DRTM Architecture for Arm

Document number:	ARM DEN 0113
Release Quality:	EAC
Document Version:	1.1
Confidentiality:	Non-Confidential
Date of Issue:	October 8, 2024

Contents

About this document

Release Information

The change history table lists the changes that have been made to this document.

Date	Issue	Confidentiality	Change
May 30, 2023	1.0	Non-confidential	• Initial release

May 7, 2024	1.0B	Non-confidential	• Minor release
			  • Fixed error in memory region descriptor caching attribute list
			  • Clarified usage of the Region type field in memory region descriptors
			  • Clarification update to DLME data diagram
			  • Minor editorial edits

October 8, 2024	1.1		• Minor version release
			  • Further defined optional image authentication:
			  • Defined the requirements of the implementation
			  • Advertisement in DRTM_FEATURES
			  • New error code
			  • New bit in DRTM_PARAMETERS, bump revision of this data structure to 2
			  • Additional measurements and event types
			  • New DLME Authorities PCR schema
			  • Added new ARM_NO_ACTION event type
			  • DRTM_DYNAMIC_LAUNCH: clarified behavior on success
			  • DRTM_DYNAMIC_LAUNCH: explicitly which error codes to return for different error cases
			  • Clarified that errors during D-CRTM phase 1 must return errors to the caller
			  • DRTM_UNPROTECT_MEMORY: added return code for success, clarified behavior if there are back-to-back calls
			  • DRTM_SET_TCB_HASH: clarified implementation behavior for OUT_OF_RESOURCES
			  • Clarified that errors in the DRTM_PARAMETERS must be returned to the caller
			  • Clarified that functions may return other return codes in addition to those explicitly defined in the function argument table.

	• Clarified the tight coupling of a Normal world DCE to the Secure DRTM implementation • For DCE public key measurement clarified which key is to be measured • For all measurements, explicitly define which event type to use • Updated system requirement for TPM • Hardware backed DRTM: updated cases where DRTM returns errors to the caller to align with D-CRTM phase 1 in firmware-backed • Editorial edits

DRTM Architecture for Arm

Arm Non-Confidential Document License ("License")

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

References

This document refers to the following documents.

Ref	Document Number	Title
[1]	Arm DDI 0487	Arm® Architecture Reference Manual for A-profile architecture
[2]	Arm DEN 0022D	Power State Coordination Interface
[3]	Arm DEN 0028D	SMC Calling Conventions
[4]		TCG D-RTM Architecture, Version 1.0.0, June 17, 2013
[5]	Arm DEN 0072	Platform Security Boot Guide
[6]		TCG PC Client Platform Firmware Profile Specification, Family “2.0”, Level 00, Revision 1.06, December 4, 2023
[7]	Arm DEN 0094A	Arm® Base System Architecture 1.0
[8]		Trusted Platform Module Library Specification, Family “2.0”, Level 00, Revision 01.59 – November 2019
[9]		TCG PC Client Platform TPM Profile Specification for TPM 2.0, Version 1.05, September 4, 2020
[10]	Arm DEN 0044A	Arm® Base Boot Requirements 1.0
[11]		TCG Glossary, Version 1.1, Revision 1.00, May 11, 2017
[12]		TCG Algorithm Registry, Family “2.0", Level 00, Revision 01.32 June 25, 2020
[13]		Advanced Configuration and Power Interface (ACPI) Specification, Version 6.4, January 2021.
[14]	Arm IHI 0069G	Arm Generic Interrupt Controller Architecture Specification, GIC architecture version 3 and version 4

Terms and abbreviations

This document uses the following terms and abbreviations.

Term	Meaning
Application PE	The term application PE refers to a PE used by the operating system or hypervisor to execute user applications or kernel threads.
Boot PE	The boot PE is the PE designated by hardware to boot the system following reset. In this architecture it is the boot PE that initiates and performs the dynamic launch and executes the phases of DRTM. All other PEs besides the boot PE are “off” during DRTM.
CRTM	Core Root of Trust for Measurement
DCE	DRTM Configuration Environment
D-CRTM	Dynamic Core Root of Trust for Measurement
Device	A peripheral or controller that can be excluded from the TCB of the system through DMA protection hardware such as an SMMU.
DLME	Dynamically Launched Measured Environment
DRTM	Dynamic Root of Trust for Measurement
Locality	Locality is a mechanism in a TPM that supports a privilege hierarchy for clients of the TPM. The platform in a system enforces access to the TPM so that clients can only access localities they have the privilege to access.
Non-Host Platform	A peripheral or controller in a system that has no DMA-protections and cannot be restricted from reading or writing the DCE or DLME.
Non-application PE	A Non-application PE is a PE that is not an application PE.
Normal world	The Non-secure privilege levels (Non-secure EL0, EL1, and EL2) and resources, for example memory, registers, and devices, that are not part of the Secure world.
Normal world DCE	A DCE component that executes at a Non-secure privilege level.
PCR	Platform Configuration Register. A protected location within a TPM containing a digest of integrity measurements.
PE	Processing element. This is the term used for a CPU core in the Arm v8-A architecture. In this specification, unless otherwise stated, the term PE refers to an application PE.
RTM	Root of trust for measurement. The Root of Trust that makes the initial integrity measurement in a measured boot flow.
Secure world	The environment that is provided by the Secure privilege levels in the Arm v8-A architecture, S-EL0, S-EL1, S-EL2, EL3, and the resources, for example memory, registers, and devices, that are accessible exclusively from the Secure privilege levels.

SRTM	Static root of trust for measurement. From the TCG glossary [11]: An RTM where the initial integrity measurement occurs at platform reset. The SRTM is static because the PCRs associated with it cannot be re-initialized without a platform reset.
TCB	Trusted computing base
TPM	Trusted Platform Module. A security module that is defined by TCG.
TCG	Trusted Computing Group
UEFI	Unified Extensible Firmware

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 such as IMPLEMENTATION DEFINED.

Also 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, which can be:

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

Feedback

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

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

Inclusive language commitment

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

This document includes language that can be offensive. We will replace this language in a future issue of this document.

To report offensive language in this document, email terms@arm.com.

1 Overview of this document

This specification defines an architecture for Dynamic Root of Trust for Measurement (DRTM) for processors based on the Arm A-profile architecture. This specification is based on concepts from the TCG D-RTM Architecture [4], but functions as a self-contained, standalone document. It uses the principles and terminology of the TCG architecture but contains significant differences as well.

The specification is structured as follows:

- Section 2, DRTM architecture overview, describes DRTM in general and an introduction to how this architecture maps DRTM to Arm-based systems. This section includes a description of differences with the TCG-defined architecture. This section also covers the security scope of DRTM on Arm and identifies threats that are in and out of scope.
- Section 3, Interface functions and data structures, describes the Secure world ABIs needed to implement DRTM on Arm.
- Section 4, Requirements for DRTM phases, describes the normative requirements that each phase of the DRTM process must comply with.
- Section 5, System requirements, describes the system-level assumptions and requirements that must be in place for a system to support the DRTM on Arm architecture.
- Section 6, Hardware-backed implementation, describes requirements specific to a hardware-backed implementation of DRTM.

In this specification, tables with requirement IDs describe the normative requirements. These requirements are distinct from the supporting informative text. The informative text provides more context to help clarify the rationale for each requirement.

2 DRTM architecture overview

2.1 DRTM background

The Trusted Computing Base (TCB) of a system consists of all hardware, firmware, and software components that are relied on to enforce the security of the system. A compromise in any TCB component affects the security posture of the entire system.

The integrity of the software TCB for a system is typically established during boot. The boot process begins in an immutable bootloader component, for example a boot ROM that loads the first mutable firmware image. Before transferring control to the loaded image a cryptographic digest of the image is computed. This digest is known as a *measurement*. That measurement might be:

- Immediately verified using a digital signature
- Securely stored in a Trusted Platform Module (TPM) for later use in security policies

The process is repeated for each loaded image in the boot chain. Critical data can be measured as well. This process forms a chain of trust that is anchored in the immutable bootloader. The chain of trust continues through all code that is executed up to the runtime environment such as a hypervisor or OS kernel.

Measured boot is a boot chain where the measurements taken during the boot process are stored securely in a root of trust for storage such as a TPM. The code in the boot ROM that begins the chain of trust is called the Core Root of Trust for Measurement (CRTM). When the CRTM executes on a processor, it is referred to as the Root of Trust for Measurement (RTM).

In measured boot flow typical boot stages include:

- Processor execution beginning with the CRTM
- Loading and executing other bootloader stages
- Executing UEFI firmware
- Executing an OS loader
- Launching the OS kernel or hypervisor

The measurements of each stage are recorded in Platform Configuration Registers (PCRs) of a TPM. These PCRs (0-15) are referred to as *static* PCRs because they cannot be re-initialized without a platform reset. The RTM in this boot flow is called the Static Root of Trust for Measurement (SRTM). See example in Figure 1.

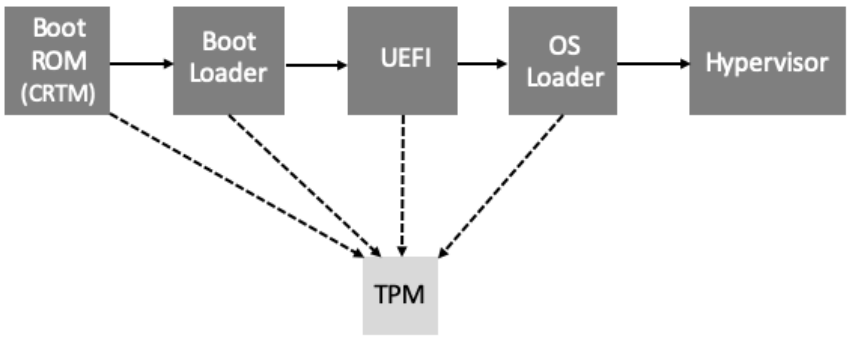


Figure 1: Example SRTM boot flow

In an SRTM boot flow every component loaded up to and including the OS kernel is measured in the static PCRs and becomes part of the system's TCB. The PCRs allow a system to attest to what software is running on the system and enables security policies such as allowing access to sealed secrets in the TPM only when the PCRs reflect that the TCB has integrity.

2.2 DRTM overview

Establishing an attestable TCB becomes difficult when the number of components in the boot chain grows or when firmware is dynamically extensible, for example by loading drivers from add-in peripherals. The larger and more complex the TCB, the greater the attack surface and the risk of untrusted code executing which can compromise security.

For example, UEFI is an extensible boot loader, where multiple EFI programs might run during the Boot Services phase. The EFI programs can include drivers, device option ROMs, and a bootloader. If compromised, these programs might be able to further compromise the target OS by tampering with the OS's code or data.

Dynamic Root of Trust for Measurement (DRTM) begins a new chain of trust by measuring and executing a protected payload. The newly started chain of trust results in a smaller TCB. DRTM is implemented by a trusted agent that ensures the following:

- All cores are placed in a known state
- The target payload is protected against modification
- A single core measures and begins running the payload
- Execution is confined to the payload
- The payload is provided with data that can be used to validate key properties of the system

When the system has been running, the same mechanism can be used again to return execution to a new measured payload without a system reset.

The event that initiates DRTM is referred to as the *Dynamic Launch Event* or DL Event.

A TPM device holds the measurements made during the dynamic launch, and these measurements are used by the launched payload to enforce a system-specific security policy. The TPM architecture defines a set of PCRs that are considered dynamic. Unlike the static PCRs used in the SRTM boot flow, the dynamic PCRs can be reset by a DL Event. This feature allows DRTM to be initiated multiple times without a system reset.

Figure 2 shows the SRTM boot chain ending with the DL Event which launches the DRTM boot chain.

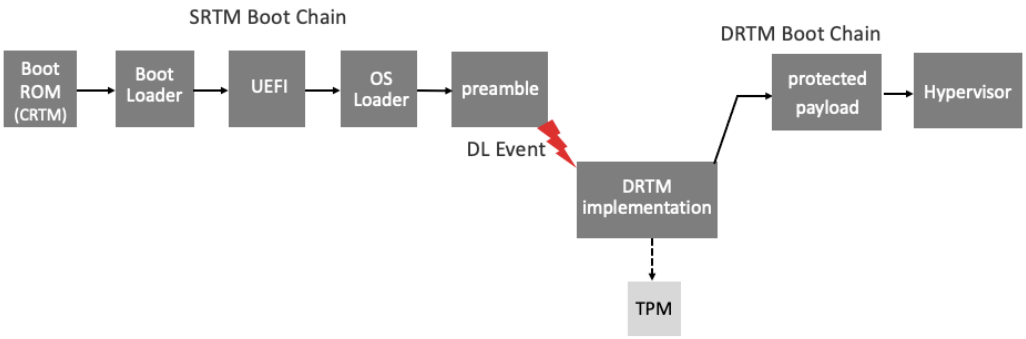


Figure 2: SRTM and DRTM boot chain illustration

A key goal of DRTM is that the new TCB established by the DRTM boot chain does not have a trust dependency on the SRTM trust chain or anything else which executed before the DL Event. A dynamic launch can be initiated any number of times on a running system.

2.3 DRTM terms

2.3.1 DCE preamble

The preamble block shown in Figure 2 is referred to as the *DRTM Configuration Environment (DCE) Preamble*. The DCE Preamble prepares the platform for DRTM by:

- Doing any needed configuration
- Loading the target payload image
- Preparing input parameters needed by DRTM
- Invoking the DL Event to start the dynamic launch

The DCE Preamble is closely coupled with the launched payload. It is typically supplied by the OS-provider or might conform to a standardized boot protocol.

2.3.2 D-CRTM and DCE

A DRTM implementation is composed of two logical components:

- The *Dynamic Core Root of Trust for Measurement* (D-CRTM)
- The *DRTM Configuration Environment* (DCE).

The D-CRTM and DCE are supplied by the silicon provider or OEM.

The D-CRTM is the trust anchor, or root of trust, for the DRTM boot sequence and is where the dynamic launch starts. The D-CRTM must be implemented in a trusted agent in the system. The D-CRTM initializes the TPM for DRTM and prepares the environment for the next stage of DRTM, the DCE. The D-CRTM:

- Measures the DCE
- Verifies its signature
- Transfers control to it

The DCE executes on an application core. The DCE:

- Verifies the state of the system
- Measures security-critical attributes of the system
- Prepares the memory region for the target payload
- Measures the payload
- Transfers control to the payload

2.3.3 DLME

The protected payload is referred to as the *Dynamically Launched Measured Environment*, or DLME. The DLME begins execution in a safe state, with a single thread of execution, DMA protections, and interrupts disabled. The DCE provides data to the DLME that it can use to verify the configuration of the system. The trustworthy

measurements made by the dynamic launch can be used to implement a system-specific security policy. The policy decides whether the system is in the expected state.

The DLME can be an operating system-specific component supplied by the same vendor that provided the DCE Preamble, or it can conform to a standardized boot protocol.

2.3.4 Devices and non-host platforms

The TCG DRTM architecture differentiates between two types of devices or controllers in a system that are capable of DMA. A *peripheral* is a device or controller that can be excluded from the TCB of the system through DMA protection hardware such as an IOMMU. A device or controller in a system that has no DMA-protections and cannot be restricted from reading or writing the DCE or DLME is referred to as a *non-host platform*. A non-host platform is trusted and is part of the system's TCB.

In this specification *device* refers to peripherals as defined in the preceding paragraph. The term non-host platform has the same usage as in the TCG specification.

2.4 DRTM on Arm

2.4.1 Overview of DRTM on Arm

Armv8-A processors have two security states: Secure and Non-Secure. Figure 3 shows the two security states.

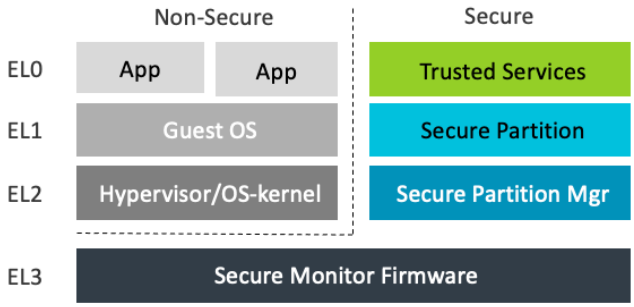


Figure 3: Exception levels and security states example

As section 2.2 describes, the objective of DRTM is to begin a new chain of trust and instantiate a smaller TCB that excludes untrusted and arbitrarily extensible components. DRTM does this by measuring and launching a protected payload.

The scope of DRTM on Arm is the Non-secure side of the processor. The launch and execution of the protected payload occurs at the highest Non-secure privilege level. The Secure privilege levels are not affected by DRTM.

A typical boot scenario for DRTM is for a Normal world OS loader to initiate a dynamic launch after loading OS images.

A DRTM implementation on Arm provides the following security guarantee:

- A trustworthy measurement is made of the DLME image and security relevant state into the TPM.
- The DLME image is launched in a safe state:
    - Single thread of execution on the boot PE. All other PEs are off
    - Asynchronous Non-secure interrupts and exceptions are disabled
    - The DLME region is protected against DMA
    - Data needed by DLME to establish the new TCB is provided. The data includes:

- A trustworthy map of the system's memory
- An event log with the measurements made during the dynamic launch
- Trustworthy ACPI tables or hashes of ACPI tables, which can be used by the DLME to validate the ones provided by Non-secure firmware

The architecture defines a new SMC function to initiate the dynamic launch. Figure 4 shows one example of how DRTM components interact in an implementation of DRTM. Section 2.5 describes possible implementation approaches.

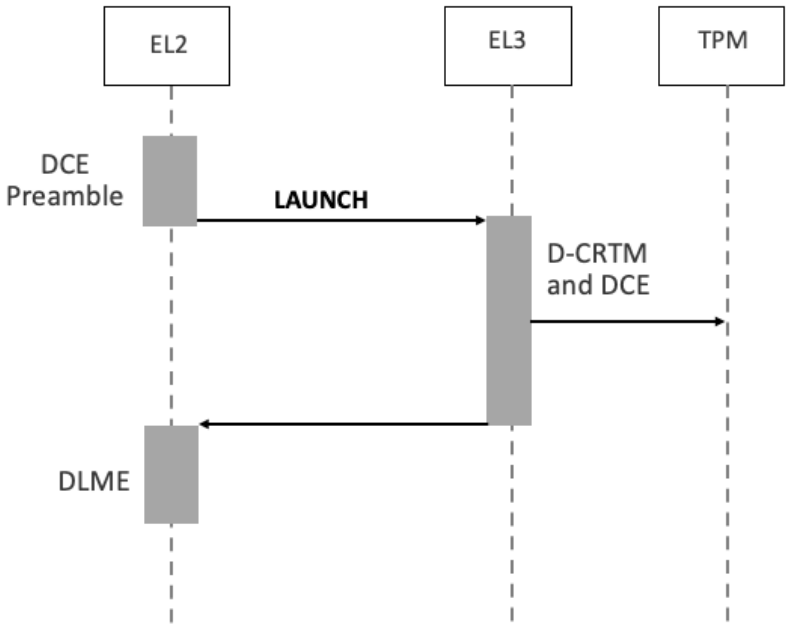


Figure 4: Example dynamic launch flow on Arm

2.4.2 DRTM SMC functions

This architecture defines a set of SMC functions which provide the interfaces needed by the Normal world DRTM client to perform initialization and initiate a DRTM dynamic launch.

