DRTM Architecture for Arm

Document number:	ARM DEN 0113
Release Quality:	BETA
Issue Number:	1
Confidentiality:	Non-confidential
Date of Issue:	November 16, 2022

Contents

About this document

Release Information

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

Date	Issue	Confidentiality	Change
February 3, 2022	Beta 0	Non-confidential	Initial beta release.
November 16, 2022	Beta 1	Non-confidential	Beta 1 updates • Misc minor clean-up • Close locality: Precisely define meaning of close • Close locality: Dynamic localities must be closed by default, clarifications on error conditions • Close locality: Locality must be relinquished before closing it • Locality 2 must be active at DLME entry • Updates to error codes • Hash algo ID changed from 32 to 16 bits • Set TCB hash: Updates to rationale and guidance for using SET_TCB_HASH • Set TCB hash: Replace Hash ID Namespace bit in table entry with Source of Entry bit so DLME can differentiate entries from NS firmware • Clarify that CRB must be protected from DMA • TPM bus must be protected in hardware-backed implementation so that locality enforcement is not bypassed • ACPI: To support SSDT allow multiple ACPI tables in TCB hash table • ACPI: Optionally allow ACPI tables to be provided in DLME data instead of hashes of tables. • Permit optional signature check of DLME • Region-based DMA protection: If necessary for platform-specific reasons, the DRTM implementation may protect a subset of the regions requested. • Region-based DMA protection: increase number of regions to 16-bits • Region-based DMA protection: number of regions advertised in FEATURES is no longer zero-based and starts with 1. Number of regions can be zero, meaning no regions are specified in the parameters and DRTM implicitly protects some number of regions.

	• Split D-CRTM requirements into 2 separate phases to differentiate requirement that change the state of the machine. • Consolidate memory protection requirements for the DCE into one table. • Added new section to system requirements specifying assumptions about firmware integrity

DRTM Architecture for Arm

Non-Confidential Proprietary Notice

References

This document refers to the following documents.

Ref	Document Number	Title
[1]	Arm DDI 0487	Armv8-A 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 1.04, June 3, 2019
[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.
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.
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	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

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 its documentation.

Feedback on this book

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

- The title (DRTM Architecture for Arm).

- The number and issue (DEN 0113 beta 0).
- The page numbers to which your comments apply.
- The rule identifiers to which your comments apply, if applicable.
- A concise explanation of your comments.

Arm also welcomes general suggestions for additions and improvements.

1 Overview

1.1 Introduction

This specification defines an architecture for Dynamic Root of Trust for Measurement (DRTM) for Armv8-A processors. 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, Architecture Overview, provides an overview of 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, defines 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, defines requirements specific to a hardware-backed implementation of DRTM.

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

2 Architecture Overview

2.1 Background

The trusted computing base, or TCB, of a system consists of all hardware, firmware, and software components in the system that are relied upon 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 digest (or measurement) of the image is computed. That measurement may be immediately verified using a digital signature or it could be 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 may be measured as well. This process forms a chain of trust that is anchored in the immutable bootloader and 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, or CRTM. When the CRTM executes on a processor it is referred to as the Root of Trust for Measurement or RTM.

In a typical measured boot flow, execution begins with the CRTM and then continues by loading and executing additional bootloader stages, executing UEFI firmware, executing an OS loader, and finally 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 or SRTM. See example in Figure 1.

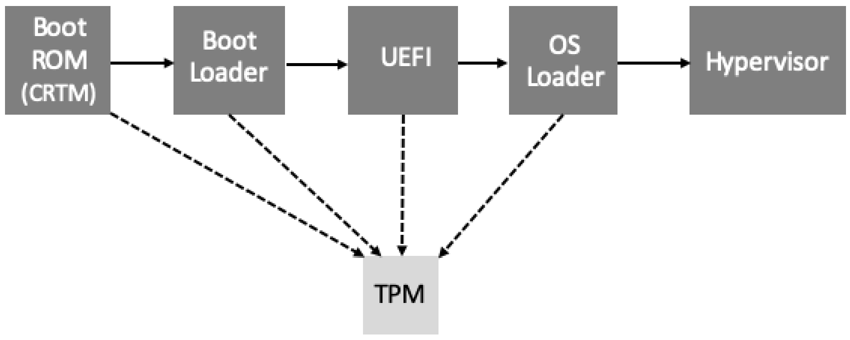


Figure 1, Example SRTM Boot Flow

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 could compromise security.

For example, UEFI is an extensible boot loader, where multiple EFI programs may run during the Boot Services phase. This may include drivers, device option ROMs, and a bootloader. If compromised, these programs may 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

Once 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, which allows DRTM to be initiated multiple times without a system reset.

