filters.

I$_{QNNTR}$ RMSD Resources are in the Realm PAS and are managed by software running at EL2 in the Realm Security state.

I$_{FPLKV}$ The term Resource X *is in* PAS Y means that Resource X *is accessible only in* PAS Y unless where the text explicitly permits the Resource to be accessible in more than one PAS.

See also:

- B2.2.1 *Granular PAS filtering*

B2.2.1 Granular PAS filtering

I$_{BYTBH}$ Granular PAS filtering is the programmable association of a Resource with a PAS at a granularity of a page (Physical Granule).

I$_{XXXPR}$ A Granular PAS filter checks the Access PAS against a Physical Granule Resource PAS as specified in a Granule Protection Table (GPT). If the check fails, the access is aborted and a Granule Protection Fault (GPF) is reported.

R$_{KGDVK}$ A Resource can be associated with a PAS using a Granule Protection Table if the following conditions are met:

- There is only a single PA within each PAS through which the Resource can be reached and the value of the PA is the same across all physical address spaces.
- The Resource can be assigned to a PAS at page granularity.

I$_{RSGHN}$ The term Granule Protection Check (GPC) refers to a requester-side Granular PAS filter as specified by [1] and [3], where such filter can be attached to an MMU or an SMMU.

I_{KQQJG} GPC is a requester-side PAS filter. By system construction, any access from an application PE or an SMMU-attached requester first goes through a requester-side PAS filter and then can reach a completer-side PAS filter. The programming of the GPT must therefore take into account the potential existence of completer-side PAS filters for specific Resources.

The Granule Protection Check can be omitted for Resources that are protected by a completer-side PAS filter, and in this scenario the Resource is marked in the GPT as "All Accesses Permitted".

I_{WVWZP} MSD guarantees that all requesters subject to the Granule Protection Check observe a consistent view of the Granule Protection Table.

I_{HDLTB} The memory encryption rules imply that granule contents are not preserved when the PAS association of a granule is changed. Nevertheless, software cannot rely on the transition of a granule to a new PAS as an implicit scrubbing event and must explicitly scrub granule contents before transitioning the granule to the Non-secure PAS.

R_{WJNMD} Granule Protection Check for on-chip Resources can only rely on Granule Protection Tables that are stored on-chip or are stored off-chip with equivalent level of integrity and replay protection.

R_{GQCQT} A Granule Protection Check that applies to non-idempotent locations does not permit any access to be speculatively performed to a non-idempotent location before the Granule Protection Check for the access is complete.

I_{KYMLS} An example for a non-idempotent location is a read-sensitive memory-mapped peripheral register. Speculative read access to non-idempotent memory can result in UNPREDICTABLE behavior. Correspondingly, a system that is using GPC for assigning non-idempotent locations to Realm, Secure, or Root PAS must complete the GPC for any non-idempotent location before permitting access to it.

R_{GFGZM} If a requester-side Granular PAS filter is in reset state, any requester that is associated with it is either in reset state or blocked from accessing memory.

I_{KVHFL} This permits for a predictable access control behavior when initializing the system.

See also:

- B2.3 *Device isolation and protection*.
- Chapter B5 *Power Management*.

B2.2.2 Cache Maintenance

I_{KWFWG} The Point of Physical Aliasing (PoPA) is a reference location for cache maintenance operations that is defined in [1].

R_{WFQKD} A PA that targets memory that can be cached is associated with a PAS until reaching the PoPA.

R_{FRMJJ} Where a PA is associated with a PAS, any PA compare operation includes the PAS.

I_{KHSYM} This rule applies to any cache or snoop filter that is before the PoPA (between the requesters and the PoPA) at any hierarchy level of the system including L1 data and instruction caches. This is needed to maintain the principle of separate physical address spaces as a global security property of the system.

I_{SQJLC} An RME system supports cache maintenance operations to the PoPA in compliance with [1]. The scope of a cache maintenance operation to the PoPA (*PoPA CMO*) is the Outer Shareable shareability domain.

R_{QBNJF} A PoPA CMO applies to any cached copy in the system with the specified {PAS, PA} regardless of both:

- The shareability domain it was cached with.
- Whether the system supports a single or multiple Outer Shareable shareability domains.

I_{JQSYD} As an example, implementations must guarantee that a PoPA CMO sent from one PE affects cache-lines that were allocated as Non-shareable by other PEs. Such a guarantee typically requires snoop filter registration for any allocation into a fully-coherent cache that is located past the Granule Protection Check, regardless of the shareability attribute of the access causing the allocation.

An implementation can support that by having application PEs artificially convert Non-shareable Cacheable accesses to either Inner Shareable Cacheable or Outer Shareable Cacheable but must guarantee that memory consistency and coherency semantics are preserved when other requesters continue to use the Non-shareable attribute when accessing the same Location.

Non-PE requesters can continue using the Non-shareable Cacheable attribute when allocating into a cache that is located before the Granule Protection Check, as later write-backs from such cache always go through a PAS protection check.

B2.2.3 Main memory (DRAM) protection

B2.2.3.1 Memory encryption and integrity

I_{FPJXZ} There are several memory encryption and integrity schemes that are applicable to an RME system.

The baseline encryption requirement is supporting encryption for external memory, using a separate encryption key or tweak for each PAS and providing spatial isolation using an address tweak. RME prevents runtime software access to ciphertext in external memory, in accordance with [2].

Cryptographic memory integrity and freshness are additional, threat-model dependent, capabilities for complying with the Arm CCA security guarantees as specified in [2].

This specification uses the term Memory Protection Engine (MPE) to describe the component that provides external memory encryption and integrity services.

A taxonomy of memory protection schemes that can be used with an Arm CCA system is defined in [2].

R_{MLFBL} External memory that is assigned to Secure PAS, Realm PAS, or Root PAS must be encrypted using a method that provides at least the following:

- A different encryption context for each PAS.
- A different address tweak for each encryption data block, such as a 128-bit memory block.
- If cryptographic memory integrity is not supported, an encryption mode that ensures bit diffusion over an encryption data block.

R_{MYWVB} Data is encrypted before being written to external memory or to any shared cache that resides past the PoPA.

See also:

- B3.2.4 *Memory Protection Engine System Requirements*.

B2.2.3.2 DRAM scrubbing

I_{DGWPK} The term *scrubbing* is used in this specification to describe an operation that guarantees that the previous contents of a memory location are no longer readable.

I_{MLKLZ} The term ECC-scrubbing is used to describe the operation of refreshing DRAM ECC state.

I_{KQPGL} On system boot, memory that could have been assigned to Secure PAS, Realm PAS, or Root PAS must be scrubbed before any requester that is not a *Trusted requester* is granted access to that memory. Because an RME system supports memory encryption, scrubbing is performed implicitly by resetting all stored copies of the memory encryption key on an RME system reset.

R_{BNSQB} An ECC-scrubbing engine located after the PoPA must not leak confidential information, for example through error record registers.

See also:

- B4.3 *RAS*.

B2.3 Device isolation and protection

MMU-attached Granule Protection Checks are applicable to Normal memory and Device memory. This system capability can be augmented with PAS filters on the completer side for isolating specific Resources. For example, a completer-side PAS filter might be useful where protection at register granularity is required or where it is required immediately after reset for an SSD resource.

R_{GDVSZ} A PA of an access to a memory-mapped peripheral is associated with a PAS until reaching the PAS filter assigned to protect the peripheral.

I_{QXXCQ} For memory-mapped peripherals that are not protected by a completer-side PAS filter the PAS tag can be stripped at an IMPLEMENTATION DEFINED location. For example, a PAS tag of an access to a PCIe memory-mapped peripheral can be stripped at the Root Port.