Table 1 shows a summary of the DRTM SMC functions.

Table 1: DRTM functions

Function	Description
DRTM_VERSION	Returns the version of the DRTM implementation. See section 3.2.
DRTM_FEATURES	Allows a client to query the DRTM implementation to determine the supported DRTM capabilities of the platform. See section 3.3.
DRTM_DYNAMIC_LAUNCH	Initiates a dynamic launch. See section 3.4.
DRTM_UNPROTECT_MEMORY	Removes the memory protection put in place by the dynamic launch. See section 3.5.
DRTM_CLOSE_LOCALITY	Closes a locality in the TPM. See section 3.6.
DRTM_GET_ERROR	Returns an error code from the previous dynamic launch. See section 3.7.

DRTM_SET_ERROR	Sets a DRTM error code indicating that a dynamic launch has failed. See section 3.8.
DRTM_SET_TCB_HASH	Used by firmware to record hashes of components of the TCB of the system before the dynamic launch. See section 3.9.
DRTM_LOCK_TCB_HASHES	Used by firmware to prevent further hashes from being recorded. See section 3.10.

Figure 5 shows an example of how the functions might be used.

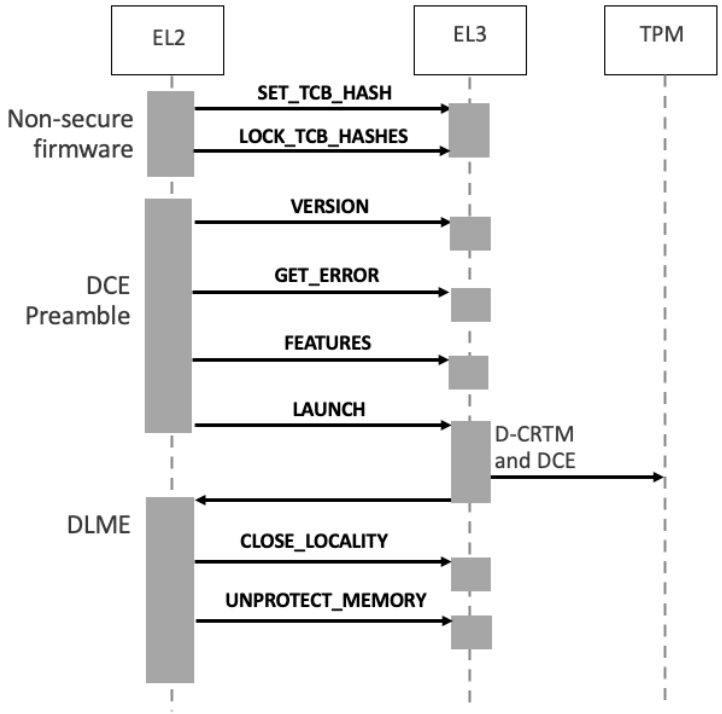


Figure 5: Example of SMC function usage

2.5 DRTM implementations

As section 2.3.2 describes, the D-CRTM must be in a trusted agent in the system. This specification describes two approaches to implementation of the D-CRTM: firmware-backed and hardware-backed.

2.5.1 Firmware-backed implementation overview

In a firmware-backed implementation the D-CRTM is implemented in secure firmware. How other DRTM components in a firmware implementation map to Arm v8 privilege levels is implementation-dependent and is not specified by this architecture. The architecture permits flexibility where the D-CRTM and DCE are implemented. Options include:

- A firmware implementation where the D-CRTM and the DCE are implemented together at EL3 as shown in Figure 4.
- A firmware implementation where the DCE is located completely in Non-secure firmware as shown in Figure 6.

- An implementation where an initial stage of the DCE is in Secure firmware and a second stage DCE is in Non-secure firmware as shown in Figure 7.

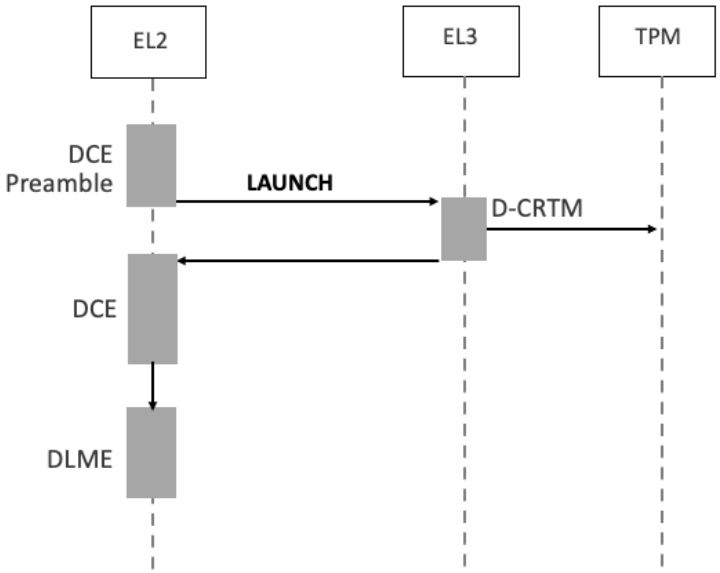


Figure 6: Firmware-backed implementation dynamic launch

2.5.2 Hardware-backed implementation overview

In a hardware-backed implementation, the D-CRTM is in a coprocessor separate from the PE that initiated the dynamic launch. A hardware-backed implementation provides a root of trust with a smaller footprint and a higher level of assurance than a firmware-backed implementation. Section 6 describes requirements for a hardware-backed implementation.

Figure 7 shows how DRTM components might map to PE privilege levels and the security coprocessor in a hardware-backed implementation.

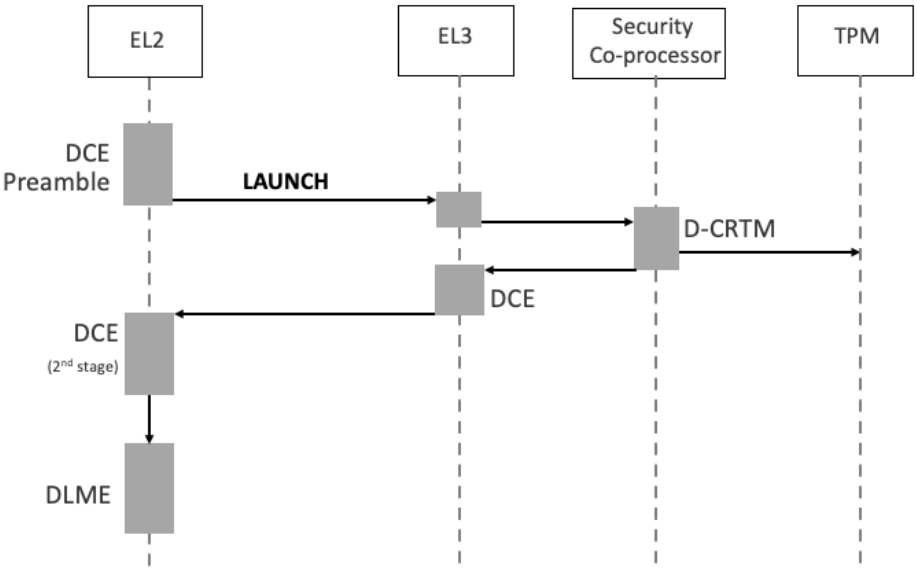


Figure 7: Hardware-backed implementation dynamic launch

2.6 Differences from the TCG DRTM specification

As the overview in section 1 describes, this specification is based on concepts from the TCG D-RTM Architecture [4]. The following are differences between the DRTM on Arm architecture compared to the TCG architecture:

- This architecture uses a different convention for PCR usage, and the terminology of PCR.Authorities, PCR.Details, and PCR.DLME.Authority defined by the TCG architecture is not used.
- This architecture supports the concept of closing localities, which is not defined by TCG.
- The concept of Sensitive Resources defined in the TCG architecture is not used, as the Arm v8-A Secure exception levels protect these kinds of resources.
- Except for the TPM event log, data structures defined by the TCG architecture are not used:
    - DRTM Resources Table and Resource Description structure. The DLME data (see section 3.12) provides equivalent functionality.
    - DRTM_PUBLIC_KEY
    - DLME_DESCRIPTOR. This structure is replaced by the DRTM parameters. See section 3.12.
    - DLME_ARGUMENTS. This structure is replaced by the DLME data. See section 3.14.
    - DLME_TRANSLATION_LIST structure.
- This architecture does not define DLME_Exit.

2.7 DRTM and the TPM

This architecture defines requirements for measurements that must be made by the DRTM implementation into a TPM during the dynamic launch. The TPM implementation must comply with the appropriate TCG specifications. See details in section 5.6. The TPM must support the dynamic localities designed for use with DRTM.

A platform must have hardware protections in place so that access to the dynamic localities of the TPM can be restricted to components with access rights to those localities. It must not be possible for software at Non-secure privilege levels to access locality 4. For a firmware-backed implementation of DRTM, a typical approach for managing the TPM is to use a Secure world TPM service where the service mediates access to the dynamic localities.

For a hardware-backed implementation, there is a requirement that platforms provide hardware enforcement of access to locality 4. This ensures that no component in the system can access locality 4 except the D-CRTM in the security coprocessor.

For a firmware-backed implementation of DRTM, you can use a firmware TPM that runs as a Secure service.

This architecture includes the concept of *closing* a dynamic locality. The dynamic localities are used in sequence by phases of DRTM: The D-CRTM uses locality 4, the DCE uses locality 3, and the DLME uses locality 2. When a given phase of DRTM is completed, access to the associated locality is closed. After a locality is closed, commands sent to the TPM through the closed locality results in an error until the next system reset or DL Event.

Section 5.6 describes the system-level requirements for a TPM implementation.

This architecture supports two approaches to the measurements made during the dynamic launch:

1) Firmware-based measurements

2) TPM-based measurements

Sections 2.7.1 to 2.7.3 provide further details.

2.7.1 Firmware-based measurements

The default approach is for the D-CRTM and DCE to implement a software-based digest algorithm for computing measurements. A measurement is computed and then extended into the TPM using the command TPM2_PCR_Extend.

The firmware-based measurement approach must be supported by a DRTM implementation.

A DRTM client can discover the firmware-based digest algorithm used through the DRTM_FEATURES function. See section 3.3.

2.7.2 TPM-based measurements

A DRTM implementation can optionally support TPM-based measurements. With TPM-based measurements, the D-CRTM and DCE make measurements using the TPM command TPM2_PCR_Event. TPM2_PCR_Event sends the data to be measured to the TPM and the digest computation is performed by the TPM. TPM2_PCR_Event extends the measurements for all digest algorithms supported by the TPM.

A DRTM client can discover whether TPM-based measurements are supported using the DRTM_FEATURES function. See section 3.3.

A DRTM client can request TPM-based measurements through the DRTM_PARAMETERS data structure passed to DRTM_DYNAMIC_LAUNCH. See section 3.12.

With the TPM-based measurements, the TPM makes the measurements with all supported hash algorithms. Measurements made with all algorithms provides an advantage because the DRTM client has maximum flexibility in choosing which digest algorithm to base a security policy on. The only limitation is the algorithms supported by the TPM.

A disadvantage of the TPM-based approach is that it can be slower than the firmware-based approach. A discrete TPM chip does not have as much processing power as an Arm v8-A PE, and the TPM must compute digests for all algorithms that it supports.

2.7.3 TPM PCR usage

This architecture can support flexibility in which TPM PCRs are used for measurements made by the D-CRTM and DCE through PCR usage schemas. The available PCR usage schemas are advertised through the DRTM_FEATURES function. See section 3.3.

When a client invokes DRTM_DYNAMIC_LAUNCH, the client requests a supported schema through a field in the DRTM_PARAMETERS. See section 3.12.

Table 2 shows a summary of the default schema. Section 4.8.2 describes the schema details.

Table 2: Default PCR usage schema

PCR	Usage
17	Measurements of DCE components, platform state, and any Secure firmware components reloaded during DRTM.
18	Measurement of the public key that signed any DCE components. Measurement of the DLME.

2.8 Memory protection

A key security requirement of this architecture is that the DLME is protected from DMA by devices during the dynamic launch. The DLME might need to measure and validate supplemental images, and these images must also be protected from DMA. The DLME and supplemental images all reside in Non-secure memory.

The DCE preamble must set up hardware-based DMA memory protections using the DRTM_PARAMETERS. See section 3.12. The architecture allows for multiple types of memory protections, and the types supported by an implementation can be discovered using the function DRTM_FEATURES. See section 3.3. The architecture currently supports the following protection types:

- *Complete DMA protection* is hardware-based enforcement at the SMMU that blocks all DMA from Non-secure devices.
- *Region-based DMA protection* is a platform-specific mechanism that provides hardware-based enforcement of DMA accesses for a number of protected memory regions. Region-based DMA protection enables a system to support active DMA during the dynamic launch process to memory that is outside the protected regions.

The DRTM implementation of DRTM_DYNAMIC_LAUNCH puts the requested memory protections in place before transitioning to the DLME. After the DLME has made any needed measurements or completed other steps to verify the state of the system, the DLME removes the memory protections using the DRTM_UNPROTECT_MEMORY function.

2.9 Security considerations

As section 2.2 describes, DRTM instantiates a smaller TCB in the Non-secure security state that excludes untrusted and arbitrarily extensible components by measuring and launching a protected payload, the DLME.

This section describes the assets that a DRTM implementation protects along with threats to those assets and corresponding mitigations. The responsibilities of key stakeholders are also identified.

The assets in scope are the DLME, the TPM, and components that comprise the DRTM implementation.

2.9.1 Security goals

The key security goal of DRTM is ensuring the integrity of the DLME and its execution environment by:

- Providing reliable measurements of the DLME and key system properties in a TPM device
- Providing a protected environment for the DLME to begin execution in and defend itself

The measurements are then used by the DLME or OS to enforce a security policy.

2.9.2 Security non-goals

There is no security goal related to confidentiality in the DRTM architecture, as DRTM does not protect or use secrets. However, DRTM depends on a TCG-compliant TPM implementation that does have confidentiality requirements. If the TPM implementation can be compromised, any DRTM-based security policy can also be compromised.

There is no security goal related to availability in the DRTM architecture. It is assumed that the adversary is in control of the component that initiates the dynamic launch and the target payload. If their goal is denial of service, the adversary can skip initiating the dynamic launch.

The following are not security goals of this architecture:

- Attacks against the availability of a system

- Defense against compromises in components within the same security boundary as the D-CRTM. In firmware-backed DRTM this includes Secure world firmware components and services, Secure devices that can be DMA-capable, and Non-host platforms. In hardware-backed DRTM this includes the security coprocessor.
- Physical attacks against the TPM, including physical man-in-the-middle attacks
- Glitches of the SoC power supply or clocks during DRTM to bypass verification checks
- Laboratory attacks in which devices are unpackaged and probed

2.9.3 Stakeholders

The following are stakeholders who each have roles in implementing the architecture:

- Silicon providers: provides silicon and potentially firmware components
- OEM or ODM: designs and manufactures the system and integrates firmware
- OSV: operating system or hypervisor provider
- End user: for example a cloud service provider that integrates components from OEM and OSV and defines a DRTM-based security policy for booting an OS

2.9.4 Threats and mitigations

Table 3 and Table 4 describe the threats in scope for this architecture, including mitigations addressed by the architecture.

Table 3: Threats in scope

Threat	Mitigation
Adversary compromises Non-secure firmware, which enables them to tamper at boot time with Normal world DCE, DLME, or the DLME's supplemental images.	The DCE measures the DLME and Normal world DCE into the TPM, allowing detection of tampering.
Adversary compromises Non-secure firmware, which enables them to tamper with security critical ACPI tables.	DRTM provides trustworthy ACPI tables or trustworthy hashes of ACPI tables in the data passed to the DLME.
Adversary compromises Non-secure firmware and modifies the UEFI system memory map and reports an attacker controlled memory-mapped I/O region to be "normal memory". This allows the attacker to spoof responses to any access made to this region.	The DCE provides a trustworthy memory map to the DLME, allowing it to detect any discrepancies in the firmware provided map.
Adversary compromises a Non-secure device and during the dynamic launch uses DMA to tamper with the Normal world DCE, DLME, or the DLME's supplemental images.	There is a system requirement that all DMA-capable devices must be behind an SMMU. The DRTM implementation verifies that DMA protections for Normal world DRTM components are in place.
Adversary uses a secondary PE to tamper with Normal world DCE, DLME, or the DLME's supplemental images.	The DRTM implementation enforces that all PEs except the boot PE are off.

Adversary gains access to locality 4 in the TPM allowing them to reset dynamic PCRs and control the PCR values. This enables them to potentially unseal secrets in the TPM or attest that the system has integrity.	The system must support hardware or Secure firmware-based enforcement to mediate and control access to the dynamic localities (localities 1-4) of the TPM.
Adversary triggers a Non-secure exception, causing an adversary-controlled ISR to run during a dynamic launch. This allows tampering with the Normal world DCE, DLME, or the DLME's supplemental images from the ISR.	The DRTM implementation must ensure that DRTM operation cannot be preempted by Non-secure asynchronous exceptions. Examples include interrupts, SError exceptions, and SDEI events.
Adversary tampers with the DRTM parameters before DL Event.	The DCE verifies the DRTM parameters before use to ensure basic validity. Parameters tampered with might still be valid values, and in this case TPM measurements made during a dynamic launch enables detection of the tampering. For example, if the address or size of the DLME image was tampered with, the measurement made of the DLME enables detection of the tampering. The DLME entry point in the DRTM parameters has a separate measurement in the TPM, allowing detection of tampering.
On an SoC with a GIC ITS, adversary abuses GIC configuration to trigger an LPI interrupt during a dynamic launch. This causes the GIC to write data into the DLME region, bypassing SMMU controls.	The DRTM implementation must disable all ITSs and ensure LPIs are disabled during the dynamic launch.

Table 4 shows the threats in scope for hardware-backed implementations, but out of scope for firmware-backed implementations:

Table 4: Threats in scope for hardware-backed implementations

Threat	**Mitigation**
Adversary tampers with Secure or Non-secure DRTM components before they are executed.	The D-CRTM in the security coprocessor verifies the signature of the initial DCE image and measures it into the TPM, allowing detection of tampering. All code executed from the D-CRTM until reaching the DLME must be verified using digital signatures.
Adversary tampers with Secure world firmware images.	Secure world firmware is reloaded, verified, and measured into the TPM, allowing detection of tampering.
Adversary compromises Secure world firmware and spoofs a dynamic launch by resetting dynamic PCRs and replaying measurements into PCRs.	Hardware protections prevent locality 4 from being accessed by any component except the D-CRTM in the security coprocessor.

Adversary compromises a Secure device and during the dynamic launch uses DMA to tamper with DRTM components.	The DRTM implementation puts DMA protections in place so that DCE components and the DLME region cannot be tampered with during the dynamic launch.
An attacker triggers a Secure device interrupt during the dynamic launch and the ISR tampers with the dynamic launch.	The DRTM implementation either: • Masks Secure asynchronous interrupts • Aborts DRTM if an interrupt occurs
An attacker uses a JTAG probe to tamper with DRTM components during the dynamic launch.	The D-CRTM detects and measures if external debug can be enabled without a reset.

