Arm® Realm Management Extension (RME) System Architecture

Document number	ARM-DEN-0129
Document quality	EAC
Document version	C.a
Document confidentiality	Non-confidential

Arm® Realm Management Extension (RME) System Architecture

Release information

Date	Version	Changes
2025/Dec/15	C.a	• Introduces RME Coherent Device Assignment, support for CXL Type-3 devices and support for the XT extensions.
2023/Nov/10	B.a	• Introduces requirements for supporting RME Device Assignment (RME-DA), Memory Encryption Contexts (MEC), and multi-chip systems.
2022/Oct/12	A.d	• Updated EAC release.
2022/Feb/07	A.c	• Updated EAC release.
2021/Nov/02	A.b	• Updated EAC release.
2021/Jun/23	A.a	• First EAC publication.

Arm Non-Confidential Document License (“License”)

This License is a legal agreement between you and Arm Limited (“**Arm**”) for the use of Arm’s intellectual property (including, without limitation, any copyright) embodied in the document accompanying this License (“**Document**”). Arm licenses its intellectual property in the Document to you on condition that you agree to the terms of this License. By using or copying the Document you indicate that you agree to be bound by the terms of this License.

“**Subsidiary**” means any company the majority of whose voting shares is now or hereafter owned or controlled, directly or indirectly, by you. A company shall be a Subsidiary only for the period during which such control exists.

This Document is **NON-CONFIDENTIAL** and any use by you and your Subsidiaries (“Licensee”) is subject to the terms of this License between you and Arm.

Subject to the terms and conditions of this License, Arm hereby grants to Licensee under the intellectual property in the Document owned or controlled by Arm, a non-exclusive, non-transferable, non-sub-licensable, royalty-free, worldwide License to:

(i) use and copy the Document for the purpose of designing and having designed products that comply with the Document;

(ii) manufacture and have manufactured products which have been created under the License granted in (i) above; and

(iii) sell, supply and distribute products which have been created under the License granted in (i) above.

Licensee hereby agrees that the Licenses granted above shall not extend to any portion or function of a product that is not itself compliant with part of the Document.

Except as expressly licensed above, Licensee acquires no right, title or interest in any Arm technology or any intellectual property embodied therein.

The content of this document is informational only. Any solutions presented herein are subject to changing conditions, information, scope, and data. This document was produced using reasonable efforts based on information available as of the date of issue of this document. The scope of information in this document may exceed that which Arm is required to provide, and such additional information is merely intended to further assist the recipient and does not represent Arm’s view of the scope of its obligations. You acknowledge and agree that you possess the necessary expertise in system security and functional safety and that you shall be solely responsible for compliance with all legal, regulatory, safety and security related requirements concerning your products, notwithstanding any information or support that may be provided by Arm herein. In addition, you are responsible for any applications which are used in conjunction with any Arm technology described in this document, and to minimize risks, adequate design and operating safeguards should be provided for by you.

Reference by Arm to any third party’s products or services within this document is not an express or implied approval or endorsement of the use thereof.

THE DOCUMENT IS PROVIDED “AS IS”. ARM PROVIDES NO REPRESENTATIONS AND NO WARRANTIES, EXPRESS, IMPLIED OR STATUTORY, INCLUDING, WITHOUT LIMITATION, THE IMPLIED WARRANTIES OF MERCHANTABILITY, SATISFACTORY QUALITY, NON-INFRINGEMENT OR FITNESS FOR A PARTICULAR PURPOSE WITH RESPECT TO THE DOCUMENT. Arm may make changes to the Document at any time and without notice. For the avoidance of doubt, Arm makes no representation with respect to, and has undertaken no analysis to identify or understand the scope and content of, third party patents, copyrights, trade secrets, or other rights.

NOTWITHSTANDING ANYTHING TO THE CONTRARY CONTAINED IN THIS LICENSE, TO THE FULLEST EXTENT PERMITTED BY LAW, IN NO EVENT WILL ARM BE LIABLE FOR ANY DAMAGES, IN CONTRACT, TORT OR OTHERWISE, IN CONNECTION WITH THE SUBJECT MATTER OF THIS LICENSE (INCLUDING WITHOUT LIMITATION) (I) LICENSEE’S USE OF THE DOCUMENT; AND (II) THE IMPLEMENTATION OF THE DOCUMENT IN ANY PRODUCT CREATED BY LICENSEE UNDER THIS LICENSE). THE EXISTENCE OF MORE THAN ONE CLAIM OR SUIT WILL NOT ENLARGE OR EXTEND THE LIMIT. LICENSEE RELEASES ARM FROM ALL OBLIGATIONS, LIABILITY, CLAIMS OR DEMANDS IN EXCESS OF THIS LIMITATION.

This License shall remain in force until terminated by Licensee or by Arm. Without prejudice to any of its other rights, if Licensee is in breach of any of the terms and conditions of this License then Arm may terminate this License immediately upon giving written notice to Licensee. Licensee may terminate this License at any time. Upon termination of this License by Licensee or by Arm, Licensee shall stop using the Document and destroy all copies of the Document in its possession. Upon termination of this License, all terms shall survive except for the License grants.

Any breach of this License by a Subsidiary shall entitle Arm to terminate this License as if you were the party in breach. Any termination of this License shall be effective in respect of all Subsidiaries. Any rights granted to any Subsidiary hereunder shall automatically terminate upon such Subsidiary ceasing to be a Subsidiary.

The Document consists solely of commercial items. Licensee shall be responsible for ensuring that any use, duplication or disclosure of the Document complies fully with any relevant export laws and regulations to assure that the Document or any portion thereof is not exported, directly or indirectly, in violation of such export laws.

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

The validity, construction and performance of this License shall be governed by English Law.

Arm Limited. Company 02557590 registered in England.

110 Fulbourn Road, Cambridge, England CB1 9NJ.

Arm document reference: PRE-21585

version 5.0, March 2024

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

Part C Appendix

Appendix 1: System flows

Appendix 2: TDISP VDMs

Part D Glossary

Glossary

Preface

This book describes a system architecture for an arm-based system that supports RME. It must be read in conjunction with the *Arm® Architecture Reference Manual for A-profile architecture* [1].

Issue C of this book introduces system requirements for RME Coherent Device Assignment (RME-CDA) and for supporting CXL devices.

It is assumed that the reader is familiar with both:

- PCI Express (PCIe).
- Compute Express Link (CXL).

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, for A-profile architecture*. (ARM DDI 0487) Arm Ltd.

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

[3] *Arm® System Memory Management Unit Architecture Specification*. (ARM IHI 0070) Arm Ltd.

[4] *PCI Express® Base Specification Revision 6.2*. PCI-SIG.

[5] *AMBA® 5 CHI Architecture Specification*. (ARM IHI 0050) Arm Ltd.

[6] *Compute Express Link (CXL) Revision 3.1*. CXL Consortium.

[7] *AMBA® AXI Protocol Specification*. (ARM IHI 0022) Arm Ltd.

[8] *AMBA® DTI Protocol Specification*. (ARM IHI 0088) Arm Ltd.

[9] *Arm® Base System Architecture Platform Design Document*. (ARM DEN 0094) Arm Ltd.

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

[11] *TDISP eXtended TEE (XT) Extensions*. PCI-SIG.

[12] *Arm® Reliability, Availability, and Serviceability (RAS) System Architecture, for A-profile architecture*. (ARM IHI 0100) Arm Ltd.

[13] *Arm® Memory System Resource Partitioning and Monitoring (MPAM) Memory System Component (MSC) Specification*. (ARM IHI 0099) Arm Ltd.

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

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

Feedback

Arm welcomes feedback on its documentation.

Feedback on this book

If you have any comments or suggestions for additions and improvements, please create a ticket at:

- https://support.developer.arm.com.

As part of the ticket, please include:

- The title (Arm® Realm Management Extension (RME) System Architecture).
- The number (ARM-DEN-0129 C.a).
- The section name 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.

Inclusive 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 Arm A-profile architecture [1] includes specification of the RME architecture for a *Processing Element* (PE), that defines the set of hardware features and properties required for a PE to comply with the Arm Confidential Compute Architecture (CCA).

The 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.

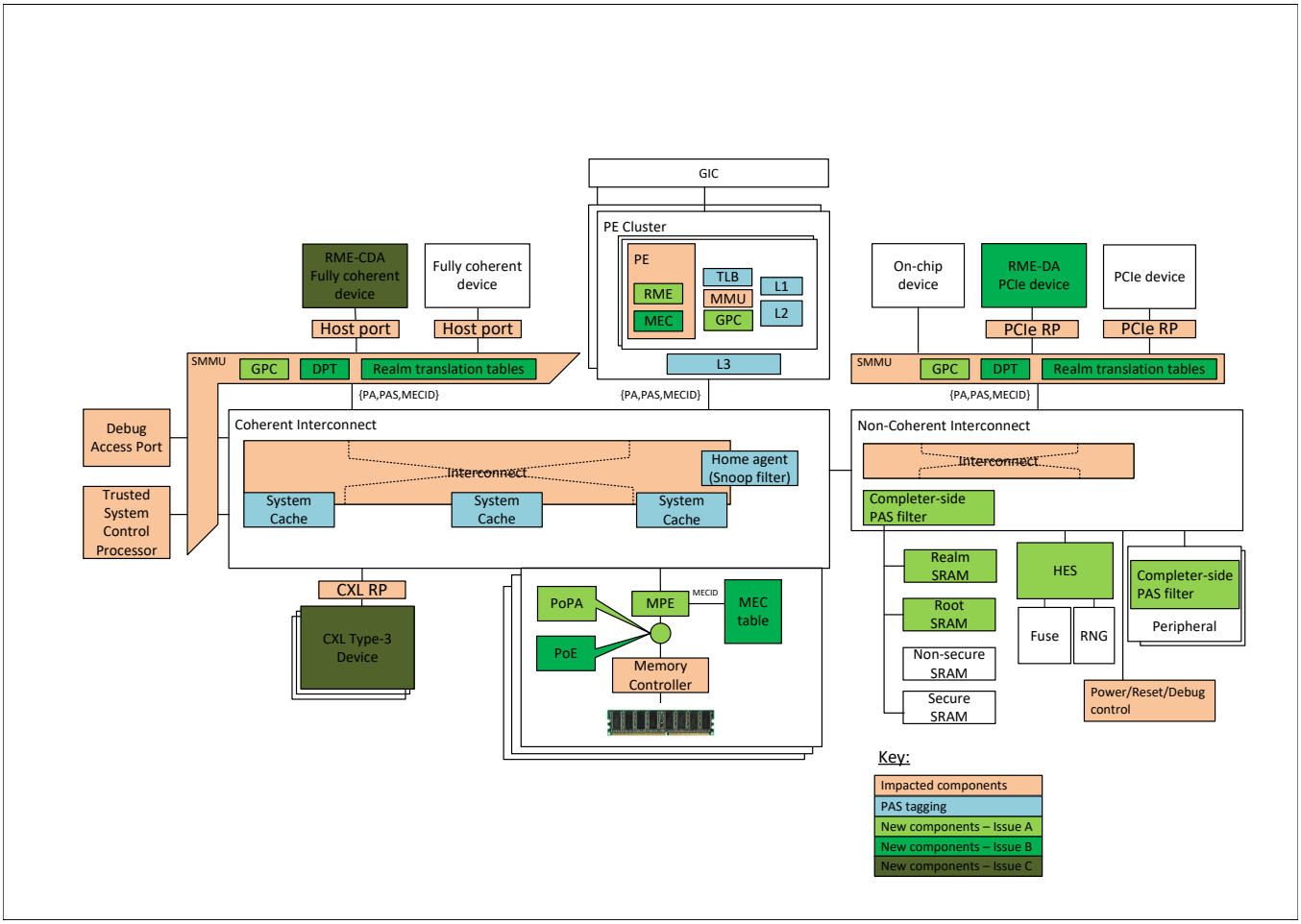

Figure A1.1: RME System Architecture

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.

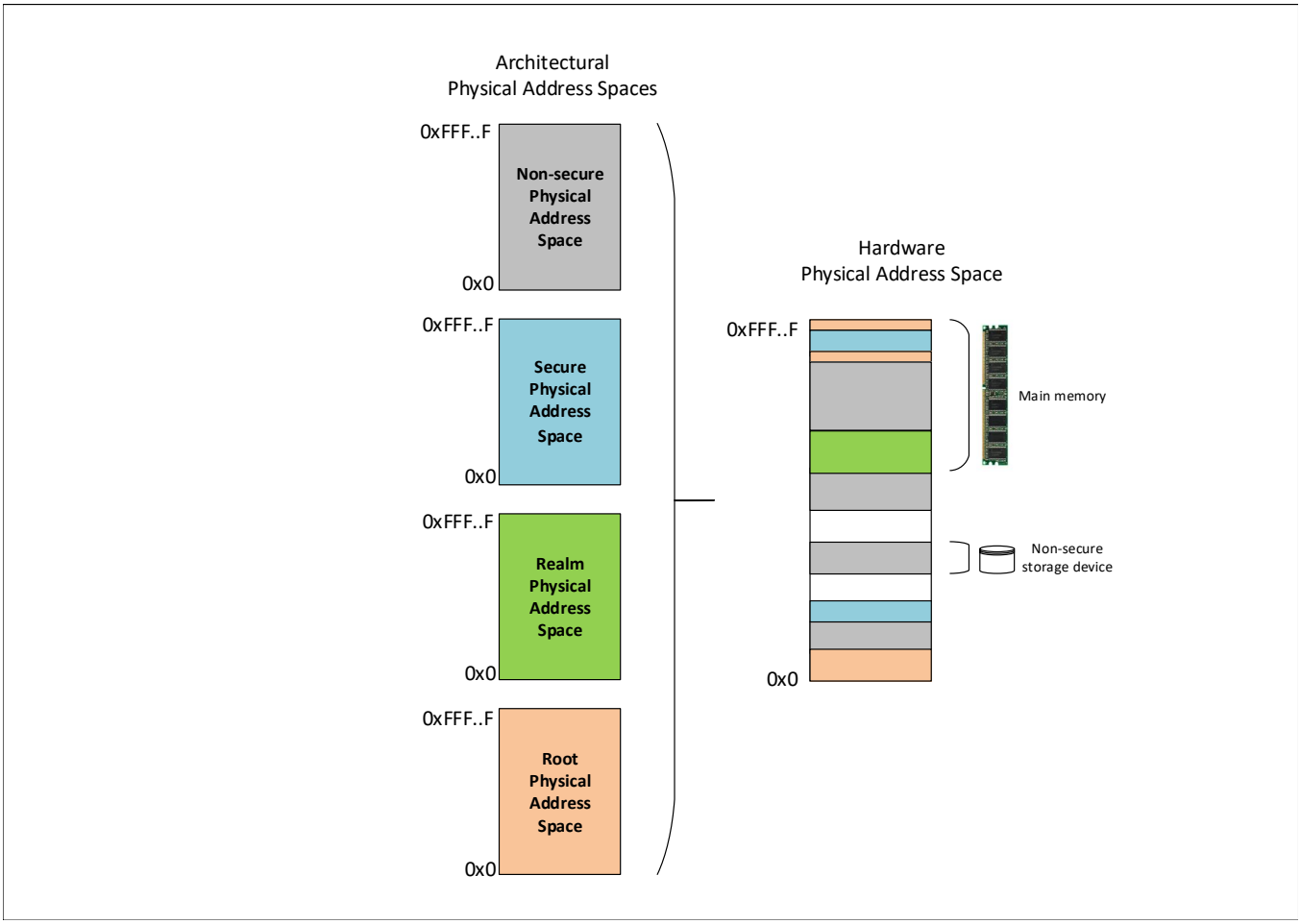

Figure B1.1: Physical Address Spaces

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 an 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 a region of memory that can be marked as cacheable retain their associated PAS until reaching the Point of Physical Aliasing (PoPA).