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

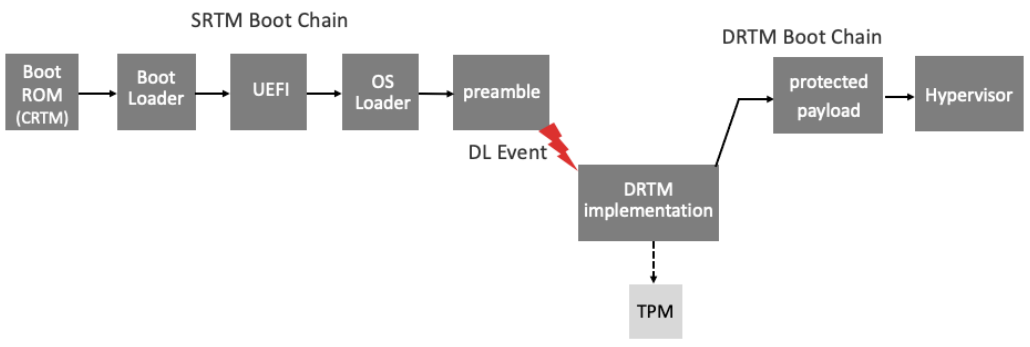


Figure 2, SRTM and DRTM Boot Chain Illustration

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 prior to the DL Event. A dynamic launch can be initiated any number of times on a running system.

2.3 DRTM Terminology

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, and preparing input parameters needed by DRTM. Finally, it invokes 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 may conform to a standardized boot protocol.

2.3.2 D-CRTM and DCE

A DRTM implementation is comprised of two logical components:

- the *Dynamic Code 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, and transfers control to it.

The DCE executes on an application core. The DCE verifies the system's state, measures security-critical attributes of the system, prepares the memory region for the target payload, measures the payload, and finally 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 that decides whether the system is in the expected state.

The DLME may be an operating system specific component supplied by the same vendor that provided the DCE Preamble, or it may 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 a *non-host platform*. A non-host platform is trusted and is part of the system's TCB.

In this specification the term *device* is used to refer to peripherals as defined above. The term non-host platform has the same usage as in the TCG specification.

2.4 DRTM on Arm

2.4.1 Overview

Armv8-A processors have two security states, Secure and Non-Secure, as illustrated in the example diagram in Figure 3.

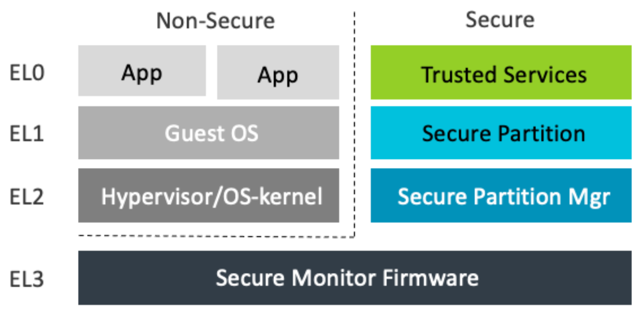


Figure 3, Exception Levels and Security States Example

As described in section 2.2 the objective of DRTM is to instantiate a smaller TCB that excludes untrusted and arbitrarily extensible components by measuring and launching a protected payload. On Arm v8-A the payload is an image that executes at the highest Non-secure privilege level.

A typical boot scenario for DRTM would be 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. See the sequence diagram in Figure 4 which illustrates an example of a DRTM implementation on an Arm system. See section 2.5 which describes further details about possibly implementation approaches.

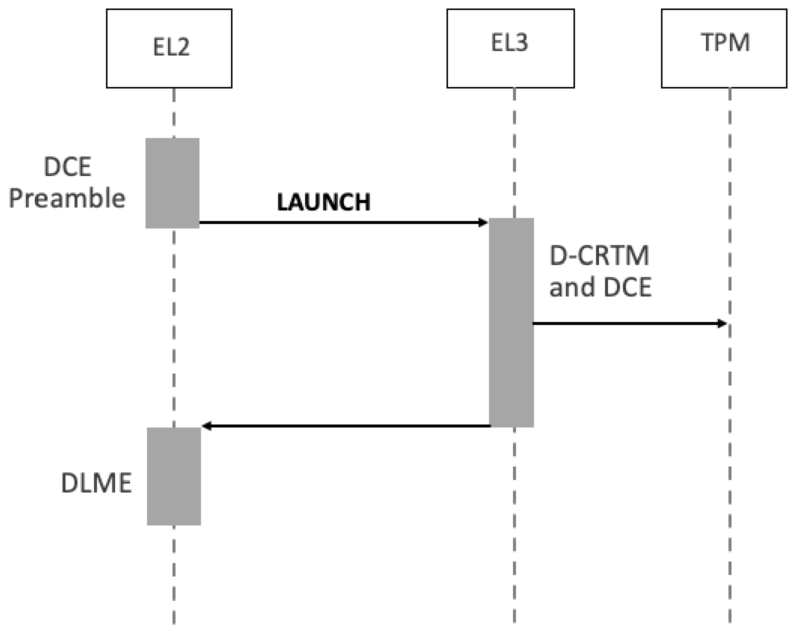


Figure 4, Example Dynamic Launch Flow on Arm

2.4.2 DRTM SMC Functions

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

See the summary in Table 1 and the sequence diagram illustrating an example use of the functions in Figure 5.