2.9.5 Security considerations for stakeholders

This section identifies security considerations that must be evaluated by the stakeholders who implement the DRTM architecture and the underlying platform. These responsibilities include ensuring that the underlying system meets the requirements defined in this specification and following best practices for threats not directly addressed by DRTM.

This architecture does not dictate a specific split of responsibility among the stakeholders that implement the DRTM architecture.

Security Consideration	**Stakeholder**
This architecture depends on a set of system requirements being met which provide the foundation for DRTM. These requirements are specified in section 5.	Silicon Provider, OEM/ODM
The security of components that comprise the system's TCB such as Non-host platforms or Secure services must be evaluated. Is there an attack surface that allows an adversary at a Non-secure or Secure privilege level to cause the component to perform a memory access that can compromise DRTM?	Silicon Provider, OEM/ODM
For hardware-backed DRTM, Secure components that comprise the system's TCB must be verified during the dynamic launch. Also, any security-critical state in the system's TCB that is preserved across the dynamic launch must be verified. Threat modeling is required to determine what state is security-critical.	Silicon Provider, OEM/ODM
This architecture requires a TCG-compliant TPM. A firmware TPM is an allowed implementation. It is beyond the scope of this specification to define the security properties for a firmware TPM implementation or for a hardware enclave-based implementation. The TPM implementation is a critical component of the system's TCB and system level threat modeling is needed to evaluate possible threats against a firmware or hardware enclave-based TPM.	Silicon Provider, OEM/ODM
There are general threats against the DRTM SMC call interface that are not specific to the DRTM architecture that must be considered by the implementer of DRTM firmware. These include: • Buffer overflows of privileged buffers	Silicon Provider, OEM/ODM

• Access control bypasses (for example, to achieve lateral or privilege escalation) The SMC Calling Conventions document [3] provides recommendations to mitigate these types of threats and should be followed by the firmware implementer and the software client invoking SMC calls.	
The security of a DRTM implementation ultimately relies on the implementation of a TPM-based security policy based on the measurements made by DRTM. This policy enforcement can be in the DLME or in the OS. For example, a flaw in how attestation is performed or how a secret is sealed to dynamic PCRs can compromise the security goals of the end user.	End User
The DRTM implementation should follow best practices to mitigate against software-induced fault-injection attacks, for example: • Rowhammer family of attacks against DRAM modules • Faults induced through power control APIs (for example, CLKSCREW vulnerability)	Silicon Provider, OEM/ODM
The dynamic launch starts the execution of the DLME in a safe state where it can defend itself and validate security-critical properties of the system. Section 4.6.2 describes steps that are recommended to be taken by the DLME. The implementer of the DLME should perform a security analysis to determine what steps are necessary to initialize and transition to the OS runtime.	OSV
The DLME might depend on ACPI tables provided by Non-secure firmware that impact the TCB of the system. This architecture defines how the DLME is provided with copies of trustworthy ACPI tables or the means to validate ACPI tables provided by Non-secure firmware. It is the responsibility of the platform manufacturer to determine which ACPI tables might impact the TCB of the system and provide hashes of these tables to the DRTM implementation.	Silicon Provider, OEM/ODM

3 Interface functions and data structures

3.1 Introduction to interface functions and data structures

The DRTM functions are invoked by using SMC calls. The functions adhere to the SMC Calling Conventions [3] and in particular, the register usage follows the specification for SMC64 calls.

The SMC Calling Conventions specification requires that all unimplemented functions return an Unknown SMC Function Identifier which maps to the NOT_SUPPORTED error code. This specification follows this convention. Therefore, a NOT_SUPPORTED error code indicates the firmware implementation does not support DRTM.

Unless otherwise specified, all in-memory data structures defined in this specification are in little-endian format.

Table 5: Interface function requirements

ID	Requirement
R31000	A DRTM instance must implement the following functions: DRTM_VERSION DRTM_FEATURES DRTM_DYNAMIC_LAUNCH DRTM_UNPROTECT_MEMORY DRTM_CLOSE_LOCALITY DRTM_GET_ERROR DRTM_SET_ERROR
R31010	If the DRTM_SET_TCB_HASH function is implemented, the DRTM_LOCK_TCB_HASHES function must be implemented.

3.2 DRTM_VERSION

Returns the version of the DRTM implementation.

Parameter	Register	Value	
uint32 Function ID	W0	0xC400_0110	
Return			
uint32	W0	On success **Bit[31]** Must be zero. **Bits[30:16]** DRTM Major version Must be 1 for this revision of the DRTM architecture for Arm. **Bits[15:0]** DRTM Minor version Must be 1 for this revision of the DRTM architecture for Arm.	
		On error: `NOT_SUPPORTED`	DRTM is not supported.

3.2.1 DRTM_VERSION usage

This function returns the version of the DRTM implementation. Each DRTM implementation must support this call and return its implementation version. For this revision of the DRTM interface, the major version is 1 and the minor version is 1.

The version number is a 31-bit unsigned integer. The upper 15 bits denote the major revision, and the lower 16 bits denote the minor revision. The following rules apply to the version numbering:

- Different major revision values indicate possibly incompatible functions. A newer major revision might:
  - Introduce new functions
  - Deprecate older functions
  - Change behavior of existing functions
- For two revisions, A and B, for which the major revision values are identical, if the minor revision value of revision B is greater than the minor revision value of revision A, then every function in revision A must work in a compatible way with revision B. However, revision B can have a higher function count than revision A.

3.2.2 DRTM_VERSION implementation responsibilities

If this function returns a valid version number, all DRTM functions defined in this specification must be implemented, unless it is explicitly stated that a function is optional.

3.3 DRTM_FEATURES

Allows a client to query the DRTM implementation to determine the supported DRTM capabilities of the platform.

Parameter	Register	Value	
uint32 Function ID	W0	0xC400_0111	
uint64 DRTM function ID or feature ID	X1	Bit[63] = 0: Parameter is function ID of the DRTM interface whose implementation must be queried. Bit[62:32]: Reserved. Must be zero. Bit[31:0]: Function ID. Bit[63] = 1: Parameter is ID of a feature supported by the DRTM implementation. Bit[62:8]: Reserved. Must be zero. Bit[7:0]: Feature ID. IDs of supported features are listed in Table 6.	
Return			
int64	X0	On success:	
		0	The function ID queried is implemented.
		> 0	The feature ID queried is implemented. See Table 6 for feature-specific capabilities.
		On error:	
		`NOT_SUPPORTED`	The function ID or feature ID specified as an input parameter is not supported.
			Note: This function may return other error codes as defined in section 3.17.

Table 6: Return values for DRTM features

DRTM Feature Name	Feature ID	Return Register	Return Value
TPM features	0x1	X1	**Bits [63:37]:** Reserved. Must be zero. **Bits [36:33]:** PCR Usage Schema. Bitmap of supported usage schemas: 0b0001: Default Schema. See section 4.8.3. 0b0010: DLME Authorities Schema. See section 4.8.4. 0b0100: Reserved 0b1000: Reserved **Bit 32:** TPM-based hash support 0: Implementation does not support TPM-based hashing. 1: Implementation supports TPM-based hashing. See section 2.7.2. The DRTM implementation measures data by sending it to the TPM using the command TPM2_PCR_Event. **Bits [31:16]** Reserved **Bits [15:0]** Firmware hash algorithm Value Identifies a single hash algorithm used by the DRTM implementation for firmware-based measurements. See section 2.7.1. Possible values are defined by the TPM_ALG_ID constants defined in the TCG Algorithm Registry [12]. A partial list is shown below. 0x000B: SHA-256 0x000C: SHA-384 0x000D: SHA-512

Minimum memory requirement	0x2	X1	**Bits [63:32]** Minimum size of Normal World DCE If the system utilizes a Normal World DCE, the value specifies the minimum amount of space needed for the Normal World DCE region. The value is specified in number of 4KB pages. If the system does not use a Normal World DCE, this value must be zero. **Bits [31:0]** Minimum size of DLME data Value specifies the minimum amount of space that the DRTM implementation needs for the DLME data. The value is specified in number of 4KB pages.
DMA protection features	0x3	X1	**Bits [63:24]** Reserved. Must be zero. **Bits [23:8]** For region-based protection, specifies the maximum number of memory regions that can be protected through the DRTM_PARAMETERS. Up to 64K regions can be supported. This field is only valid if region-based DMA protection is supported. A value of 0 is valid, which means that no regions can be specified in DRTM parameter, but the DRTM implementation implicitly protects one or more regions. **Bits [7:0]** DMA protection support. Bitmap with the types of DMA protection supported by the DRTM implementation: 0b00000001: Complete DMA protection. 0b00000010: Region-based DMA protection. For further details on DMA protection, see section 5.8.
Boot PE ID	0x4	X1	**Bits [63:0]** Identifies the boot PE. The encoding is the same as the *target_cpu* parameter to CPU_ON as defined in the PSCI specification [2]. The PSCI encoding is based on the affinity fields of the MPIDR register.

TCB hash features	0x5	X1	**Bits [63:8]** Reserved. Must be zero. **Bits [7:0]** Maximum number of supported TCB hashes that can be recorded with DRTM_SET_TCB_HASH. See section 3.9. A value of zero indicates that hashes cannot be recorded with DRTM_SET_TCB_HASH.
DLME image authentication features	0x6	X1	**Bits [63:1]** Reserved. Must be zero. **Bit 0** DLME image authentication support 0: Implementation does not support DLME image authentication. 1: Implementation supports DLME image authentication.

3.3.1 DRTM_FEATURES usage

This function can be used to query the capabilities of the DRTM implementation. The caller passes a DRTM function ID or feature ID and the return value advertises features and capabilities.

A return value of 0 means the function ID or feature ID is implemented.

A return value of greater than 0 means the feature ID is implemented, and there are other feature-specific feature or capability bits available in other return value registers.

See Table 6 for supported feature IDs and features.

See section 2.8 for an overview of memory protection.

See section 2.7 for an overview of PCR usage and TPM-based and firmware-based measurements.

3.4 DRTM_DYNAMIC_LAUNCH

Initiates a DRTM dynamic launch.

Parameter	Register	Value	
uint32 Function ID	W0	0xC400_0114	
uint64 DRTM parameters	X1	64-bit physical address of the DRTM_PARAMETERS.	
Return			
int64	X0	On success:	On success, this function does not return to the caller. After a successful dynamic launch, control is transferred to the DLME image entry point. See section 4.6.1 for a description of the initial state of the DLME.
		On error:	
		`NOT_SUPPORTED`	DRTM is not supported.
		`INVALID_PARAMETERS`	The address of the DRTM parameters was invalid, or the contents of the DRTM parameters were invalid.
		`DENIED`	The system is not in a state where a dynamic launch can be initiated.
		`MEM_PROTECT_INVALID`	Some or all memory protection requests passed in the DRTM_PARAMETERS are invalid.
		`SECONDARY_PE_NOT_OFF`	Secondary PEs were detected to be on.
			Note: This function may return other error codes as defined in section 3.17.

3.4.1 DRTM_DYNAMIC_LAUNCH usage

This function initiates the dynamic launch process. It takes as a parameter the physical address of the DRTM_PARAMETERS structure. See section 3.12.

The function returns an error if the parameters are invalid or if the platform is incapable of initiating the launch in its current state. Otherwise, the dynamic launch is performed and this function never returns. When the launch is complete, execution begins at the DLME entry point.

The DRTM_FEATURES function can be used to query features implemented by the DRTM implementation, including:

- Hash algorithm used by the implementation
- Whether the implementation supports TPM-based measurements
- The minimum size of the DLME region and the size of the Normal world DCE region
- The type of memory protection supported by the implementation

3.4.2 DRTM_DYNAMIC_LAUNCH caller responsibilities

Prior to invoking DRTM_DYNAMIC_LAUNCH the caller must prepare the system state as described in the requirements in sections 4.1 and 4.2. This includes:

- Halting all PEs except the boot PE
- Preparing and initializing a DLME region
- Defining a memory protection table
- Initializing the DRTM_PARAMETERS

3.4.3 DRTM_DYNAMIC_LAUNCH implementation responsibilities

The implementation carries out the DRTM dynamic launch and executes the D-CRTM followed by executing the DCE. The dynamic launch is complete when control is transferred to the DLME entry point.

The following are responsibilities of the implementation:

- Implement the requirements specified in section 4.3, *Dynamic launch event*.
- Implement the requirements specified in section 4.4, *Firmware-backed D-CRTM* requirements.
- Implement the requirements specified in section 4.5, *DCE requirements*.

3.5 DRTM_UNPROTECT_MEMORY

Removes the memory protection requested by the DRTM_PARAMETERS and put in place during the dynamic launch.

Parameter	Register	Value	
uint32 Function ID	W0	0xC400_0113	
Return			
int64	X0	On success: SUCCESS	The operation succeeded.
		On error:	
		NOT_SUPPORTED	DRTM is not supported.
		DENIED	The system is not in a state where memory can be unprotected.
			Note: This function may return other error codes as defined in section 3.17.

3.5.1 DRTM_UNPROTECT_MEMORY usage

This function is called by the DLME to remove the DMA protections put in place during the dynamic launch. The function takes no arguments.

The DLME calls this function when it has done sufficient verification and initialization so that it can defend itself from DMA attacks.

3.5.2 DRTM_UNPROTECT_MEMORY implementation responsibilities

The following are responsibilities of the implementation:

- If the caller invokes DRTM_UNPROTECT_MEMORY and there are no memory protections in place, the implementation must return the error value DENIED. If the caller invokes DRTM_UNPROTECT_MEMORY multiple times, the first instance would remove the memory protections and subsequence calls would return DENIED.
- For complete DMA protection the implementation must not make any changes to the SMMU. The function must be a NOP with respect to the SMMU.
- For region-based DMA protection the implementation must remove the memory protections requested by DRTM_PARAMETERS and put in place during the dynamic launch.

3.6 DRTM_CLOSE_LOCALITY

Closes a locality in the TPM.

Parameter	Register	Value	
uint32 Function ID	W0	0xC400_0115	
uint32 Locality	X1	Specifies the locality to close. Valid values are 2 and 3.	
Return			
int64	X0	On success: `SUCCESS`	The operation succeeded.
		On error: `NOT_SUPPORTED`	DRTM is not supported.
		`INVALID_PARAMETERS`	The locality specified is not valid.
		`ALREADY_CLOSED`	The locality is already closed.
		`DENIED`	The locality has not been relinquished.
			Note: This function may return other error codes as defined in section 3.17.

3.6.1 DRTM_CLOSE_LOCALITY usage

This function is used by the DLME to close dynamic localities in the TPM. Localities that can be closed with this function are 2 and 3.

Locality 2 is a dynamic locality used by the DLME to make any needed measurements. After the DLME has completed its measurements, it invokes DRTM_CLOSE_LOCALITY to prevent further measurements from being made.

Locality 3 is a dynamic locality used by the DCE to make any needed measurements. A Normal world DCE component that runs at a Non-secure exception level can use this function to close locality 3 after all needed measurements are made.

Note: if there is only a single stage DCE, the DCE executes in the Secure world and must close locality 3 when it finishes making measurements.

3.6.2 DRTM_CLOSE_LOCALITY caller responsibilities

The caller must relinquish the locality it is trying to close at the TPM before invoking this function. Failure to do this results in an error.

3.6.3 DRTM_CLOSE_LOCALITY implementation responsibilities

The implementation must interface to the TPM implementation in a platform-specific manner to close the target locality. If the locality is already closed or has not been relinquished the implementation must return an error to the caller.

3.7 DRTM_GET_ERROR

Returns an error code from the previous DRTM dynamic launch.

Parameter	Register	Value	
uint32 Function ID	W0	0xC400_0116	
Return			
int64	X0	On success: `SUCCESS`	Error code from previous launch is returned in X1.
		On error: `NOT_SUPPORTED`	DRTM is not supported.
		`NOT_FOUND`	An error occurred, and the error code was not found.
			Note: This function may return other error codes as defined in section 3.17.
int64	X1	If X0 == `SUCCESS`	Error code. See error encoding in section 3.11

3.7.1 DRTM_GET_ERROR usage

This function can be called by a DRTM client to determine if a previous invocation of DRTM_DYNAMIC_LAUNCH failed and entered remediation. See section 4.7. If a previous invocation of DRTM_DYNAMIC_LAUNCH had not occurred, then the value returned by DRTM_GET_ERROR is undefined.

An error code value of 0 indicates that the previous DRTM_LAUNCH did not enter remediation.

3.7.2 DRTM_GET_ERROR caller responsibilities

The DRTM_GET_ERROR function can be invoked in the DCE Preamble before the dynamic launch to determine the error code from a previous dynamic launch. If a previous dynamic launch has not occurred, then DRTM_GET_ERROR should not be invoked, as it will return an undefined value. The caller must determine using implementation-specific means whether a dynamic launch had been previously initiated.

An error code value of zero indicates that no error occurred.

3.7.3 DRTM_GET_ERROR implementation responsibilities

If an error is detected during a dynamic launch and the DRTM implementation entered remediation (see section 4.7), the implementation must record an error code in a location that is preserved across the remediation-initiated reset. How the error code is recorded is implementation-dependent. The implementation of DRTM_GET_ERROR must return the recorded error code.

Calling the DRTM_GET_ERROR function does not clear the error. Multiple calls to DRTM_GET_ERROR returns the same error code value.

The error code must be cleared by the DCE before transfer of control to the DLME.

3.8 DRTM_SET_ERROR

Sets a DRTM error code indicating that a dynamic launch has failed.

Parameter	Register	Value	
uint32 Function ID	W0	0xC400_0117	
int64 Error Code	X1	64-bit error code describing the reason for a dynamic launch failure. See error encoding in section 3.11	
Return			
int64	X0	On success: `SUCCESS`	The set error operation succeeded.
		On error: `NOT_SUPPORTED`	DRTM is not supported.
		`DENIED`	An error code has been previously set.
			Note: This function may return other error codes as defined in section 3.17.

3.8.1 DRTM_SET_ERROR usage

This function can be called by a Normal world DCE component or the DLME to report that an error was detected. This function can only be called once. A second or subsequent invocation results in an error with a return value of DENIED.

3.8.2 DRTM_SET_ERROR caller responsibilities

The DRTM_SET_ERROR function must only be called at most once and must be called either during the DCE stage of DRTM or by the DLME.

Following the invocation of DRTM_SET_ERROR, the system must be reset to complete remediation.

3.8.3 DRTM_SET_ERROR implementation responsibilities

The implementation must persist the DRTM error code in a non-volatile location not accessible by the Normal world so that it can be returned to a client using the DRTM_GET_ERROR function following the system reset.

3.9 DRTM_SET_TCB_HASH

Used by firmware to record hashes of components of the TCB of the system.