Accesses to a region of memory that cannot be marked as cacheable, for example memory-mapped peripheral registers, retain their associated PAS at least until reaching the PAS filter assigned to protect that region.

A component, for example a peripheral or an interconnect, is said to be *PAS tag-aware* if that component observes the full PAS tag such that it can distinguish between all physical address spaces defined by RME.

B1.2 Memory Encryption Context Identifier

I_{FDJJW} *Memory Encryption Contexts* (MEC) [1] is an optional RME system feature for encrypting all memory Locations of a Realm, using an encryption context unique to that Realm. In an RME system with MEC, each access to a physical address is assigned a *Memory Encryption Context Identifier* (MECID), which associates the access with a memory encryption context.

I_{JQGRL} *Common MECID width* is a parameter indicating the number of MECID bits that an RME system supports. It is resolved as the minimal MECID width that all MEC-capable components in the system can support.

I_{RPPVX} Each physical address space has an independent MECID namespace. In the Realm physical address space, the usable MECID namespace is from zero to 2^*common MECID width* - 1. The Root, Secure and Non-secure physical address spaces each have one MECID, which is the default MECID of zero.

I_{PXJSN} For systems with a high number of concurrently executing Realms, such as in Cloud and Data Center deployments, Arm recommends a common MECID width of at least 12 bits.

See also:

- B2.2 *Memory isolation and protection*

Chapter B2
System capabilities

This chapter specifies 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 the Arm A-profile architecture [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 the following:

- 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.
  - 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.
  - 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.
  - 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.
  - The Root Security state cannot be read or modified by an agent associated with any other Security state.
- An assignable device interface 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.
- An assignable device interface associated with a VMID in the Realm Security state can only access memory associated with that VMID, or memory associated with the Non-secure Security state.

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

I_{JTCVV} The RME security guarantee applies to a system in the Secured lifecycle state.

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 the Arm CCA Security Model [2]. Figure B2.1 provides an illustration of the security model.

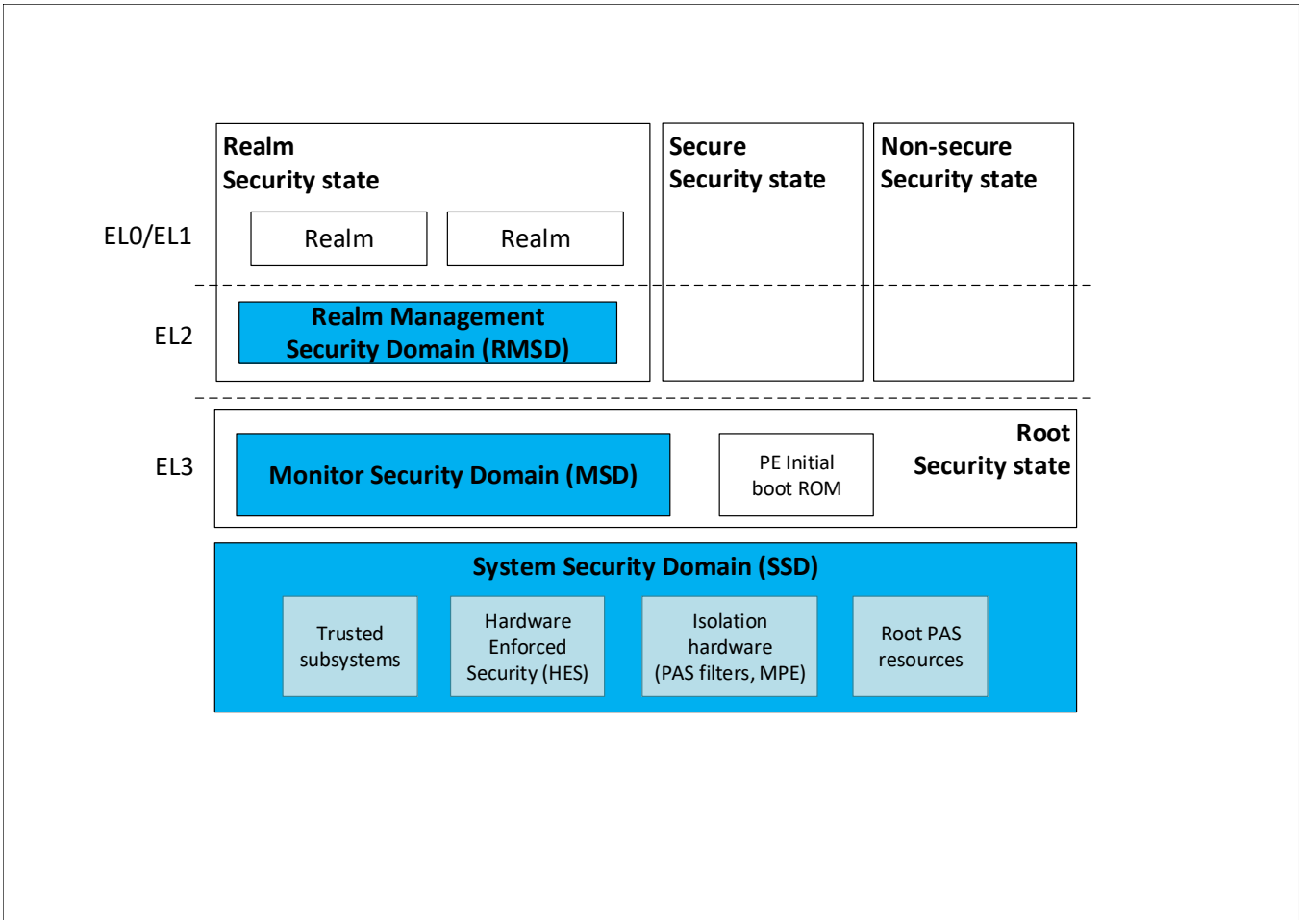

Figure B2.1: Arm CCA security model

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 Security states through these spaces.

I_{BHZXC} The physical address spaces that can be reached from each Security state are defined in the Arm A-profile architecture [1] and captured in Table B2.1:

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

I_{YLXPX} A *Resource* is an addressable physical entity that is formally defined in the Arm A-profile architecture [1]. A Resource in an RME system is accessible only when it is associated with a *Resource PAS*.

R_{QDWVC} Association of a Resource with a Resource PAS is controlled by either SSD or MSD.

I_{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.

R_{BJVZS} An access to a Resource is associated with:

- An *Access PAS*, in accordance with Table B2.1.
- A MECID, in accordance with the rules specified in the Arm A-profile architecture [1] and the SMMU architecture [3].

I_{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 and MECID in compliance with R_{BJVZS} 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.

I_{TFRNC} The Arm A-profile architecture [1] and the SMMU architecture [3] define the programming model for setting the Access PAS from each Security state.

R_{YXFMV} A requester that is accessing memory-mapped resources not through a stage 1 or stage 2 MMU/SMMU must support a method that is enforced by SSD hardware for tagging accesses with an Access PAS, in accordance with the PAS Access Table (Table B2.1).

For example:

- A Debug Access Port (DAP) can expose a programming register to an external debugger that allows setting an Access PAS to one of the permitted values, as implied by the debug authentication interface state, for any access that targets main memory or an APB peripheral.
- If the debug authentication interface permits RMSD external debugging but not Secure external debugging then DAP hardware would reject an attempt to program the register to Access PAS == Secure.
  - Furthermore, if the debug authentication interface permits RMSD external debugging then DAP hardware can permit accesses with Access PAS == Realm to specify a programmed MECID.

R_{SCDLL} Once assigned, the value of an Access PAS cannot be altered.

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

R_{YKVJK} A *PAS filter* enforces the PAS protection check by permitting access to a Resource only if the Access PAS matches a Resource PAS associated with that Resource.

R_{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, including PCIe devices or 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 the Arm A-profile architecture [1] and the SMMU architecture [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

B2.2.2.1 Point of Physical Aliasing (PoPA)

I_{KWFWG} The Point of Physical Aliasing (PoPA) is a reference location for cache maintenance operations that is defined in the Arm A-profile architecture [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} R_{FRMJJ} 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 the Arm A-profile architecture [1]. The scope of a cache maintenance operation to the PoPA (*PoPA CMO*) is the Outer Shareable shareability domain.

R_{QBNJF} A PoPA CMO affects any cached copy in the system with the specified {PAS, PA} regardless of all of the following:

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

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.

B2.2.2.2 Point of Encryption (PoE)

I_{VDHGS} The Point of Encryption (PoE) is a reference location for cache maintenance operations, as defined in the Arm A-profile architecture [1]. An RME system with MEC supports cache maintenance operations to the PoE in compliance with [1].

R_{TRBZM} An access to a cacheable memory Location is associated with a MECID until reaching the PoE.

I_{CRDLH} When a copy of a Location is allocated to cache, it is stored with the MECID of the allocating access.

R_{KMNQX} Memory accesses resulting from a cache Clean operation, due to cache maintenance operations and natural evictions, use the MECID that the entry was cached with.

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 the Arm CCA Security Model [2].

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

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

- A unique encryption context for each PAS.

- A unique 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.
- In a system with MEC, a different encryption context for each MECID in the Realm PAS.

External memory here includes any memory that is protected using the GPC and is not reset to a known value upon an RME system reset, for example DRAM, CXL Type-3 memory or CXL Type-2 memory.

R_{MYWVB} Data is encrypted before being written to external memory or to any shared cache that resides past the PoPA. In a system with MEC, data is encrypted before being written to external memory or to any shared cache that resides past the PoE.

I_{JMZJK} In a system with MEC, MECIDs identify encryption contexts such as encryption keys or tweaks, that might be stored in MECID-indexed tables or MECID-tagged caches.

R_{RQZBK} Memory-mapped data structures that store encryption contexts must reside in SMEM in the Root PAS, such as MSD SMEM.

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, in an RME Device Assignment (RME-DA) system the PAS tag of an access to a PCIe memory-mapped peripheral is only stripped at the Root Port after IDE stream association.

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)

The term *Non-PE requester* as defined in I_{CDDPM} is a generic term for describing DMA-capable system components, for example PCIe devices, interrupt controllers and control processors.

R_{RHBJN} The Security state of a non-PE requester in an RME system can be one of the following:

- Non-secure.
- Secure.
- Realm, if the system supports RME Device Assignment (RME-DA) and the requester is a *TEE Device Interface* (TDI) in the *RUN* TDISP [4] state.
- Root, if the requester is a Trusted subsystem.

I_{CPKZT} The Security state of a non-PE requester defines which PA spaces it is allowed to access, in accordance with the PAS access table (Table B2.1).

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 that is not part of the System Security Domain (SSD) will not observe coherency traffic for addresses in the Secure, Realm, or Root PAS.

I_{PPVTC} For example, a fully coherent external device that is attached to an RME system over a coherent link must not be able to track access patterns by observing snoop requests in the Secure, Realm or Root PAS. A cache of a coherent non-PE requester that is part of the System Security Domain is permitted to observe snoop requests in any PAS but will not make these requests visible to the hosted context.

B2.3.3 Programmable completer-side filters

I_{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.

R_{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.

R_{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.

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

B2.3.4 RME Device Assignment

I_{LFSPW} The term *Assignable Device Interface* refers to a portion of a device that can be independently assigned to software executing in one of the Security states. An assignable device interface can act as an independent requester and have its own private memory-mapped resources.

RME Device Assignment (RME-DA) is an RME system feature that enables the secure assignment of assignable device interfaces to the Realm Security state.

I_{PTDLG} There is the following related terminology:

PCIe refers to devices that comply with the TEE Device Interface Security Protocol (TDISP [4]) as *TEE-I/O capable devices*. This specification refers to them as *TDISP-compliant devices*.

PCIe uses *TEE Device Interface* (TDI) to refer to an assignable device interface of TDISP-compliant devices.

TDISP defines both:

- A *TEE Security Manager* (TSM) which is a logical entity at the host that enforces security policies.
- A *Device Security Manager* (DSM) which is a logical entity in the device that enforces security policies on the device.

R_{WBJJT} TSM functionality in RME-DA is implemented within RMSD.

I_{XQHNF} SMMU for RME-DA [3] defines SMMU requirements for supporting the assignment of TDIs to software executing in the Realm Security state.

The RME system architecture defines memory system and PCIe Root Port requirements for supporting the assignment of PCIe TDIs to the Realm Security state.

See also:

- B3.2.6 *PCIe Root Port requirements for RME-DA*

B2.3.4.1 Device Permission Table (DPT)

I_{NMXPZ} SMMU for RME-DA [3] introduces a Device Permission Table (DPT) that contains permission attributes associated with physical addresses and specifies a corresponding set of DPT checks that apply to translated accesses from devices.

R_{QRMPD} A translated access from a TDI that is assigned to Realm state is subject to DPT checks, unless where stated otherwise. This includes all host-routed Peer-to-Peer (P2P) accesses with the exception of traffic associated with a Direct P2P IDE Stream, if the host supports Direct P2P IDE Stream routing.

R_{PGSTQ} An RME system can include integrated TDISP-compliant devices, for example an on-chip RCiEP, that are measured and attested by HES or MSD. For such a device:

- DPT checks can be skipped.
- GPC cannot be skipped.

B2.3.5 RME Coherent Device Assignment

I_{GWFRT} A *coherent device* is a non-PE agent that can participate in the coherency protocol of the host system as a fully-coherent node. For example, this can be a CXL Type-1 or Type-2 device, or a device that implements a CHI-based interface.

This section introduces *RME Coherent Device Assignment* (RME-CDA), an RME system feature that enables the assignment of coherent devices to the Realm Security state. Initial support includes CHI-based devices only.

I_{WCVYY} A host node in an RME system is a discrete component that constitutes part of the System Security Domain (SSD). For example, it can be a compute node, which is a node that contains application PEs. A coherent device could be implemented as an on-chip element forming an integral part of a host node or as a discrete element that is directly connected to a host node over a Chip-to-Chip (CTC) coherent link. A discrete coherent device may be concurrently connected to the same host node over multiple coherent and non-coherent links or channels, where a channel is a logical partition of a physical link.

A *coherent host port* is the entry and exit point on a host node for coherent link traffic. For example, this can be a CXL Root Port or a CHI-based interface port. For on-chip coherent devices the corresponding host port is IMPLEMENTATION DEFINED but must comply with the rules in this chapter.

I_{FKQNC} In an RME system, multiple host nodes form a single SSD of a host system. The SSD can also include coherent device nodes where such a device is accepted into SSD as part of platform attestation, in accordance with the Arm CCA Security Model [2].

The rules specified in this section apply to coherent devices that are not part of SSD and enable RME-CDA, a system capability which allows RME-CDA coherent devices to be assigned to Realms while maintaining the RME security guarantee.

An RME-CDA coherent device is only trusted by the set of Realms it is assigned to. This means that the RME-DA trust model is extended to include the following properties:

- If a Realm trusts a coherent device, the Realm trusts that the device does not violate the system coherency rules.
- If a Realm does not trust a coherent device, the Realm is not required to trust it to respect the system coherency rules.

RME-CDA defines the appropriate protections that a system must implement to prevent a violation of the RME security guarantee.