B2.3.1 Peripheral isolation

I_{SKDDD} A peripheral can include a private completer-side PAS filter for autonomously controlling access to its memory-mapped registers.

R_{DVPGT} A private PAS filter allows access to a register only if the Access PAS matches a Resource PAS that the register is associated with.

I_{LDBCM} A peripheral might associate a memory-mapped register with multiple physical address spaces, for example to expose different values to software in different Security states. In such a case, the PAS filter allows access to the register from multiple physical address spaces.

I_{FLNFT} An interconnect can include a PAS filter for controlling access to completer nodes that are not PAS-aware.

I_{TKRBJ} For example, an interconnect can be configured to assign an attached device to a specific PAS, and block any accesses to that device unless they use the correct PAS.

B2.3.2 Non-PE requesters (Devices)

R_{RHBJN} The Security state of a non-PE requester that is not a Trusted subsystem can be one of the following:

- Non-secure.
- Secure.

I_{CPKZT} A non-PE requester that is not a Trusted subsystem cannot access memory in the Realm PAS or the Root PAS. Providing non-PE requesters such as IO devices or accelerators with access to the Realm PAS is not supported in this version of the RME architecture.

I_{CRPCL} An RME system can include non-PE requesters which are fully coherent and therefore capable of emitting snoop responses.

R_{MCMSH} A fully coherent non-PE requester will not provide snoop data in response to snoop requests to addresses in the Realm PAS or Root PAS.

I_{PPVTC} This rule applies both to on-chip devices and off-chip devices that are attached to the SoC over a coherent link. One possible method to achieve it is by not forwarding a snoop request to addresses in the Realm PAS or Root PAS to non-PE requesters and by ignoring unsolicited snoop responses with data.

B2.3.3 Programmable completer-side filters

$\mathrm{I}_{\mathrm{TPLYX}}$ A completer-side PAS filter can be software programmable. Examples include:

- A completer-side PAS filter that can be programmed to assign memory-mapped portions of a Resource or granules of a Resource to a specific PAS.
- A completer-side PAS filter for SMEM that can be programmed to assign an SMEM address range to a specific PAS.
- A completer-side PAS filter on an interconnect port that can be programmed to assign a completer attached to that port to a specific PAS.

$\mathrm{R}_{\mathrm{RGQRT}}$ If a programmable completer-side PAS filter can assign resources to all physical address spaces then:

- The registers that control the filter are in the Root PAS.
- On an RME system reset, Resources controlled by the filter are either assigned to the Root PAS or are reset to a known value.

$\mathrm{R}_{\mathrm{GLLZY}}$ If a programmable completer-side PAS filter assigns resources only to the Secure PAS and Non-secure PAS then:

- The registers that control the filter are in the Secure PAS or in the Root PAS.
- On an RME system reset, Resources controlled by the filter are either assigned to the Secure PAS or the Root PAS or are reset to a known value.

$\mathrm{I}_{\mathrm{TCSGH}}$ Resources suitable for protection using a completer-side PAS filter include memory-mapped peripherals and on-chip SRAM (SMEM).

Chapter B3
Resources and Components

This chapter outlines properties of system resources and components that are part of an RME system.

R_{JSDVG} All RME structures and fields use little-endian convention.

B3.1 Shielded memory

This section describes shielded memory requirements for MSD and RMSD.

I_{PLSGD} The term *shielded memory* (SMEM) refers to memory that provides confidentiality, integrity, and replay protection against off-chip attacks. A typical example for SMEM is on-chip SRAM. Shielded memory can be local to a PE or globally visible to all requesters in a system.

I_{FTDTY} MSD and RMSD use SMEM for storing sensitive code and state in accordance with requirements in [2]. MSD SMEM is used for storing MSD code, data and translation tables, and also for storing the level 0 GPT. RMSD SMEM can be used for storing RMSD code, data, measurements, and cryptographic context.

R_{CSSDG} MSD SMEM is in the Root PAS.

R_{SPLKT} The address ranges of MSD SMEM are either defined statically or defined by SSD following an RME system reset.

R_{NXJLB} On an RME system reset MSD SMEM is either immediately assigned to the Root PAS or scrubbed and is available for access by the PE boot ROM as soon as it starts executing.

R_{CMMCZ} RMSD SMEM is in the Realm PAS.

R_{ZVQGS} The address ranges of SMEM assigned to the Realm PAS and Secure PAS are either defined statically or by SSD or MSD.

R_{ZQQSQ} SMEM that can be dynamically assigned to the Realm PAS or the Secure PAS is either immediately assigned to the Root PAS or scrubbed on an RME system reset.

R_{ZCJHY} The access control path that protects SMEM is not affected by state from non-shielded memory.

I_{SQQWY} For example, if the Level 1 GPT is stored in non-shielded memory it cannot be used for assigning on-chip SRAM to the Root PAS.

I_{JRLRX} For example, if SMEM is on-chip SRAM then SMEM access control can be implemented using a completer-side PAS filter, where:

- On an RME system reset all SMEM Resources are assigned to the Root PAS.
- SMEM Resources can be assigned to the Realm and Secure PAS using registers in the Root PAS.

This prevents leakage of state following system reboot.

I_{GCJRZ} Arm recommends that global RMSD SMEM and global MSD SMEM support cacheable accesses. [1] specifies GPT shareability and cacheability configuration that is common to all levels of the GPT. An implementation that locates the level 0 GPT in non-cacheable SMEM can use an IMPLEMENTATION DEFINED method to guarantee this does not prohibit cacheability of other levels of the GPT.

B3.2 Components

B3.2.1 PE

I_PCXDR The term *application PE* refers to a PE used by the operating system or hypervisor to execute user applications or kernel threads.

R_GSRPS All A-profile application PEs in the system implement the Realm Management Extension (RME).

I_ZCLYG Non-application PEs are not required to implement the Realm Management Extension. As any other requesters, non-application PEs are subjected to PAS filtering.

I_TXKMT External access to PE resources through a memory-mapped interface such as a utility bus must comply with RME system architecture rules for peripheral isolation.

For example, an implementation must guarantee that for features that are exposed through a memory-mapped interface:

- When a PE is in the Realm Security state, Non-secure or Secure accesses by another agent through the memory-mapped interface, do not return any information about Realm Security state execution or affect it.
- Similarly, when a PE is in the Secure Security state, Non-secure or Realm accesses by another agent through a memory-mapped interface, do not return any information about Secure Security state execution or affect it.

I_FHLLF A PE that implements RME and might be integrated in legacy systems should support a LEGACY_TZ_EN input tie-off. When set, the LEGACY_TZ_EN input tie-off forces the PE to hide the RME capability and fallback to supporting two Security states and two physical address spaces.

See also:

- B2.3.1 *Peripheral isolation*.
- B8.1 *Using RME IP in a legacy system*.

B3.2.2 SMMU

R_NJRPC An SMMU in an RME system complies with [3].

B3.2.3 Interconnect and caches

This section defines rules related to components that connect between multiple requesters and completers, such as clusters, non-coherent interconnects, and coherent interconnects.

I_YBKWT A PAS tag is assigned to any transaction that carries a PA.

I_FLYMQ The integration of an RME-aware component must guarantee that the PAS tag is propagated to that component through any interconnect or bus on the way. For example:

- A coherent or non-coherent interconnect attaching to global SMEM that is protected with a completer-side PAS filter.
- A system bus, such as AMBA APB, attaching to a memory-mapped interface of a PE.

R_XBKYB All bus and interconnect decoding components between the point where the Access PAS is assigned and the PoPA are PAS tag-aware.

I_SFRSB In practice, a forwarding component can decide to ignore the PAS tag when making a forwarding decision for certain address ranges, such as memory ranges that are subject to granular PAS filtering.