Parameter	Register	Value	
uint32 Function ID	W0	0xC400_0118	
uint64 TCB Hash Table Address	X1	64-bit physical address of a TCB hash table.	
Return			
int64	X0	On success: `SUCCESS`	The operation succeeded. See supplemental value in X1.
		On error: `NOT_SUPPORTED`	DRTM is not supported.
		`INVALID_PARAMETERS`	The address in the parameter was invalid or there was an error in the header of the TCB_HASH_TABLE.
		`INVALID_DATA`	An entry in the TCB_HASH_TABLE contains an error. See supplemental value in X1.
		`OUT_OF_RESOURCE`	The maximum number of hashes has been exceeded. See maximum advertised by DRTM_FEATURES.
		`DENIED`	The implementation is locked against recording further hashes.
			Note: This function may return other error codes as defined in section 3.17.
int64	X1	If X0 == `SUCCESS`	Specifies how many valid entries are populated in the table.
		If X0 == `INVALID_DATA`	Indicates which table entry contains the error.

3.9.1 DRTM_SET_TCB_HASH usage

This function can be called by Normal world firmware before the dynamic launch to record hashes of components belonging to the TCB, such as ACPI tables. These hashes are made available to the DLME in the DLME data.

The function takes as a parameter the address of a TCB_HASH_TABLE containing one or more hashes.

3.9.2 DRTM_SET_TCB_HASH caller responsibilities

The hashes recorded by this function are made available to the DLME which uses them to validate the TCB of the system, and so it is critical that firmware invoking this function be in an execution phase where all firmware components are under the authority of the platform manufacturer. For UEFI-based firmware this is the phase of execution before the End of DXE event.

The following are responsibilities of the caller:

- The caller must compute TCB hashes using the hash algorithm in use by the DRTM implementation when it performs firmware-based measurements (see 2.7.1). The caller determines the hash algorithm using the DRTM_FEATURES function.
- The caller constructs a TCB_HASH_TABLE which consists of entries defining each hash. For each TCB component to be measured, the caller must compute the digest of the component and record the digest value in the TCB_HASH_TABLE.
- The caller should not set the Source of Entry field in a table entry, because it will be ignored.
- DRTM_SET_TCB_HASH can be invoked multiple times, for example once for each component being measured. Alternatively, a single TCB_HASH_TABLE can be constructed containing all TCB hashes with DRTM_SET_TCB_HASH being invoked once.

3.9.3 DRTM_SET_TCB_HASH implementation responsibilities

The following are responsibilities of the implementation:

- The implementation must verify the validity of the referenced TCB_HASH_TABLE based on the requirements in Table 16 and Table 17. If there are any errors in TCB_HASH_TABLE, the return value INVALID_PARAMETERS must be returned.
- The maximum number of entries in the hash table is implementation-specific. The maximum is advertised through DRTM_FEATURES. If this maximum is exceeded, the implementation must return the return value OUT_OF_RESOURCE and the TCB hash table must be left as it was before the invocation of this function.
- The implementation must perform error checking on the TCB_HASH_TABLE passed to this function before recording any of the hashes. If there are any errors, the TCB hash table must be left as it was before the invocation of this function.
- The implementation must ignore the Source of Entry if set in any table entry.
- The implementation must preserve the measurements made with this function so that a single list of measurements can later be presented in the DLME data during the dynamic launch.
- After the function DRTM_LOCK_TCB_HASHES is invoked, the set of hashes is considered locked and subsequent calls to this function must result in the error code DENIED.

3.10 DRTM_LOCK_TCB_HASHES

Used by firmware to prevent further hashes from being recorded.

Parameter	Register	Value	
uint32 Function ID	W0	0xC400_0119	
Return			
int64	X0	On success: `SUCCESS`	The operation succeeded.
		On error: `NOT_SUPPORTED`	DRTM is not supported.
		`DENIED`	Hashes are already locked.
			Note: This function may return other error codes as defined in section 3.17.

3.10.1 DRTM_LOCK_TCB_HASHES usage

After recording TCB hashes using DRTM_SET_TCB_HASH, firmware can invoke this function to prevent further hashes from being recorded.

3.10.2 DRTM_LOCK_TCB_HASHES caller responsibilities

As section 3.9 describes, the function DRTM_SET_TCB_HASH can be used to record hashes of TCB components. These measurements must be performed when Non-secure firmware is in an execution phase where all components are under the authority of the platform manufacturer. For UEFI-based firmware this is the phase of execution before the End of DXE event. Before transitioning to the next execution phase, firmware uses DRTM_LOCK_TCB_HASHES to prevent further hashes from being recorded.

3.10.3 DRTM_LOCK_TCB_HASHES implementation responsibilities

This function must be implemented if DRTM_SET_TCB_HASH is implemented.

The implementation must record state to indicate that TCB measurement are locked. The implementation of DRTM_SET_TCB_HASH uses this state data to prevent further measurements from being made. If the TCB hashes are already locked, the error DENIED must be returned.

3.11 DRTM error encoding

Table 7: DRTM error encoding

Value
Bits [63:11] Error code specific data. Each error code might define more fields specific to that error code. **Bits [10:3]** Error code. See Table 8. **Bits [2:0]** DRTM phase when error occurred. 0: No error occurred 1: D-CRTM on PE 2: D-CRTM on coprocessor 3: DCE 4: Normal world DCE 5: DLME 6-7: reserved

Table 8: error codes

Description	Value
No error occurred	0x0
Reserved	0x1
Error in DRTM_PARAMETERS	0x2
Secure exception occurred during DRTM	0x3
TCB hashes were recorded but were not locked	0x4
An error occurred at the TPM	0x5
An error occurred verifying a digital signature	0x6
The DLME image format was not recognized	0x7
Reserved	0x8 – 0xFE
Vendor-specific error	0xFF

3.12 DRTM_PARAMETERS

The DRTM_PARAMETERS are created by the DCE preamble and are the input parameters to the DRTM dynamic launch. The parameters describe dynamic launch features, the DLME region (see section 3.14), and any Normal world DCE image. The existence of a Normal world DCE image is implementation-dependent.

Table 9: DRTM_PARAMETERS definition

Field	Byte Offset	Byte Length	Description
Revision	0	2	Revision of this data structure. Must have a value of 2 for this revision of DRTM architecture for Arm. For a given DRTM major version number (see DRTM_VERSION, 3.2), this structure is always extended in a backwards compatible manner.
Reserved	2	2	Must be zero.
Launch Features	4	4	Selects optional features, if supported by the implementation: Bits[31:7]: Reserved. Must be zero. Bit[6]: DLME image authentication 0: No DLME image authentication requested. 1: DLME image authentication requested. Bits[5:3]: Memory protection type. 0: *All*. Complete DMA protection requested. 1: *Region*. Region-based DMA protection requested. 2-7: Reserved Bits[2:1]: Requested PCR Usage Schema. 0: Default Schema (see section 4.8.3) 1: DLME Authorities Schema (see section 4.8.4) 2-3: Reserved Bit[0]: Type of hashing 0: Request firmware-based hashing (see section 2.7.1). 1: Request TPM-based hashing (see section 2.7.2).
DLME region address	8	8	Starting physical address of the DLME region.
DLME region size	16	8	Size of the DLME region in bytes.

DLME image start	24	8	Offset in bytes to the start of the DLME image within the DLME region. See Figure 8.
DLME entry point offset	32	8	Offset in bytes from the start of the DLME image to the DLME image entry point.
DLME image size	40	8	Size of the DLME executable image in bytes.
DLME data offset	48	8	Offset in bytes to the DLME data within the DLME region. See Figure 8.
Normal world DCE region address	56	8	Physical address of the Normal world DCE region. The address must be 4KB aligned. Must be zero if a Normal world DCE is not in use.
Normal world DCE region size	64	8	Size of the Normal world DCE region in bytes. Must be zero if a Normal world DCE is not in use.
Memory protection table address	72	8	For a memory protection type value of *Region* in the Launch Features, this field specifies the 64-bit physical address of the memory protection table describing all memory regions to be protected. The address must be 4KB aligned. Must be zero if region-based DMA protection is not used.
Memory protection table size	80	8	For a memory protection type value of *Region* in the Launch Features, this field specifies the size of the memory protection table in bytes. Must be zero if region-based DMA protection is not used.

Table 10 describes the requirements for the DRTM_PARAMETERS:

Table 10: DRTM_PARAMETERS requirements

ID	Requirement
R312000	The DRTM_PARAMETERS must be in Non-secure, physically contiguous memory.
R312010	The DRTM_PARAMETERS must start at a 4KB aligned address.
R312020	The address ranges described by the parameters must not overlap.
R312030	The address ranges described by the parameters must not wrap around.
R312040	The DLME image size must not extend beyond the bounds of the DLME region.

R312050	The DLME entry point offset must be within the bounds of the DLME image.
R312060	The DLME region must meet the requirements in section 3.14.
R312070	If in use, the Normal world DCE region must be in Non-secure physically contiguous memory.
R312080	If in use, the Normal world DCE region must start at a 4KB aligned address.
R312090	It is an error to request the DLME Authorities Schema unless DLME image authentication is also requested.

3.13 MEMORY_REGION_DESCRIPTOR_TABLE

The MEMORY_REGION_DESCRIPTOR_TABLE is a data structure that describes regions of physical memory. The table header defines the number of regions in the table. Each region is described by its physical address, size, and type.

Table 11: MEMORY_REGION_DESCRIPTOR_TABLE definition

Field	Byte Offset	Byte Length	Description
Revision	0	2	Revision of this data structure. Must be the value 1 for this version of the DRTM architecture. For a given DRTM major version (see DRTM_VERSION, 3.2) this structure is always extended in a backwards compatible manner.
Reserved	2	2	Must be zero.
Number of regions	4	4	Specifies the number of regions described in this data structure.
Region 0 address	8	8	Starting physical address memory region 0
Region 0 size/type	16	8	Bits [63:57]: Reserved. Must be zero. Bits[56-55]: Cacheability attributes. Only applicable for region type 1. Attributes and MAIR attribute encoding Attr<n> [7:4] [3:0] are shown below: 0 – Not cacheable: 0000 0000 1 – Write combine: 0100 0100 2 – Write through: 1011 1011 3 – Write back: 1111 1111 Bits[54:52]: Region type 0 – normal, usable memory 1 – normal, usable memory with cacheability attribute requirements. See bits 56-55.

			2 – device memory, memory-mapped I/O 3 – non-volatile memory 4 – the region is reserved, and cannot be used 5 – Reserved 6 – Reserved 7 – Reserved Bits[51:0]: Number of 4KB pages in region 0
Region 1 address	24	8	Starting physical address memory region 1
Region 1 size/type	32	8	Type and number of 4KB pages in region 1. See definition for region 0.
Region N address	(16*N)+8	8	Starting physical address memory region N
Region N size/type	(16*N)+16	8	Type and number of 4KB pages in region N. See definition for region 0.

Table 12 describes the requirements for a MEMORY_REGION_DESCRIPTOR_TABLE:

Table 12: Memory region descriptor requirements

ID	Requirement
R313000	The descriptor table must be in physically contiguous memory.
R313010	All region addresses in the table must be 4KB aligned.
R313020	Regions described by the table must not overlap.

3.14 DLME region

The DLME region contains the DLME image and DLME data. See Figure 8. The DCE preamble determines the address and size of the DLME region, loads the DLME image, and passes details of the region in the DRTM_PARAMETERS. The DLME data is populated by the DRTM implementation.

The DRTM_PARAMETERS describes the key characteristics of the DLME region:

- Physical address and size of the region
- Size of the DLME executable image
- Offset to the start of the DLME image
- Offset of the DLME image entry point within the image
- Offset to the DLME data

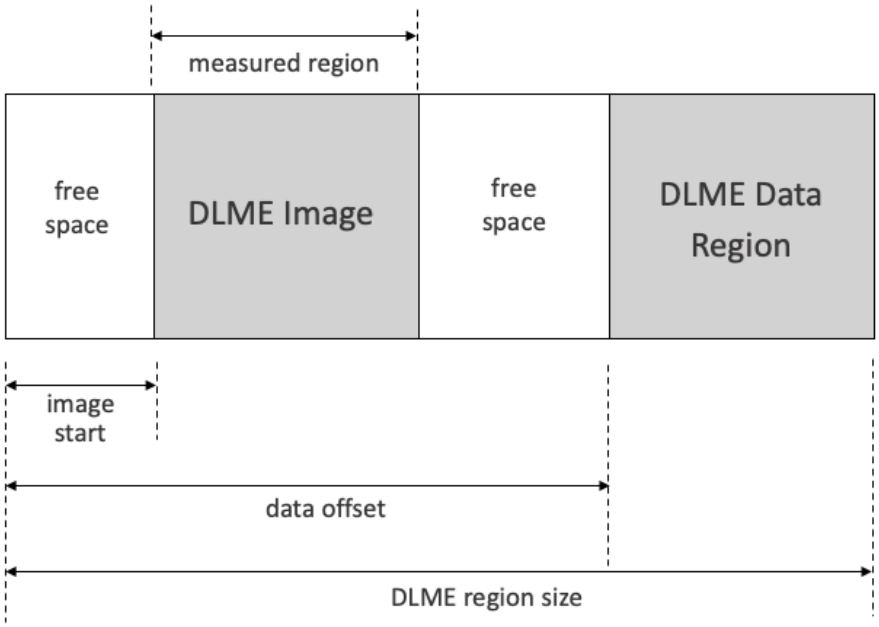


Figure 8: DLME region

Table 13 describes the requirements for the DLME region:

Table 13: DLME region requirements

ID	Requirement
R314000	The DLME region must be in Non-secure physically contiguous memory.
R314010	The DLME region must start at a 4KB aligned address.
R314020	The DLME image must start at a 4KB aligned address.
R314030	The DLME data must start at a 4KB aligned address.
R314040	The DLME image must come before the DLME data and must not overlap with it.

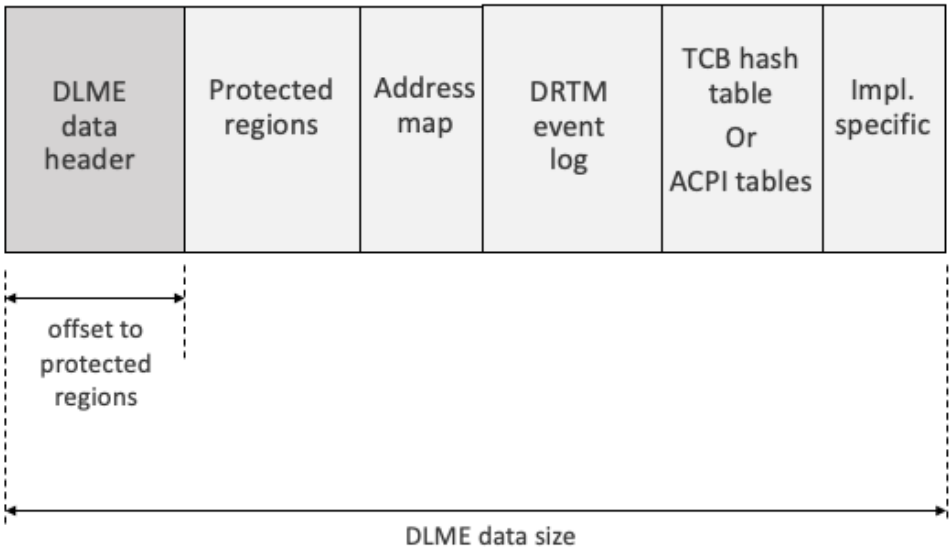
Space in the DLME region that is outside the DLME image and the DLME data can be used by the DCE preamble and the DLME for OS-specific purposes. See the regions labeled “free space” in Figure 8.

The DLME data consists of a header followed by a number of sub-regions containing data for use by the DLME. The DCE populates the fields of the DLME data which contains the following sub-regions within it:

- Protected memory regions
- Address map
- DRTM event log
- A region containing trustworthy ACPI tables or a table of hashes of TCB components.

The sub-regions within the DLME data are consecutive in memory with no padding between sub-regions. The DLME can find the start of each sub-region using the sizes in the DLME_DATA_HEADER.

Figure 9: DLME data



The DLME_DATA_HEADER describes the size each sub-region in the DLME data.

Table 14: DLME_DATA_HEADER definition

Field	Byte Offset	Byte Length	Description
Revision	0	2	Revision of this data structure. Must be the value 1 for this version of the DRTM architecture. For a given DRTM major version number (see DRTM_VERSION, 3.2) this structure is always extended in a backwards compatible manner.
Size	2	2	The size in bytes of the DLME_DATA_HEADER
Reserved	4	4	Must be zero.
DLME data size	8	8	Total size of the DLME data in bytes, including the DLME_DATA_HEADER and all data regions referenced by the parameters.
Protected regions size	16	8	Size in bytes of the protected regions memory descriptor table.
Address map size	24	8	Size in bytes of the address map memory descriptor table.
DRTM event log size	32	8	Size in bytes of the DRTM event log.
TCB hash table size	40	8	Size in bytes of the TCB hash table region. If the TCB hash table is not present, the size must be zero.

ACPI tables region size	48	8	Size in bytes of the ACPI tables region. If the ACPI table region is not present, the size must be zero.
Implementation-specific region size	56	8	Size of region for implementation-specific use. The presence of this region is optional. If it is not present, the size must be zero.

Note, the DLME data can be reclaimed for use by the OS after the DLME image has finished using it.

Table 15 describes the requirements for the DLME data:

Table 15: DLME data requirements

ID	Requirement
R314050	All the sub-regions referenced by the DLME_DATA_HEADER must be within the DLME region.
R314060	The following sub-regions must be present in the DLME data: • Protected regions • Address map • DRTM event log
R314070	It is strongly recommended that the ACPI table or TCB hash table region be present in the DLME data. See section 4.5 for further details.
R314080	The sub-regions referenced by the DLME_DATA_HEADER must be in the order described in Table 14 and must not overlap.
R314090	The size of fields in the DLME_DATA_HEADER must not extend beyond the bounds of the DLME data.
R314100	The address map must be formatted as a MEMORY_REGION_DESCRIPTOR_TABLE (see section 3.13) and must define all regions of normal memory, memory mapped I/O, and non-volatile memory accessible to the Normal world. The Region type field in the memory region descriptors must identify the type of region. For regions of normal memory, the Region type field must be set to type 1 and the Cacheability attributes field must be populated.
R314110	The protected regions must be formatted as a MEMORY_REGION_DESCRIPTOR_TABLE (see section 3.13) and must define all regions of memory protected by the DCE preamble through the DRTM_PARAMETERS. The Region type field in the memory region descriptors must be set to type 0, indicating normal memory.
R314120	For the MEMORY_REGION_DESCRIPTOR_TABLE defining the address map and the protected regions, the regions must be sorted with the first descriptor in the table being the lowest address. Regions in the table that are directly adjacent to each other must be coalesced into a single region.
R314130	The TCB hash table, if present, must be formatted as a TCB_HASH_TABLE. See section 3.15.

R314140	The ACPI tables region, if present, must start with an XSDT table at offset 0 which contains the physical addresses of all the other ACPI tables in the region. See the ACPI specification [13] for the definition of the XSDT.
R314150	The TCB hash table and ACPI tables regions are mutually exclusive and cannot both be present.

3.15 TCB_HASH_TABLE

The TCB_HASH_TABLE is a table data structure that describes one or more hashes. Each entry in the table represents one hash and contains the following fields: a unique hash ID and the digest of the hashed data. The table header defines the number of table entries.