R_{SHRMN} The following constraints apply to coherent links, coherent devices, and their relations in an RME-DA system:

- For each coherent link, at most a single coherent device is permitted to be attached to a host node over that link. A single coherent device can be connected to a host node over one or more links.
- Device attestation applies to the connected device and to any logic within it or attached to it that might affect the RME security guarantee. This includes caching agents and coherency tracking structures within the device as well as any components attached to the device through an external interface.
- A coherent device is permitted to include multiple requesters, for example fully coherent Request Nodes (CHI RN-Fs), and must enforce the RME security guarantee between these.
- A coherent device is permitted to include multiple home agents, for example fully coherent Home Nodes (CHI HN-Fs).

I_{SJQLP} The term *host coherent memory* (HCM) refers to any memory that is directly attached to a host node and that can be coherently accessed by the host and by coherent devices.

The term *device coherent memory* (DCM) refers to any memory that is directly attached to a coherent device and that can be coherently accessed by the host, the device, and other coherent devices.

RME-CDA allows one coherent device to access DCM of another coherent device only where that access is channelled through a host node.

I_{CZGJL} Coherent devices are attested and assigned to Realms in a similar manner to PCIe devices. This section uses terms from TDISP [4], for example *TDI* and *DSM*, for describing general security properties of coherent devices. A successful attestation of an RME-CDA coherent device implies device compliance with all rules in this section.

B2.3.5.1 Chip-to-Chip link requirements

I_{HZMML} A *device-to-host* access is an access over a CTC coherent link that is initiated by the device. A *host-to-device* access is an access over a CTC coherent link that is initiated by the host or channeled through the host, and is sent to the device.

The possible memory targets of an access are defined by its type and direction. For example:

- A *device-to-host* memory request can target HCM or DCM of another coherent device.
- A *device-to-host* snoop request targets Locations in DCM for that device.
- A *host-to-device* memory request targets DCM for that device.
- A *host-to-device* snoop request can target Locations in HCM, DCM of the device or DCM of another coherent device.

I_{PQGTC} A CHI-based CTC coherent link between a host node and a coherent device supports all of the following:

- For any device-to-host access sent on the CHI REQ channel that specifies an address:
  - A 16-bit Requester ID (RID) associating an access with a TDI requester. The format of the Requester ID complies with PCIe Routing ID rules and can be qualified by a Segment Number at the host port to form an SMMU StreamID.
  - A SEC_SID identifier associating an access with a Security state, which can be Non-secure or Realm.
- For any device-to-host access sent on the CHI SNP channel that specifies an address, a PAS identifier associating an access with a PA space, which can be Non-secure or Realm.
- For any host-to-device access sent on the CHI REQ or SNP channels that specifies an address, a PAS identifier associating an access with a PA space, which can be Non-secure or Realm.

Encodings of new identifiers are specified in the AMBA CHI architecture [5].

B2.3.5.2 General coherent device requirements

R_{NVWMC} An RME-CDA coherent device complies with all of the following:

- Supports a programming interface for device measurement and device interface (TDI) assignment in compliance with TDISP [4] or CXL [6].
- Conforms to the security requirements specified in TDISP [4] unless stated otherwise in this section.
- Supports memory encryption in compliance with this document.
- If the device supports MEC, complies with MEC rules in the Arm A-profile architecture [1] and in this document.
- Conforms to the applicable coherency protocol, for example CHI or CXL.
- Conforms to all rules in this section.

I_{CYPWV} When a TDI of a coherent device is in the *RUN* TDISP state, it can access the Realm PAS and can cache Locations in the Realm PAS.

R_{QWNBN} When a TDI of a coherent device is in the *ERROR* TDISP state, then for any cached Location in the Realm PAS that is associated with the TDI all of the following apply:

- The coherent device will poison write-backs of that Location from the cache.
- The coherent device will respond with a poison to snoop requests to that Location.

I_{WBCQS} Poisoning cached data is required as its integrity may have been compromised once the TDI is in *ERROR* state. In compliance with R_{WWBPM}, a TDI in *ERROR* state is no longer permitted to access cached data of Locations in the Realm PAS or allocate it to the cache.

R_{LMFLX} When a TDI transitions to the *CONFIG_UNLOCKED* TDISP state, the coherent device will invalidate any content associated with the TDI that is stored in the device, including copies of Locations that were cached by the TDI and ATC entries.

Dirty data stored in the cache must be written-back and invalidated. If a write-back is not possible, for example because the IDE Stream is in Insecure state, then the cache line is invalidated without performing a write-back.

R_{VPKVF} The following operations must not be affected by the TDISP state of a TDI:

- Host-to-device requests on the CHI SNP channel. If the TDI is in the *ERROR* state snoop responses are issued in compliance with R_{QWNBN}.
- Device-to-host requests on the CHI SNP channel. For example, the device must be able to issue snoops to DCM Locations associated with the TDI even if the TDI is in the *ERROR* state.
- Host-to-device requests on the CHI REQ channel. For example, the device must be able to service CMOs to DCM Locations associated with the TDI.

I_{LBGKN} Note that R_{VPKVF} does not apply to a specific TDI but to global device functionality. For example, after a TDI transitions to the *ERROR* TDISP state, all of the following apply:

- The device should be able to continue sending device-to-host snoop requests specifying any DCM Location in the Realm PAS.
- The device must complete host-to-device CleanInvalid or MakeInvalid requests to DCM Locations that were previously cached by that TDI and mark with poison any associated snoop responses for Locations in the Realm PAS.
- The device must complete host-to-device SnpCleanInvalid or SnpMakeInvalid requests to HCM Locations that were previously cached by that TDI and mark with poison snoop responses for Locations in the Realm PAS.
- R_{VPKVF} implies that the device is able to continue servicing host-to-device memory requests to any DCM Location, including Locations that were assigned to the TDI.
- The TDI is not allowed to perform memory accesses to HCM or DCM Locations in the Realm PAS, in compliance with R_{WWBPM}.

I_{FNSDT} In compliance with this document, a cache in a coherent device must maintain PA space separation by associating a PA with a PAS and including the PAS in any PA compare operation. Devices obtain the PAS information from the TE bit in ATS Translation completions, as specified in the *TDISP eXtended TEE (XT) Extensions* ECN.

I_{DPSLW} The rules that govern device cache behavior in the different TDISP states do not apply to native CXL Type-1 and Type-2 devices, and the permitted behavior for these devices may be affected by further definition in CXL [6].

I_{RZMHP} RMSD expects that a DCM range associated with a TDI is reported in the TDISP DEVICE_INTERFACE_REPORT using an MMIO_RANGE with a range ID value of `0xFFFF`.

B2.3.5.2.1 Device configuration and reset

R_{WFYLW} Any configuration register in an RME-CDA coherent device that affects the RME security guarantee must comply with one of the following:

- The register is exclusively controlled by the device DSM.
- The register is locked and verified by the device DSM before responding to a TDISP LOCK_INTERFACE_REQUEST, and DSM fails the request if a verification error is detected.

This includes registers that control address decoding, offsets and masks, memory controllers, cache state invalidation and any other functionality that may affect the locations, contents and access control of DCM.

The verification process includes consistency checks of address decoders to prevent aliasing, ensuring that DCM is only accessible through host-visible address ranges, that any memory integrity features are enabled, and that any memory-related error injection or debug capabilities are disabled.

R_{JZQCP} An RME Coherent device must perform protection operations upon any reset or power state transition of the device that has one or more of the following consequences:

- Might affect the confidentiality or integrity of any Location in the Realm PAS, for example memory contents at rest or in transit.
- Resets cache state or invalidates cache lines without a write-back.
- Resets coherency structures state.
- Resets a protected register or any context that was configured by DSM, for example PAS filter or encryption context.

The protection operations are the following:

- The IDE Stream transitions to Insecure state.
- All TDIs transition to the TDISP *ERROR* state and are then permitted to transition to *CONFIG_UNLOCKED* state, in compliance with TDISP [4].
- Memory encryption keys are reset to a known default value. Subsequently, device completer-side PAS filters are permitted to transition all DCM Locations from the Realm PAS to the Non-secure PAS.
- The device is permitted to clean and invalidate device caches upon the event. If clean and invalidate of device caches does not occur upon the event, it still occurs when the associated TDIs transition to *CONFIG_UNLOCKED* state, in compliance with R_{LMFLX}. Write-backs that occur during cache invalidation are not guaranteed to complete successfully, as the device-side PAS check might fail these.

R_{HDXTM} When a TDI transitions to the *CONFIG_UNLOCKED* TDISP state, the coherent device must scrub all contents of DCM Locations associated with the TDI. For example, this can be accomplished by resetting memory encryption contexts associated with the TDI to a known default value after invalidating cached content associated with the TDI, or by explicitly setting memory contents to a known value in a coherent manner.

I_{KGWBJ} DCM Locations associated with the TDI include any DCM ranges that have been reported in the TDISP DEVICE_INTERFACE_REPORT request for the TDI. Memory scrubbing operations and invalidation of DCM Locations cached at the device must succeed, even if device-to-host snoops cannot be issued, for example because the IDE Stream is in Insecure state. RMSD is responsible for ensuring that DCM Locations cached at the host have been invalidated.

B2.3.5.3 Host-to-device access

A rule in this subsection can either apply to a coherent device or to the host.

I_{QVMRW} Assignment of DCM is restricted as follows:

- DCM is only permitted to be assigned to the following:
  - Realms that have established trust with the device using TDISP.
  - Non-secure PAS.
- DCM is not permitted to be assigned to the following:
  - Root PAS.
  - Secure PAS.
  - RMSD. RMSD is permitted to access DCM but will not store private code or data in DCM.
  - Realms that do not trust the device.

I_{CWKGR} Arm recommends that DCM supports any memory feature that is supported by HCM. For example:

- If MEC is supported for HCM, it is also supported for DCM.
- If RAS is supported for HCM, it is also supported for DCM.

I_{LSTDW} A memory feature that is not supported by DCM might lead to the feature being globally disabled for any client using the device.

R_{KRCWK} An access from a PE to any memory location, including to DCM, is subject to host-side GPC.

I_{SNHGK} DCM can be accessed by the following access types:

- Internal - an access by an agent that is part of the attested device, for example a device TDI.

- External - an access by a non-internal agent, for example a host PE or a peer device initiating an access that is channeled through the host.

I_{GVDRS} A coherent device can have the capability of performing the following:

- Autonomously resolving the Access PAS of internal accesses to DCM.
- Autonomously performing the PAS protection check for internal and external access to DCM using a programmable completer-side PAS filter.

Support for this capability is advertised by the device through the *PAS_CHECK* attestable property.

A coherent device that reports PAS_CHECK == TRUE is permitted to access its DCM directly without doing the following:

- Using ATS for host-side stage 2 translation.
- Going through host-side GPC.

R_{QRMSC} If PAS_CHECK == FALSE, all of the following are true:

- Internal access to DCM is subject to host-side GPC.
- The device is not permitted to access DCM without first using ATS for host-side address translation and GPC.
- The device is permitted to access DCM directly using ATS-translated addresses, in compliance with TDISP [4] rules for translated requests.

If PAS_CHECK == TRUE, all of the following are true:

- Internal access to DCM is subject to the device-side PAS protection check.
- The device is permitted to use an IMPLEMENTATION DEFINED mechanism for assigning PA ranges of DCM to TDIs, and for specifying the Access PAS of each such PA range.
- The device is permitted to access DCM directly without using ATS, in compliance with TDISP [4] rules for translated requests.

I_{XBTCM} CHI-based devices report their support for the PAS_CHECK property using an Arm-specified TDISP VDM request.

I_{GLQWV} The GPC and the device-side PAS check are consistently programmed to produce the same access control result for any translation regime.

RMSD guarantees that the GPT information correctly reflects the PAS association of DCM ranges as reported by the device, for example using TDISP. The device-side PAS check is programmed by the Device Security Manager (DSM).

R_{WWBPM} An RME-CDA coherent device does not permit an internal access to DCM to specify Access PAS == Realm if the access is not from a TDI in the *RUN* TDISP state.

R_{FZBMW} An RME-CDA coherent device does not permit an external access to DCM to specify Access PAS == Realm if the access did not arrive over an IDE Stream provisioned through TDISP.

R_{KNZGC} If a device-side PAS protection check applies to read-sensitive locations, the device must not permit speculative execution of accesses to any location that might be read-sensitive, until the check is complete.

R_{VHZCR} If PAS_CHECK == TRUE, device-side PAS protection checks apply both to internal access and external access to DCM.

R_{STQHS} The host is permitted to forward a host-to-device snoop request to a coherent device if and only if the snoop request specifies at least one of the following:

- A Location that is known to be resident in the device cache. For example, this condition is met when all home agents in the system, either in host nodes or in coherent devices, precisely track cached Locations.
- A Location that the device is permitted to access. For example, this condition is met if the host has a method to associate Physical Addresses with a set of devices that have access to these Locations.
- A MECID that the device is permitted to use. For example, this condition is met if the host has a method to associate MECIDs with a set of devices that have access to these.
- An address in the Non-secure PA space.

I_{KMFFT} It is IMPLEMENTATION DEFINED how an RME system complies with R_{STQHS}. As this version of RME does not define host-side mechanisms for associating PAs or MECIDs with device identifiers, home agents in coherent devices must track cached Locations in a way that ensures that device snoops are not forwarded to unknown targets.

R_{HTTNB} If a host-to-device snoop request cannot be forwarded to a coherent device, for example because the IDE Stream is in Insecure state, and the host port is tracking the Location as potentially owned by the device in a Dirty state, the host port must respond with poison and fixed data. If the host port cannot associate the response with a correct MECID, it must use the default MECID of zero.

B2.3.5.4 Device-to-host access

A rule in this subsection can either apply to a coherent device or to the host.

I_{PKYGW} For any coherent device, including RME-CDA coherent devices and Non-secure coherent devices, the following apply:

- Device-to-host accesses are subject to host-side GPC, as specified in this document and the SMMU architecture [3].
- GPC for device-to-host snoop requests are permitted to be skipped, as specified in this section.

For RME-CDA coherent devices, the following additionally apply:

- Device-to-host accesses carrying SEC_SID and RID are subject to DPT checks, as specified in this document and the SMMU architecture [3].
- DPT checks for device-to-host snoop requests are permitted to be skipped, as specified in this section.

R_{YDSYL} An RME-CDA coherent device guarantees that on any device-to-host memory request from a TDI or a cache associated with a TDI, the CHI SEC_SID and CHI RID fields are set as follows:

- If the TDI is in the *RUN* TDISP state, CHI SEC_SID == Realm. Otherwise, CHI SEC_SID == Non-secure.
- The CHI RID field is set to the Requester ID of the TDI. For a TDI in the *RUN* TDISP state this value is equal to the Requester ID field in the TDISP-specified INTERFACE_ID of the TDI.
- A device-to-host access from a cache specifies the SEC_SID and RID fields of the allocating access.

R_{SKJNK} An RME-CDA coherent device guarantees that any PA specified on a device-to-host memory request over a coherent or a non-coherent channel is obtained by the device from the host SMMU using ATS or an ATS-equivalent protocol.

R_{YJVJR} An SMMU in an RME-CDA system supports populating the *TE Memory Attribute* bit (TE bit) in ATS Translation completions with the resolved PA space.

R_{FRFJG} An RME-CDA coherent device guarantees that a TDI can generate device-to-host snoop requests only to PAs that it is allowed to directly access.