R_XTSXB An RME coherent interconnect supports cache maintenance operations to the PoPA in compliance with [1].

R_{FXQCD} A PoPA CMO applies to any cache before the PoPA, including system caches that are located beyond the Point of Coherency.

R_{LCXDB} Completion of a PoPA CMO for a given PA guarantees that both:

- Any dirty cached or transient state associated with the PA before the PoPA has been cleaned to after the PoPA.
- Any cached or transient state associated with the PA before the PoPA has been invalidated.

I_{CDSDR} Any system cache before the PoPA must comply with the cache lockdown rules that are specified in [1].

R_{CMMDG} For any cache before the PoPA, cache prefetching across granule-boundary is allowed only after querying the GPC for the PAS association of the next granule.

R_{PSGCM} A cache maintenance operation performed on a *Clean* cache entry never results with a write of entry content past the PoPA.

I_{RMKKN} Writing clean data past the PoPA can lead to data corruption as a granule transitions between Physical Address Spaces for certain cache and MMU implementations that make use of Speculative data read accesses. The Armv8-A architecture forbids writing clean data outside of a shareability domain (See [4], D4.4.1 General behavior of the caches).

See also:

- B8.1 *Using RME IP in a legacy system*.

B3.2.3.1 DVM operations

I_{WWYZX} Granular PAS filters can cache GPT content. [1] Defines the rules for caching GPT content and invalidating cached GPT content. The scope of all GPT cache invalidations is the Outer Shareable domain which guarantees that all GPT caches in the system are reachable by GPT cache invalidations.

R_{JRJSQ} An RME coherent interconnect complies with a Distributed Virtual Memory (DVM) version that supports Realm Translation Regimes and *TLB Invalidate by PA* operations.

B3.2.4 Memory Protection Engine System Requirements

I_{ZNLMR} A Memory Protection Engine (MPE) in an RME system is either configured by MSD or a Trusted subsystem or implemented as an autonomous SSD component.

R_{KSPKN} Encryption keys used by MPE are stored in registers that are reset to a known default value on an RME system reset.

I_{KZZCG} An MPE supports a different encryption key or tweak for each PAS.

R_{QDPVN} Any PAS other than the Non-secure PAS must have encryption enabled.

R_{VSMPS} The decision to enable encryption for the Non-secure PAS is either hardwired or defined at boot and immutable once set.

I_{WHCDD} An MPE can include functionality that verifies the integrity, that is, the correctness and freshness, of data read from memory.

R_{YHXPH} An MPE integrity error is reported as an external abort to a software or hardware agent consuming the error.

R_{YJDSJ} Any captured details of an MPE integrity error are only visible to MSD.

I_{NDPVG} If a speculative access is made to a memory location that is integrity protected and that access is tagged with an incorrect PAS, the integrity check can fail but it must not result in a fatal error.

An example of a speculative access that might have an incorrect PAS is a read access generated for a translation table walk for which the granule protection check for the address being accessed has not been architecturally resolved.

I_{KCTYS} The properties of Memory Protection Engines are reflected through the following structures:

- Global MPE properties like "Encryption using a per-PAS key" can be captured in the System Properties structure in Root Non-volatile Storage.
- Programming interface details of MSD controlled MPEs are identified through MSD resource discovery.

R_{LPQSN} An MPE property that is reported through the System Properties structure in Root Non-volatile Storage (RNVS) is supported for all external memory ports in the system.

See also:

- B3.3 *Resource discovery*.
- B4.3 *RAS*.

B3.2.5 Trusted System Control Processor

I_{MCHBN} The RME system architecture anticipates the presence of a System Control Processor (SCP) within an implementation which can be responsible for operations such as system initialization and run-time power management. Where such a processor must have the ability to access locations in the Root PAS, it becomes a Trusted subsystem and is referred by this specification as a Trusted SCP.

R_{SXCFK} A Trusted SCP is an on-chip control processor that is trusted by MSD and can access resources in the Root PAS.

R_{ZHJQJ} A Trusted SCP is considered a Trusted subsystem and must meet the applicable security requirements defined in [2], for example, supporting Secure boot and having attestable firmware.

I_{CLYQH} An implementation must guarantee that the Trusted SCP is able to specify the correct PAS when accessing any system resource and specifically Root PAS resources.

Root PAS resources include:

- MSD SMEM.
- MSD-Protected Registers (MPRs).
- Memory-mapped resources of Trusted subsystems and Trusted requesters.
- Main memory (such as DRAM) assigned to the Root PAS.

R_{MZDXV} It is permitted for a Trusted SCP to have a mechanism to bypass a PAS filter which filters its transactions.

I_{MMVRL} The Trusted SCP must be able to access registers in the system before any programmable PAS filter, including an SMMU-attached Granule Protection Check, has been configured by MSD.

B3.3 Resource discovery

I$_{JDXSD}$ The method by which MSD discovers RME resources in the system is out of scope for this specification.

This specification uses the term *MSD Resource Discovery* to describe the mechanism that allows enumerating these resources.

How MSD resource discovery is implemented is IMPLEMENTATION DEFINED. For example, resources can be discovered through a firmware structure that is part of MSD firmware.

MSD resource discovery includes information like the following:

- Locations of system PAS filters, for example SMMUs, and system Memory Protection Engines that are managed by MSD.
- Location of MSD SMEM.
- Location of Root Non-volatile Storage (RNVS).
- Locations of DRAM carve-outs that are reserved for MSD.
- Locations of Storage Class Memory in the system address map.

Chapter B4
System security properties

This chapter defines general security properties and capabilities that must be supported by the System Architecture to ensure RME security.

This chapter extends other security specifications that may be part of a system's certification profile, for example Trusted Based System Architecture.

B4.1 Root of Trust Services

B4.1.1 Non-volatile storage

This section describes System Architecture requirements for Root Non-Volatile Storage.

I$_{QPVZF}$ *Root Non-Volatile Storage* (RNVS) is an on-chip non-volatile storage resource, such as fuses or on-chip flash that stores Arm CCA immutable boot parameters.

R$_{WNPYD}$ A programming interface that allows read and write access to RNVS must be in the Root PAS.

I$_{VZVBQ}$ RNVS parameters can be conceptually partitioned to two categories, public parameters and confidential parameters.

R$_{QCHPW}$ The system supports a method for permanently blocking write access from application PEs to all RNVS parameters.

R$_{LMSSL}$ The system supports a method for permanently blocking read access from application PEs to RNVS confidential parameters.

I$_{MZTMB}$ The following table provides examples for RNVS public parameters.

Table B4.1: RNVS public parameters

Parameter	Description
MSD firmware Anti-rollback	For example: a monotonic counter specifying minimal firmware version.
System Properties	A vector in which each property is assigned a single bit:
	“0” - property is supported.
	“1” - property is not supported.
	System Properties can be included in the Realm attestation report.
MSD Verification Key	Public key for validating the MSD firmware signature.

I$_{RZNNN}$ System Properties is a structure that allows the system integrator to report to MSD the set of chosen implementation options of an RME system.

R$_{VXBYG}$ System support for any memory protection property reported in System Properties is immutable and applicable for all DRAM memory controllers in the system.

I$_{TGKMJ}$ RNVS confidential parameters are immutable confidential material such as a Hardware Unique Key (HUK) and other private keys. These are specified in [2].

I$_{SBYFQ}$ The physical implementation and factory provisioning methods of RNVS depend on the security certification profile of the system and must comply with requirements for *Shielded locations* or *Isolated locations* as specified in [2].

B4.1.2 Root watchdog

I$_{FDTWY}$ The term *Root watchdog* refers to a watchdog that is exclusively controlled by MSD or a Trusted subsystem. An RME implementation can include a Root watchdog for detecting and recovering from MSD functional faults.