This table is used:

- By firmware to report hashes of TCB components using the DRTM_SET_TCB_HASH function
- By the DRTM implementation to provide TCB hash data to the DLME

Table 16: TCB_HASH_TABLE definition

Field	Byte Offset	Byte Length	Description
Revision	0	2	Revision of this data structure. Must be the value 1 for this version of the DRTM architecture. For a given DRTM major version (see DRTM_VERSION, 3.2) this structure is always extended in a backwards compatible manner.
Number of hashes	2	2	Specifies the number of entries in this table.
Hash algorithm	4	2	Value identifies the algorithm used for hashes recorded in this table. Possible values are defined by the TPM_ALG_ID constants defined in the TCG Algorithm Registry [12]. A partial list is shown below. 0x000B: SHA-256 0x000C: SHA-384 0x000D: SHA-512 Note: The hash algorithm determines the size of the hash digest fields in the entries of this table. For example: SHA-256: 32 bytes SHA-384: 48 bytes SHA-512: 64 bytes
Reserved	6	2	Reserved. Must be 0.

Hash 0 ID	8	4	A unique value that identifies the data that was hashed. Bit[31]: Source of Entry 0: DRTM implementation. 1: DRTM_SET_TCB_HASH function. Bits[30:0]: Hash ID For the ACPI namespace this field consists of 4 ASCII bytes with the signature of the ACPI table.
Hash 0 digest	12	X	The digest of the TCB data. Consists of an array of X bytes, with the array length depending on the hash algorithm used.
Hash N ID	12+X	4	A unique value that identifies the data that was hashed.
Hash N digest	16+X	X	The digest of the TCB data. Consists of an array of X bytes, with the array length depending on the hash algorithm used.

Table 17 describes the requirements for a TCB_HASH_TABLE:

Table 17: TCB_HASH_TABLE requirements

ID	Requirement
R315000	The table must be in physically contiguous memory.
R315010	The Source of Entry bit in each table entry must be set by the DRTM implementation to identify whether the entry came from an invocation of the DRTM_SET_TCB_HASH function or directly from the DRTM implementation. Note: the Source of Entry bit is ignored by the DRTM_SET_TCB_HASH function.
R315020	For table entries with an ID namespace of ACPI, the ID field must be the signature of the table.
R315030	The hash algorithm field must reflect the same algorithm used by the DRTM implementation when it performs firmware-based measurements (see 2.7.1).
R315040	The maximum number of entries in the table is defined by the DRTM implementation and advertised in DRTM_FEATURES.
R315050	It is permitted for there to be entries with duplicate IDs. It is the responsibility of the DLME to evaluate duplicate entries.

3.16 DRTM event log

When measurements are made and extended into dynamic TPM PCRs during the dynamic launch the measurement values are also recorded into an event log in memory. The event log makes it possible to determine the set of measurements that make up a PCR value later.

The event log structure is specified in the TCG PC Client Platform Firmware Profile Specification [6].

3.16.1 DRTM event log requirements

Table 18 describes the requirements for the DRTM event log:

Table 18: Event log requirements

ID	Requirement
R316000	The DRTM event log must be in the DLME data which is allocated by the DCE preamble before the dynamic launch event.
R316010	There must be a minimum of 64KB of space allocated for the event log.
R316020	The DRTM event log must follow the crypto agile event log structure defined in the TCG PC Client Platform Firmware Profile Specification [6].
R316030	All measurements made into the TPM by the D-CRTM and DCE must be placed in the DRTM event log. See section 4.8 for PCR numbers and event type details for a given PCR usage schema.
R316040	The DRTM event log represents the most recent dynamic launch event. If multiple launch events occur while a system is powered up a new log is written each time.

3.16.2 Event types

When measurements are extended into TPM PCRs during the phases of DRTM, the measurements are also recorded in a TCG compliant event log. The table below defines the DRTM event types and the associated event data.

Table 19: DRTM event types

Event Type	Value	Description
EVTYPE_ARM_BASE (EB)	0x9000	The base of the DRTM event types.
EVTYPE_ARM_PCR_SCHEMA	EB + 1	Measurement of the PCR schema specified in the DRTM_PARAMETERS. See section 3.12 for the definition of possible schema values. The schema is measured as a 1-byte value. For example, if the schema used was b'0001 the measured value is 0x01. • Digest = Hash(PCR Schema) • Event data size = 0

EVTYPE_ARM_DCE	EB + 2	Measurement of the DCE image. This event has no event data. • Digest = Hash(DCE image) • Event data size = 0
EVTYPE_ARM_DCE_PUBKEY	EB + 3	Measurement of the public key used to authenticate the DCE. The DCE signing scheme is implementation specific, and the selection of the public key that is measured is up to the DRTM implementer. • Digest = Hash(public key used to authenticate the DCE) • Event data = The certficate chain associated with the public key used to authenticate the DLME image.
EVTYPE_ARM_DLME	EB + 4	Measurement of the DLME image. This event has no event data. • Digest = Hash(DLME image) • Event data size = 0
EVTYPE_ARM_DLME_ENTRY_POINT	EB + 5	Measurement of the 8-byte DLME entry point offset passed in the DRTM_PARAMETERS. • Digest = Hash(DLME image entry point offset) • Event data size = 0
EVTYPE_ARM_DEBUG_CONFIG	EB + 6	During development it can be necessary to perform a dynamic launch on systems that have debug or trace enabled. To permit the DLME to detect this situation, a measurement is made by the D-CRTM or DCE to record the debug and trace state. • If debug or hardware trace are enabled: Digest = Hash(0x01) • If debug and hardware trace are disabled: Digest = Hash(0x00) • Event data size = 0
EVTYPE_ARM_NONSECURE_CONFIG	EB + 7	During development it might be necessary to perform a dynamic launch on systems that are in a non-deployment security lifecycle state. To permit the DLME to detect this situation, a measurement is made by the D-CRTM or DCE to record the security lifecycle state. • If system is a non-secure lifecycle state: Digest = Hash(0x01)

		• If system is in a deployed/secure lifecycle state: Digest = Hash(0x00) • Event data size = 0
EVTYPE_ARM_DCE_SECONDARY	EB + 8	Measurement of a secondary DCE image. This event has no event data. • Digest = Hash(DCE image) • Event data size = 0
EVTYPE_ARM_TZFW	EB + 9	Measurement of any Secure firmware components. • Digest = Hash(Secure firmware component) • Event data size = implementation defined • Event data = an implementation-specific string that identifies the Secure firmware component.
EVTYPE_ARM_SEPARATOR	EB + 10	This is the last event extended by the DCE before transferring control to DLME. • Digest = Hash("ARM_DRTM") • Event data size = 8 • Event data = "ARM_DRTM"
EVTYPE_ARM_DLME_PUBKEY	EB + 11	Measurement of the public key used by the DCE to authenticate the DLME image. The DLME signing scheme is implementation specific, and the selection of the public key that is measured is up to the DCE implementer. • Digest = Hash(public key used to authenticate DLME) • Event data = The certficate chain associated with the public key used to authenticate the DLME image.
EVTYPE_ARM_DLME_SVN	EB + 12	Measurement of the security version number of the DLME image. This event has no event data. • Digest = Hash(DLME security version number) Event data size = 0
EVTYPE_ARM_NO_ACTION	EB + 13	This is an informative event recorded in the event log, but not extended into a PCR. • PCR = 0 • Digest = all zeros • Event data = [*a null terminated, event specific ASCII string*]

		• Event size = size of string, including terminating null

3.17 Return codes

Table 20 defines the possible values for error codes used with the interface functions. The error return type is a 64-bit signed integer. Zero and positive values denotes success and negative values indicates error.

Table 20: Return codes and values

Name	Description	Value
SUCCESS	The call completed successfully.	0
NOT_SUPPORTED	DRTM is not supported by this platform.	-1
INVALID_PARAMETERS	Some or all parameters passed to the call are invalid.	-2
DENIED	The call is not allowed because of the current system state.	-3
NOT_FOUND	The requested entity was not found.	-4
INTERNAL_ERROR	There was an internal error in the DRTM implementation.	-5
MEM_PROTECT_INVALID	Some or all memory protection requests were invalid.	-6
COPROCESSOR_ERROR	There was an error making a request to the coprocessor. (Hardware-backed implementation only)	-7
OUT_OF_RESOURCE	Resources needed to complete the request are not available.	-8
INVALID_DATA	Data supplied to the function contains invalid values.	-9
SECONDARY_PE_NOT_OFF	One or more secondary PEs were detected to not be off.	-10
ALREADY_CLOSED	The specified locality is already closed	-11
TPM_ERROR	The DRTM implementation encountered an error at the TPM.	-12

4 Requirements for DRTM phases

4.1 Non-secure firmware and TCB-critical data

As section 2.2 describes, the goal of DRTM is to establish a new smaller TCB that does not have a trust dependency on components in the SRTM boot chain. In addition to code, the TCB can include data such as security-critical ACPI tables that must be trustworthy so the new TCB can defend itself.

The recommended approach is for the DRTM implementation to directly provide hashes of security-critical ACPI tables or copies of ACPI tables in the DLME data. See section 3.14.

There might be cases where UEFI firmware must generate or modify ACPI tables that impact the TCB of the system, which prevents the DRTM implementation from directly providing ACPI table data. To accommodate a wide range of systems, Non-secure firmware can report hashes of ACPI tables to the DRTM implementation if the firmware can do this securely before any extensible components are loaded.

For UEFI firmware, before the End of DXE event the firmware is under the control of the platform manufacturer and no extensible components are part of the boot flow. Before End of DXE the firmware can use the DRTM_SET_TCB_HASH function to record the hashes of ACPI tables that impact the TCB.

See the considerations below:

- DRTM_SET_TCB_HASH should only be used if it is unavoidable for firmware to generate or modify TCB critical ACPI tables. The preferred alternative is for the DRTM implementation to directly provide hashes or the contents of TCB-critical ACPI tables in the DLME data.
- The use of the DRTM_SET_TCB_HASH function introduces a trust dependency on the Non-secure firmware component that uses the function.
- A Non-secure firmware component that uses the DRTM_SET_TCB_HASH function becomes part of the TCB, and the system must take steps to establish the trustworthiness of the Non-secure firmware. See section 5.10.
- The DRTM_LOCK_TCB_HASHES function must be used before extensible components are loaded to prevent further updates to the TCB hashes.

Table 21: Non-Secure firmware requirements

ID	Requirement
R41000	For any ACPI tables whose hashes or content are not provided directly by the DRTM implementation, it is strongly recommended that Non-secure firmware record hashes of all ACPI tables that impact the TCB of the system using the DRTM_SET_TCB_HASH function.
R41010	When using DRTM_SET_TCB_HASH, Non-secure firmware must be in an execution phase where all firmware components are under the authority of the platform manufacturer. For UEFI-based firmware this is the phase of execution before the End of DXE event.
R41020	If Non-secure firmware uses DRTM_SET_TCB_HASH, it is strongly recommended that hashes of the following ACPI tables be provided: MADT MCFG GTDT IORT TPM2

R41030	The RSDT, XSDT, FADT, DSDT, and SSDT should be measured by Non-secure firmware if they do not change after transitioning to an execution phase where firmware components might not be under the control of the platform manufacturer.
R41040	For other ACPI tables that are believed to not impact the TCB, it is still recommended that Non-secure firmware record hashes of these tables if possible to provide an extra level of security.
R41050	Before transitioning to an execution phase where firmware components might not be under the control of the platform manufacturer, the DRTM_LOCK_TCB_HASHES function must be used to prevent further hashes from being recorded.

4.2 DCE preamble

The dynamic launch is a single-threaded operation that happens on the boot PE. The DCE preamble ensures that all PEs except the boot PE are off and this is verified by the D-CRTM. It is the responsibility of the DRTM client to take platform-specific steps to halt or quiesce Secure and Non-secure software in the system that can be affected by DRTM. Examples of Secure software that can be affected by DRTM include Trusted Applications loaded and used by Non-secure software or an open session established between Non-secure software and a Secure service.

The DCE Preamble is responsible for preparing the platform for DRTM and initiating the dynamic launch.

Table 22 specifies the requirements for the DCE Preamble.

Table 22: DCE preamble requirements

ID	Requirement
R42000	If a previous dynamic launch occurred, the DCE preamble should check for errors from previous dynamic launch events using the DRTM_GET_ERROR function. Note: It is the responsibility of the DCE preamble to coordinate with the DLME to determine whether a dynamic launch might have previously occurred and failed. For example, the DCE preamble can set a flag in an OS-specific location indicating the initiation of a dynamic launch, with the DLME clearing the flag.
R42010	The DCE preamble must ensure that all PEs except the boot PE are off using the CPU_OFF PSCI function [2].
R42020	The DCE preamble must allocate memory for the DLME region with sufficient space for the DLME image and the DLME data. See section 3.14 for a description of the DLME region and its requirements.
R42030	The DCE preamble must load the DLME image into the DLME region.
R42040	The DCE preamble must determine if the implementation utilizes a Normal world DCE using the DRTM_FEATURES function. If the minimum memory requirement for a Normal world DCE is greater than zero a Normal world DCE is used. See section 3.3 for more details.

R42050	If a Normal world DCE is in use, the DCE preamble must allocate sufficient memory for the Normal world DCE. The amount of memory required is platform-specific and can be determined using the DRTM_FEATURES function.
R42060	The DCE preamble must prepare the DRTM_PARAMETERS. See section 3.12 for the requirements for the DRTM_PARAMETERS.
R42070	If region-based DMA protection is in use, the DCE Preamble must protect the DLME region by defining a memory protection table formatted as a MEMORY_REGION_DESCRIPTOR_TABLE. See section 3.13.
R42080	If region-based DMA protection is in use, the DCE Preamble can request protection of other memory regions besides the DLME region.
R42090	If region-based DMA protection is in use, the Region type field for all regions described by the table must be set to 0 (normal, usable memory). The address and size of the table must be passed in the DRTM_PARAMETERS.
R42100	If a Normal world DCE is in use, the DCE preamble must ensure the Normal world DCE region is protected using the memory protection table passed in the DRTM_PARAMETERS.
R42110	If a TPM Command Response Buffer Interface is in normal memory, the DCE preamble must ensure that the locality 3 CRB is protected using the memory protection table passed in the DRTM_PARAMETERS. The DCE preamble may protect the CRB regions for locality 2 and 1.
R42120	The DCE preamble must ensure there are no active TPM localities at the time of the dynamic launch.
R42130	The DRTM_PARAMETERS and any memory protection table must be created so that they can be accessed coherently with the following attributes and data access permissions: • Normal Write-Back Cacheable. • Non-transient Read-Allocate. • Non-transient Write-Allocate. • Inner Shareable. • Read-Write. Arm recommends that the DRTM_PARAMETERS and any memory protection table be mapped Execute-Never.
R42140	The DCE preamble may use space in the DLME region that is outside the DLME image and DLME data for passing OS-specific data to the DLME.

4.3 Dynamic launch event

The Dynamic Launch Event, or DL Event, begins the DRTM process and moves execution into the D-CRTM. The DL Event is initiated by the function DRTM_DYNAMIC_LAUNCH. See section 3.4.

Table 23 specifies the requirements for the DL Event.

Table 23: DL Event requirements

ID	Requirement
R43000	The invocation of DRTM_DYNAMIC_LAUNCH must be in AArch64 state.
R43010	It must be possible to initiate a DRTM_DYNAMIC_LAUNCH repeatedly without a system reset.

4.4 Firmware-backed D-CRTM requirements

The invocation of DRTM_DYNAMIC_LAUNCH transitions control to the D-CRTM which is a trusted agent where the dynamic launch process begins. For firmware-backed DRTM the trusted agent is implemented in Secure firmware.

The first phase of the D-CRTM performs checks such as verifying the boot PE is being used, without changing the state of the system. During this phase, the D-CRTM can return errors to the caller of DRTM_DYNAMIC_LAUNCH.

After the D-CRTM makes changes to the state of the system, such as resetting dynamic PCRs in the TPM, it enters a second phase of execution and can no longer return errors to the caller of DRTM_DYNAMIC_LAUNCH. The D-CRTM must enter remediation (see section 4.7) if any errors occur during this phase.

Table 24 specifies the requirements for the first phase of the D-CRTM in a firmware-backed implementation, when errors can be returned to the caller.

Table 24: D-CRTM phase 1 requirements

ID	Requirement
R44000	The D-CRTM must be immutable and tamper-resistant with respect to the Normal world.
R44010	The D-CRTM must only execute on the boot PE. If an error is detected the D-CRTM must return the return code DENIED to the caller of DRTM_DYNAMIC_LAUNCH.
R44020	The D-CRTM must verify that the DCE preamble used PSCI CPU_OFF to turn off all PEs except the boot PE, and if not return the return code SECONDARY_PE_NOT_OFF to the caller of DRTM_DYNAMIC_LAUNCH.
R44030	An implementation of DRTM may require the boot PE to be a specific PE in the topology of the SoC. In this case the D-CRTM must return the error code DENIED if the PE used to initiate the dynamic launch is not the correct one.
R44040	The D-CRTM must ensure that DRTM operation on the boot PE cannot be preempted by Non-secure asynchronous exceptions unless explicitly enabled by the DLME. Examples include interrupts, SError exceptions, and SDEI events.
R44050	If the caller of DRTM_DYNAMIC_LAUNCH was not in AArch64 state, the D-CRTM must return the return code DENIED to the caller of DRTM_DYNAMIC_LAUNCH.
R44060	The D-CRTM must verify that no TPM localities are active, and if not return the return code DENIED to the caller of DRTM_DYNAMIC_LAUNCH. If there is an error accessing the TPM the return code TPM_ERROR must be returned to the caller.
R44110	The D-CRTM must map the DRTM_PARAMETERS Execute-Never.

R44111	The D-CRTM must verify the DRTM_PARAMETERS comply with the requirements defined in Table 10. If an error is detected, the D-CRTM must return the return code INVALID_PARAMETERS to the caller.
R44112	If a memory protection table is specified in the DRTM_PARAMETERS, the D-CRTM must verify that the table complies with the requirements defined in Table 12. If an error is detected, the D-CRTM must return the return code MEM_PROTECT_INVALID to the caller.
R44065	If the DRTM_PARAMETERS request features that are not supported by the DRTM implementation, the D-CRTM must return the return code INVALID_PARAMETERS to the caller.
R44066	If the DRTM implementation encounters an internal error, it must return the error code INTERNAL_ERROR to the caller.
R44070	If the D-CRTM returns an error to the caller, the open or closed state of localities 1, 2, and 3 must be left unmodified.
R44075	If the D-CRTM returns an error to the caller of DRTM_DYNAMIC_LAUNCH the state of the system must be left unmodified.

Table 25 specifies the requirements for the second phase of the D-CRTM in a firmware-backed implementation where errors cannot be returned to the caller.