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 prior to the dynamic launch. See section 3.9.
DRTM_LOCK_TCB_HASHES	Used by firmware to prevent additional hashes from being recorded. See section 3.10.

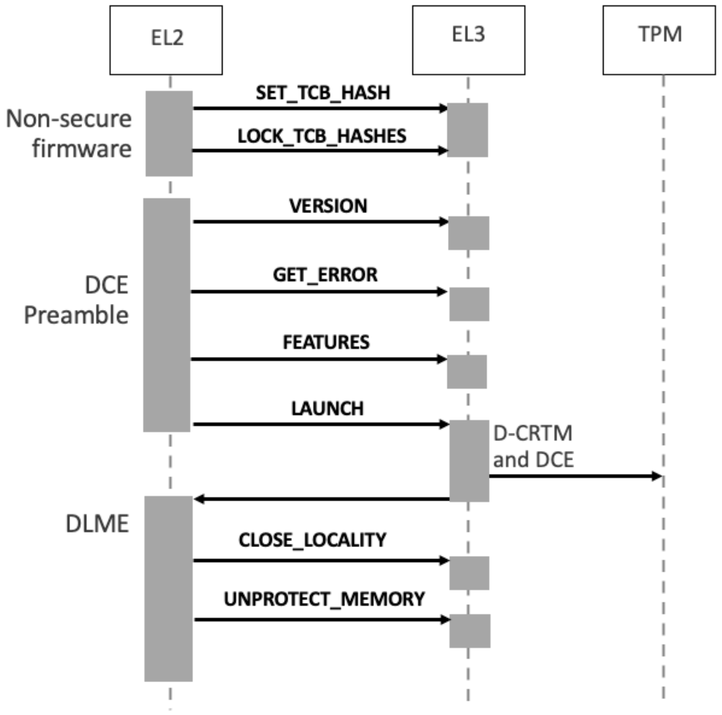


Figure 5, Example of SMC Function Usage

2.5 Firmware and Hardware-backed Implementations

As described in section 2.3.2, the D-CRTM must be located 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

In a firmware-backed implementation the D-CRTM is implemented in EL3 firmware. How additional 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 in where the DCE is implemented. Options include:

- A firmware implementation where the D-CRTM and the DCE are implemented together 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 located in Non-secure firmware as shown in Figure 7.

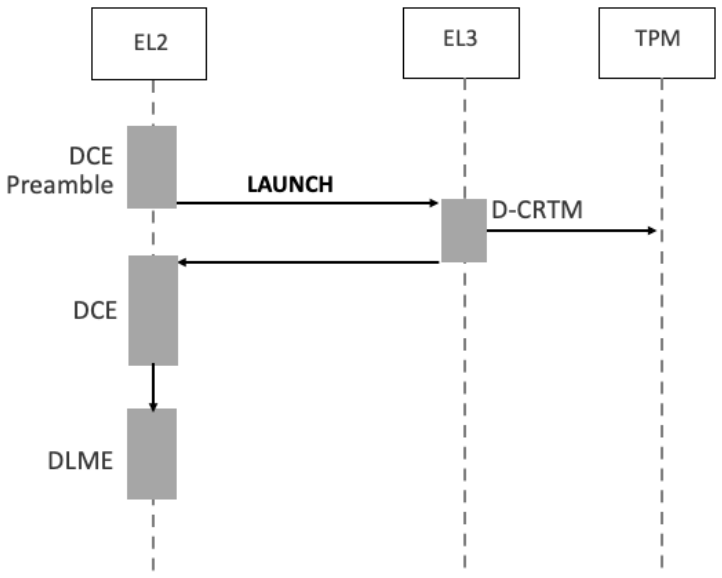


Figure 6, Firmware-backed Implementation Dynamic Launch

2.5.2 Hardware-backed

In a hardware-backed implementation the D-CRTM is located in a co-processor 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. Requirements for a hardware-backed implementation are described in section 6.

See the sequence diagram in Figure 7 which illustrates how DRTM components may map to PE privilege levels and the security co-processor in a hardware-backed implementation.