R$_{ZHBBL}$ The memory-mapped registers of a Root watchdog are in the Root PAS.

R$_{VXGBP}$ A Root watchdog is capable of triggering an RME system reset when predefined expiration conditions are met.

I_{ZYNSW} For a generic watchdog implementation as specified by [5], this can be supported by routing Watchdog Signal 1 (WS1) to a Trusted subsystem such as a Trusted SCP or hardware that directly trigger an RME system reset.

I_{ZDCNQ} A Root watchdog can generate an interrupt to EL3 when predefined expiration conditions are met. The method for securely delivering the watchdog interrupt to EL3 is IMPLEMENTATION DEFINED.

B4.1.3 Random Number Generator

R_{QYRGG} MSD and RMSD are provided with a private interface for accessing a True Random Number Generator (TRNG) that meets the certification profile of the system.

I_{MQMBF} [1] specifies the related requirements for Armv8 RNG architecture extension support.

B4.1.4 Cryptographic Services

I_{DKFMS} The RME System Architecture does not specify hardware requirements for generic cryptographic services, for example:

- Encryption cipher modes and Message Authentication Codes.
- Public Key Cryptography.
- Key generation and Key Derivation Functions (KDF).
- Secure key storage and secure measurement storage.
- Monotonic counters and trusted time.

MSD and RMSD can implement required cryptographic functionality such as AES-GCM-256 or SHA using a software library that leverages the various ARMv8 cryptographic extensions.

Other cryptographic functionality might be offered as a system service implemented by SSD and exposed through a memory-mapped interface in the Root PAS.

B4.1.5 Hardware Enforced Security

I_{HTMDC} This section specifies rules for RME systems that support Hardware Enforced Security (HES) as defined in [2].

HES moves critical Arm CCA security features off application PEs and into an isolated Trusted subsystem responsible for Arm CCA initial measurements, identity and attestation services, debug authentication, and lifecycle state management.

HES functionality can be hosted on a dedicated Trusted subsystem or as an isolated tenant within a multi-tenant Trusted subsystem.

A HES implementation integrates RNVS functionality, cryptographic functionality, and secure measurement storage forming a private execution environment that shares no resources with application PEs. This allows for the following:

- A guarantee that firmware running on an application PE cannot influence Arm CCA initial measurements or Arm CCA initial boot state.
- Physical isolation of critical Arm CCA resources by preventing direct access to root secrets, or to operations on root secrets, from application PEs.

Arm strongly recommends that RME systems support HES.

$\mathrm{I}_{\mathrm{YBZMD}}$ The HES implementation has private execution resources, memory resources, and cryptographic resources that are not shared with application PEs or other hardware agents. The system can further provide a HES implementation with private reset, clock and power domain to allow it to maintain context independently to the system power state.

The HES implementation can include an updatable component such as a firmware image. The HES updatable component is measured by immutable HES logic and the measurement result is included in attestation tokens that are composed and signed by HES.

$\mathrm{R}_{\mathrm{NWQBJ}}$ If HES is hosted as a tenant within a multi-tenant Trusted subsystem, HES functionality must be isolated from other tenants, such that tenants must not be able to monitor HES functionality or impact HES functionality or integrity.

$\mathrm{R}_{\mathrm{HJSSG}}$ The HES implementation exposes a private interface to SSD components such as Trusted subsystems for requesting HES services.

$\mathrm{I}_{\mathrm{PMWXL}}$ Services provided by HES for SSD are defined in [2] and can include:

- Extending one or more measurements with a digest calculated by SSD.
- Calculating a digest of an image or boot state provided by SSD, for example a Trusted SCP image, and extending a measurement with that digest.

$\mathrm{R}_{\mathrm{CGDVX}}$ The HES implementation exposes a programming interface in the Root PAS, shared by all application PEs, allowing MSD and PE Initial boot ROM to request for HES services.

$\mathrm{I}_{\mathrm{BNKKC}}$ Services provided by HES for PE Initial boot ROM include:

- Extending a measurement with a digest calculated by PE Initial boot.

Services provided by HES for MSD include:

- Composing and signing an initial attestation token for MSD.
- Extend one or more measurements with a digest calculated by MSD, for example a digest measuring RMSD image.

$\mathrm{R}_{\mathrm{BQPFG}}$ HES has exclusive read and write access to RNVS confidential parameters.

$\mathrm{R}_{\mathrm{BTWVY}}$ A measurement register can be either extended using a secure hash algorithm, locked or reset.

$\mathrm{R}_{\mathrm{DFPJL}}$ HES has exclusive access to extend, lock and reliably obtain the value of a measurement register it owns.

$\mathrm{R}_{\mathrm{FWSRF}}$ Once locked, a measurement cannot be further extended until it is reset.

$\mathrm{R}_{\mathrm{WYSLK}}$ An RME system reset is the only method to reset a measurement owned by HES.

$\mathrm{R}_{\mathrm{XCRMH}}$ On an RME system reset, HES state is reset to a known value, including all measurements and ephemeral cryptographic context.

B4.2 System isolation properties

B4.2.1 System configuration integrity

I_{PDPGT} MSD must establish the correctness and integrity of any System register that can risk its security guarantees.

For example, MSD must assert that memory controller configuration is consistent as well as any configuration that controls the mapping of physical address spaces to memory controllers.

I_{NKJLK} A MSD-Protected Register (MPR) is a configuration register holding a value that can compromise MSD functionality.

MPRs must be in the Root PAS and can only be configured by MSD or by a Trusted subsystem.

I_{WVSXK} The list of MPRs in a system is IMPLEMENTATION DEFINED and [2] provides details on their possible classes. Here are some examples:

- Interconnect Registers that control mapping of physical addresses to DRAM memory ports.
- Registers that allow modifying the address or data fields of a memory access or cache maintenance message or can impact its delivery.
- Memory controller registers that allow modifying the value of a memory location, modifying the address or data of an access or violate RME RAS rules.
- Registers that might store confidential information or that provide access to memory, buffers, or caches that might store it.
- Registers controlling debug/DFT access to memory, buffers, or caches.
- Registers controlling power and reset settings or Trusted SCP registers.
- Registers controlling the system counter functionality.

Identifying all MPRs in a system requires a careful security analysis that addresses both direct channels and side channels.

B4.2.1.1 Temporal isolation

This section defines properties and rules related to temporal isolation of memory-mapped registers.

I_{DDMPS} A register is write-lockable if MSD has a method to block writes to it from all requesters at any Security state.

A register is read-lockable if MSD has a method to block reads to it from all requesters at any Security state.

I_{PCLND} Examples for how the write-lockable and read-lockable properties can be implemented are:

- Using write-lock and read-lock signals that are controlled by a register in the Root PAS.
- Using the *No accesses permitted* GPC encoding specified in [1], if the properties can be applied to all registers in a single Physical Granule.

R_{VDFYZ} A register that is located outside of the Root PAS but can affect a service provided by MSD must be implemented as a *measurable register*.

R_{YLVDB} A measurable register is a write-lockable register that MSD has a trusted method to obtain its value.

I_{GSPZK} As an example, measurable registers can be used for a configuration that defines a protection policy or a property of an address range that are not directly controlled by MSD. After MSD sets the write-lock of a measurable register it must have a method to read its value using a path that is controlled by SSD, MSD or by other measurable registers.

B4.2.2 Reporting of critical errors

I_VZDVG A functional error that can violate the RME security guarantee such as a GPC walk fault is reported either to MSD or to a Trusted subsystem.

[1] describes how GPC faults can be reported to EL3 using synchronous GPC exceptions.