Table 25: D-CRTM phase 2 requirements

ID	Requirement
R44080	If the SoC implements a GIC ITS (Interrupt Translation Service [14]), the D-CRTM must: • Ensure all ITSs are disabled (GITS_CTLR.Enabled = 0 and GITS_CTLR. Quiescent = 1) • Ensure LPIs are disabled in all Redistributors (GICR_CTLR.EnableLPIs = 0 and GICR_CTLR.RWP = 0) These settings ensure that the GIC makes no memory accesses during the dynamic launch. Implementation Note: In some GICv3.0 implementations it might not be possible to clear GITS_CTLR.Enabled after it is set. For this case the D-CRTM must: • Verify the location and size of the LPI Configuration tables and the LPI Pending tables to ensure that they do not overlap any DRTM-protected memory region. • Verify that GICR_PENDBASER and GICR_PROPBASER do not violate any of the rules specified in the GIC architecture that lead to behavior defined as UNPREDICTABLE.
R44090	The D-CRTM must reset the dynamic PCRs in the TPM using TPM_HASH_START (or equivalent) at locality 4.
R44100	The D-CRTM must open localities 1, 2, and 3 in an implementation-specific way.
R44120	If the D-CRTM is distinct from the DCE image, the D-CRTM must load and authenticate the DCE image. Authentication means: • Verifying the integrity of the image through a digital signature check • Verfiying that the image signing public key is rooted to a trusted authority.

	The definition of the DCE image format and signature format are implementation specific.
R44130	Prior to loading a DCE image, the D-CRTM must prepare the memory region the DCE will execute from and ensure it is protected from DMA.
R44140	If the D-CRTM is distinct from the DCE image, the D-CRTM must extend a measurement of the DCE image into the TPM and record the measurement in the event log using the event type EVTYPE_ARM_DCE. See PCR schema details in section 4.8.
R44150	If the D-CRTM is distinct from the DCE image, the D-CRTM must extend a measurement of the public key used to authenticate the DCE image into the TPM and record the measurement in the event log using the event type EVTYPE_ARM_DCE_PUBKEY. See PCR schema details in section 4.8.
R44160	If the D-CRTM and DCE images are not distinct and a digest of the DCE cannot be computed, the D-CRTM must: • Extend the digest of the 1-byte value of zero into the TPM for the DCE image and record the measurement in the event log using the event type EVTYPE_ARM_DCE. • Extend the digest of the 1-byte value of zero into the TPM for the public key that signed the DCE image and record the measurement in the event log using the event type EVTYPE_ARM_DCE_PUBKEY. See PCR schema details in section 4.8.
R44170	The D-CRTM must measure the PCR schema passed in the DRTM_PARAMETERS into the TPM and record the measurement in the event log using the event type EVTYPE_ARM_PCR_SCHEMA. See PCR schema details in section 4.8.
R44180	If the platform supports a mechanism that permits a PE to detect if external debug (invasive or non-invasive) is enabled, the D-CRTM must detect this, and: • Extend a Boolean value into the TPM • Record the measurement in the DRTM event log using the event type EVTYPE_ARM_DEBUG_CONFIG. See PCR schema details in section 4.8. If external debug can be enabled dynamically without a system reset, the D-CRTM must measure debug as enabled.
R44190	If it is possible in a deployed lifecycle state for a hardware trace feature to write trace data to Non-secure memory and if the platform supports a mechanism that permits a PE to detect if hardware trace is enabled, the D-CRTM must detect this and extend a Boolean value into the TPM and record the measurement in the DRTM event log using the event type EVTYPE_ARM_DEBUG_CONFIG. See PCR schema details in section 4.8. If the enabling of trace cannot be detected by a DRTM implementation or if trace can be enabled dynamically without a system reset the D-CRTM must measure trace as enabled.
R44200	The D-CRTM may detect and measure the security lifecycle state of the platform into the TPM and record the measurement in the DRTM event log using the event type EVTYPE_ARM_NONSECURE_CONFIG. See PCR schema details in section 4.8. Note: if the D-CRTM does not make this measurement, it must be done by the DCE.

R44210	If the D-CRTM is unable to log a measurement because there is no available space in the event log region, the D-CRTM must extend a hash of the 1-byte value 0xFF into PCR[17] and PCR[18] and enter remediation.
R44220	If the D-CRTM encounters an error, it must return an error to the caller of DRTM_DYNAMIC_LAUNCH or enter remediation. See section 4.7.
R44230	The D-CRTM must only be updateable through a secure firmware update procedure that meets the requirements specified in the Arm Platform Security Boot Guide [5]
R44240	Updates to the D-CRTM must be protected against rollback as specified by the requirements in the Arm Platform Security Boot Guide [5].

4.5 DCE requirements

The DCE can consist of multiple components. It is not a requirement that the DCE be a single, monolithic image. The requirements in this section are applicable to both firmware and hardware backed implementations of DRTM.

Table 26 specifies the requirements for the DCE images.

Table 26: DCE image requirements

ID	Requirement
R45000	The DCE must only be updateable through a secure firmware update procedure that meets the requirements specified in the Arm Platform Security Boot Guide [5].
R45010	Updates to the DCE must be protected against rollback as specified by the requirements in the Arm Platform Security Boot Guide [5].
R45020	The DCE image(s) must be digitally signed by the DCE provider following the requirements specified in the Arm Platform Security Boot Guide [5].

Table 27 specifies general requirements for the DCE.

Table 27: General DCE requirements

ID	Requirement
R45030	The DCE must map the DRTM_PARAMETERS Execute-Never.
R45040	The DCE must verify the fields of the DRTM_PARAMETERS before use. See section 3.12.
R45050	If the DCE is unable to log a measurement because there is no available space in the event log region, the DCE must extend a hash of the 1-byte value 0xFF into PCR[17] and PCR[18] and enter remediation.

R45060	If not done by the D-CRTM, the DCE must read and measure the security lifecycle state of the platform and extend it into the TPM and record the measurement in the DRTM event log using the event type EVTYPE_ARM_NONSECURE_CONFIG. See PCR schema details in section 4.8.
R45070	If the DCE encounters an error, it must enter remediation. See section 4.7.

Table 28 specifies memory protection requirements.

Table 28: Memory protection requirements

ID	Requirement
R45080	For complete DMA protection the DCE must configure the SMMUs in the system to block DMA from all devices. To guarantee completion of all outstanding device accesses the DRTM implementation must perform an invalidation of all Non-secure TLB information at the SMMU.
R45090	For region-based DMA protection, the DCE must verify any memory protection table specified in the DRTM_PARAMETERS before using the table.
R45100	For region-based DMA protection, the DCE must verify that the DLME region is in the memory protection table specified in the DRTM_PARAMETERS.
R45110	For region-based DMA protection, if a TPM Command Response Buffer Interface is in normal memory, the DCE must verify that the locality 3 CRB is protected using the memory protection table specified in the DRTM_PARAMETERS.
R45120	For region-based DMA protection, the DCE must use a platform-dependent hardware enforcement mechanism to protect the requested regions.
R45130	For region-based DMA protection the DLME region must be protected. However, for memory outside of the DLME region, it is permitted for the implementation to protect a subset of the protected regions required in the DRTM_PARAMETERS if required for platform-dependent reasons.
R45140	For region-based DMA protection, regions protected by the implementation must be described in the protected regions area in the DLME data. The protected regions in the DLME data might differ from the regions requested in the DRTM_PARAMETERS.
R45150	For a firmware-backed implementation, the DMA protection for both complete and region-based DMA protection must prevent DMA accesses from Non-secure devices.
R45160	If complete DMA protection is in use, the DCE must define a single region in the protected regions in the DLME data consisting of address 0x0 and the maximum region size. These sentinel values do not define a protected region of memory, but instead communicate to the DLME that all DMA is disabled.

The DCE can be composed of multiple components, where an initial DCE component started by the D-CRTM loads, verifies, and measures a next stage DCE component. Table 29 defines requirements specific to an implementation consisting of multiple DCE components.

Table 29: Multiple stage DCE requirements

ID	Requirement
R45170	The prior stage DCE must load the next stage DCE image into a platform-appropriate memory region and must enforce verification of the cryptographic signature on the next stage DCE image.
R45180	The prior stage DCE may optionally extend a measurement of the next stage DCE image into the TPM and record the measurement in the event log using the event type EVTYPE_ARM_DCE_SECONDARY. See PCR schema details in section 4.8.

Table 30 specifies requirements that the DCE must perform with respect to the DLME.

Table 30: DCE requirements for the DLME

ID	Requirement
R45190	The DCE must verify that DLME region is in Non-secure memory and that the requirements described in section 3.14 are met.
R45200	The DCE must verify that the DLME data fits within the DLME region.
R45210	Before the DLME image is measured the DCE must verify that the DLME region is protected from DMA based on the memory protection specified in the DRTM_PARAMETERS.
R45220	Before the DCE measures the DLME image or initializes the DLME data, it must clean and invalidate the DLME region for all data caches to the Point of Coherency.
R45230	If required by the PCR schema in use, the DCE must extend a measurement of the DLME image into the TPM and record the measurement in the DRTM event log using the event type EVTYPE_ARM_DLME. See PCR schema details in section 4.8. The DLME image region is defined by the DLME image offset and DLME image size fields of the DRTM_PARAMETERS. See section 3.12.
R45250	The DCE must initialize the DLME_DATA_HEADER. See section 3.14.
R45260	The DCE must populate the address map in the DLME data. See section 3.14.
R45270	The DCE must populate the protected regions in the DLME data. See section 3.14.
R45280	The DCE may use the implementation-specific region in the DLME data for implementation-specific purposes. See section 3.14.
R45290	If firmware provides TCB hashes using the DRTM_SET_TCB_HASH function, firmware must also lock the set of hashes using the DRTM_LOCK_TCB_HASHES function. If the hashes are not locked, the DCE must record the error code 0x4 and enter remediation. See section 3.11.
R45300	Before the DLME is called, the DCE must extend the event type EVTYPE_ARM_SEPARATOR into the TPM and record the measurement in the DRTM event log. See PCR and event type details in section 4.8.
R45310	Before the DLME is called, the DCE must invalidate all instruction caches.

R45320	Before the DLME is called, any unused space in the DRTM event log region must be initialized to zeros.
R45330	Before the DLME is called, the DCE must close locality 3.
R45340	Before the DLME is called, locality 2 must be active.
R45350	Before the DLME is called the DCE must set the DRTM error code to zero.
R45360	The DCE must identify the entry point of the DLME image from the offset in the DRTM_PARAMETERS. See section 3.12.
R45370	The DCE must extend a measurement of the DLME entry point offset into the TPM and record the measurement in the DRTM event log using the event type EVTYPE_ARM_DLME_ENTRY_POINT. See PCR schema details in section 4.8.
R45380	The physical address of the DLME region must be passed to the DLME in X0 and the offset to the DLME data area must be passed in X1.
R45390	The DCE must transfer control to the DLME entry point.

A Normal world DCE is a DCE component that executes in the Normal world. The DRTM implementation in the Secure world is responsible for loading and starting the Normal world DCE. How the Secure world DRTM implementation and Normal world DCE interact is implementation defined. For example, how the DRTM_PARAMETERS are made available to the Normal world DCE is implementation defined. Because of this it is expected that the Secure world implementation of DRTM and the Normal world DCE are tightly coupled and provided by the same vendor.

If a Normal world DCE component is in use, the DCE preamble allocates space for the Normal world DCE to execute in and provides the address and size of that memory region in the DRTM_PARAMETERS. Table 31 specifies requirements for the component that loads and starts the Normal world DCE.

Table 31: Normal world DCE requirements

ID	Requirement
R45400	The DRTM implementation must determine the address and size of the Normal world DCE region from the DRTM_PARAMETERS.
R45410	The DRTM implementation must verify that the Normal world DCE region is protected from DMA based on the protection specified in the DRTM_PARAMETERS.

For the DLME to defend itself, it might need a description of TCB hardware components. During the normal boot flow of a system, the system firmware provides a set of ACPI tables that provide a system description for use by the OS. To enable the DLME to detect tampering of TCB-critical ACPI tables it is strongly recommended that the DCE provide: A) trustworthy hashes of TCB-critical ACPI tables, or B) trustworthy copies of TCB-critical ACPI tables. The DLME data defines region where the hashes or copies of tables can be provided.

For the TCB hash table, the DCE can determine the contents of the table in two ways:

1. From data provided by firmware using the DRTM_SET_TCB_HASH function

2. Through implementation-specific means. For a simple system, the DRTM implementation might have a static set of hashes that can be provided in the TCB hash table. If the DRTM implementation uses implementation-specific means, it might advertise a value of zero in the TCB hash features of DRTM_FEATURES.

It is permitted for TCB hash table entries to come from both the DRTM_SET_TCB_HASH function and directly from the DRTM implementation through implementation-specific means. The DLME can differentiate between the two sources of TCB hash data through the Source of Entry bit in each table entry.

The set of ACPI tables or other data that can impact the TCB of a system is system-specific. It is the responsibility of the platform manufacturer to determine which ACPI tables or other data impact the TCB and make hashes or copies of these components available in one of the ways described above.

Table 32 specifies requirements for the DCE-provided TCB hash table.

Table 32: ACPI table requirements

ID	Requirement
R45420	The DCE must provide a TCB hash table region or copies of TCB-critical ACPI tables in the DLME data.
R45430	The DCE must set the Source of Entry bit in each TCB hash table entry to identify whether the entry came from the DRTM_SET_TCB_HASH function or from the DRTM implementation.
R45440	It is strongly recommended that trustworthy hashes or copies of the following ACPI tables be provided: MADT MCFG GTDT IORT TPM2

Table 33 specifies how the boot PE must be configured when control is transferred to the DLME.

Table 33: Boot PE requirements

ID	Requirement
R45450	Except as noted below, the boot PE must be configured as defined in the PSCI specification [2] in the definition of the initial state of a PE after a CPU_ON operation. See the section *Initial state after CPU_ON, CPU_SUSPEND* in the PSCI specification. The following requirements supersede the definition in PSCI: • Execution state must be AArch64 • Exception level must be the highest Non-secure exception level • Little-endian • Register X0 contains the physical address of the DLME region • Register X1 contains the offset from the start of the DLME region to the DLME data
R45460	Non-secure asynchronous exceptions must be masked on the boot PE.
R45470	Debug exceptions must be masked on the boot PE.

4.5.1 DLME image authentication

In addition to measuring the DLME image, the DCE may also support authentication of the DLME image using digital signatures. The DRTM_FEATURES advertise whether the image authentication capability is available.

The definition of supported image formats, a security version number, and signature formats for image authentication are out of scope of this specification and should be provided by the vendor of the platform.

If DLME image authentication is in use, an alternate DLME Authorities PCR schema may be requested when initiating a dynamic launch. The DLME Authorities schema measures the public key used to authenticate the DLME image and the DLME image's security version number. See details in section 4.8.4.

Table 34 defines requirements related to image authentication. These requirements are only applicable if DLME image authentication is requested.

Table 34: DLME image authentication requirements

ID	Requirement
R45240	The DCE may perform authentication of the DLME image using digital signatures. Authentication means: • Verifying the integrity of the image through a digital signature check • Verifying that the image signing public key is rooted to a trusted authority
R45241	If the DCE supports DLME image authentication, this capability must be advertised in the DRTM_FEATURES (see section 3.3).
R45242	If the DCE preamble requests DLME image authentication in the DRTM_PARAMETERS, the DCE must perform digital signature authentication of the DLME image.
R45243	If the DLME image format is not recognized by the DCE, it must enter remediation.
R45244	If the digital signature check of the DLME image fails to verify the integrity of the image, the DCE must enter remediation.
R45245	If the check whether the signing key is rooted to a trusted authority fails, the DCE must log the event type EVTYPE_ARM_NO_ACTION with the event data string "DLME AUTHORITY FAIL" to the event log and continue the dynamic launch. Note, the EVTYPE_ARM_NO_ACTION event does not modify any TPM PCR. See section 3.16.2.
R45246	If DLME image authentication succeeds and the DLME Authorities PCR schema is in use, the DCE must extend a measurement of the public key used to authenticate the DLME image into the TPM and record the measurement in the event log using the event type EVTYPE_ARM_DLME_PUBKEY. See the DLME Authorities PCR schema details in section 4.8.4.
R45247	If DLME image authentication is succeeds and the DLME Authorities PCR schema is in use and if a security version number (SVN) is present in the DLME image, the DCE must extend a measurement of the SVN into the TPM and record the measurement in the event log using the event type EVTYPE_ARM_DLME_SVN. See the DLME Authorities PCR schema details in section 4.8.4.

4.6 DLME

4.6.1 DLME initial state

When the DLME entry point is executed, it can assume the following about the system state:

- The DLME receives execution control on the boot PE. All other PEs are off.
- The boot PE is configured as defined by CPU_ON in the PSCI specification [2], except as noted in Table 33
- Register X0 contains the physical address of the DLME region
- Register X1 contains the offset from the start of the DLME region to the DLME data
- The DMA protections requested in the DRTM_PARAMETERS are in effect
- TPM locality 2 is active. The DLME does not need to request locality 2
- The DLME data is populated as defined in section 3.14

4.6.2 DLME operation

The DLME might be an operating system-specific component supplied by the same vendor that provided the DCE Preamble component, or it might conform to a standardized boot protocol.

The DLME begins execution in a new minimal TCB established by DRTM and is in a protected state. See section 4.6.1. From this starting point, it prepares the operational environment for the runtime operation of the hypervisor or operating system kernel. It is critical that the DLME not utilize any code or data outside of the DLME region unless it has been validated in some way. Code can be verified using digital signatures or using measurements extended into the TPM. Data can be validated to confirm it is safe to consume.

The DLME can itself be the OS kernel, or it might prepare the runtime environment and then hand off control to the OS.

The following describes the responsibilities of the DLME to defend itself and prepare for the OS runtime:

- **Boot PE Initialization**. The DLME begins execution on the boot PE which is configured as defined in Table 33. The DLME is responsible for initializing the boot PE into an operational state. This process typically includes creating address translation tables, enabling the MMU, and putting exception vectors in place.
- **Determining trust.** The system must establish a security policy based on the measurements made by DRTM in PCR[17] and PCR[18] (see section 4.8). The policy detects any tampering of the DLME and establishes whether the boot flow should continue.
    - The system can use local attestation to determine trust, where a security policy can be implemented based on sealing to PCR[17] and/or PCR[18]. For example, a secret such as a disk encryption key needed to boot the system can be sealed against dynamic PCRs.
    - The system can use remote attestation where it sends a TPM quote to a remote verifier that can determine whether the state measured by DRTM is as expected. For example, the verification of the quote can be a criteria for access to a secure network.
    - The system can use the DRTM event log in the DLME data to implement a policy around specific system state measured by DRTM. For example, if DRTM measured that external debug was enabled in the system the DLME can determine this from the event log. The DLME can then implement an appropriate policy.
- **DLME data**.