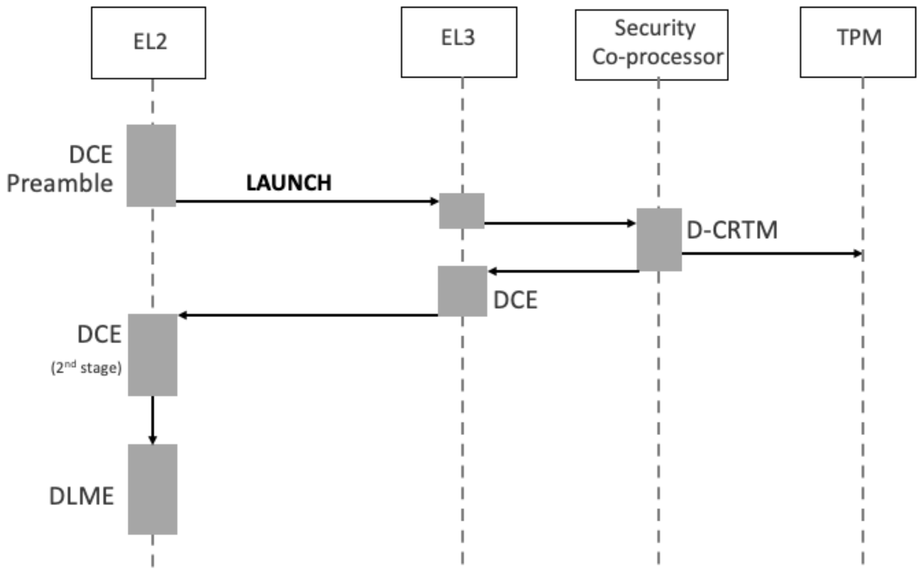


Figure 7, Hardware-backed Implementation Dynamic Launch

2.6 Differences from the TCG DRTM Specification

As described previously, 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.
- With the exception of 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 is replaced by the DRTM parameters (see section 3.12).
    - DLME_ARGUMENTS. This 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 around measurements that must be made by the DRTM implementation into a TPM during the dynamic launch. The TPM implementation used 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 directly access locality 4. For a firmware-backed implementation of DRTM, a typical approach for managing the TPM would be 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 the platform provide hardware enforcement of access to locality 4 so that no component in the system may access locality 4 except the D-CRTM in the security co-processor.

For a firmware-backed implementation of DRTM, it is permissible to 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.

See section 5.6 for details of 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

The sections that follow 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 may optionally support TPM-based measurements. With this approach the D-CRTM and DCE make measurements using the TPM command TPM2_PCR_Event, which sends the data to be measured to the TPM and the digest computation is performed by the TPM itself. TPM2_PCR_Event computes the digest and 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. This provides an advantage because the DRTM client has maximum flexibility in choosing which digest algorithm to base a security policy upon. The only limitation is the algorithms supported by the TPM.

A disadvantage of the TPM-based approach is that it may 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).

The default schema for usage of dynamic PCRs is summarized in Table 2. Details for the default schema are described in section 4.8.2.

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 be protected from DMA by devices during the dynamic launch. The DLME may 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 is responsible for setting 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 additional steps to verify the state of the system, the DLME removes the memory protections using the DRTM_UNPROTECT_MEMORY function.

2.9 Security Considerations

2.9.1 Overview

As described previously, the objective of DRTM is to instantiate 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 examines 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.

2.9.2 Security Goals

The key security goal of DRTM is ensuring the integrity of the DLME and its execution environment by: A) providing reliable measurements of the DLME and key system properties in a TPM device, and B) 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.

2.9.3 Security Non-goals

There is no security goal related to confidentiality in the DRTM architecture, as DRTM does not protect or make use of secrets. However, DRTM depends on a TCG-compliant TPM implementation that does have confidentiality requirements. If the TPM implementation could be compromised, any DRTM-based security policy could 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. The adversary can simply skip initiating the dynamic launch if their goal is denial of service.

The following are not security goals of this architecture:

- Attacks against the availability of a system.
- Defense against a compromised TCB. DRTM depends upon the integrity a TCB consisting of software and hardware components that are trusted such as:
    - Secure devices that may be DMA-capable
    - Non-host platforms
    - The security co-processor in a hardware-backed implementation
    - Secure firmware and Secure services at Secure privilege levels
- Physical attacks against the TPM, including physical man-in-the-middle attacks.
- Glitches of the SoC power supply or clocks during DRTM in order to bypass verification checks.
- Laboratory attacks in which devices are unpackaged and probed.

2.9.4 Stakeholders

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

- Silicon Providers – provides silicon and potentially firmware components
- OEM/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.

2.9.5 Threats and Mitigations

The tables below specify the threats in scope for this architecture, including mitigations addressed by the architecture.