When an error is detected asynchronously, it can be signalled to application PEs using the SError interrupt or can be wired directly to a Trusted subsystem. Examples include:

- A GPC fault detected by an SMMU that is reported in SMMU_ROOT_GPT_CFG_FAR as specified in [3].
- A GPT walk fault detected by an MMU upon a write to the Trace Buffer.
- A RAS critical error detected by a Trusted requester.

B4.3 RAS

The RME architecture specification [1] provides implementation requirements for RAS in PEs and system components using the Arm RAS architecture [6]. This section provides additional rules for system components described in this specification.

B4.3.1 Confidential information in RAS Error Records

The RME architecture [1] describes the concept of *confidential information* and provides the requirements on the content that can be recorded in RAS error record registers.

$\mathrm{R}_{\mathrm{GNGMB}}$ Only SSD or MSD are capable of controlling whether recording is performed for error records that might contain confidential information.

B4.3.2 RAS Error signaling

The following rules describe constraints on how errors must be signaled to PEs or other system components in an RME system using the Arm RAS architecture.

$\mathrm{I}_{\mathrm{BHNYL}}$ For rules on signaling errors to PEs, see the RME architecture specification [1].

$\mathrm{R}_{\mathrm{GZTVL}}$ Critical Error Interrupts (CI) must be wired to a Trusted subsystem that will respond with an RME system reset.

$\mathrm{R}_{\mathrm{LWVCX}}$ An uncontainable error results in an RME system reset.

$\mathrm{I}_{\mathrm{HYXZN}}$ Components that implement RAS can be reset with an Error Recovery Reset, so that RAS syndrome information is not lost.

$\mathrm{R}_{\mathrm{JNBWJ}}$ Only SSD or MSD can enable or disable the generation of a CRI.

B4.3.3 RAS for Memory Protection Engine

Section B3.2.4 *Memory Protection Engine System Requirements* introduces the concept of MPE and its associated requirements. This section covers RAS requirements associated with MPEs.

$\mathrm{R}_{\mathrm{XPCTR}}$ Where an MPE provides support for integrity, if it detects an integrity error it can perform one of the following responses:

- Respond by returning poison back to the consumer and record the error as a deferred error.
- Respond with an in-band error response and record the error as an uncorrected error.

$\mathrm{R}_{\mathrm{HSVLQ}}$ Only SSD or MSD must be able to control the abilities of detecting, propagating, and reporting MPE integrity errors.

$\mathrm{I}_{\mathrm{DKXXG}}$ Arm recommends MPEs with integrity support implement the Arm RAS system architecture [6].

$\mathrm{R}_{\mathrm{GZHTD}}$ In addition to providing encryption and, where implemented, integrity capabilities, the MPE is able to pass poison information:

- If a requester above the MPE defers errors by writing poison, then the MPE must be able to pass this value through to the memory system below it as poison.
- If a requester above the MPE consumes a memory location that has been marked as poison, either as a result of that access or a previous access, the MPE must pass that poison to consumer.

$\mathrm{I}_{\mathrm{TMKYF}}$ Passing poison through the MPE does not weaken its security properties.

B4.4 MPAM

I$_{WRPFN}$ The complete definition of MPAM changes for RME is specified in [7].

I$_{MYBCG}$ A PE that implements RME is capable of generating a 2-bit MPAM_SP encoded as:

- 0b00 - Secure.
- 0b01 - Non-secure.
- 0b10 - Root.
- 0b11 - Realm.

I$_{QZQNG}$ Non-PE requesters in an RME system are capable of generating the following MPAM_SP encodings:

- 0b00 - Secure.
- 0b01 - Non-secure.

Other encodings for non-PE requesters are in the scope of future versions of the RME Architecture.

I$_{ZFXNS}$ A Four-space MSC is an MSC that supports 4 PARTID spaces.

I$_{QNCKT}$ An MSC that supports only 2 PARTID spaces can have a PARTID space mapper that allows associating the PARTID space of an incoming message with one of the PARTID spaces supported by the MSC. The registers controlling the PARTID space mapper are in the Root PAS and are defined in [7].

R$_{RFSYB}$ An RME system propagates a 2-bit MPAM_SP field to all MSCs that are either a Four-space MSC or have a PARTID space mapper.

I$_{BTNRR}$ An RME system can include a combination of Four-space MSCs, MSCs that support a PARTID space mapper and Two-space MSCs.

I$_{VSQFK}$ An RME system can include a Two-space MSC only where this does not leak confidential information.

I$_{FXLRK}$ An access to a Granule Protection Table by a PAS filter is tagged with the PARTID and PMG of the memory access being filtered. This is consistent with MPAM rules on PARTID selection for translation table walk accesses.

B4.5 MTE

I_{FTYRS} [1] defines the required changes to the Memory Tagging Extension (FEAT_MTE) when RME is implemented. Memory Tagging makes use of Allocation Tags that can be assigned for Normal memory locations in the system. One implementation option for storing Allocation Tags is using an array located in main memory.

I_{DFCMZ} Where an implementation is using addressable memory for storing Allocation Tags, the hardware physical address range of such memory (MTE carve-out) can only be accessible to:

- Hardware responsible for the association of Allocation Tags with physical addresses.
- An Access in Root PAS - for implementations that expose Allocation Tags through regions of the data PA space.

The MTE carve-out must be protected by a filter located either at the completer or at the requester side that guarantees this behavior.

Encryption of Allocation Tags in main memory is an optional defence-in-depth capability for mitigating attempts to read or corrupt tags. If performed, and the MTE carve-out is accessible in the Root PAS then encryption must be done using the Root PAS encryption key. Otherwise, encryption may be done using a key associated with the PAS of the data that corresponds to the Allocation Tag.

R_{JYMQD} Allocation and protection of the address range assigned to an MTE carve-out are controlled by either SSD or MSD.

B4.6 Side channel resistance

B4.6.1 System PMU counters

I_{DSBRJ} Performance Monitoring Unit (PMU) counters are a potential side channel for leaking confidential information such as access patterns or control flow of execution in a Security state protected by the RME security guarantee. The rules provided below augment existing rules specified in [8] for guaranteeing that system PMUs do not leak confidential information. These rules apply to any PMU counter in the system, including architectural and IMPLEMENTATION DEFINED MPAM monitors.

R_{HRVJB} A system PMU counter that is accessible in the Secure PAS can only count events that are attributable to the Secure PAS or to the Non-secure PAS.

R_{BSZPN} A system PMU counter that is accessible in the Realm PAS can only count events that are attributable to the Realm PAS or to the Non-secure PAS.

R_{TMSNN} A system PMU counter that is accessible in the Root PAS can count events that are attributable to any PAS.

R_{MMPWY} A system PMU counter that is accessible in the Non-secure PAS can count events that are attributable to a specific PAS if there is a per-PAS authentication control that can permit events from that PAS to be counted.

R_{PLXZB} A per-PAS authentication control, as specified in [8] can be driven by a debug authentication interface signal or by a register accessible in the corresponding PAS or in the Root PAS.

R_{CFYKS} An event that is not explicitly associated with a PAS but can leak confidential information is implicitly associated with the Root PAS.

B4.6.2 Fault attacks using signal and power manipulations

I_{NSXXG} The integrity and robustness of clock, reset, and voltage supplies of a system are an important aspect of its security guarantee. While this specification does not define explicit requirements for addressing fault injection attacks, it is strongly recommended that RME implementations consider the system immunity against attacks that may make use of these physical properties. Examples for possible measures include:

- Detection of glitches on input power supplies and of voltage levels dropping or exceeding the levels allowed by specification.
- Increasing noise-immunity of reset control signals.
- Reliable delivery of clocks to system components with duty-cycle and jitter that meet specification boundaries.
- Implementing any register that controls voltage settings, power switch on/off settings, clock duty cycle, PLL settings, or power-on-reset settings as an MSD protected register (MPR).