    - The DLME can determine the address of the DLME data from the DLME region address and DLME data offset arguments passed to the DLME. See section 4.6.1.
    - Space in the DLME region that is outside the DLME image and the DLME data can be used for DLME-specific purposes. See the regions labeled “free space” in Figure 8. If the DLME requires a specific amount of free space before or after the DLME image, it must determine the DLME image size and then determine the available space in conjunction with the DLME region address and the DLME data offset received as arguments.
- **Protected regions**. The protected regions in the DLME data describe the DMA protections requested by the DCE Preamble and put in place by the DRTM implementation.
    - If complete DMA protection was requested, this is indicated by a sentinel entry in the protected regions.
    - If region-based DMA protection was requested, the protected regions always include the DLME region, which is guaranteed to have DMA protections. If additional memory protections beyond the DLME region were requested, the DLME must examine the protected regions to verify they are as expected. If necessary for platform-specific reasons, the DRTM implementation might protect a subset of the regions requested.
- **Memory map**
    - The memory map provided in the DLME data identifies all regions of normal memory, memory mapped I/O, and non-volatile memory accessible to the Normal world. For normal memory regions the memory map includes cacheability attributes.
    - Before utilizing any address space outside of the DLME region, the DLME should check that regions used are consistent with the address map in the DLME data.
    - Before using an address map provided by untrusted firmware, the DLME should verify that the map is consistent with the address map in the DLME data.
- **ACPI Tables**. The DLME must not use any untrusted data components, such as unverified ACPI tables provided by Non-secure firmware, to establish the TCB.
    - The DLME data contains: A) hashes of TCB-critical ACPI tables, or B) copies of ACPI tables. The DLME can use this data to verify the ACPI tables provided by Non-secure firmware. The DLME can determine which type of data is present by inspecting the DLME_DATA_HEADER.
    - When processing a TCB hash the DLME should consider the following:
        - The source of each table entry (DRTM_SET_TCB_HASH or the DRTM implementation) can be distinguished with the Source of Entry bit in each entry.
        - There might be entries with duplicate IDs in the TCB hash table. For ACPI tables such as the SSDT this may be expected. For other type of IDs duplicates are not valid. It is the responsibility of the DLME to evaluate any duplicate IDs and enter remediation if necessary.
    - Untrusted tables can be used as long as the content in them is not used to establish the TCB. The DLME is free to make a policy decision around the ACPI tables that it considers security critical and can refuse to boot or might disable functionality if the DRTM implementation does not provide hashes or copies of required ACPI tables. Platform manufacturers must be aware of the requirements for OSes they intend to support.
- **Execution of other software**. The DLME must not execute any untrusted components unless it has validated the component or constrained its execution. This requirement means, for example, that the DLME must take

care in utilizing a component such as UEFI Runtime Services which has not been measured as part of DRTM and is not trusted. Validation of code can include digital signatures or measurements.

- **TPM usage**. The DLME can use the TPM for making implementation or OS-specific measurements.
    - If the DLME uses the TPM2 ACPI table, it should verify the integrity of the table using the ACPI information in the DLME data.
    - The dynamic PCRs 19-22 are reserved for use by the DLME.
    - The DLME can make measurements using locality 2.
    - If preventing further measurements into PCRs 17 and 18 is a requirement of the security policy of the system, the DLME must close locality 2 using the DRTM_CLOSE_LOCALITY function.
- **DRTM event log**. The DLME should make the DRTM event log available to the OS runtime in an OS-specific manner.
- **DMA protections**
    - If the DLME uses the IORT ACPI table, it should verify the integrity of the table using the ACPI information provided in the DLME data.
    - The DLME should put appropriate DMA protections in place for the OS runtime before transitioning to the OS.
    - For complete DMA protection the DLME should have no expectation that the SMMU be restored to any particular state following a call to DRTM_UNPROTECT_MEMORY. The DLME should assume nothing about the state of the SMMU.
- **Device I/O**. If the DLME uses the MCFG ACPI table, it should verify the integrity of the table using the ACPI information provided in the DLME data.
- **Interrupts**. The DLME should verify the integrity of the MADT ACPI table using the ACPI information provided in the DLME data.
- **Secrets in memory**. The system must provide a mechanism to mitigate against platform reset attacks. See section 5.9. The DLME should use the provided mechanism to ensure secrets are scrubbed from memory on a reset.
- **Error handling**. Errors that occur during a dynamic launch will never reach the DLME. The DRTM implementation might return errors to the DCE Preamble or might enter remediation which will reset the system. Errors detected by the DLME are handled in a platform-specific manner. The DLME can use DRTM_SET_ERROR to report detected errors.
- **Remove memory protections**. The DLME must use the DRTM_UNPROTECT_MEMORY function to remove the memory protections put in place by the DCE preamble.

4.7 Error handling and remediation

During the dynamic launch, errors can occur that prevent the launch from succeeding. The actions a platform takes when errors occur are referred to as *remediation* in this architecture.

When the DCE preamble initiates a dynamic launch, there are a class of errors that can be directly returned to the caller as section 3.4 describes. For these errors, any remediation needed is the responsibility of the DCE preamble.

As the dynamic launch progresses through the D-CRTM and DCE phases it might not be possible to return errors to the caller of DRTM_DYNAMIC_LAUNCH. For these errors, remediation consists of the DRTM implementation

storing an error code and initiating a system reset. The error code can be retrieved by a DRTM client using the DRTM_GET_ERROR function following the system reset.

For errors that require platform remediation the requirements are specified in Table 35.

Table 35: Remediation requirements

ID	Requirement
R47000	The DRTM implementation must place an error code, as defined in section 3.11, in a storage location so that the value persists across the system reset and the DCE preamble can determine why the previous dynamic launch failed.
R47010	An implementation may record implementation-defined errors using the error code 0xFF and an implementation-specific data value. See the DRTM error encoding in section 3.11.
R47020	If a Normal world DCE is used, it must record the error code using the DRTM_SET_ERROR function. See section 3.8.
R47030	As a final step in remediation, the DRTM implementation must initiate a system reset.
R47040	An implementation may report DRTM errors to other error logging infrastructure that might exist on a platform.

Errors detected by the DLME are handled in a platform-specific manner. The DLME can record error codes using the DRTM_SET_ERROR function.

4.8 TPM measurements

4.8.1 TPM measurement requirements

The TPM measurement requirements in Table 36 are applicable to all phases of DRTM.

Table 36: TPM measurement requirements

ID	Requirement
R48000	For firmware-based measurements (see 2.7.1), if the TPM implementation supports SHA-384, all measurements must be made with a SHA-384 or stronger hashing algorithm. A hashing algorithm of equivalent security strength to SHA-384 is also permitted. If the TPM implementation does not support SHA-384, then measurements must be made with a SHA-256 or stronger hashing algorithm (or equivalent).
R48010	If the PCR schema in use does not specify an order in which the measurements are to be made, the order of the measurements must be deterministic between runs.

4.8.2 PCR schemas

This architecture supports a flexible approach by defining schemas that specify how TPM PCRs are used for measurements. The available PCR usage schemas in a DRTM implementation are advertised using a bitmap in DRTM_FEATURES. A DRTM client requests the schema to use through a value in the DRTM_PARAMETERS passed to DRTM_DYNAMIC_LAUNCH.

The architecture defines a default schema that must be present in all implementations (see details in section 4.8.3).

If DLME image authentication is supported (see section 4.5.1) an additional DLME Authorities schema will be available. See definition in section 4.8.4.

4.8.3 Default PCR schema

The default schema uses PCR[17] and PCR[18]. For each PCR the definition includes:

- The component or item measured
- Event type for the DRTM event log
- Any ordering requirements for the measurements

The goal of the default PCR schema is to provide a stable PCR[18] measurement that the DLME or OS can use to implement a security policy.

Table 37: Default PCR schema requirements

ID	Requirement
R48020	For the default PCR usage schema the components or items measured, the measurement order, and event type must comply with the requirements in Table 38 and Table 39 for PCR[17] and PCR[18]. See Event types, 3.16.2, for details of how each event type is logged.

Table 38: Default PCR[17] schema details

Order	Component Measured	Event Type	Usage
1	DCE image	EVTYPE_ARM_DCE	Required
2	PCR schema value in the DRTM_PARAMETERS	EVTYPE_ARM_PCR_SCHEMA	Required
3	Secure world TCB images	EVTYPE_ARM_TZFW	Required if measured
4	Debug/trace state	EVTYPE_ARM_DEBUG_CONFIG	Required if measured
5	Security lifecycle state	EVTYPE_ARM_NONSECURE_CONFIG	Required
6	Secondary DCE image(s)	EVTYPE_ARM_DCE_SECONDARY	Required if measured
7	Separator signifying the end of DCE measurements.	EVTYPE_ARM_SEPARATOR	Required

Table 39: Default PCR[18] schema details

Order	Component	Event Type	Usage

1	PCR schema value in the DRTM_PARAMETERS	EVTYPE_ARM_PCR_SCHEMA	Required
2	Public key used to authenticate the DCE image(s)	EVTYPE_ARM_DCE_PUBKEY	Required if measured
3	DLME image	EVTYPE_ARM_DLME	Required
4	DLME image entry point	EVTYPE_ARM_DLME_ENTRY_POINT	Required
5	Separator signifying the end of DCE measurements.	EVTYPE_ARM_SEPARATOR	Required

4.8.4 DLME Authorities PCR schema

The DLME Authorities PCR schema is available if DLME image authentication (see section 4.5.1) is supported by the DRTM implementation. Like the default schema (see section 4.8.3) it uses PCR[17] and PCR[18].

The goal of the DLME Authorities schema is to provide a stable PCR[18] based on the measurement of the public key used to authenticate the DLME image and the DLME's security version number.

PCR[17] is identical between the default schema and DLME Authorities schema.

Table 40: DLME Authorities PCR schema requirements

ID	Requirement
R48030	For the DLME Authorities PCR usage schema, the components or items measured, the measurement order, and event type must comply with the requirements in Table 38 for PCR[17] and Table 41 for PCR[18]. See Event types, 3.16.2, for details of how each event type is logged.

Table 41: DLME Authorities PCR[18] schema details

Order	Component	Event Type	Usage
1	PCR schema in DRTM_PARAMETERS	EVTYPE_ARM_PCR_SCHEMA	Required
2	Public key used to authenticate the DCE image(s)	EVTYPE_ARM_DCE_PUBKEY	Required if measured
3	Public key used by the DCE to authenticate the DLME.	EVTYPE_ARM_DLME_PUBKEY	Required if DLME image authentication succeeds
4	DLME image security version number.	EVTYPE_ARM_DLME_SVN	Required if DLME image authentication succeeds and an SVN is present in the DLME image
5	DLME image entry point	EVTYPE_ARM_DLME_ENTRY_POINT	Required

6	Separator signifying the end of DCE measurements.	EVTYPE_ARM_SEPARATOR	Required

5 System requirements

This section describes the hardware and software requirements needed to support DRTM.

5.1 Processing elements

In this specification, a processing element (PE) refers to an Arm core on which an operating system or hypervisor runs, not cores that function as devices or peripherals. Table 42 specifies the requirements for the application processor PEs to support DRTM:

Table 42: System requirements for PEs

ID	Requirement
R51000	PEs must implement the Armv8-A or Armv9-A Architecture [1].
R51010	PEs must comply with the requirements in the Arm Base System Architecture 1.0 [7].

5.2 Multiple sockets

The rules and requirements in this specification apply equally to single- and multi-socket systems. The expectation is that the boot PE and security coprocessor on which the DRTM boot flow is performed can perform the needed checks and tasks defined by the architecture regardless of which socket a resource might be located on.

5.3 SMMU and DMA capable devices

As section 2.8 describes, a key security requirement of this architecture is that the DLME and supplemental images be protected against DMA attacks during a dynamic launch.

Table 43 specifies the requirements related to SMMU and DMA capable devices:

Table 43: System requirements for SMMUs

ID	Requirement
R53000	All DMA capable devices in a system must be behind an SMMU, including both Secure and Non-secure devices. The requirement does not apply to non-host platforms.
R53010	An SMMU implementation must be compliant with the requirements specified in the Arm Base System Architecture 1.0 [7].

5.4 Non-host platforms

A non-host platform (NHP) is a controller, peripheral, or PE in the system that is part of the system's TCB and has the ability to access memory without the protection of an SMMU. A non-application PE is a non-host platform.

Table 44: System requirements for non-host platforms

ID	Requirement
R54000	If a NHP can access Non-secure memory at an address chosen by or influenced by software running at a Non-secure privilege level the NHP must provide a means for the DRTM implementation to quiesce, disable, or reset the NHP function that results in Non-secure memory accesses.

5.4.1 GIC

An Arm Generic Interrupt Controller (GIC) implementation might include support for Locality-specific Peripheral Interrupts (LPIs) or an Interrupt Translation Service (ITS). These features can result in Non-secure memory accesses when an interrupt is triggered and due to this the GIC is treated as a Non-host platform.

Table 45: System requirements for GIC

ID	Requirement
R54010	A GIC that implements Locality-specific Peripheral Interrupts (LPIs) should support clearing GICR_CTLR.EnableLPIs.
R54020	A GIC implementation that does not support clearing GICR_CTLR.EnableLPIs after it is set must not permit modification of GICR_PENDBASER when GICR_CTLR.EnableLPIs == 1.

5.4.2 Hardware trace

A system might support an embedded hardware trace capability where captured trace data is written into system memory. Trace can be controlled by off-chip means or by on-chip self-hosted trace software. A hardware trace feature that accesses system memory is considered a non-host platform and is considered part of the TCB of the system.

Table 46: System requirements for trace

ID	Requirement
R54030	In a deployed lifecycle state, if it is possible for hardware trace to write trace data to Non-secure memory, the system should provide a means for a DRTM implementation to detect and measure whether trace is enabled or not.

Note: If the enabling of trace cannot be detected by a DRTM implementation or if trace can be enabled dynamically without a system reset the DRTM implementation measures trace as enabled.

5.5 Security lifecycle

A system must ensure that the protection of assets and the availability of system functions follow a prescribed and constrained path from manufacture to system disposal. Therefore, the system must have a state machine that it can use to make appropriate security decisions within a particular context. This is known as a security lifecycle.

Each security state in the security lifecycle defines the security properties of the system. The security state depends on software measurements, hardware configuration, debug mode, and the lifecycle phase. For example, a lifecycle state can cover scenarios such as development, provisioning, deployment, returns, and end-of-life. Table 47 specifies the lifecycle requirements.

Table 47: System requirements for lifecycle

ID	Requirement
R55000	A system must have a security lifecycle that can be read and measured by the D-CRTM or DCE and must include a way to distinguish between deployed and pre-deployment states.

5.6 TPM

5.6.1 TPM requirements

This architecture permits the following implementations of a TPM 2.0:

- Discrete chip
- Implementation in a hardware enclave in the SoC. A hardware enclave implementation executes on a coprocessor other than the PEs that initiated DRTM.
- Firmware implementation in the Secure world

Table 48 specifies the requirements for the TPM implementation:

Table 48: System requirements for TPMs

ID	Requirement
R56000	A system must have a TPM implementation that is compliant with the TCG PC Client Platform TPM Profile Specification for TPM 2.0 [9].
R56010	A discrete TPM 2.0 chip must be certified by TCG.
R56020	The system must support hardware-based enforcement to protect locality 4 of the TPM, and it must not be possible for locality 4 to be accessed by Non-secure privilege levels. Implementation note: For a firmware-backed implementation of DRTM, a common approach to protect locality 4 is to make a discrete TPM chip a Secure device and accessible only at Secure privilege levels. This approach allows a TPM service running at a Secure privilege level to mediate access to the TPM.
R56030	The platform interface to access the TPM must support the localities 1-4 of the TPM.
R56040	The implementation must support closing localities 2 and 3. See section 5.6.2.
R56050	A system-level reset must reset the TPM.

It is beyond the scope of this specification to define the security properties for a firmware TPM implementation or for a hardware enclave-based implementation. The TPM implementation is a critical component of the TCB of a system and system level threat modeling is required to evaluate possible threats.

5.6.2 Closing localities

This architecture defines the concept of closing TPM localities. When a locality is closed it means that an attempt by software to request that locality is ignored by the TPM. Localities 3 and 2 can be closed to prevent further extend operations into PCRs 17 and 18 following the dynamic launch. The DCE always closes locality 3. If preventing further measurements into PCRs 17 and 18 is a requirement of the system's security policy, the DLME can close locality 2 using the DRTM_CLOSE_LOCALITY function.

Table 49 specifies requirements for the TPM implementation with respect to closing localities:

Table 49: Requirements for closing TPM localities

ID	Requirement
R56060	When a locality is closed: • For a FIFO interface, the TPM must ignore writes to the TPM_ACCESS_x register. • For a CRB interface, the TPM must ignore writes to the TPM_LOC_CTRL_x register. • The behavior of all other TPM registers must conform to the definition in the TCG PTP specification [9] when the active locality is “Not Set”. See the PTP sections “FIFO Interface Locality Usage per Register” and “CRB Interface Locality Use per Register”.
R56070	Following a system reset, the TPM must treat the dynamic localities 1, 2, and 3 as closed.
R56080	The TPM must provide a means for the D-CRTM to open localities 1, 2, and 3 as part of the dynamic launch sequence.
R56090	The TPM implementation must support closing locality 3. See the DRTM_CLOSE_LOCALITY function, section 3.6.
R56100	The TPM implementation must support closing locality 2. See the DRTM_CLOSE_LOCALITY function, section 3.6.
R56110	After a locality is closed, an attempt by a TPM client to access the closed locality must result in an error returned by the interface used to send commands to the TPM.

How the closing of localities is implemented is platform-specific. As section 5.6.1 describes, access to the TPM implementation must be mediated. The mediation layer that clients use to access the TPM is responsible for enforcing access to closed localities.

5.7 ACPI

This specification assumes the platform uses an ACPI-based system description. Table 50 specifies the requirements for ACPI:

Table 50: System requirements for ACPI

ID	Requirement
R57000	The system must implement a system description based on ACPI that is compliant with the definition in the Base Boot Requirements specification [10].
R57010	If Non-secure firmware updates or patches to ACPI tables that impact the TCB, the firmware should provide measurements of the tables to the DRTM implementation using the

	DRTM_SET_TCB_HASH function. This step must occur while the Non-secure firmware is in a phase of execution where all Non-secure firmware components are under the control of the platform manufacturer. The implication of this requirement is that Non-secure firmware should not perform late patching or updates to ACPI tables that impact the TCB.

5.8 DMA protection

A key security requirement of this architecture is that the DLME is protected against DMA by devices during the dynamic launch. The DLME might need to measure and validate supplemental images and these images must also be protected from DMA. The DLME and supplemental images all reside in Non-secure memory.

DMA protection is specified by the DCE preamble and protections are enabled at the time of the dynamic launch. The protections must remain enabled during the DRTM phases until explicitly removed by the DLME.

Complete DMA protection is hardware-based enforcement at the SMMU that blocks all DMA from Non-secure devices. A DRTM implementation advertises this capability using the DRTM_FEATURES function and a DRTM client requests the protection through the DRTM_PARAMETERS.

Region-based DMA protection is a platform-specific mechanism that provides hardware-based enforcement against DMA for a number of memory regions. A DRTM implementation advertises this capability and the number of regions supported using the DRTM_FEATURES function. A client defines a memory protection table specifying the physical address and size of each region to be protected and passes the table in the DRTM_PARAMETERS. Region-based protection allows DMA to continue during the dynamic launch while maintaining the protections requested in the protection table. If necessary for platform-specific reasons, the DRTM implementation might protect a subset of the regions requested. The DLME can determine all memory regions that are protected using the DLME data.