Table 3, Threats in Scope

Threat	**Mitigation**
Adversary compromises NS 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 NS 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 NS 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	There is a system requirement that all DMA-capable devices must be behind an SMMU.

with the Normal world DCE, DLME, or the DLME's supplemental images.	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 pre-empted by Non-secure asynchronous exceptions. Examples include interrupts, Serror exceptions, and SDEI events.
Adversary tampers with the DRTM parameters prior to DL Event.	The DCE verifies the DRTM parameters before use to ensure basic validity. Parameters tampered with could be valid values, and in this case TPM measurements made (for example, of the DLME) during DRTM would enable detection of the tampering. The DLME entry point in the DRTM parameters has its own 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 which 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.

The following threats are 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 security co-processor (D-CRTM) 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 that are re-loaded during the dynamic launch.	Reloaded Secure world firmware is 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 located in the security co-processor.
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: a) masks Secure asynchronous interrupts or b) 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 could be enabled without a reset.

2.9.6 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 would allow an adversary at a Non-secure or Secure privilege level to cause the component to perform a memory access that may compromise DRTM?	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 specifications 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:	Silicon Provider, OEM/ODM

• buffer overflows of privileged buffers • access control bypasses (e.g., 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 may 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 may compromise the end user's security goals.	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 via power control APIs (e.g., 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. Steps that are recommended to be taken by the DLME are described in section 4.6.2. The implementer of the DLME should perform a security analysis to determine what steps are necessary in order to initialize and transition to the OS runtime.	OSV
The DLME may 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 may 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

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 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
	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
	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 0 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 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 0 and the minor version is 1.

The version number is a 31-bit unsigned integer, with the upper 15 bits denoting the major revision, and the lower 16 bits denoting 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, it is possible for revision B to have a higher function count than revision A.

3.2.2 Implementation Responsibilities

If this function returns a valid version number, all the functions that are described 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 or feature 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.

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: Reserved 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 4 KB 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 4 KB 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 may be recorded with DRTM_SET_TCB_HASH. See section 3.9. A value of zero indicates that hashes may not be recorded with DRTM_SET_TCB_HASH.

3.3.1 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 additional feature-specific feature or capability bits available in other return value registers.

See Table 6 for supported feature IDs and features.

For an overview of memory protection see section 2.8.

For an overview of PCR usage and TPM-based and firmware-based measurements see section 2.7.

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 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 of the memory protection request passed in the DRTM_PARAMETERS is invalid.
		`SECONDARY_PE_NOT_OFF`	Secondary PEs were detected to be on.

3.4.1 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 may return 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 may 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 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, and initializing the DRTM parameters.

3.4.3 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 error:	
		`NOT_SUPPORTED`	DRTM is not supported.
		`DENIED`	The system is not in a state where memory can be unprotected.

3.5.1 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 would typically call this function when it has done sufficient verification and initialization so that it can defend itself from DMA attacks.

3.5.2 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.
- 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	X0	0xC400_0115	
uint32 Locality	X1	Specifies the locality to close. Valid values are 2 and 3.	
Return			
int64	X0	On success: `SUCCESS`	
		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.

3.6.1 Usage

This function is used by the DLME to close dynamic localities in the TPM. Localities that may 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 may 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 Caller Responsibilities

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

3.6.3 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 that entered remediation.

Parameter	Register	Value	
uint32 Function ID	W0	0xC400_0116	
Return			
int64	X1	Error code. See error encoding in section 3.11.	
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.

3.7.1 Usage

This function may 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 Caller Responsibilities

The DRTM_GET_ERROR function may be invoked in the DCE Preamble prior to 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. It is the responsibility of the caller to 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 Implementation Responsibilities

If an error is detected during a dynamic launch and the DRTM implementation entered remediation (see section 4.7), it is the responsibility of the implementation to 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 would return the same error code value.

The error code must be cleared by the DCE prior to 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.

3.8.1 Usage

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

3.8.2 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 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 additional hashes.
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 Usage

This function may be called by Normal world firmware prior to 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 table containing one or more hashes.

3.9.2 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 thus 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 prior to 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, as it will be ignored.
- DRTM_SET_TCB_HASH may be invoked multiple times, for example once for each component being measured. Alternatively, a single TCB_HASH_TABLE could be constructed containing all TCB hashes with DRTM_SET_TCB_HASH being invoked once.

3.9.3 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.
- 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 prior to 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 to the DLME 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 additional 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.

3.10.1 Usage

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

3.10.2 Caller Responsibilities

As described in section 3.9, the function DRTM_SET_TCB_HASH may 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 prior to 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 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 will use 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 may define additional 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 co-processor 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 occured at the TPM	0x5
Reserved	0x5 – 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 1 for this revision of DRTM architecture for Arm. For a given DRTM major version number (see DRTM_VERSION, 3.2) this structure will always be 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[32:6]: Reserved. Must be zero. 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) 1-3: Reserved Bit[0]: 0: Request firmware-based hashing (see 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 4 KB 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 4 KB 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.