B4.7 Architectural differences

I_{JFJVD} Architectural differences between application PEs in an RME system present a potential security exposure and increase management software complexity. For example, supporting a different physical range per PE by having different values in ID_AA64MMFR0_EL1.PARange could compromise the enforcement of memory isolation. Supporting different cache policies through CTR_EL0.L1Ip could compromise Realm memory confidentiality through incorrect cache maintenance operations.

The rules in this section aim to minimize architectural differences between application PEs and guarantee that where a difference exists it does not compromise the RME security guarantee.

R_{SQMWT} Application PEs in an RME system do not have architectural differences unless this is explicitly permitted by this specification.

I_{BFCFX} An architectural feature must be implemented using the same revision and the same feature IDs across all application PEs that support it. This guarantees predictable software behaviour and simplifies the reporting of supported architectural extensions and features in an attestation report.

Non-uniform support of an architectural feature is possible only if the feature is not exposed to a Realm and not used by RMSD. Arm strongly recommends that all application PEs in an RME system implement uniform support of architectural features and where this is not possible guarantee that controls at EL2 or EL3 can be used to hide non-uniform features from Realms. These can be register and instruction trap controls at EL2 or feature enablement configuration at EL3. Using trap controls for guaranteeing feature uniformity has the drawback of requiring RMSD software to be platform-dependant and may not completely prevent hostile attempts to use the feature.

I_{JVTXC} Application PEs are allowed to have differences in the following architected registers:

Description	Short-Form	Permitted Differences
Main ID Register	MIDR_EL1	Part number [15:4], Revision [3:0], Variant [23:20], Implementer [31:24].
Virtualization Processor ID Register	VPIDR_EL2	Same fields as MIDR_EL1, writable by hypervisor.
Multiprocessor ID Register	MPIDR_EL1	Bits [39:32] and Bits [24:0]. Affinity fields and MT bit.
Virtualization Multiprocessor ID Register	VMPIDR_EL2	Same fields as MPIDR, writable by hypervisor.
Revision ID Register	REVIDR_EL1	Specific to implementation indicates implementation specific Revisions/ECOs. All bits can vary.
Auxiliary ID Register	AIDR_EL1	IMPLEMENTATION DEFINED identification information
Current Cache Size ID Registers	CCSIDR_EL1 CCSIDR2_EL1	The number of Sets and the associativity of Caches can be different on each PE.

The following differences are allowed by the Server Base System Architecture [5] but are not allowed by the RME system architecture:

Description	Short-Form	SBSA differences *not* allowed by RME
Cache type register	CTR_EL0	Bits [15:14] Level 1 Instruction Cache Policy (L1IP).

Description	Short-Form	SBSA differences *not* allowed by RME
AArch64 Memory Features Register	ID_AA64MMFR0_EL1	Bits [3:0] describing the supported physical address range.

R_{CFYBJ} An IMPLEMENTATION DEFINED property of an architecture extension or an IMPLEMENTATION DEFINED difference between application PEs must not create an exposure that could break the RME security guarantee.

R_{XKBNZ} PE behavior is UNPREDICTABLE when the following are true:

- An IMPLEMENTATION DEFINED difference between application PEs is visible to software, for example through different System register values across PEs.
- There is a mismatch between the register value assumed by software running on a PE and the actual hardware value of the PE.
  - An example where such mismatch could occur, is if software obtained the value by reading it on a different PE.

I_{DTDWX} UNPREDICTABLE behavior as defined in the Arm architecture is constrained to forbid privilege escalation.

Chapter B5
Power Management

This chapter specifies RME system power management rules.

Power management flows define how the system and its components transition between various power states and how associated operations like switching-off power domains and managing context are executed.

The RME Power Management requirements described in this chapter address the following:

- Prevent compromise of the RME security guarantee using power management manipulations, for example due to loss of context.
- Minimize RME impact on power management functionality and system power consumption.

B5.1 System power management

B5.1.1 Power states

R_{LRQXZ} A software-initiated power state transition in an RME system at any level of the system hierarchy (PE, PE-cluster, System) is validated by MSD or by a Trusted subsystem.

B5.1.2 PE power management

I_{TGBQF} At the exit from any PE low-power state in which PE context is lost, instruction execution begins at MSD. Before allowing code in other Security states to run, MSD sets up PE context to maintain the RME security guarantee at the PE level and at the system level.

R_{WJVRX} Save/Restore operations for MSD PE context can only be done by MSD or a Trusted subsystem and use storage that is not accessible from Realm, Secure and Non-secure states.

R_{MVZHF} Save/Restore operations for RMSD PE context can only be done by RMSD, MSD, or a Trusted subsystem and use storage that is not accessible from Secure and Non-secure states.

R_{RCLYM} Save/Restore operations for PE context of Secure state can only be done by MSD or a Trusted subsystem or software running in the Secure state and use storage that is not accessible from Realm and Non-secure states.

B5.1.3 System and PE-cluster power management

I_{JMNJC} System power management operations that involve power-gating caches or interconnect elements present an attack surface and therefore must be managed by trusted power control which can be implemented through MSD firmware, a Trusted subsystem such as a Trusted SCP, SSD power control logic, or a combination thereof.

As an example, before powering-down a component within a level of the system hierarchy (PE, PE-Cluster, System) power control must establish that a coherent cache clean operation for all data caches in that level has been performed.

When powering-up a level of the system hierarchy (PE, PE-Cluster, System) power control must establish that all data and instruction caches are in INVALID state.

R_{GVJYZ} Any register that affects a system power policy or a hardware power mode is implemented as an MSD-Protected Register (MPR).

I_{PKLDS} *MSD state* is defined to be any system state that affects MSD behavior. *MSD state* includes the following:

- System structures storing MSD PE context.
- SMEM in the Root PAS.
- DRAM assigned to the Root PAS, for example GPT.
- MPRs and measurable registers.
- PAS filters context and MPE context.
- Cache state and Snoop filter state.

R_{KYXMR} Any power management operation that can affect MSD state or the RME security guarantee must be validated by MSD or a Trusted subsystem.

I_{KVFFP} For example, a Trusted SCP can expose an interface to Non-secure software to modify the performance attributes of the system but must not permit the programming of invalid values or invalid value combinations.

B5.1.4 System power states

I_{YDHQF} Once all PEs have been placed in a power state in which PE context is lost the system can enter a power state in which its context is lost, for example by calling PSCI SYSTEM_OFF. If PEs are placed in a low power state in which PE context is preserved, the system can enter a low power state in which its context is preserved, for example by calling PSCI SYSTEM_SUSPEND.

R_{MLJVR} On an exit from a low power state in which system context is preserved, power control guarantees that MSD state is fully preserved. If MSD state is not preserved, power control applies an RME system reset.

I_{FVZXC} Managing MSD state during a power state in which system context is preserved can be done using system retention structures or using Save/Restore operations for lost context.

R_{ZNLSZ} Save/Restore operations for MSD state can only be done by MSD or a Trusted subsystem and use on-chip storage that is not accessible from Realm PAS, Secure PAS or Non-secure PAS.

I_{WDNPT} On an exit from a low power state, MSD establishes if any MSD state is no longer valid, in which case MSD performs initialization operations in a cold boot manner.

B5.2 RME components power management

I_{ZPKNM} The term ACTIVE is used in this section to describe a power mode in which a component is fully operational.

A component in any low-power mode, for example clock-gated, power-gated, or retention is no longer in the ACTIVE power mode.

Also, a component that has not fully restored its context following an exit from a low-power mode is not in the ACTIVE mode.