R_{NSMGT} A host port must be able to identify that a PA of a device-to-host access falls within the DCM range of the coherent device attached to that port.

R_{SHSXK} For any device-to-host request that specifies a PA, and that was sent on a CHI REQ channel or a non-coherent channel, the host enforces the following checks, in compliance with the SMMU architecture [3]:

- GPC.
- If the access has SEC_SID == Realm, a DPT check.

The host is permitted to skip the GPC and DPT checks for these requests if the access PA falls within the DCM range of the device.

I_{LHPSV} The host can optionally enforce a DPT check on accesses that have SEC_SID == Non-secure.

R_{DMSPT} The Requester ID (RID) field of any device-to-host request that is sent on a CHI REQ channel or a non-coherent channel, is checked by the host port against a permitted range of Requester ID values for the device.

I_{RLRMC} The host port sets the SMMU StreamID and SubstreamID fields using the RID and PASID fields of the request in accordance with Arm BSA [9] and SBSA [10] specifications. These fields are further qualified by the SEC_SID field specified on the access.

R_{JSYPS} For any device-to-host request sent on a CHI SNP channel that specifies a PA, the host port enforces the following:

- If the access PA falls within the DCM range, then the request is subject to an Access PAS check and the host port will permit PAS == Realm in compliance with R_{GTVGZ}.
- If the access PA does not fall within the DCM range, the request is *rejected with error* by the host port.
- In all cases, GPC and DPT checks are skipped for the request.

R_{HVCNR} If a device-to-host snoop request cannot be forwarded because the IDE Stream is in Insecure state, and the device is tracking the Location as potentially owned by an external caching agent in a Dirty state, the device behavior shall be consistent with receiving a host-to-device snoop response with poison.

R_{ZTQYT} An RME-CDA coherent device is not permitted to send CHI requests with `DVMOp` or `SnpDVMOp` opcodes.

See also:

- B3.2.9 *Coherent host port requirements for RME-CDA*

B2.3.5.5 MEC requirements for Coherent Device Assignment

I_{RNGWY} When MEC is enabled, the MECID field must be populated on any CHI message that includes that field, in compliance with the AMBA CHI architecture [5], with the following exception:

- The MECID field on device-to-host requests that are sent on the CHI REQ channel is not required to be present or populated as the SMMU resolves the MECID based on the RID.

I_{TXBRS} A single MECID value is associated with a Realm and is used when accessing granules of that Realm both in HCM and DCM.

I_{ZMVQP} All caches, including coherent device caches, must ignore the MECID when processing a Cache Maintenance Operation (CMO) so that coherency management is not affected by MECID mismatches. This is also specified by the AMBA CHI architecture [5].

R_{DRVTK} The host must guarantee that all device-to-host requests and device-to-host responses are associated with a MECID that the device is permitted to use.

I_{QNLLK} For device-to-host memory requests this rule is enforced by the host SMMU resolving the MECID association directly.

For device-to-host snoop responses the device is permitted to specify a MECID on the response in compliance with the AMBA CHI architecture [5]. Spoofing the MECID of a snoop response may enable a certain type of attack, where a malicious device could use the snoop response as a method to encrypt arbitrary data with a victim's MEC. An attacker that gains physical access to memory can then extract the ciphertext and apply pattern matching methods against the victim's ciphertext.

As a mitigation, the host must be able to verify that the device is permitted to use the MECID specified on a snoop response. For example, this can be achieved using a table at the host that associates a set of MECIDs with a device identifier.

It is IMPLEMENTATION DEFINED how an RME system complies with this requirement for snoop responses as this version of RME does not specify host-side mechanisms for validating a MECID supplied by the device.

I_{KVQFQ} A method for programming a MECID to a TDI of a coherent device is needed so that the device can include the MECID in snoop responses from the device cache. For CHI-based devices, this programming will be supported using an Arm-specified TDISP VDM request.

See also:

- B3.2.9 *Coherent host port requirements for RME-CDA*

Chapter B3
Resources and Components

This chapter outlines properties of system resources and components that are part of an RME system, including systems that support RME-DA or MEC.

R_{JSDVG} All structures and fields defined by this specification 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 the Arm CCA Security Model [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. Where R_{ZCJHY} applies to the GPT, then this is only for locations that are not marked in the GPT as “All Accesses Permitted”.

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. The Arm A-profile architecture [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:

- Non-secure or Secure accesses by another agent through the memory-mapped interface, do not return information about Realm Security state execution or affect it.
- Similarly, Non-secure or Realm accesses by another agent through a memory-mapped interface, do not return 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 any other capability dependent on RME, 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 SMMU for RME [3] and, if the system supports RME-DA or MEC, with SMMU for RME-DA.

R_{PXDQJ} In a system that supports RME-DA, any access from a TDISP-compliant device is subject to SMMU translation.

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} Coherency tracking logic within any interconnect in an RME system must be PAS tag-aware of all PA spaces specified by RME, regardless of the type of completer nodes attached to it.

Forwarding logic within an interconnect is permitted 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 the Arm A-profile architecture [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_{MNGJT} In an RME system with MEC, R_{LCXDB} also applies to any cached or transient state associated with the PA before the PoE.

I_{CDSDR} Any system cache before the PoPA must comply with the cache lockdown rules that are specified in the Arm A-profile architecture [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 Arm A-profile architecture [1] forbids writing clean data outside of a shareability domain. See *General behavior of the caches* in chapter *AArch64 System Level Memory Model* in [1].

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. The Arm A-profile architecture [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.

B3.2.3.2 Interconnect support for RME-DA

This section specifies interconnect requirements for supporting RME-DA.

R_{DNFTD} A PA of an access to a PCIe Root Port is associated with a PAS until reaching the Root Port.

R_{TTPLM} Interconnect registers that control mapping of PAs to PCIe Root Ports are implemented as MSD-Protected registers (MPRs).

I_{GLSZD} The list of all interconnect registers that must be implemented as MPRs is IMPLEMENTATION DEFINED. The following are some examples related to RME-DA:

- Registers that allow modifying the address or data fields of a memory access targeting an RME-DA Root Port.
- Registers that can affect the routing of a memory access that is either targeting an RME-DA Root Port or emitted from it.

R_{XZTPC} An RME-DA system guarantees that all of the following parameters are either defined statically or exclusively controlled by MSD firmware or a Trusted subsystem, for example by implementing any related registers as MPRs or as RMSD write-protect registers:

- The Bus Number, Device number, and corresponding ECAM address range assigned by a host bridge for each RP and RCiEP in the system.

- The memory ranges assigned by a host bridge or system interconnect for each RP and RCiEP in the system.

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 or any other confidential memory encryption context that is used by an 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 context, for example a key or a tweak, for each PAS.

I_{RKSDY} A MEC-capable MPE supports a different encryption context, for example a key or a tweak, for each MECID in the Realm 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 or MECID, 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_{CNHPC} In an RME system with MEC, MPEs are permitted to cache information from MECID-indexed tables in local cache structures. The mechanism for invalidating such caches when an encryption context is revoked or updated is IMPLEMENTATION DEFINED.

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) that is 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 the Arm CCA Security Model [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 that 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.

B3.2.6 PCIe Root Port requirements for RME-DA

This section defines requirements for an RME-DA Root Port (RP), in order to securely associate TDIs of TDISP-compliant devices with EL1 Realms, in compliance with PCIe TDISP [4].

I_{WLFYM} This specification uses the term *outgoing* for traffic that enters the RP from its host interface and targets its PCIe hierarchy domain.

I_{ZDYJB} This specification uses the term *incoming* for traffic that enters the RP from its PCIe hierarchy domain.

R_{LGXBX} An RME-DA RP sets the *TEE-IO Supported bit* in the Device Capabilities Register.

B3.2.6.1 Integrity and Data Encryption (IDE) support

I_{DKVYQ} This section defines requirements for PCIe IDE [4] support in an RME-DA RP.

R_{GRCKL} An RME-DA RP supports all the following IDE features:

- At least one Selective IDE Stream.
  - The number of supported Selective IDE Streams scales with:
    - The number of TDISP-compliant devices that each RP might support.
    - The number of PCIe Traffic Classes (TC) supported by the system.
    - The number of RID ranges that can be assigned to each TDISP-compliant device.
  - *NUM_SEL_STR* denotes the number of Selective IDE Streams supported by the Root Port.
- At least two Address Association register blocks for each Selective IDE Stream.

I_{HDPRK} RME-DA requires Selective IDE Stream support for setting up IDE with TDISP-compliant devices, that can be located behind a PCIe switch or directly attached to the RME-DA RP. This enables Requester ID (RID) checks to be performed at the RP for all TDISP-compliant devices.

Having two Address Association register blocks per Selective IDE Stream allows programming separate 32-bit and 64-bit ranges.

Arm recommends that an RME-DA RP supports the *TEE-Limited Stream* IDE capability.

I_{QQFBP} An IDE stream is identified by an IDE Stream ID, and can be in *IDE Insecure state* or *IDE Secure state*.

B3.2.6.1.1 Key programming

R_{BDLXG} An RME-DA RP exposes an IMPLEMENTATION DEFINED IDE key programming interface for the following IDE Key Management (IDE_KM) data objects:

- KEY_PROG.
- K_SET_GO.
- K_SET_STOP.

R_{VCRRM} An RME-DA RP must support IDE key refresh operations in compliance with [4].

I_{XQTTY} IDE_KM data objects have a *Key Set* field for configuring two different key sets per IDE Stream. Two different key sets are required for key refresh operations. The IDE key programming interface is allowed to have a method for indicating that it is temporarily not ready to receive a new data object, for example a *Busy* flag. When the Busy flag is cleared, the interface must be able to receive an IDE_KM data object of any type and for any Key Set value.

I_{TFMGB} The IDE key programming register format is IMPLEMENTATION DEFINED. This means that RMSD access to the key programming interface is mediated through MSD firmware and implemented by IMPLEMENTATION DEFINED code that can execute either in MSD firmware or in a Trusted subsystem.

B3.2.6.1.2 Key refresh schedule

I_{TLZJN} TDISP defines that the TSM is responsible for scheduling IDE key refreshes.

In an RME system, this task can be facilitated by having the RME-DA RP detect and report when an IDE key requires a refresh. For example, a counter with a pre-programmed threshold might be used. When the counter reaches the threshold, the RP uses an interrupt to report to a Trusted subsystem that the key associated with that counter requires a refresh. The RME-DA RP is also permitted to detect that a key set has expired and therefore must no longer be used. On detecting a key expiry, the RP autonomously transitions the corresponding IDE Stream to Insecure state.

R_{FSFST} The RP IDE logic must be able to detect that an IDE key set requires a refresh and perform one or more of the following:

- Assert a dedicated interrupt that shall be delivered to a Trusted subsystem.
- Transition the corresponding IDE Stream to Insecure state.

I_{DCBJS} Arm recommends that the RP tracking logic is capable of identifying at least the following key refresh event and expose the corresponding counter value:

- Initialization Vector (IV) counter crossed a threshold.

I_{SMMDW} Because RME does not support trusted delivery of interrupts to RMSD or MSD, key refresh must be scheduled using an interrupt handled by a Trusted subsystem. This could be either a direct interrupt from the RP or a timer-based interrupt managed by the Trusted subsystem.

B3.2.6.1.3 Selective IDE Stream programming

R_{BWFTS} RMSD ensures that Selective IDE Streams are configured such that different streams are assigned RID ranges and address ranges that do not overlap.

B3.2.6.2 Root Port registers

I_{STTSV} A register in this section refers to any of:

- An RP PCIe configuration space register accessed through the PCIe Enhanced Configuration Access Mechanism (ECAM).
- An RP PCIe memory-mapped register, for example in the Root Complex Register Block (RCRB).
- An RP IMPLEMENTATION DEFINED register.

I_{FLPRX} *RMSD write-protect* is a register security property which means that the register can only be written by accesses in the Realm PAS or the Root PAS. Reads are permitted in any PAS.

RMSD full-protect is a register security property which means that the register can only be read or written by accesses in the Realm PAS or the Root PAS.

RMSD write-detect is a register security property which means that any write to the register in the Non-secure or Secure PAS transitions all hosted IDE Streams to IDE Insecure state and clears all IDE Selective Stream Lock bits. Accesses in the Realm PAS or the Root PAS to an RMSD write-detect register are always permitted.

I_{LGWNF} RMSD write-detect, write-protect, and full-protect require the RP to implement a PAS filter on any register that can have one of these properties. Where write-protect applies to PCIe registers, if the PAS protection check fails for a write, this must not result in a fatal error. This means that the write is ignored and a PCIe System Error is not reported.

I_{SCDMH} An RME-DA RP implements the RME-DA Designated Vendor-Specific Extended Capability (DVSEC).

R_{DVJRV} The RME-DA DVSEC is implemented in compliance with PCIe [4] and has the following format:

Offset	Register name	Details
0x0000	RMEDA_ECH	See B3.2.6.2.1 *RME-DA Extended Capability Header*
0x0004	RMEDA_HEAD1	See B3.2.6.2.2 *RME-DA DVSEC Header 1*
0x0008	RMEDA_HEAD2	See B3.2.6.2.3 *RME-DA DVSEC Header 2*
0x000C	RMEDA_CTL1	See B3.2.6.2.4 *RME-DA Control register 1*
0x0010	RMEDA_CTL2	See B3.2.6.2.5 *RME-DA Control register 2*

B3.2.6.2.1 RME-DA Extended Capability Header

PCI Express Extended Capability Header (RMEDA_ECH).

Bit [15:0], ECH_ID PCI Express Extended Capability ID

Value	Meaning
0x0023	DVSEC

Bit [19:16], ECH_CAP_VER Capability Version

Value	Meaning
0x1	Version 1

Bit [31:20], NEXT_CAP_OFF Next Capability Offset

IMPLEMENTATION DEFINED

B3.2.6.2.2 RME-DA DVSEC Header 1

RME-DA DVSEC Header 1 (RMEDA_HEAD1).

Bit [15:0], DVSEC_VENDOR_ID DVSEC Vendor ID

Value	Meaning
0x13b5	Arm

Bit [19:16], DVSEC_REVISION DVSEC revision

Value	Meaning
0x0	Revision 0

Bit [31:20], DVSEC_LENGTH DVSEC length in bytes

Value	Meaning
0x014	20 bytes

B3.2.6.2.3 RME-DA DVSEC Header 2

RME-DA DVSEC Header 2 (RMEDA_HEAD2).

Bit [15:0], DVSEC_ID Vendor-defined ID that indicates the nature and format of the DVSEC structure

Value	Meaning
0xFF01	RME_DA

B3.2.6.2.4 RME-DA Control register 1

The RME-DA Control register 1 (RMEDA_CTL1) contains architectural RP controls for RME-DA.

Access to this register is RW. Unspecified bits are RES0 (RsvdP).

Bit [0] TDISP_EN TDISP Enable

Controls whether TDISP functionality for RME-DA is enabled in the RP

Value	Meaning
0b0	TDISP functionality for RME-DA is disabled.
0b1	TDISP functionality for RME-DA is enabled.

This bit resets to 0.

B3.2.6.2.5 RME-DA Control register 2

The RME-DA Control register 2 (RMEDA_CTL2) contains the IDE Selective Stream Lock vector.

Access to this register is RW. Unspecified bits are RES0 (RsvdP).

Bit [31:0], SEL_STR_LOCK A vector of IDE Selective Stream Lock bits.