The following are the requirements for the DRTM parameters:

Table 10, DRTM Parameters Requirements

ID	Requirement
	The DRTM parameters must be located in Non-secure, physically contiguous memory.
	The DRTM parameters must start at a 4 KB aligned address.
	The address ranges described by the parameters must not overlap.
	The address ranges described by the parameters must not wrap around.
	The DLME image size must not extend beyond the bounds of the DLME region.
	The DLME entry point offset must be within the bounds of the DLME image.
	The DLME region must meet the requirements in section 3.14.

	If in use, the Normal world DCE region must be located in Non-secure physically contiguous memory.
	If in use, the Normal world DCE region must start at a 4 KB aligned address.

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. This 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 will always be 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: 1011 1011 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 can't be used

			5 – Reserved 6 – Reserved 7 – Reserved Bits[51:0]: Number of 4 KB 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 4 KB 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 4 KB pages in region N. See definition for region 0.

The following are the requirements for a MEMORY_REGION_DESCRIPTOR_TABLE:

Table 12, Memory Region Descriptor Requirements

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

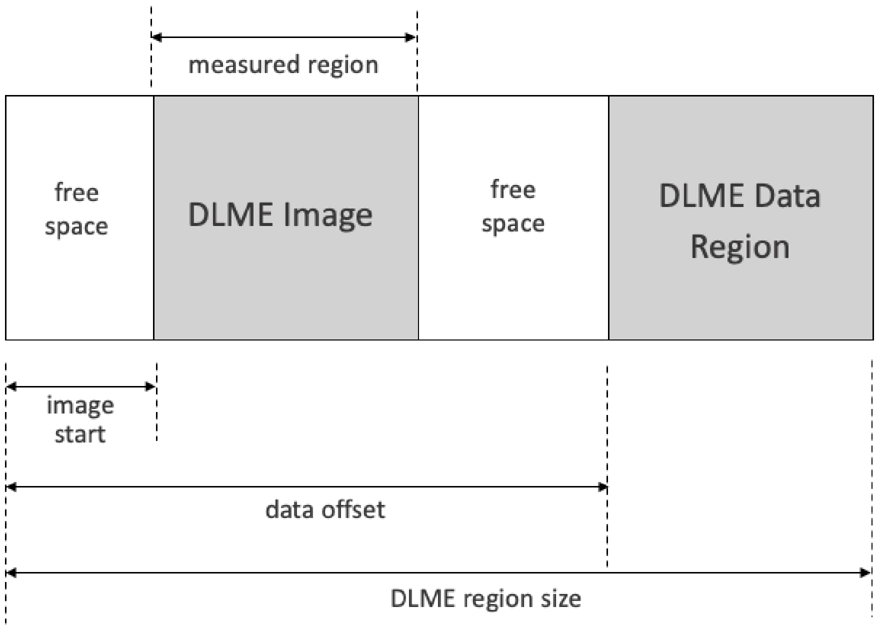
3.14 DLME Region

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 key characteristics of the DLME region are described in the DRTM_PARAMETERS:

- 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



The following are the requirements for the DLME region:

Table 13, DLME Region Requirements

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

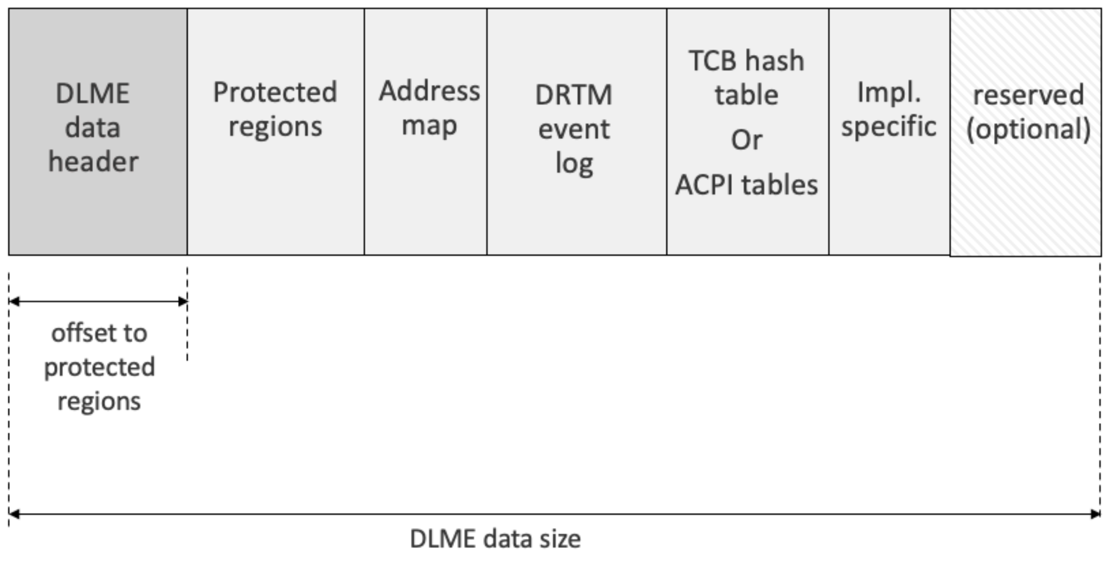
Space in the DLME region that is outside the DLME image and the DLME data may be used by the DCE preamble and the DLME for OS-specific purposes. See the regions labeled “free space” in Figure 8.