I_{ZQCHL} A PAS filter or MPE can be placed in a non-ACTIVE mode if this does not violate the RME security guarantee. For example, if all requesters associated with an MMU-attached PAS filter are reset or power-gated MSD can permit the power-gating of the filter. A completer-side PAS filter can be power-gated when the corresponding peripheral is power-gated.

A transition of a PAS filter or MPE to a non-ACTIVE mode in which context is retained can be done autonomously. Otherwise, if context is lost the transition can be carried by trusted power control. Power control verifies that the transition is allowed, and that context is correctly restored before moving back to ACTIVE mode.

R_{DQTSG} An MPE or a PAS filter in a non-ACTIVE mode in which context is not fully retained blocks its operation and does not service requests until it is in ACTIVE mode again.

R_{JDBCS} An MMU-attached PAS filter in a non-ACTIVE mode either continues to respond to GPT cache invalidations, or invalidates any cached state when moving back to ACTIVE mode.

Chapter B6
Debug

I_{LRRRJ} The term RMSD external debugging is used in this chapter to describe any system or PE external debugging capability that enables the following:

- Monitoring or modifying RMSD behavior.
- External access to the Realm PAS or Realm Security state.

The term Root external debugging is used in this chapter to describe any system or PE external debugging capability that enables the following:

- Monitoring or modifying MSD behavior.
- External access to the Root PAS or Root Security state.

This includes both invasive and non-invasive hardware debugging capabilities that provide access to a component used by MSD or RMSD, including Debug Access Ports, trace hardware, or debug registers.

I_{GPSGG} A Secured Arm CCA system is a system that is provisioned to run Arm CCA at the Secured lifecycle state as defined in [2].

R_{QSXBZ} RMSD external debugging and Root external debugging are disabled by default on a Secured Arm CCA system.

R_{HLTLK} RMSD external debugging can only be authorized following an RME system reset and before RMSD firmware is loaded.

R_{XVNFV} Root external debugging can only be authorized following an RME system reset and before MSD firmware is loaded.

I_{DSKVX} RMSD external debugging and Root external debugging can only be enabled on a Secured Arm CCA system after performing the following operations:

- Authenticating a signed request such as a debug certificate for enabling a debugging mode.
- For RMSD external debugging:

    - Guaranteeing that RMSD external debugging state is visible to MSD.
- For Root external debugging:
    - Guaranteeing that RNVS confidential parameters and any other SSD or MSD non-volatile confidential parameters are inaccessible.
    - Guaranteeing that any confidential state from previous boot that may reside in TLBs, caches, or PE registers is invalidated or scrubbed.

R_{GTPGZ} When Root external debugging is enabled, the RNVS confidential parameters are either inaccessible, scrubbed, or populated with debug values.

R_{RHGKX} Access to a Secured Arm CCA system through an external debug or test interface, including debug access ports, JTAG ports, and scan interfaces is disabled by default. Debug access can be enabled following validation of a debug certificate or password which is injected via an external debug interface.

I_{VNSTB} [1] describes how the Debug Authentication Interface defined in [4] is extended to express the RMSD external debugging and Root external debugging capabilities. In compliance with the rules specified in this section, Debug Authentication Interface signals that enable RMSD external debugging and Root external debugging can only be set before MSD starts executing.

Chapter B7
System boot

This chapter describes requirements for initializing an RME system.

B7.1 Reset requirements

I_{YRDRZ} An RME system reset is any system event that resets the entire global functional state of the system.

An RME system reset includes the resetting of PEs, PE-clusters, system core logic and auxiliary logic, all system buses, and all system peripherals.

From MSD perspective, an RME system reset event is observed as the logical equivalent to power cycling the platform.

On coming out of reset, execution starts at an address in the Root PAS.

R_{HJHRL} On an RME system reset, all Trusted requesters and Trusted subsystems are reset. Any Trusted subsystem state that might include MSD or RMSD confidential information is reset to known values.

I_{ZVHWK} Examples for Trusted requesters and Trusted subsystems are GPCs, MPEs, a Trusted SCP, and a Trusted subsystem hosting HES.

R_{KKSQB} An RME system reset propagates to PEs as either a Cold reset, Warm reset, or Error recovery reset.

R_{HLKZP} An RME system reset might propagate to any component that implements RAS [6] as an Error recovery reset.

R_{SSGMJ} The reset of a system component that affects the RME security guarantee can only be controlled by MSD or a Trusted subsystem, or driven by an RME system reset.

I_{CRGLD} An RME system reset is not required to explicitly reset external memory. For example, a SYSTEM_WARM_RESET operation as defined by PSCI [9] is a permitted variant of an RME system reset.

See also:

- Reset chapter in [1]
- B4.3 *RAS*

B7.2 RME disable

I_{NMQLP} An RME system can fall back to supporting only two Security states and two physical address spaces if it includes a LEGACY_TZ_EN system tie-off.

R_{KQLKN} LEGACY_TZ_EN is not permitted to change value after RME system reset has been deasserted.

I_{TJPLF} Disabling RME functionality can be controlled by a fuse that drives the value of the LEGACY_TZ_EN tie-off before RME system reset deasserts.

Disabling RME as a firmware boot option is not currently supported. A boot time option for disabling RME complicates the security analysis due to potential leakage of confidential information across boot cycles. It also implies a functional challenge in synchronizing the resource transition from Root PAS to Secure PAS across multiple system components with the MSD firmware transition from Root state to Secure state.

The Granule Protection Check can be disabled by MSD firmware if Granular PAS filtering is not required.

See also:

- Chapter B8 *System construction*

Chapter B8
System construction

B8.1 Using RME IP in a legacy system

I$_{TLFMJ}$ RME IP that might be integrated in systems that support only two Security states can have an input tie-off (*LEGACY_TZ_EN*) that enables a legacy behavior.

R$_{KXMHF}$ A system that contains RME components, that have the LEGACY_TZ_EN input, will drive a common tie-off input value into all components.

I$_{LRMWL}$ IP with the LEGACY_TZ_EN tie-off might include PEs, SMMUs, components with a completer-side PAS filter, and logic that enforces the PAS Access Table (Table B2.1).

R$_{HCGZN}$ If LEGACY_TZ_EN is TRUE, PAS[1] is driven to 0b0 by any logic that enforces the PAS Access Table (Table B2.1).

R$_{CLKXF}$ A PE that supports the LEGACY_TZ_EN tie-off hides the RME capability if LEGACY_TZ_EN is TRUE and reverts all functionality defined by RME.

B8.1.1 Peripheral isolation in legacy systems

I$_{QGRFK}$ A peripheral with a private completer-side PAS filter which might be used in a system that supports only two Security states can implement the LEGACY_TZ_EN input tie-off.

When set, the LEGACY_TZ_EN input tie-off forces the peripheral HW to fallback to supporting only two physical address spaces.

If LEGACY_TZ_EN is TRUE:

- Peripheral registers that are in the Root PAS and affect global functionality are assigned to the Secure PAS.
- Peripheral registers that are in the Realm PAS can be assigned to the Non-secure PAS or become inaccessible.

See also:

- B3.2.1 *PE*

B8.2 Using legacy IP in an RME system

I_{DFSNF} An RME system might include legacy requesters or completers that support only two physical address spaces.

R_{CKBGZ} A legacy completer is attached to an RME IP by driving the NS signal of the completer from PAS[0] of the RME IP.

R_{YKSSD} A legacy requester is attached to an RME IP by driving PAS[0] of the RME IP from the NS signal of the legacy requester and driving PAS[1] of the RME IP to 0b0.

I_{WKZHW} A legacy requester that is fully coherent must not receive or respond to snoop requests in the Realm PAS or Root PAS.