A Selective Stream Lock bit has the following encodings:

Value	Meaning
0b0	The Selective IDE register blocks associated with the Lock bit are Unlocked
0b1	The Selective IDE register blocks associated with the Lock bit are Locked

NUM_SEL_STR denotes the number of Selective IDE Streams supported by the Root Port. A Selective IDE register block has an index *STR_INDEX* in the range of zero to (*NUM_SEL_STR*-1) and is associated with a Selective Stream Lock bit SEL_STR_LOCK[*STR_INDEX*]. If *NUM_SEL_STR* is bigger than 32 then streams with *STR_INDEX*>31 cannot be used with a TDISP-capable device. If *NUM_SEL_STR* is smaller than 32 then SEL_STR_LOCK[31:*NUM_SEL_STR*] are RES0 (RsvdP).

This field resets to 0.

R_{XHMDQ} When RMEDA_CTL1.TDISP_EN == 1 the following registers are RMSD write-protect:

- IMPLEMENTATION DEFINED registers that can impact the RME security guarantee and that are programmed by MSD firmware or a Trusted subsystem. For Example:
  - Registers that allow reading or modifying any *Transaction Layer Packet* (TLP) parameters, such as its address or data, or that might lead to a drop, corrupt, replay or reorder of a TLP before IDE is applied (for outgoing TLPs) or after the IDE check (for incoming TLPs).
  - Registers that allow forwarding a Poisoned TLP as a non-Poisoned TLP.
  - Registers that define the method of signaling an Unsupported Request (UR) over the host interface.
  - A register that controls the RP ID or the PCIe Segment Number of the RP.
  - Registers that might affect the correctness of IDE functionality, for example error injection controls.

I_{PTQSC} Both the RP, and the Root-complex Host bridge it is connected to, might have IMPLEMENTATION DEFINED registers that impact the RME security guarantee. Where such registers can be configured by MSD firmware or a Trusted subsystem, RMSD write-protect is the preferable register security property for guaranteeing their integrity, as it does not require a flow for validating the configuration and allows for runtime updates.

R_{NXJKQ} When RMEDA_CTL1.TDISP_EN == 1, the following registers are RMSD full-protect:

- IDE key programming registers.
- Registers that store IDE confidential information, for example Initialization Vectors (IV) or IMPLEMENTATION DEFINED confidential state.
- Registers that store payload from TLPs that have IDE T-bit == 1 or XT-bit == 1.

R_{PCRFM} When RMEDA_CTL1.TDISP_EN == 1, the following registers are RMSD write-detect:

- RP configurations that are not allowed to be modified when the RP has an IDE Stream bound to a TDI as specified in TDISP [4].
- IMPLEMENTATION DEFINED registers that can impact the RME security guarantee and that must be programmed by Non-secure state. For example, RP registers that perform address translation between system hardware address space and PCIe address space.

R_{HCMWC} When RMEDA_CTL1.TDISP_EN transitions from 1 to 0, all hosted IDE Streams transition to IDE Insecure state.

R_{RNQNM} When RMEDA_CTL1.TDISP_EN == 0, all of the following apply:

- For any incoming request with IDE T-bit == 1 or XT-bit == 1, the RP either:
  - Forces IDE T-bit == 0 and XT-bit == 0.
  - Rejects the request.
- The RP rejects with error an outgoing request if it needs to be sent with IDE T-bit == 1 or XT-bit == 1.

I_{XSNPG} The term *reject* means that the request can be silently dropped and that the RP is permitted but is not required to handle the request as UR in compliance with PCIe. The term *reject with error* applies to outgoing requests and means that a posted request can be silently dropped and for a non-posted request the RP returns an error response compatible with UR, for example Read-As-Ones (RAO) or NDERR.

R_{NPGJV} The RMEDA_CTL registers are RMSD write-protect by hardware default.

I_{SQPZR} RMEDA_CTL1 must be RMSD write-protect regardless of the value of RMEDA_CTL1.TDISP_EN. This guarantees that the IDE T-bit and the XT-bit are 0 for any transaction arriving on an RP that was not configured by RMSD or that is not visible to RMSD.

R_{NWSJB} All RPs in an RME-DA system must implement the RME-DA DVSEC.

I_{JGJYF} This provides software with a standard method for discovering RME-DA capabilities and enforcing correct IDE T-bit and XT-bit values across the system.

I_{VTBSP} A Selective IDE register block and a Selective IDE Stream associated with it can be in one of the following lock states: Unlocked, Locked. The lock state of a Selective IDE register block is defined by the value of the Selective Stream Lock bit it is associated with.

RYHQQL When a Selective IDE register block is Unlocked (SEL_STR_LOCK is 0):

- The block registers do not have any register security property.
- The associated Selective IDE Stream is in Unlocked state.

When a Selective IDE register block is Locked (SEL_STR_LOCK is 1):

- The block registers are RMSD write-detect.
- The associated Selective IDE Stream is in Locked state.

B3.2.6.3 Root Port checks and IDE association for outgoing traffic

RGKHSZ An RME-DA RP performs the following for all outgoing TLPs:

- Associates the TLP with an IDE Stream.
- Sets the IDE T-bit and XT-bit of the TLP to the appropriate value.

B3.2.6.3.1 IDE T-bit

RCFQBW An RME-DA RP sets the IDE T-bit for an outgoing PCIe Memory Request or Configuration Request based on the request PAS. If the request PAS is Realm or Root, then IDE T-bit is 1, otherwise it is 0.

IZGFNF The PAS is derived from outgoing request attributes based on its interface type.

For example, IDE T-bit for requests arriving on an AMBA AXI [7] interface is extracted as follows:

AxNSE	AxPROT[1]	PAS	IDE T-bit
0	0	Secure	0
0	1	Non-secure	0
1	0	Root	1
1	1	Realm	1

RSWBSV An RME-DA RP sets the IDE T-bit of PCIe messages as follows:

- For messages generated from AMBA DTI requests, the IDE T-bit is extracted from the request in compliance with AMBA DTI. See AMBA DTI [8] and SMMU for RME-DA [3].
- For Vendor-Defined messages, the IDE T-bit is permitted to be 1 if the RP has a method to associate the message with the Root or Realm Security state. Otherwise, the IDE T-bit is set to 0.
- For any other message, the IDE T-bit is set to 0. For example, Power Management messages.

RCKJMN IDE T-bit for PCIe completions is set in compliance with IDE and TDISP [4]. This means that:

- For ATS Translation Requests, the host will set the IDE T-bit on the corresponding ATS Translation Completion to match the IDE T-bit value of the request.
- For ATS-translated read requests, the host will set the IDE T-bit value on the corresponding read completion to match the value of the request, with the following exception:
  - If a P2P read request with IDE T-bit == 1 is forwarded through the host to a non-TDISP device, the host is permitted but not required to set IDE T-bit == 0 on the corresponding completion.

For a description of how an RP computes the XT-bit for outgoing requests and PCIe completions see B3.2.6.6 *Root port support for the TDISP eXtended TEE (XT) Extensions*.

B3.2.6.3.2 IDE Stream association

IYRQDN An RP that supports IDE performs a check on every outgoing request, to determine whether the request should be associated with a Selective IDE Stream. For PCIe memory requests, the association is based on IDE Address Association registers. For PCIe configuration requests and other ID-routed messages such as ATS invalidations, the association is based on IDE RID Association registers.

R_{DVKPF} An outgoing request that needs to be sent with IDE T-bit == 1 or XT-bit == 1, but that cannot be associated with a Selective IDE Stream that is Locked and in the IDE Secure state, is rejected with error by the RP.

I_{QWGVX} This specification mandates that an RME-DA RP supports at least one Selective IDE stream, for enabling RMSD to always use a Selective IDE stream for communicating with a TDISP-compliant PCIe device. RME-DA does not support using a Link IDE Stream to communicate with a TDISP-compliant PCIe device.

See also:

- B3.2.6.6 *Root port support for the TDISP eXtended TEE (XT) Extensions*

B3.2.6.4 Root Port checks and IDE association for incoming traffic

I_{SDRYY} In compliance with rule R_{RNQNM}, when RMEDA_CTL1.TDISP_EN == 0, the RP enforces IDE T-bit == 0 and TX-bit == 0 for any incoming request.

R_{KZBHV} When RMEDA_CTL1.TDISP_EN == 1, the RP permits an incoming request to have IDE T-bit == 1 or XT-bit == 1 if either:

- The request arrived on a Selective IDE Stream that is Locked and in the IDE Secure state.
- An IMPLEMENTATION DEFINED configuration controlled by MSD firmware or a Trusted subsystem enables the incoming request to have IDE T-bit == 1.

Otherwise, the RP either forces IDE T-bit == 0 and XT-bit == 0, or rejects the request.

I_{QVGRT} When RMEDA_CTL1.TDISP_EN == 1, the RP does not permit an incoming request arriving on a Link IDE Stream with IDE T-bit == 1 or XT-bit == 1, unless an IMPLEMENTATION DEFINED configuration controlled by MSD firmware or a Trusted subsystem permits it.

I_{PGWPR} In compliance with TDISP [4]:

- An incoming request received on a Selective IDE Stream is handled as an *Unsupported Request* (UR) if the RID field of the request is less than the RID Base or greater than the RID Limit in the Selective IDE RID Association Register block of the stream.
- Any incoming request that arrived on an IDE Stream that is not the IDE Secure state is rejected.

R_{MYKFH} When an RP forwards an incoming request over a host interface, it sets the SMMU SEC_SID, StreamID, and SubstreamID fields as follows:

- If XT Enable is cleared for the IDE stream and if the request has IDE T-bit == 1, SEC_SID is set to `0b10` (Realm). Otherwise, SEC_SID is set to `0b00` (Non-secure).
- If XT Enable is set for the IDE stream, see B3.2.6.6 *Root port support for the TDISP eXtended TEE (XT) Extensions*.
- SMMU StreamID and SubstreamID are set using the RID and PASID fields in accordance with Arm BSA [9] and SBSA [10] specifications.

I_{JPTYW} For example, AXI [7] host interface signals are set as follows based on the fields of an incoming TLP:

- AxMMUSECSID[1:0] = {T-bit || (XT-bit && XT Enable), 0}.
- AxMMUSID[SID_WIDTH-1:0] = {Segment ID, RID[15:0]}.
- AxMMUSSIDV is set to 1 if the TLP has a PASID prefix and to 0 otherwise.
- AxMMUSSID[19:0] = PASID[19:0].

Where:

- Segment ID is a static parameter associated with the RP, denoting its PCIe segment.
- XT Enable is the value of the XT Enable field for the corresponding selective IDE Stream.

I_{GBDBV} In compliance with IDE and TDISP [4], the RP performs the following checks:

- That outgoing read requests with IDE T-bit == 1 are responded to with read completions that use the same IDE Stream as the read request. If this is not met, the RP rejects the completion and reports an IDE Check Failed error.

- That outgoing ATS invalidation requests with IDE T-bit == 1 are responded to with ATS invalidation completions that have IDE T-bit == 1 and that use the same IDE Stream as the invalidation request. If this is not met, the RP rejects the completion.
- That any incoming IDE TLP is associated with an IDE Stream. If not, the RP rejects the incoming IDE TLP and reports a Misrouted IDE TLP error.

R_{MDPKR} When P2P traffic between two TDISP devices is routed through the Root Complex, then for any non-posted request that is forwarded by the Root Complex from a source peer to a target peer, the Root Complex must guarantee that the corresponding completion will be forwarded back to the source peer only if it originated from the target peer.

I_{GLCGT} For example, the Root Complex can achieve this by guaranteeing that a completion is forwarded back to the source peer only if it arrived on the same IDE stream on which the non-posted request was forwarded on to the target peer. This prevents completion spoofing by untrusted completers.

See also:

- B3.2.6.6 *Root port support for the TDISP eXtended TEE (XT) Extensions*

B3.2.6.5 Root Port general security requirements

R_{LMFSV} When RMEDA_CTL1.TDISP_EN == 1, any RP debug functionality that might affect the RME security guarantee is disabled unless explicitly enabled by one of:

- An access to a write-protect register.
- An assertion of a debug authentication signal indicating that either RMSD external debugging or Root external debugging is enabled.

R_{GSTJC} Any of the following events transitions all hosted IDE Streams to IDE Insecure state:

- A reset or loss of state of a write-detect, write-protect, or full-protect register.
- A reset or loss of state of an RP component that affects the RME security guarantee.

I_{KDXVQ} TDISP [4] defines how a TDI must respond to a power state transition that might be initiated by Non-secure state. A system power state transition might include peripherals as well as TDISP-compliant devices and must first be approved by MSD or a Trusted subsystem.

B3.2.6.5.1 RP as a requester

I_{KKCLM} RPs with Direct Memory Access (DMA) capabilities are able to initiate transactions as requesters, over their host interface and PCIe interface.

R_{MJNLW} Requests that are autonomously initiated by the RP over its host interface are tagged with PAS == Non-secure. Likewise, requests autonomously initiated by the RP over its PCIe interface must have IDE T-bit == 0.

I_{TJBQM} For example, an MSI message generated by the RP must have PAS == Non-secure. An RP with DMA capability is subject to GPC like any other non-PE requester.

I_{YRQDP} RP IMPLEMENTATION DEFINED logic can initiate requests with PAS == Realm where that logic is associated with a TDISP-compliant RCiEP.

See also:

- B3.2.7 *Root Complex Integrated Endpoint (RCiEP) requirements for RME-DA*

B3.2.6.5.2 Event and error handling

I_{FKQRD} If the following events are not reported using the Advanced Error Reporting (AER) mechanism, Arm recommends that the RP reports them using a different mechanism, for debugging purposes.

- An event that transitions an IDE Stream to the IDE Insecure state.
- An event that causes a request to be rejected in accordance with this specification.

For example, the RP can log an RMSD write-detect violation in an IMPLEMENTATION DEFINED error status register.

I_{CVRJC} An RP is permitted to use a Non-secure interrupt for reporting the following events to host software:

- A transition of an IDE Stream to IDE Insecure state.
- Key refresh is due for an IDE Stream.

Host software then forwards the interrupt to RMSD, which can then query the RP for the event details using an IMPLEMENTATION DEFINED mechanism exposed to RMSD by MSD firmware.

R_{PJGJK} When an RP encounters an uncorrectable data integrity error, it must do one of:

- Poison any TLP affected by the error.
- Transition any IDE Stream that the affected TLPs are associated with, or could be associated with, to the IDE Insecure state.

The RP is permitted to log and report the error as an *Uncorrectable Internal Error* using AER.

I_{SBTDM} An RP is permitted to use a Non-secure interrupt for reporting to host software the following errors as AER Uncorrectable Errors:

- Uncorrectable Internal Error.
- IDE Check Failed.
- Misrouted IDE TLP.

I_{VHVRB} The RME security guarantee does not include hardware-enforced interrupt integrity and delivery to the Realm and Root Security states.

I_{DXGRL} In compliance with RAS rules for RME, an uncontainable error in the RP must result in an RME system reset, for example by reporting it as a RAS Critical Error Interrupt to a Trusted subsystem.

B3.2.6.6 Root port support for the TDISP eXtended TEE (XT) Extensions

The TDISP eXtended TEE (XT) extensions ECN [11] specifies the XT-bit as a method to extend the semantics expressed by TEEs and TEE-assigned TDIs for accessing memory. The XT-bit is evaluated together with the T-bit to indicate the TEE attributes associated with a TLP.