The DLME disables the protections with the DRTM_UNPROTECT_MEMORY function following the dynamic launch.

Table 51: System requirements for DMA protection

ID	Requirement
R58000	A platform must support at least one type of DMA protection specified in this architecture.
R58010	For a firmware-backed implementation the DMA protection, whether complete or region-based, must prevent DMA accesses from Non-secure devices.
R58020	For region-based DMA protection, DMA must be permitted to regions of memory not included in the protected regions, subject to previously configured SMMU stream tables, context descriptors, and translation tables.

5.9 Platform

Some platform implementations can include mechanisms in the SoC interconnect that enable configuration of the physical address map of the system or how interconnect transactions are routed. Any such configuration must only be possible from the Secure world.

Table 52: Platform configuration requirements

ID	Requirement
R59010	It must not be possible for components operating at Non-secure privilege levels to change the physical memory map of a system. This requirement includes I/O regions such as a PCIe ECAM. This requirement does not apply to the programming of PCI BARs.

Physical memory aliasing is where a system permits configuration of the physical address map of the system so that a single memory location can be accessed through multiple physical addresses.

Table 53: Physical memory aliasing

ID	Requirement
R59020	A platform must not have memory that is physically aliased.

5.10 Firmware

It is assumed that system implements best practices to establish and protect the integrity of system firmware.

Table 54: Firmware integrity requirements

ID	Requirement
R510000	For a firmware-backed DRTM implementation the integrity of the D-CRTM must be established by the system using measurements or digital signatures.
R510010	If the DRTM_SET_TCB_HASH function is used by Non-secure firmware to report hashes of TCB-critical data, this brings the Non-secure firmware component into the new TCB established by DRTM. In this case, the integrity of the Non-secure firmware must be established using measurements or digital signatures.

A platform reset attack or cold boot attack is where an attacker forcibly resets a system without a clean shutdown and then attempts to extract secrets from the system's memory.

Table 55: Requirements to mitigate platform reset attacks

ID	Requirement
R510020	A platform must provide a mechanism to mitigate against platform reset attacks. Arm recommends at a minimum an implementation of the MEM_PROTECT function specified in PSCI 1.1 [2].

5.11 Dynamic launch errors

Dynamic launch errors can involve entering error remediation and a system-level reset. See section 4.7. In this case the DRTM implementation records an error code in a location that persists across the system reset. Following reset a DRTM client can then retrieve the error code using DRTM_GET_ERROR. For a successful dynamic launch, the DCE clears the error code before transferring control to the DLME.

Table 56: System requirements for DRTM launch errors

ID	Requirement
R511000	A platform must provide a means to store a 64-bit DRTM error code that persists across an error remediation-initiated system reset. The error code should be tamper-resistant with respect to the Normal world.
R511010	The error code may be cleared by the platform as part of a power-on-reset or a system reset not caused by DRTM error remediation.

5.12 SMCCC and PSCI requirements

The Arm SMC Calling Convention (SMCCC) [3] defines a common calling mechanism for SMC calls. The Arm Power State Coordination Interface [2] specification defines a mechanism to determine the SMCCC version implemented by firmware.

Table 57: SMCCC and PSCI requirements

ID	Requirement
R512000	System firmware must implement PSCI 1.0 or later.
R512010	System firmware should implement the MEM_PROTECT function specified in PSCI 1.1.
R512020	System firmware must implement SMCCC 1.0 later. Note: The version of SMCCC can be determined using the PSCI_FEATURES function with the SMCCC_VERSION function ID. See the “Discovery of Arm Architecture Service functions” in the SMCCC specification appendices.

5.13 Secure services

A system can contain Secure services that execute on PEs at Secure privilege levels. Secure services can be part of the TCB of a system, and so it is critical for the architecture of a system to comprehend possible threats to DRTM posed by Secure services that can run asynchronously and access Non-secure memory.

Secure services can perform functions based on requests made by Non-secure privilege levels. These requests can be acted on synchronously on the requesting PE or asynchronously if triggered by Secure interrupts. The interactions between the requester and the service might involve data shared in Non-secure memory buffers.

Table 58: Secure services

ID	Requirement
R513000	If a Secure service can be invoked asynchronously and can access Non-secure memory at addresses chosen by or influenced by software running at Non-secure privilege levels, firmware in the Secure execution state must provide a mechanism for the DRTM implementation to quiesce, disable, or reset the Secure service.

6 Hardware-backed implementation

6.1 Hardware-backed overview

As section 2.1 describes, the trusted computing base (TCB) of a system is made up of the components that together enforce the security posture of a system. As section 2.9 describes, hardware-backed DRTM includes in its security scope detecting compromised Secure world components that are part of the TCB of the system, so providing a higher level of assurance in the measurements made by DRTM. Hardware-backed DRTM accomplishes this by the following:

- A dynamic launch transitions control to the D-CRTM which is a trusted agent where the dynamic launch process begins. For hardware-backed DRTM the trusted agent is in a coprocessor separate from the Arm v8-A PE that initiated the launch. This approach protects and isolates the D-CRTM so that it cannot be tampered with even if other Secure firmware in the system might have been compromised.
- System hardware enforces that only the D-CRTM and coprocessor have access to locality 4 in the TPM. This protection provides hardware-based enforcement so that only the D-CRTM can initiate a dynamic launch, reset the dynamic PCRs, and make the initial measurement of the DCE.
- The System's TCB is reestablished. All Secure world firmware components that are part of the system's TCB are reloaded, verified, and measured.
- DMA and interrupts from Secure devices are not permitted.
- The chain of trust rooted in the D-CRTM on the coprocessor extends to the DCE and DLME which run on the boot PE.

From the point of view of the DRTM client, the DRTM architectural interfaces in a hardware-backed implementation remain identical to a firmware-backed implementation. The same SMC functions are used by the DCE preamble to prepare the environment and initiate the dynamic launch. A DRTM firmware component proxies function requests made by the DRTM client running on the PE to the DRTM coprocessor. See the sequence diagram in Figure 10 which shows an example of the flow of a dynamic launch in a hardware-backed implementation.

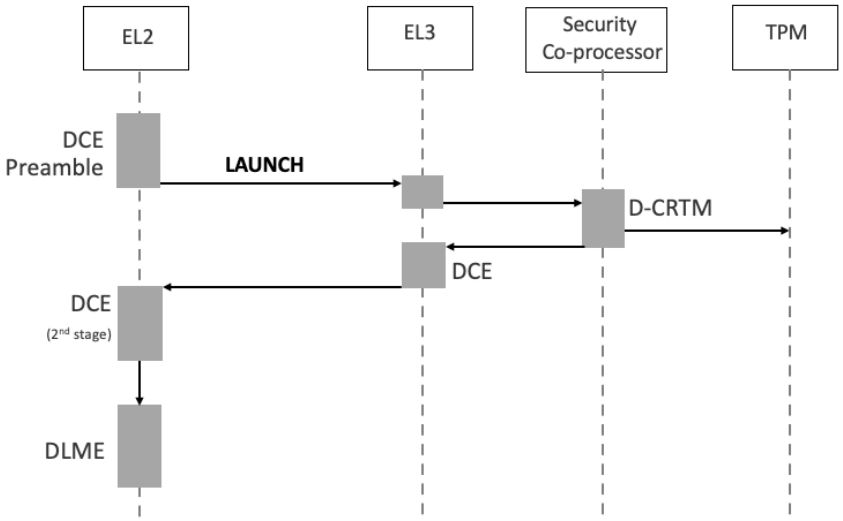


Figure 10: Hardware-backed implementation example

6.2 Hardware-backed D-CRTM requirements

In a hardware-backed implementation, the D-CRTM is in a coprocessor that is distinct from the PE that initiated the dynamic launch. After the DCE Preamble initiates the launch on the boot PE, Secure firmware in the DRTM implementation receives the dynamic launch request and proxies it to the coprocessor where the D-CRTM is located. See section 6.4.2 for further details about the coprocessor that hosts the D-CRTM.

Table 59 describes requirements for the DRTM firmware that transitions the dynamic launch to the D-CRTM in the coprocessor. During this phase of the dynamic launch errors are returned to the caller.

Table 59: DRTM Firmware requirements for transition to D-CRTM

ID	Requirement
R62000	The DRTM implementation must verify that the DCE preamble used PSCI CPU_OFF to turn off all PEs except the boot PE and if not return the return code SECONDARY_PE_NOT_OFF to the caller of DRTM_DYNAMIC_LAUNCH.
R62010	An implementation of DRTM might require the boot PE to be a specific PE in the topology of the SoC. In this case the DRTM implementation must return the error code DENIED if the PE used to initiate the dynamic launch is not the correct one.
R62011	If the caller of DRTM_DYNAMIC_LAUNCH was not in AArch64 state, the DRTM implementation must return the return code DENIED to the caller of DRTM_DYNAMIC_LAUNCH.
R62012	The DRTM implementation must verify that no TPM localities are active, and if not return the return code DENIED to the caller of DRTM_DYNAMIC_LAUNCH. If there is an error accessing the TPM the return code TPM_ERROR must be returned to the caller.
R62013	The DRTM implementation must verify the DRTM_PARAMETERS comply with the requirements defined in Table 10. If an error is detected, the DRTM implementation must return the return code INVALID_PARAMETERS to the caller.
R62014	If a memory protection table is specified in the DRTM_PARAMETERS, the DRTM implementation must verify that the table complies with the requirements defined in Table 12. If an error is detected, the DRTM implementation must return the return code MEM_PROTECT_INVALID to the caller.
R62015	If the DRTM_PARAMETERS request features that are not supported by the DRTM implementation, the DRTM implementation must return the return code INVALID_PARAMETERS to the caller.
R62020	If the DRTM implementation returns an error to the caller of DRTM_DYNAMIC_LAUNCH the state of the system must be left unmodified.
R62030	The DRTM implementation must map the DRTM_PARAMETERS Execute-Never and pass the parameters to the D-CRTM by an implementation-specific method.
R62040	The DRTM implementation must start execution of the D-CRTM by an implementation-specific method. If this fails, the error code COPROCESSOR_ERROR must be returned to the caller.

R62050	If the DRTM implementation encounters an internal error, it must return the error code INTERNAL_ERROR to the caller.

Table 60 specifies the requirements for the D-CRTM in a hardware-backed implementation.

Table 60: Hardware-backed D-CRTM requirements

ID	Requirement
R62060	The D-CRTM must hold all PEs in a reset state where all the logic on which the PE executes is reset, including the integrated debug functionality. This reset state is referred to as a *Cold reset* in the Arm Architecture [1]. Note: this is not referring to a system-level cold boot or reset which conventionally refers to a full system restart following the removal of power from the system.
R62070	The D-CRTM must verify no TPM localities are active. If this is not the case it must enter remediation.
R62080	The D-CRTM must reset the dynamic PCRs in the TPM using TPM_HASH_START or equivalent at locality 4.
R62090	The D-CRTM must open localities 1, 2, and 3 in an implementation-specific way.
R62100	The D-CRTM must measure the PCR schema passed in the DRTM_PARAMETERS and record the measurement in the event log using the event type EVTYPE_ARM_PCR_SCHEMA. See PCR schema details in section 4.8.
R62110	The D-CRTM must prepare a tamper-resistant memory space for the DCE to execute from. The memory space must be protected against Secure devices that are DMA capable.
R62120	The D-CRTM must load the DCE image and must enforce verification of the cryptographic signature on the DCE.
R62130	The D-CRTM must extend a measurement of the DCE image into the TPM and record the measurement in the event log using the event type EVTYPE_ARM_DCE. See PCR schema details in section 4.8.
R62140	The D-CRTM must extend a measurement of the public key used to authenticate the the DCE image into the TPM and record the measurement in the event log using the event type EVTYPE_ARM_DCE_PUBKEY. See PCR schema details in section 4.8.
R62150	The D-CRTM must detect if external debug mechanisms are enabled in the platform. If external debug is detected to be enabled, the D-CRTM must record a measurement of the debug state into the TPM and record the measurement in the event log using the event type EVTYPE_ARM_DEBUG_CONFIG. See PCR schema details in section 4.8. If external debug can be enabled dynamically without a system reset, the D-CRTM must measure debug as enabled.
R62160	If it is possible in a deployed lifecycle state for a hardware trace feature to write trace data to Non-secure memory and if the platform supports a mechanism that permits the D-CRTM to detect if hardware trace is enabled, the D-CRTM must detect this and extend a Boolean

	value into the TPM and record the measurement in the DRTM event log with the event type EVTYPE_ARM_DEBUG_CONFIG. See PCR schema details in section 4.8. If the enabling of trace cannot be detected by the D-CRTM or if trace can be enabled dynamically without a system reset the D-CRTM must measure trace as enabled.
R62170	The D-CRTM may detect and measure the security lifecycle state of the platform into the TPM and record the measurement in the event log with the event type EVTYPE_ARM_NONSECURE_CONFIG. See PCR schema details in section 4.8. Note: if the D-CRTM does not make this measurement, it must be done by the DCE.
R62180	The D-CRTM must release the boot PE from reset and cause execution to begin at the DCE entry point.
R62190	If the D-CRTM is unable to log a measurement because there is no available space in the event log region, the D-CRTM must extend a hash of the 1-byte value 0xFF into PCR[17] and PCR[18] and enter remediation.
R62200	If the D-CRTM encounters an error, it must enter remediation. See section 4.7.
R62210	The D-CRTM must only be updateable through a secure firmware update procedure that meets the requirements specified in the Arm Platform Security Boot Guide [5].
R62220	Updates to the D-CRTM must be protected against rollback as specified by the requirements in the Arm Platform Security Boot Guide [5].

6.3 Hardware-backed DCE requirements

After the D-CRTM has released the boot PE from reset, execution begins in the DCE. Table 61 defines requirements specific to the DCE in a hardware-backed implementation.

Table 61: Hardware-backed DCE requirements

ID	Requirement
R63000	DCE components that execute at the Secure EL0, Secure EL1, Secure EL2, or EL3 privilege levels must abort DRTM and enter remediation if an exception occurs during DCE execution. DCE components that execute at Secure privilege levels may mask asynchronous interrupts.
R63010	All Secure world firmware components that are part of the system's TCB must: • Be loaded by the DCE with verification of the cryptographic signature of the component. • Be measured by the DCE with the image measurement extended into the TPM and recorded in the event log with event type EVTYPE_ARM_TZFW. See PCR schema details in section 4.8. Note: These requirements can preclude a system from using TCB components that are dynamically loaded.
R63020	Any security critical state in the system's TCB that is preserved across the dynamic launch must be verified.

R63030	The DCE must ensure that DRTM operation on the boot PE cannot be preempted by Non-secure asynchronous exceptions unless explicitly enabled by the DLME. Examples include interrupts, SError exceptions, and SDEI events.
R63040	If the SoC implements a GIC ITS (Interrupt Translation Service [14]), the DCE must: • Ensure all ITSs are disabled: GITS_CTLR.Enabled = 0 and GITS_CTLR.Quiescent = 1 • Ensure LPIs are disabled in all Redistributors: GICR_CTLR.EnableLPIs = 0 and GICR_CTLR.RWP = 0 These requirements ensure that the GIC makes no memory accesses during the dynamic launch. Implementation Note: In some GICv3.0 implementations it might not be possible to clear GITS_CTLR.Enabled after it is set. For this case the DCE must: • Verify the location and size of the LPI Configuration tables and the LPI Pending tables to ensure that they do not overlap any DRTM-protected memory region. • Verify that GICR_PENDBASER and GICR_PROPBASER do not violate any of the rules specified in the GIC architecture that lead to behavior defined as UNPREDICTABLE.

6.4 Hardware-backed system requirements

6.4.1 TPM

Section 5.6 describes the base requirements for a TPM implementation. Table 62 specifies additional TPM requirements for a hardware-backed implementation of DRTM.

Table 62: TPM requirements in a hardware-backed implementation

ID	Requirement
R64000	The TPM 2.0 must be a discrete chip or hardware enclave implementation.
R64010	Hardware must enforce access to locality 4 so that it is not possible for locality 4 to be accessed by anything in the system except by the D-CRTM.
R64020	If the TPM implementation is a discrete chip, it must not be possible for any component in the system, except for the coprocessor, to bypass the system's locality enforcement mechanism and directly access the bus to which the TPM is connected.
R64030	A mechanism must be provided to close localities 2 and 3.

6.4.2 Coprocessor

In a hardware-backed implementation, the DRTM process is rooted in a D-CRTM that executes on a security coprocessor that is distinct from the PE that initiated the dynamic launch. The D-CRTM is responsible for preparing the DCE for execution, measuring the DCE, and transferring control to the DCE.

The hardware-backed D-CRTM requirements are specified in section 6.2.

Table 63 specifies the requirements for the DRTM security coprocessor:

Table 63: Coprocessor requirements

ID	Requirement
R64040	The coprocessor must be a compute element isolated from all other hardware agents, including the application PEs. The coprocessor must not share caches with the application PEs.
R64050	The coprocessor must be tamper-resistant. The specific tamper-resistant properties required depend on the system's threat model.
R64060	The coprocessor must host the D-CRTM code that starts the dynamic launch.
R64070	The coprocessor must provide an interface that allows the DRTM firmware to make function requests and get return codes for DRTM functions handled by the coprocessor.
R64080	The coprocessor must support a verified boot process to ensure the integrity of the D-CRTM.
R64090	The coprocessor must support a secure firmware update process that ensures that only authorized updates to the D-CRTM are allowed.
R64100	The coprocessor must have rollback protection for firmware, so the D-CRTM can be protected.
R64110	The coprocessor must be able to prevent external debug of the coprocessor itself.
R64120	The coprocessor must be able to detect if external debug mechanisms are enabled.
R64130	The coprocessor must be able to issue a Cold reset, as defined in the Arm Architecture [1], to the Arm v8-A PEs on which DRTM is being performed.
R64140	The coprocessor must be able to access the memory containing the DCE image.
R64150	The coprocessor must be able to cause the boot PE to execute the DCE entry point.
R64160	The coprocessor must be able to record a persistent error code if an error occurs during the dynamic launch before entering remediation.
R64170	The coprocessor must be able to initiate a system-level reset if an error occurs during the dynamic launch to complete remediation.

Table 48 in section 5.6.1 specifies the system requirements for TPM devices. Table 64 specifies additional requirements for a hardware-backed implementation for how the security coprocessor relates to the TPM:

Table 64: Requirements for TPMs in a hardware-backed implementation

ID	Requirement
R64180	The coprocessor must have exclusive access to access locality 4 of the discrete TPM.

6.4.3 DMA protection

Table 65 specifies DMA protection requirements for a hardware-backed implementation.

Table 65: DMA protection requirements

ID	Requirement
R64190	Whether complete or region-based, the DMA protection must prevent DMA accesses from both Non-secure and Secure devices.

6.5 Hardware-backed DRTM function requirements

6.5.1 DRTM_DYNAMIC_LAUNCH

The following are hardware-backed requirements for the DRTM_DYNAMIC_LAUNCH function:

- The dynamic launch request must be proxied to the D-CRTM in the coprocessor using an implementation-specific method.
- The DMA protections put in place by the caller must prevent DMA accesses from both Non-secure and Secure devices.