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. This 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 will always be 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 may be reclaimed for use by the OS after the DLME image has finished using it.

The following are the requirements for the DLME data:

Table 15, DLME Data Requirements

ID	Requirement
	All the sub-regions referenced by the DLME_DATA_HEADER must be within the DLME region.
	The following sub-regions must be present in the DLME data: • Protected regions • Address map • DRTM event log
	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.
	The sub-regions referenced by the DLME_DATA_HEADER must be in the order described in Table 14 and must not overlap.
	The size of fields in the DLME_DATA_HEADER must not extend beyond the bounds of the DLME data.
	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 “type” field in the memory region descriptors must identify the type of region.
	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.
	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.
	The TCB hash table, if present, must be formatted as a TCB_HASH_TABLE (see section 3.15).
	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.
	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. This 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 will always be 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 ACPI table's signature.
Hash 0 digest	12	X	The digest of the TCB data. This is 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. This is an array of X bytes, with the array length depending on the hash algorithm used.

The following are the requirements for a TCB_HASH_TABLE:

Table 17, TCB_HASH_TABLE Requirements

ID	Requirement
	The table must be located in physically contiguous memory.
	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.
	For table entries with an ID namespace of ACPI, the ID field must be the table's signature.
	The hash algorithm field must reflect the same algorithm used by the DRTM implementation when it performs firmware-based measurements (see 2.7.1).
	The maximum number of entries in the table is defined by the DRTM implementation and advertised in DRTM_FEATURES.
	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. This makes it possible to determine at a later time the set of measurements that make up a PCR value.

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

3.16.1 DRTM Event Log Requirements

The following are the requirements for the DRTM event log:

Table 18, Event Log Requirements

ID	Requirement

	The DRTM event log must be in the DLME data which is allocated by the DCE preamble prior to the dynamic launch event.
	There must be a minimum of 64 KB of space allocated for the event log.
	The DRTM event log must follow the crypto agile event log structure defined in the TCG PC Client Platform Firmware Profile Specification [6].
	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.
	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 that signed the DCE. This event has no event data. • Digest = Hash(public key that signed DCE) • Event data size = 0
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 may be necessary to perform a dynamic launch on systems that have debug or trace enabled. In order 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 may be necessary to perform a dynamic launch on systems that are in a non-deployment security lifecycle state. In order 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. This event has no event data. • 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”

3.17 Return codes

The table below 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 of the 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 of the memory protection request was invalid.	-6
COPROCESSOR_ERROR	There was an error making a request to the co-processor. (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.	

4 Requirements for DRTM Phases

4.1 Non-secure firmware and TCB-critical data

As described in 2.2, 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 may 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 may 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. In order to accommodate a wide range of systems, the DRTM architecture permits Non-secure firmware to report hashes of ACPI tables to the DRTM implementation if it can do this securely before any extensible components are loaded.

For UEFI firmware, prior to 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. Prior to End of DXE the firmware may use the DRTM_SET_TCB_HASH function to record the hashes of ACPI tables that impact the TCB.

The following considerations should be taken into account:

- 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 prior to extensible components being loaded to prevent further updates to the TCB hashes.

Table 21, Non-Secure Firmware Requirements

ID	Requirement
	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.
	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 prior to the End of DXE event.
	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

	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 may not be under the control of the platform manufacturer.
	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 additional level of security.
	Before transitioning to an execution phase where firmware components may 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 may be affected by DRTM. Examples of Secure software that may 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
	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 may have previously occurred and failed. For example, the DCE preamble may set a flag in an OS-specific location indicating the initiation of a dynamic launch, with the DLME clearing the flag.
	The DCE preamble must ensure that all PEs except the boot PE are off using the CPU_OFF PSCI function [2].
	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.
	The DCE preamble must load the DLME image into the DLME region.
	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 additional details.