I_{JKCDK} For an RME-DA RP that supports the XT extensions all of the following apply:

- The RP sets the *XT Supported* bit in the IDE Capability register, implements the *XT Enable* bit in the Selective IDE Stream Control Register, and complies with the XT extensions ECN [11].
- The RP complies with the rules specified in this section.

Arm recommends that an RME-DA system provides support for the XT extensions. In this case all of the following apply:

- All RPs support the XT extensions.
- The SMMU has SMMU_R_IDR3.XT=1.

R_{MJSFF} An RME system must not include an SMMU that supports the XT extensions, unless all RPs either support the XT extensions or override the XT-bit to 0 for incoming requests.

R_{JXRNG} If XT extensions are supported, then all of the following apply:

- If the XT Enable bit is set for the IDE stream, then the following applies to outgoing TLPs:
  - For any outgoing request:
    - The RP sets XT-bit == 1 if it sets T-bit == 1. For example, this applies to requests in the Realm PAS.
    - The RP sets XT-bit == 0 if it sets T-bit == 0. For example, this applies to requests in the Non-secure PAS.
  - For any outgoing message the IDE XT-bit is always set to 0. This includes:
    - Messages generated from AMBA DTI requests, for example ATC Invalidates or PRG responses. For a PRG response, this can also be implemented by reflecting the XT value from the Page request in the corresponding PRG response.
    - Vendor-Defined messages.
  - The host sets the IDE XT-bit and T-bit values on an outgoing read completion to match the values of the corresponding read request, with the following exception:

        - If a P2P read request with IDE XT-bit == 1 is forwarded through the host to a non-TDISP device, the host is permitted but not required to set IDE XT-bit == 0 on the corresponding completion.
      - The host sets the IDE XT-bit and T-bit values on an outgoing ATS Translation Completion to match the values of the corresponding ATS Translation Request.
    - If the XT Enable bit is set for the IDE stream, then the following applies to incoming TLPs:
      - For any incoming request or message forwarded by the RP over a host interface:
        - If after applying the RP overrides the request has IDE T-bit == 1 or XT-bit == 1, SEC_SID is set to Realm. Otherwise, SEC_SID is set to Non-secure.
        - If SEC_SID == Realm then:
          - If {XT-bit,T-bit} = {1,1} the PAS is set to Realm.
          - If {XT-bit,T-bit} = {1,0} the PAS is set to Non-secure.
          - If {XT-bit,T-bit} = {0,1} the PAS is Unknown. In AXI this is signaled using the AxMMUPASUNKNOWN semantic.
      - If the host interface supports the XT indication then the RP forwards the XT indication. Depending on the host interface type, this can be an explicit forward of the XT value or an encoded value of the combination {XT-bit,T-bit} = {0,1}.
    - If the XT Enable bit is clear for the IDE stream, then in compliance with TDISP [4]:
      - For outgoing requests or messages the RP sets XT-bit == 0.
      - For incoming requests with IDE XT-bit == 1, the RP forces XT-bit == 0.

I_{VVRFD} In an RME-DA system it is not possible to specify the combination {XT-bit,T-bit}={1,0} on outgoing transactions issued by a PE. Realm access to TDI locations marked as IS_NON_TEE_MEM == 1 must be in the Non-secure PAS, and if XT Enable is set, the RP converts this to {XT-bit,T-bit}={0,0}. This limitation also applies to host-routed P2P transactions in a system where the host takes ownership of the transaction. In this case, if the transaction specifies {XT-bit,T-bit}={1,0}, the Root Complex converts it to {XT-bit,T-bit}={0,0}.

B3.2.7 Root Complex Integrated Endpoint (RCiEP) requirements for RME-DA

I_{JDBRW} An on-chip peripheral that is implemented as a Root Complex Integrated Endpoint (RCiEP) and supports RME-DA must comply with TDISP [4]. An RCiEP is not required to implement IDE [4] for supporting RME-DA.

R_{GBVTS} As a completer of memory requests, a TDISP-compliant RCiEP extracts the request IDE T-bit from the request PAS. If PAS is Realm or Root then IDE T-bit is 1, otherwise it is 0.

R_{ZJJMZ} As a requester, an RCiEP sets the SMMU SEC_SID, StreamID, and SubstreamID fields of a request as follows:

- If the request must be sent with IDE T-bit == 1, the RCiEP sets SEC_SID to `0b10` (Realm). Otherwise the RCiEP sets SEC_SID to `0b00` (Non-secure).
- SMMU StreamID and SubstreamID are set using the RID and PASID fields in accordance with Arm BSA [9] and SBSA [10] specifications.

R_{QNTYC} The PCIe segment and RIDs that are allocated to an RCiEP are either defined statically or configured using an RMSD write-protect register.

B3.2.8 MEC support in system components

I_{XQKRQ} Arm recommends that all RME system components support the same MECID width, to avoid faulty behavior.

If the MECID bit width differs at the interface between two components, it can be zero-extended or have the most-significant bits truncated, as appropriate. This adjustment will only produce correct MEC operation if the *common MECID width* (defined in the Arm A-profile architecture [1]) is correctly resolved.

I_{MYRTS} For memory-mapped Resources that are not encrypted, such as peripheral registers or SRAM, the MECID can be stripped at an IMPLEMENTATION DEFINED location. For example, the MECID of an access to a PCIe memory-mapped peripheral can be stripped at the PCIe Root Port.

See also:

- B1.2 *Memory Encryption Context Identifier*

B3.2.9 Coherent host port requirements for RME-CDA

B3.2.9.1 Link protection

R_{GBGQX} A CTC coherent link that is exposed to physical attacks, for example external probing, manipulation of traffic, or device detaching, must support link protection in the form of cryptographic encryption and integrity, with a programming interface complying with CXL IDE [6].

I_{CGTGG} To support coherent link protection based on CXL IDE:

- Both the coherent host port and coherent device expose a CXL.cachemem IDE programming interface [6].
- The coherent host port exposes an IMPLEMENTATION DEFINED CXL IDE key programming interface for the following CXL IDE key management (CXL_IDE_KM) data objects:
  - KEY_PROG.
  - K_SET_GO.
  - K_SET_STOP.
- The coherent device must provide full support for the CXL_IDE_KM protocol for IDE key programming.

R_{DFWKW} A coherent host port that supports RME-CDA implements the Coherent RME-DA DVSEC (RME-CDA DVSEC).

R_{HMXTF} The system address range associated with a coherent host port must be controlled by one of the following:

- A dedicated CXL HDM decoder in a host bridge associated with that port.
- A dedicated RMSD write-protect or write-detect register associated with that port.

B3.2.9.2 Coherent host port registers

I_{HQBGM} The RME-CDA DVSEC is specified as a super-set of the RME-DA DVSEC, and supports both coherent traffic, for example CHI, and non-coherent traffic, for example PCIe or CXL.io.

It can be instantiated in both coherent and non-coherent ports.

I_{DHTFS} The term *Coherent Link IDE* is used when describing fields that apply to the coherent link protection protocol, for example CXL.cachemem IDE.

R_{GVRQC} The RME-CDA DVSEC is implemented in compliance with PCIe [4] and has the following format:

Offset	Register name	Details
0x0000	RMECDA_ECH	See B3.2.9.2.1 *RME-CDA Extended Capability Header*
0x0004	RMECDA_HEAD1	See B3.2.9.2.2 *RME-CDA DVSEC Header 1*
0x0008	RMECDA_HEAD2	See B3.2.9.2.3 *RME-CDA DVSEC Header 2*
0x000C	RMECDA_CTL1	See B3.2.9.2.4 *RME-CDA Control register 1*
0x0010	RMECDA_CTL2	See B3.2.9.2.5 *RME-CDA Control register 2*
0x0014	RMECDA_CTL3	See B3.2.9.2.6 *RME-CDA Control register 3*
0x0018	RMECDA_CTL4	See B3.2.9.2.7 *RME-CDA Control register 4*

B3.2.9.2.1 RME-CDA Extended Capability Header

PCI Express Extended Capability Header

Bit [15:0], ECH_ID PCI Express Extended Capability ID

Value	Meaning
0x0023	DVSEC

Bit [19:16], ECH_CAP_VER Capability Version

Value	Meaning
`0x1`	Version 1

Bit [31:20], NEXT_CAP_OFF Next Capability Offset

IMPLEMENTATION DEFINED

B3.2.9.2.2 RME-CDA DVSEC Header 1

RME-CDA DVSEC Header 1.

Bit [15:0], DVSEC_VENDOR_ID DVSEC Vendor ID

Value	Meaning
`0x13b5`	Arm

Bit [19:16], DVSEC_REVISION DVSEC revision

Value	Meaning
`0x0`	Revision 0

Bit [31:20], DVSEC_LENGTH DVSEC length in bytes

Value	Meaning
`0x01C`	28 bytes

B3.2.9.2.3 RME-CDA DVSEC Header 2

RME-CDA DVSEC Header 2.

Bit [15:0], DVSEC_ID DVSEC Identifier

A vendor-defined ID that indicates the nature and format of the DVSEC structure.

Value	Meaning
`0xFF03`	Coherent RME-DA

B3.2.9.2.4 RME-CDA Control register 1

The RME-CDA Control register 1 (RMECDA_CTL1) contains architectural host port controls for Coherent RME-DA. Access to this register is RW. Unspecified bits are RES0 (RsvdP).

Bit [0], TDISP_EN TDISP Enable

Controls whether TDISP functionality for Coherent RME-DA is enabled in the host port.

Value	Meaning
0b0	TDISP functionality for Coherent RME-DA is disabled.
0b1	TDISP functionality for Coherent RME-DA is enabled.

This bit resets to 0.

Bit [1], LINK_STR_LOCK Coherent Link IDE Stream Lock bit.

The Coherent Link IDE Stream Lock bit has the following encodings:

Value	Meaning
0b0	The Coherent Link IDE Stream is Unlocked.
0b1	The Coherent Link IDE Stream is Locked.

This bit resets to 0.

This bit is reserved (RES0) for PCIe RPs.

Bits [28:16], FIRST_TEE_CKID FIRST TEE CKID

CKID associated with Realm MECID 0.

This field resets to 0.

This field is reserved (RES0) for CHI-based coherent ports and PCIe RPs.

B3.2.9.2.5 RME-CDA Control register 2

The RME-CDA Control register 2 (RMECDA_CTL2) contains the IDE Selective Stream Lock vector.

Access to this register is RW. Unspecified bits are RES0 (RsvdP).

Bit [31:0], SEL_STR_LOCK A vector of IDE Selective Stream Lock bits.

A host port implements this register if it supports Selective IDE streams, for example for CXL.io or PCIe.

A Selective Stream Lock bit has the following encodings:

Value	Meaning
0b0	The Selective IDE register blocks associated with the Lock bit are Unlocked.
0b1	The Selective IDE register blocks associated with the Lock bit are Locked.

NUM_SEL_STR denotes the number of Selective IDE Streams supported by the Root Port. A Selective IDE register block has an index *STR_INDEX* in the range of zero to (*NUM_SEL_STR* - 1) and is associated with a Selective Stream Lock bit SEL_STR_LOCK[*STR_INDEX*]. If *NUM_SEL_STR* is bigger than 32 then streams with *STR_INDEX*>31 cannot be used for RME-CDA. If *NUM_SEL_STR* is smaller than 32 then SEL_STR_LOCK[31:*NUM_SEL_STR*] are RES0 (RsvdZ).

This field resets to 0.

B3.2.9.2.6 RME-CDA Control register 3

The RME-CDA Control register 3 (RMECDA_CTL3) contains parameters for Coherent Link IDE RID Association.

Access to this register is RW. Unspecified bits are RES0 (RsvdP).

Bit [23:8], RID_LIMIT RID Limit

Indicates the highest value RID in the range associated with the Coherent Link IDE Stream.

This field is reserved (RES0) for CXL-based coherent ports and PCIe RPs.

B3.2.9.2.7 RME-CDA Control register 4

The RME-CDA Control register 4 (RMECDA_CTL4) contains parameters for Coherent Link IDE RID Association.

Access to this register is RW. Unspecified bits are RES0 (RsvdP).

Bit [23:8], RID_BASE RID Base

Indicates the lowest value RID in the range associated with the Coherent Link IDE Stream.

This field is reserved (RES0) for CXL-based coherent ports and PCIe RPs.

Bit [0], RID_RANGE_VALID RID Range Valid

When set, indicates that the RID_BASE and RID_LIMIT fields have been programmed.

This field resets to 0.

This field is reserved (RES0) for CXL-based coherent ports and PCIe RPs.

B3.2.9.3 Requirements for coherent host port registers

R_{VSFPJ} When RMECDA_CTL1.TDISP_EN == 1, the following registers in a coherent host port are RMSD write-protect:

- Any host port register that can impact the RME security guarantee and that must be programmed by Non-secure state. For example:
  - Host port registers that perform address translation between system hardware address space and CXL address space.
  - CXL or PCIe configuration space registers that are not allowed to be modified when an IDE Stream is bound to a TDI as specified in CXL [6]. For example, the CXL Extended Security Capability register and HDM Decoder Global Control Register.
- IMPLEMENTATION DEFINED registers that can impact the RME security guarantee and are programmed by MSD firmware or a Trusted subsystem. For Example:
  - Registers that allow reading or modifying any Transaction Layer Packet (TLP) parameters, such as TLP address or data, or that may lead to a drop, corrupt, replay or reorder of a TLP before IDE is applied (for host-to-device packets) or after the IDE check (for device-to-host packets).
  - Registers that control transaction Poison state.
  - Registers that define the method of signaling an Unsupported Request (UR) over the host interface.
  - Registers that control the port ID or the Segment Number of the host port.
  - Registers that may affect the correctness of IDE functionality, for example error injection controls.

In cases where the coherent host port is a CXL Root Port, this rule also applies to the associated CXL Host bridge, unless it is protected by other methods.

R_{PHCGC} When RMECDA_CTL1.TDISP_EN == 1, the following registers are RMSD full-protect:

- IDE key programming registers.
- Registers that store IDE confidential information, for example Initialization Vectors (IV) or IMPLEMENTATION DEFINED confidential state.
- Registers that store payload from TLPs that have SEC_SID == Realm.

R_{FDVZC} When RMECDA_CTL1.TDISP_EN transitions from 1 to 0, all IDE Streams transition to IDE Insecure state.

R_{NYCLL} When RMECDA_CTL1.TDISP_EN == 0:

- If a device-to-host memory request has SEC_SID == Realm or PAS == Realm, the host port rejects with error the request .
- If a host-to-device memory request has PAS == Realm, the host port rejects with error the request.

R_{WPGJB} The RMECDA_CTL registers are RMSD write-protect by hardware default.

I_{GPBKX} This means that the Coherent Link IDE RID Association parameters cannot be programmed by Non-secure software and must be directly programmed by RMSD.

R_{PLYKV} All host ports in a system that supports Coherent RME-DA implement the RME-CDA DVSEC.

I_{GPQDW} The Coherent Link IDE stream can be in either the Unlocked or Locked state, as defined by the value of the Coherent Link IDE Stream Lock bit.

R_{DHNWR} When RMECDA_CTL1.LINK_STR_LOCK is 0:

- The IDE capability structure registers do not have any register security property.
- The Coherent Link IDE Stream is in the Unlocked state.

When RMECDA_CTL1.LINK_STR_LOCK is 1:

- The IDE capability structure registers are RMSD write-protect.
- The Coherent Link IDE Stream is in the Locked state.

R_{WYVCQ} If RMECDA_CTL1.LINK_STR_LOCK == 0, host-to-device requests that have PAS == Realm must not be associated with a Coherent Link IDE Stream and must be rejected with error by the host port. This means that writes are dropped and reads are responded to with poison.