B8.3 Memory hot plug

I_{QXFYH} An RME System can support memory hot-insertion and hot-removal operations if the following conditions are met:

- MSD has a method to establish if a memory slot is populated and learn about hot-insertion and hot-removal events of memory modules.
- An access to an unpopulated memory slot produces an error equivalent to an MPE integrity error.

Once a memory module is hot-inserted, MSD can assign granules that use this memory to a protected PAS after establishing that the memory slot configuration is correct and that MPE capabilities are enabled for the slot.

Chapter B9
Appendix: System flows

B9.1 System Initialization flow

This section provides an example of an initialization flow of an RME system, and describes the relations between system components and corresponding security considerations for a system boot sequence. This section does not include specific details about secure boot sequences, or software measurement flows.

Reset Deassertion:

- RME system reset deasserts.
  - SMMU-attached GPCs assume a default policy that blocks all memory accesses.
    - This does not block Trusted requesters such as HES from accessing system resources.
  - MMU-attached GPCs assume a default policy that permits MSD to access the system.
  - MPEs are in reset state. All encryption key state is set to zero by HW, which effectively scrubs DRAM content.
- HES starts executing and measures SSD state, for example, firmware images of Trusted subsystems.
- A Trusted SCP starts executing and performs system initialization that can include DRAM configuration.

Application PE Initial Boot:

- Performed by application PE Initial boot code, for example a PE boot ROM executing at EL3, and HES. In the following example, the PE boot ROM performs the measurement of MSD firmware.
  - HES releases application PE reset.
  - PE boot ROM loads MSD firmware image into MSD SMEM locks, measures it, and submits the MSD measurement to HES.
  - PE boot ROM validates MSD firmware, for example using MSD Verification Key, and launches MSD.
  - MSD establishes if global initialization is required, for example if this is the Primary PE after a cold reset.

    - ∗ If yes, perform MSD Initialization. Otherwise, skip to *Initial Boot Exit* after completing local PE initialization.
    - ∗ Until MMU is enabled, all PE memory accesses are in the Root PAS.

MSD Initialization:

- MSD firmware completes system initialization operations. This includes:
  - Configuring any MPRs that were not configured by Trusted SCP or HES.
  - As an optional step, launch an S-EL2 firmware image in order to perform system initialization operations such as DRAM and interconnect configuration:
    - ∗ Firstly, perform GPT Initialization but configure only the Level 0 GPT entries with block descriptors set to “All accesses permitted”.
      - · This maintains the behavior that GPC is always enabled when executing in EL2 or lower Exception levels.
    - ∗ Secondly, load, validate, and launch the S-EL2 image. S-EL2 firmware executes and returns back control to MSD Initialization.
  - Lock and validate all measurable registers in the system.
  - Perform GPT Initialization.

GPT Initialization:

- SMMU-attached GPCs and MPEs in the system are identified through MSD resource discovery.
- MSD configures SMMU-attached GPCs and MPEs of the system to enable a secure initialization of the GPT.
  - Invalidate all data/unified caches in the system that are before the PoPA, including both private and shared caches, to prevent GPT corruption using dirty cache lines.
    - ∗ If GPT Initialization occurs immediately after reset, this might be guaranteed by the system default behavior such that no explicit invalidation is required.
  - Establish that MPEs enforce the required encryption and integrity properties of Root PAS main memory, such as DRAM.
  - Establish that non-Trusted requesters are blocked from accessing DRAM during the GPT initialization.
- MSD initializes GPT:
  - Resolve the system physical address space size, main memory ranges, and physical granule size.
    - ∗ This can be done using information from a firmware table.
  - Resolve the available Root PAS SMEM and DRAM resources allocated for the GPT, for example through MSD resource discovery.
    - ∗ An example of a GPT allocation:
      - · Level 0 GPT in SMEM with size defined by the Protected Physical Address Size and the window size of a level 0 GPT entry. For example, if the Protected Physical Address Size is 8TB then the Level 0 GPT requires 64KB of SMEM.
      - · Leve1 1 GPTs in DRAM with size defined by the total amount of DRAM in the system.
  - Initialize the level 0 GPT default values. For example:
    - ∗ Level 0 entries of DRAM regions: configure a level 1 GPT Table Descriptor.
    - ∗ Level 0 entries of non-DRAM regions: configure as “All accesses permitted”.
  - Initialize the level 1 GPT default values.
    - ∗ In the case of the address range that stores the GPT itself, assign to Root PAS.
    - ∗ Optionally, allocate DRAM carve-out memory ranges to Realm PAS and Secure PAS.
    - ∗ For the rest of the main memory address range, assign to Non-Secure PAS.
  - MSD completes GPC configuration by:
    - ∗ Enabling the MMU-attached GPC and SMMU-attached GPCs.
    - ∗ Invalidating all GPT caches, to enforce new GPT configuration.
    - ∗ After the MMU-attached GPC is enabled the MMU can be enabled.

Initial Boot Exit (performed by all application PEs):

- Before allowing boot operations for other Security states:
  - Scrub any PE register that might have held RMSD or Realm state, unless it is guaranteed to be reset to a constant value by a PE warm or cold reset.
  - Scrub any SMEM content that might have been assigned to RMSD on previous boot.

  - Invalidate PE GPT caches to flush stale state before GPC is enabled.
  - Once GPT Initialization is complete by Primary PE, enable MMU-attached GPC.
    * After GPC is enabled the MMU can be enabled.
  - For the Primary PE, guarantee that all PE TLBs, private caches, and shared caches have been invalidated. This is so that any potential secret from previous boot is flushed.
  - For all other PEs, guarantee that all PE TLBs and private caches have been invalidated. This is so that any potential secret from previous boot is flushed.

RMSD Initialization:

- Before MSD loads, validates and launches an RMSD image it performs the following:
  - Read System Properties from RNVS and verify that minimal conditions for enabling RMSD functionality are met.
  - Allocate required RMSD SMEM and DRAM resources. DRAM resources might have already been assigned during GPT Initialization.
  - Establish RMSD external debugging state. This has an impact on how RMSD boot state is derived.
  - Derive RMSD boot state as defined in [2] using RNVS parameters.
- RMSD performs the following as part of its boot:
  - Query MSD for the location and size of SMEM and DRAM resources allocated to RMSD.
  - Initialize RMSD using RMSD boot state.

PE entering a low-power state in which PE context is lost:

- MSD image performs operations that guarantee that the PE exits coherency domains without leaving dirty copies in any cache that is powered-down as part of the entry into the low power state.
- MSD image can then disable GPC for the PE before it enters the low-power state.

During low-power state the PE does not make accesses. On exit from low-power state instruction execution begins at MSD.

See also:

- B3.3 *Resource discovery*
- Chapter B5 *Power Management*

Glossary

Application PE

A PE used by the operating system or hypervisor to execute user application or kernel threads.

GPC

Granule Protection Check. The process of checking whether an access to a Location is permitted by the Granule Protection Table. This term is also used to refer to an MMU-attached or SMMU-attached Granular PAS filter that implements the Granule Protection Check.

GPT

Granule Protection Table

HES

Hardware Enforced Security

Location

An address in a Physical Address Space

MPE

Memory Protection Engine

MSD

Monitor Security Domain

PAS

Physical Address Space

PG

Physical Granule

PoPA

Point of Physical Aliasing (A point in the system that spans multiple Physical Address Spaces)

Resource

A physical entity that can be accessed at one or more Locations

RME

Realm Management Extension

RMSD

Realm Management Security Domain

RNVS

Root non-volatile storage

SCP

System Control Processor

SMEM

Shielded memory

SoC

System on Chip

SSD

System Security Domain