R_{GTVGZ} When RMECDA_CTL1.TDISP_EN == 1, a device-to-host request that has SEC_SID == Realm or PAS == Realm is handled by the host port as follows:

- If RMECDA_CTL1.LINK_STR_LOCK == 1, and the IDE Stream is in the Active state or IDE is not supported, then the request is permitted.
- Otherwise, the host port handles the request as if RMECDA_CTL1.TDISP_EN == 0 as described in R_{NYCLL}.

R_{XQHNG} If RMECDA_CTL1.LINK_STR_LOCK == 1, a device-to-host request is rejected with error by the host port in the following cases:

- If the Requester ID field of the request is less than the RMECDA_CTL4.RID_BASE or greater than RMECDA_CTL3.RID_LIMIT.
- If RMECDA_CTL4.RID_RANGE_VALID is 0.

I_{JTGYR} The Selective Stream Lock bits (RMECDA_CTL2.SEL_STR_LOCK) apply to CXL.io Selective IDE streams and have the same properties as Selective Stream Lock bits of a PCIe Root Port.

I_{FKVQX} Host port checks and link protection requirements apply to both coherent and non-coherent links associated with the device. For CXL IDE implementations, the requirements apply regardless of whether a packet arrived over CXL.io or over CXL.cachemem.

A host port implementation is permitted to use a single RME-CDA DVSEC for multiple coherent links that connect to a single TDISP-compliant device.

RMSD can guarantee that multiple links will be bound to a single TDISP-compliant device by provisioning link protection keys for all interfaces through the single SPDM session that is set up with the TDISP-compliant device.

See also:

- B3.2.6.1.2 *Key refresh schedule*
- B3.2.6.2 *Root Port registers*
- B3.2.6.5 *Root Port general security requirements*

B3.2.10 CXL.mem requirements

The CXL.mem interface specified in CXL [6] enables the attachment of different types of memory devices to a system. A device that supports CXL.mem can be one of the following:

- A Type-3 *memory expansion device*, which is a generic memory device that expands the main memory of the system.
- A Type-3 *special-purpose device*. For example, this can be a non-volatile memory device or a different form of non-generic device such as an accelerator with direct-attached memory.
- A Type-2 device, which is a CXL device that supports both CXL.mem and CXL.cache interfaces.

CXL-TSP [6] is a CXL feature that specifies mechanisms for adding CXL devices to the trust boundary of TEEs and supporting confidential compute scenarios using these devices. The first version of CXL-TSP specifies security requirements for CXL.mem HDM-H devices only. HDM-DB requirements will be added in a later version of this specification.

I_{XFHFR} Arm recommends that all CXL Type-3 memory expansion devices in an RME system support CXL-TSP. This specification permits non-TSP memory expansion devices to be used as well under certain restrictions.

R_{LQMCY} In an RME system, if a Type-3 memory expansion device does not support target-side memory encryption then its memory must be encrypted by a host-side MPE.

R_{HCQWS} A host-side MPE shall comply with initiator-based memory encryption requirements specified in CXL-TSP. For example, when handling a partial write, a MemRdFill should be used for the under fill read.

I_{PLCMC} A CXL-TSP Type-3 device that supports target-side memory encryption implements the *CKID-based Encryption* capability and advertises the number of CKIDs it supports using the *Number of CKIDs* field. In an RME system with MEC, *Number of CKIDs* should not significantly restrict the *common MECID width* of the system. For example, for a *common MECID width* of 9, the *Number of CKIDs* needs to be at least 513, to allow for 512 TVMCKIDs and 1 OSCKID.

R_{XWJNN} If the link between a host node and a CXL memory expansion device is exposed to physical attacks, such as device detaching, external probing, or external manipulation, then link protection using CXL IDE must be supported.

R_{CNSLJ} An RME system can include a Type-3 memory expansion device that does not support CXL-TSP if all of the following conditions are met:

- The CXL device memory is encrypted by a host-side MPE.
- The CXL device supports *Component Measurement and Authentication* (CMA).
- The CXL device memory integrity is guaranteed either by a host-side MPE that supports cryptographic integrity and replay protection, or by having the CXL device and any switch that it is connected to be exclusively controlled by RMSD or MSD. This means that all memory-mapped configuration registers of the CXL device are effectively RMSD write-protect registers. This includes both IMPLEMENTATION DEFINED and CXL-specified registers such as HDM decoder registers.
- All non-PE requesters in the system, including coherent and non-coherent devices, access the CXL Type-3 device through a host node. *UIO Direct P2P* [6] is not enabled for the device.
- Back-invalidate snoops are not enabled for the device.

I_{CCZBQ} R_{CNSLJ} ensures that CXL devices that do not support CXL-TSP are not exposed to confidentiality and integrity attacks by manipulating device registers.

I_{SKMSX} In an RME system that uses CXL Type-3 devices, RMSD and MSD are responsible for guaranteeing the protection of Realm content stored within a Type-3 device. This includes:

- Setting up an SPDM session with the device over a DOE mailbox.
- Obtaining device identity and measurement and including these in attestation reports.
- Managing CXL-TSP device states and programming its security features, for example CKID and cryptographic contexts.
- Setting up a CXL IDE link between the host and the device by programming IDE at the CXL Root Port and the CXL device. This requires implementing the IDE_KM protocol in compliance with the CXL IDE Establishment flow [6].

Depending on the system feature set, the following additional support may be required:

- Supporting both single logic device (SLD) and multi logic device (MLD) types.
- Identifying Shared memory regions in MLD memory map and excluding these ranges from being assigned to the Realm, Secure, or Root PAS.
- CXL switch authentication if CXL-TSP Type-3 devices are connected to the host through a CXL-TSP switch.
- Supporting dynamic allocation/de-allocation of device physical memory backing up an HDM range.

I_{HLBVL} CXL-TSP defines a *TEE Exclusive State* (TE State) memory property that indicates whether the content of the memory is for TEE or non-TEE data. Correspondingly, accesses are tagged with a TEE-attribute referred to as *TEE Intent*. CXL-TSP further specifies a TE State access control mechanism for verifying the TEE Intent against the TE State of the memory being accessed.

In an RME system, TE State access control is not required for HDM-H memory, but will be required for HDM-DB memory. Since this version of the RME system architecture does not include support for HDM-DB memory, it does not define mechanisms related to TE State access control or to the *TEUpdate* memory transaction.

When RMECDA_CTL1.TDISP_EN == 1, RMSD will disable HDM-DB for a memory device attached to the CXL Root Port by enforcing that *BI Enable* == 0 for it.

R_{HHMVM} If a CXL Root Port does not implement the RME-CDA DVSEC or has RMECDA_CTL1.TDISP_EN == 0 then for any Back-invalidate snoop request sent on the BISnp channel, the Root Port must tag the snoop with PAS == Non-secure. A Root Port is permitted to reject the snoop if the access PA does not fall within the HDM range for the device.

B3.2.10.1 Cache maintenance

R_{PTGGP} If a PA of a PoE CMO or a PoPA CMO specifies a memory Location in a CXL Type-3 device, the host must guarantee that the CMO reaches any cache that is located before the CXL Root Port associated with the device. This includes caches that might exist within a host-side MPE.

If Back-invalidate snoops are enabled for the device, the host must guarantee that PoPA and PoE CMOs have the same effect on the device as a CMO to the PoC. That is, PoPA, PoE, and PoC CMOs are all converted to the same CXL.mem semantic that guarantees that device-side caches are flushed.

B3.2.10.2 CXL Root Port requirements

R_{BYTYV} For a CXL Root Port (CXL RP) in an RME system, one of the following applies:

- The CXL RP supports the RME-CDA DVSEC.
- The CXL RP forces SEC_SID == Non-secure and PAS == Non-secure for any device-to-host request. In this case, the RP also rejects with error host-to-device requests with PAS == Realm, unless an IMPLEMENTATION DEFINED mechanism controlled by MSD firmware or a Trusted subsystem permits this.

I_{YGPHL} Arm recommends that the RME-CDA DVSEC is supported by any CXL RP that is used for memory expansion.

R_{KJYPB} If a CXL RP is not subject to GPC then CXL.cache must be disabled for it using an MSD-protected register.

R_{PHWMM} A CXL Root Port that implements the RME-CDA DVSEC must comply with CXL-TSP.

I_{BNCSB} In the following text, *TEE-attribute* indicates whether a CXL transaction is marked with a TEE opcode or with a non-TEE opcode.

R_{JXPZP} A CXL Root Port that implements the RME-CDA DVSEC must perform the following mapping from {PAS, MECID} to {TEE-attribute, CKID} for CXL.mem requests:

PAS	Result
Non-Secure	If MECID >= RMECDA_CTL1.FIRST_TEE_CKID then report an Uncorrectable Error. Otherwise: CKID = MECID. TEE-attribute = 0.
Realm	CKID = RMECDA_CTL1.FIRST_TEE_CKID + MECID. TEE-attribute = 1. If CKID > `0x1FFF` then report an Uncorrectable Error.
Root	CKID = RMECDA_CTL1.FIRST_TEE_CKID. TEE-attribute = 1.
Secure	Report an Uncorrectable Error.

I_{DSNGX} In compliance with CXL [6], a transaction that generates an Uncorrectable Error is either dropped or poisoned. This applies to R_{JXPZP} as well.

I_{WLLSQ} R_{JXPZP} implies that a CXL-TSP device cannot host locations in the Secure PAS but is permitted to host locations in the Root PAS. This is meant to enable storing a level 1 GPT in a CXL-TSP device that protects memory locations within that device. A Root PAS access uses the same CKID as a Realm PAS access with MECID == 0.

I_{DKQMJ} In compliance with R_{WYVCQ}, a CXL Root Port must reject with error a request that has PAS == Realm if RMECDA_CTL1.TDISP_EN == 0.

R_{DWRKS} When RMECDA_CTL1.TDISP_EN == 1 for a CXL Root Port, the following registers are RMSD write-protect in addition to the registers specified in R_{VSFPJ}:

- CXL Root Port and Host Bridge configurations that must be protected as specified by CXL-TSP [6] (11.5.4.8.2 Considerations for Securing the Host). For example:
  - CXL HDM Decoder Global Control register.
  - CXL IDE Control register.
  - CXL BI Decoder register.
  - CXL Extended Security Capability registers (Device Trust Level).

See also:

- B3.2.9.2 *Coherent host port registers*

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 the Arm CCA Security Model [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 the Arm CCA Security Model [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 [10], 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} The Arm A-profile architecture [1] specifies the related requirements for A-profile 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 A-profile 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 the Arm CCA Security Model [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.

I_{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.

R_{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.

R_{HJSSG} The HES implementation exposes a private interface to SSD components such as Trusted subsystems for requesting HES services.

I_{PMWXL} Services provided by HES for SSD are defined in the Arm CCA Security Model [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.

R_{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.

I_{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.

R_{BQPFG} HES has exclusive read and write access to RNVS confidential parameters.

R_{BTWVY} A measurement register can be either extended using a secure hash algorithm, locked or reset.

R_{DFPJL} HES has exclusive access to extend, lock, and reliably obtain the value of a measurement register it owns.

R_{FWSRF} Once locked, a measurement cannot be further extended until it is reset.

R_{WYSLK} An RME system reset is the only method to reset a measurement owned by HES.

R_{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 the Arm CCA Security Model [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 the Arm A-profile architecture [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.

The Arm A-profile architecture [1] describes how GPC faults can be reported to EL3 using synchronous GPC exceptions.

When an error is detected asynchronously, it can be signaled 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 the SMMU architecture [3].
- A GPT walk fault detected by an MMU on a write to the Trace Buffer.
- A RAS critical error detected by a Trusted requester.

B4.3 RAS

The Arm A-profile architecture [1] provides implementation requirements for RAS in PEs, and the RAS System Architecture [12] provides implementation requirements for system components using the Arm RAS architecture. This section provides additional rules for system components described in this specification.

B4.3.1 Confidential information in RAS Error Records

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

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

I_{HBRCT} The RME security guarantee for the protection of confidential information applies to systems in the Secured lifecycle state. If an RME system is not in the Secured state or if Root external debugging is enabled, then:

- Recording can be enabled for any error record.
- Any RAS debugging feature can be enabled.

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.

I_{BHNYL} For rules on signaling errors to PEs, see the Arm A-profile architecture [1].

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

R_{LWVCX} An uncontainable error results in an RME system reset.

I_{HYXZN} Components that implement RAS can be reset with an Error Recovery Reset, so that RAS syndrome information is not lost.

R_{JNBWJ} Only SSD or MSD can enable or disable the generation of a CI.

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.

R_{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.

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

I_{DKXXG} Arm recommends MPEs with integrity support implement the Arm RAS system architecture [12].

R_{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.

I_{TMKYF} Passing poison through the MPE does not weaken its security properties.

B4.4 MPAM

I_{WRPFN} The definition of MPAM support for RME is specified in the MPAM PE Architecture [1] and *Arm® Memory System Resource Partitioning and Monitoring (MPAM) Memory System Component (MSC) Specification*.

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 MPAM [13].

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} The Arm A-profile architecture [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 defense-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) and Activity Monitoring Unit (AMU) 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 following rules augment existing rules specified in the CoreSight PMU architecture [14] 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 the CoreSight PMU architecture [14] 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.

I_{LZNLQ} Certain platform use cases may require Non-secure software to have visibility to PMU or AMU events associated with other Security states. This could pose a side-channel risk if the events are accessible to Non-secure software in an uncontrolled manner. An RME system can control and enable such visibility in different methods, as long as the enabled level of visibility is reported to the Realm owner through platform attestation.

For example:

- An RME system is permitted to implement the per-PAS authentication indicated in R_{MMPWY} such that different controls enable the counting of different PMU event types.
- MSD can associate a set of PMU and AMU event types with a specific policy, and enable the counting of this set in counters that are accessible to Non-secure software, while permitting the counting of events that are associated with the Realm state.
  - An example for such policy is permitting System PMU events that do not leak address patterns to be visible to Non-secure software.
- MSD reports the enabled policy to the Realm through platform attestation.

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 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 address 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 behavior 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-dependent 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 [10] 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).
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 component has been performed.

When powering-up a component within a level of the system hierarchy (PE, PE-Cluster, System) power control must establish that all data and instruction caches are in INVALID state, and that the level of system hierarchy is configured with power settings that allow it to operate correctly.

R_{GVJYZ} A register that affects a system power policy or a hardware power mode must have at least one of the following properties:

- The register is implemented as an MPR.
- SSD hardware guarantees that the value programmed to the register cannot violate the power specification of the system or cause corruption of state.

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.

See also:

- B4.2.1 *System configuration integrity*

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 the Arm CCA Security Model [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 and cannot change state until a subsequent RME system reset.

R_{XVNFV} Root external debugging can only be authorized following an RME system reset and before MSD firmware is loaded and cannot change state until a subsequent RME system reset.

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} RME extends the Debug Authentication Interface defined by the Arm A-profile architecture [1] to include **RTPIDEN** and **RLPIDEN** as signals for authorizing Root external debugging and RMSD external debugging, respectively.

In compliance with rules R_{HLTLK} and R_{XVNFV}:

- A Trusted subsystem is permitted to set **RTPIDEN** or **RLPIDEN** to TRUE only before MSD starts executing, and the system must not change the value of these signals until a subsequent RME system reset.
- MSD is permitted to set **RLPIDEN** to TRUE only before RMSD starts executing, and must not change the value of this signal until a subsequent RME system reset.

Correspondingly, an attestation report will include the authorization state of Root and RMSD external debugging as directly reflected by these signals, even if external debugging is disabled for other reasons, for example since a Non-secure debug authentication signal is not asserted.

R_{QLPNL} When external debugging is enabled for any Security state, external requests to power-up a component within a level of the system hierarchy (PE, PE-Cluster, System) are permitted but must be executed by trusted power control.

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.

I_{CFDQY} The term *firmware update* refers to the operation of writing a new firmware image to non-volatile storage. A *controlled firmware update* describes the case where the update of the firmware image, and of any other image it depends on for booting, is carried by an authorized agent, for example a Trusted subsystem.

R_{HJHRL} On an RME system reset:

- All Trusted requesters and Trusted subsystems are reset.
- Any state in Trusted requesters or Trusted subsystems that might include confidential information must be set to known values by by trusted mechanisms.
- Any *system state* that might contain confidential information and that is accessible to software, including SSD firmware or MSD firmware executing at any Exception level, or is accessible through external debug, must be set to known values by trusted mechanisms.

Trusted mechanisms may include any of the following, provided in a decreasing order of trust:

- SSD hardware.
- Immutable firmware. For example, this can be a hardware ROM or firmware that is immutable through other means, such as a hardware hash-lock.
- SSD or MSD firmware that are updatable using controlled firmware update.
- SSD or MSD firmware that are auditable by the Realm owner, for example through a public firmware review framework.
- SSD or MSD firmware that are not updatable using controlled firmware and are not auditable by the Realm owner.

The chosen set of trusted mechanisms that an RME system relies on for complying with this rule can be reported through platform attestation.

I_{ZVHWK} *System state* includes caches, PE registers that are affected by a cold reset, and IMPLEMENTATION DEFINED or micro-architectural state.

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

A controlled firmware update can help in ensuring that an update of MSD firmware or SSD firmware followed by an RME system reset never exposes confidential state of Realms that were running prior to the reset.

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 [12] 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 [15] is a permitted variant of an RME system reset.

See also:

- In the *Arm® Architecture Reference Manual for A-profile architecture*:
  - *Reset Types* in chapter *AArch64 System Level Programmers' Model*.
  - *Error Recovery Reset* in chapter *RAS System Architecture*.

- 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 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.

B8.4 Multi-chip systems

I_{XDJZS} A *multi-chip system* comprises multiple discrete nodes (host nodes) that are interconnected using chip-to-chip (CTC) interfaces. Host nodes could be, for example, compute nodes or I/O nodes, implemented as monoltihic SoCs or as chiplets. A CTC interface might be vulnerable to physical attacks, particularly if it is externally exposed.

R_{LYXGC} A CTC interface in a multi-chip RME system supports all of:

- Transport of the PAS tag with any access that specifies a physical address (PA).
- Transport of the MECID with any access that specifies a PA, if the RME system supports MEC.
- Transport of CMO and DVM messages that RME and MEC [1] specify.

R_{HXJRC} If an RME system supports 4 MPAM PARTID spaces, the CTC interface transports the MPAM_SP[1:0] indication.

R_{GZMNH} An RME system reset in a multi-chip system affects all nodes.

B8.4.1 Link protection

R_{CMMZS} A CTC interface that is vulnerable to physical attacks, for example by external probing or manipulation of traffic, supports *Link protection* in the form of cryptographic encryption and integrity protection.

R_{VZCPW} The security level and cryptographic strength of Link protection is equivalent to IDE [4] or better. This means that:

- The encryption algorithm and cryptographic parameter sizes (MAC, IV, Key) are as supported by IDE or better.
- There is encryption of at least the payload data of transmitted packets.
- There is integrity protection against corruption, drop, replay, and reorder of packets.

R_{MSRGY} Link protection is mandatory for both:

- Transactions associated with a PA in the Root, Realm and Secure physical address spaces.
- Transactions not associated with a physical address space, for example DVM messages.

I_{KKNVS} Link protection for transactions associated with a PA in the Non-secure physical address space is permitted but not required.

R_{KYCPH} The following CTC interface registers are implemented as MSD-Protected registers or as measurable registers:

- CTC programming registers that allow reading or modifying transaction parameters such as packet address or data, or that could cause corruption, drop, replay, or reorder of packets, either:
  - Before Link protection is applied (for outgoing traffic).
  - After the Link protection check (for incoming traffic).
- CTC programming registers that control Link protection context and properties, for example key programming registers and enable bits.

I_{VPSBV} The key programming register format for Link protection, and the mechanisms for detecting and handling key refresh events for Link protection, are IMPLEMENTATION DEFINED. MSD firmware or a Trusted subsystem can manage key refresh events.

See also:

- B3.2.6.1.2 *Key refresh schedule*

B8.4.2 Multi-chip RME system initialization

I_{VZWQW} During multi-chip RME system initialization, before the earliest PE completes PE initial boot, SSD or HES must securely establish a global system state for all the following system properties:

- The authenticity of any node in the system.
- The external debug state of the system.
- The lifecycle state of the system.
- The global system memory map.

R_{LXPMB} Establishing a global system state complies with all the following:

- If a Debug Authentication Interface signal is enabled for one node, it is enabled for the system.
- The lifecycle state of the system is set to *Secure* only if it is *Secure* for all nodes.
- For memory-mapped Resources, the value of the PA through which a Resource can be reached is the same across all nodes.
- All nodes observe the same GPT information at any level of the GPT.

I_{KDPSY} During system initialization, a set of Root Non-volatile Storage (RNVS) parameters common to all nodes is established. The method to securely negotiate and resolve the common set of parameters is out of scope of this specification. The following guidelines are provided:

- If the system policy permits different nodes to have variance in an RNVS parameter, then any globally negotiated parameter must be applicable to all nodes.
  - For example, if different nodes report different values for LEGACY_TZ_EN, the negotiated value is TRUE and must be observed by all nodes regardless of their local value.
- The system only enables a security feature, for example MEC, if all nodes in the system support it.

See also:

- *Arm® Confidential Compute Architecture (CCA) Security Model* [2].

Part C
Appendix

Appendix 1: System flows

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 confidential encryption context in tables, registers or caches 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 the Arm CCA Security Model [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*

Appendix 2: TDISP VDMs

In this section a set of Arm-specific TDISP VDM requests and responses are defined that comply with the VDM_REQUEST format introduced by TDISP [4].

TDISP VDM requests enable vendors to define messages that are specific for their use-cases. The VDMs defined in this section are meant to address properties that are unique to CHI-based coherent devices.

R_{FPYMV} On a successful completion of an Arm TDISP VDM request, a corresponding response message is returned by the device. Upon an unsuccessful completion of a request, the TDISP_ERROR response message is returned by the device.

I_{QKTCB} The following Arm TDISP VDM request and response types are defined:

Requests:

Type	Name	Permitted TDI states
0x01	GET_VERSION_REQ	Any
0x02	GET_DEV_PROP_REQ	Any
0x05	SET_INTERFACE_REQ	CONFIG_UNLOCKED

Responses:

Type	Name	Permitted TDI states
0x11	GET_VERSION_RESP	Any
0x12	GET_DEV_PROP_RESP	Any
0x15	SET_INTERFACE_RESP	CONFIG_UNLOCKED

R_{WFPXR} The device is permitted to respond to any Arm TDISP VDM request with the TDISP_ERROR message as specified by TDISP [4].

GET_VERSION_REQ

RPXLFY The TDISP VDM request GET_VERSION_REQ is used for querying the Arm-specified protocol versions supported by the device.

The request does not apply to a specific INTERFACE_ID and the field is ignored by the device.

The request has the following format:

Offset	Field name	Details
0x0000	ARM_VDM_HEADER	See ARM_VDM_HEADER.
0x0004	REQ_RESP_HEADER	See Request/Response Header.

ARM_VDM_HEADER

Bit [7:0], REGISTRY_ID ID of the registry assigning the VENDOR_ID.

Value	Meaning
0x00	PCI-SIG

Bit [15:8], VENDOR_ID_LEN Length of the VENDOR_ID field.

Value	Meaning
0x02	2 bytes

Bit [31:16], VENDOR_ID Vendor ID.

Value	Meaning
0x13b5	Arm

Request/Response Header

Bit [7:0], ARM_VDM_VERSION ARM VDM version.

8-bit version entry where each entry is formatted as:

- 7:4 – Major Version Number.
- 3:0 – Minor Version Number.

Value	Meaning
0x00	Version 0.0

Bit [15:8], MESSAGE_TYPE ARM VDM message type.

See: Arm TDISP VDM request and response types.

Bit [31:16], RES0

GET_VERSION_RESP

R_{WFWLF} The TDISP VDM response GET_VERSION_RESP has the following format:

Offset	Field name	Details
0x0000	ARM_VDM_HEADER	See ARM_VDM_HEADER.
0x0004	REQ_RESP_HEADER	See Request/Response Header.
0x0008	GET_VERSION_DATA	See GET_VERSION_DATA.

GET_VERSION_DATA

Bit [7:0], VERSION_COUNT

Number of version entries, N. Max value of N is 256.

Bit [(N-1)*8+15:8], VERSIONS

A list of 8-bit version entries in ascending order where each entry is formatted as:

- 7:4 – Major Version Number.
- 3:0 – Minor Version Number.

SET_INTERFACE_REQ

R_{XDKDT} The TDISP VDM request SET_INTERFACE_REQ associates memory properties with a TDI.

The request applies to a specific INTERFACE_ID as specified in the TDISP message header.

The request has the following format:

Offset	Field name	Details
0x0000	ARM_VDM_HEADER	See ARM_VDM_HEADER.
0x0004	REQ_RESP_HEADER	See Request/Response Header.
0x0008	PROPERTIES	See SET_INTERFACE_REQ PROPERTIES.

SET_INTERFACE_REQ PROPERTIES

Bit [15:0], PMECID

Primary MECID associated with the TDI.

Bit [31:16], RES0

SET_INTERFACE_RESP

R_{FMHST} The TDISP VDM response SET_INTERFACE_RESP has the following format:

Offset	Field name	Details
0x0000	ARM_VDM_HEADER	See ARM_VDM_HEADER.
0x0004	REQ_RESP_HEADER	See Request/Response Header.

GET_DEV_PROP_REQ

R_{GRPDP} The TDISP VDM request GET_DEV_PROP_REQ is used for querying properties of the device.

The request does not apply to a specific INTERFACE_ID and the field is ignored by the device.

The request has the following format:

Offset	Field name	Details
0x0000	ARM_VDM_HEADER	See ARM_VDM_HEADER.
0x0004	REQ_RESP_HEADER	See Request/Response Header.

GET_DEV_PROP_RESP

R_{GHDCB} The TDISP VDM response GET_DEV_PROP_RESP has the following format:

Offset	Field name	Details
0x0000	ARM_VDM_HEADER	See ARM_VDM_HEADER.
0x0004	REQ_RESP_HEADER	See Request/Response Header.
0x0008	PROPERTIES	See GET_DEV_PROP_RESP PROPERTIES.
0x000C	REGISTER_LIST	See GET_DEV_PROP_RESP Register list.

GET_DEV_PROP_RESP PROPERTIES

Bit [0], PAS_CHECK The PAS_CHECK property for the coherent device.

Value	Meaning
0b0	The PAS_CHECK property is not supported by the device.
0b1	The PAS_CHECK property is supported by the device.

Bit [1], MEC MEC support for the coherent device.

Value	Meaning
0b0	Device does not support MEC.
0b1	Device supports MEC.

Bit [7:2], RES0
Bit [12:8], MECID_BITWIDTH

MECID Width that the device supports, expressed in number of bits.

Bit [31:13], RES0

GET_DEV_PROP_RESP Register list

This structure is composed of a list of configuration registers that the device provides to the host so that it can check their validity. Such registers are typically programmed by Non-secure software and locked by the device once a TDI is in the CONFIG_LOCKED state.

The list of registers is IMPLEMENTATION DEFINED.

Part D
Glossary

Glossary

Application PE

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

BAR

Base Address Register

CCA

Confidential Compute Architecture

CDA

Coherent Device Assignment. See RME Coherent Device Assignment.

CHI

Coherent Hub Interface. CHI is an Advanced Microcontroller Bus Architecture (AMBA) protocol that classifies different components in a system by node type and provides a means for communication between nodes.

CMO

Cache Maintenance Operation

Coherent

Data accesses from a set of observers to a byte in memory are coherent if accesses to that byte in memory by the members of that set of observers are consistent with there being a single total order of all writes to that byte in memory by all members of the set of observers. This single total order of all to writes to that memory location is the *coherence order* for that byte in memory.

Coherent device

A non-PE agent that can participate in the coherency protocol of the host system as a fully-coherent node. See I_{GWFRT}.

Completer

An agent in a computing system that responds to and completes a memory transaction that was initiated by a Requester.

CTC

Chip-to-chip

DA

Device Assignment

DCM

Device Coherent Memory

DPT

Device Permission Table. DPT contains permission attributes associated with physical addresses and specifies a corresponding set of DPT checks that apply to translated accesses from devices.

DSM

Device Security Manager. TDISP term for the logical entity in a device that enforces security policies.

DVSEC

Designated Vendor-Specific Extended Capability

ECAM

Enhanced Configuration Access Mechanism. A PCIe-specified flat memory-mapped address space to access device configuration registers.

ECC-scrubbing:

The operation of refreshing DRAM Error correction code (ECC) state.

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

HCM

Host Coherent Memory

HES

Hardware Enforced Security

IDE

Integrity and Data Encryption

Location

An address in a Physical Address Space

MECID

Memory Encryption Context Identifier. An identifier assigned to memory accesses, associating each access with a memory encryption context.

MPE

Memory Protection Engine

MPR

MSD-Protected Register

MSD

Monitor Security Domain

PAS

Physical Address Space

PG

Physical Granule

PoE

Point of Encryption (A point in the system where writes are encrypted using a MECID)

PoPA

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

RCRB

Root Complex Register Block

Requester

An agent in a computing system that is capable of initiating memory transactions.

Resource

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

RID

Requester ID. A unique identifier of a Requester in a PCIe Hierarchy.

RME

Realm Management Extension

RME-CDA

RME Coherent Device Assignment. An RME system feature that enables the secure assignment of fully-coherent assignable device interfaces to the Realm Security state. See RME Coherent Device Assignment.

RME-DA

RME Device Assignment. An RME system feature that enables the secure assignment of assignable device interfaces to the Realm Security state. See RME Device Assignment.

RMSD

Realm Management Security Domain. See I_{YRMHM}.

RNVS

Root non-volatile storage

RP

Root Port

SCP

System Control Processor

SMEM

Shielded memory. SMEM provides confidentiality, integrity, and replay protection against off-chip attacks.

SoC

System on Chip

SSD

System Security Domain

TDI

TEE Device Interface

TDISP

TEE Device Interface Security Protocol

TEE

Trusted Execution Environment. A term used by TDISP [4] to describe virtualization-based environments that host confidential computing workloads.

TLP

Transaction Layer Packet. A PCIe packet generated in the Transaction Layer to convey a Request or Completion.

TSM

TEE Security Manager. TDISP term for the logical entity at the host that enforces security policies.

UR

Unsupported Request. A PCIe-specified status that applies to a posted or non-posted Request that specifies some action or access to some space that is not supported by the Completer.

Write-back

The operation of writing data from a cache to memory, either due to capacity eviction, or due to an explicit instruction.