	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.
	The DCE preamble must prepare the DRTM_PARAMETERS. See section 3.12 for the requirements for the DRTM_PARAMETERS.
	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).
	If region-based DMA protection is in use, the DCE Preamble may request protection of other memory regions besides the DLME region.
	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.
	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.
	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.
	The DCE preamble must ensure there are no active TPM localities at the time of the dynamic launch.
	The DRTM_PARAMETERS and any memory protection table must be created such 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.
	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
	The invocation of DRTM_DYNAMIC_LAUNCH must be in Aarch64 state.
	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 may return errors to the caller of DRTM_DYNAMIC_LAUNCH.

After the D-CRTM makes changes to the state of the system (e.g., resetting dynamic PCRs) 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 may be returned to the caller.

Table 24, D-CRTM Phase 1 Requirements

ID	Requirement
	The D-CRTM must be immutable and tamper-resistant with respect to the Normal world.
	The D-CRTM must only execute on the boot PE.
	The D-CRTM must verify that the DCE preamble used PSCI CPU_OFF to turn off all PEs except the boot PE and if this is not the case return an error to the caller of DRTM_DYNAMIC_LAUNCH or enter remediation.
	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 an error if the PE used to initiate the dynamic launch is not the correct one.
	The D-CRTM must ensure that DRTM operation on the boot PE cannot be pre-empted by Non-secure asynchronous exceptions unless explicitly enabled by the DLME. Examples include interrupts, Serror exceptions, and SDEI events.
	If the caller of DRTM_DYNAMIC_LAUNCH was not in Aarch64 state, the D-CRTM must return an error.
	The D-CRTM may perform digital signature verification of the DLME image if it detects a digital signature in the image. If digital signature verification fails, the D-CRTM must return an error to the caller.
	The D-CRTM must verify that no TPM localities are active, and if this is not the case return an error to the caller of DRTM_DYNAMIC_LAUNCH or enter remediation.
	If the D-CRTM returns an error to the caller, localities 1, 2, and 3 must be left closed.

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
	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) This ensure that the GIC makes no memory accesses during the dynamic launch. Implementation Note: In some GICv3.0 implementations it may 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 would lead to behavior defined as UNPREDICTABLE.
	The D-CRTM must reset the dynamic PCRs in the TPM using TPM_HASH_START (or equivalent) at locality 4.
	The D-CRTM must open localities 1, 2, and 3 in an implementation specific way.
	The D-CRTM must map the DRTM parameters Execute-Never.
	If the D-CRTM is distinct from the DCE image, the D-CRTM must load the DCE image and must enforce verification of the cryptographic signature on the DCE.
	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.
	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. See PCR schema details in section 4.8.
	If the D-CRTM is distinct from the DCE image, the D-CRTM must extend a measurement of the public key that signed DCE image into the TPM and record the measurement in the event log. See PCR schema details in section 4.8.
	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 value of zero (1 byte in size) into the TPM for the DCE image and record the measurement in the event log. • extend the digest of the value of zero (1 byte in size) into the TPM for the public key that signed the DCE image and record the measurement in the event log. See PCR schema details in section 4.8.

	The D-CRTM must measure the PCR schema passed in the DRTM_PARAMETERS. See PCR schema details in section 4.8.
	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 and record the measurement in the DRTM event log. 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.
	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. See PCR schema details in section 4.8. If the enabling of trace can't 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.
	The D-CRTM may detect and measure the security lifecycle state of the platform into the TPM. 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.
	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 value 0xFF (1 byte in size) into PCR[17] and PCR[18] and enter remediation.
	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.
	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]
	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 may 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 image(s).

Table 26, DCE Image Requirements

ID	Requirement
	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].

	Updates to the DCE must be protected against rollback as specified by the requirements in the Arm Platform Security Boot Guide [5].
	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
	The DCE must map the DRTM parameters Execute-Never.
	The DCE must verify the fields of the DRTM parameters before use. See section 3.12.
	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 value 0xFF (1 byte in size) into PCR[17] and PCR[18] and enter remediation.
	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. See PCR schema details in section 4.8.
	If the DCE encounters an error, it must enter remediation. See section 4.7.

Table 28 specifies requirements with respect to memory protection.

Table 28, Memory Protection Requirements

ID	Requirement
	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.
	For region-based DMA protection, the DCE must verify any memory protection table specified in the DRTM parameters before using the table.
	For region-based DMA protection, the DCE must verify that the DLME region is in the memory protection table specified in the DRTM_PARAMETERS.
	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.
	For region-based DMA protection, the DCE must use a platform-dependent hardware enforcement mechanism to protect the requested regions.
	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.

	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 may differ from the regions requested in the DRTM_PARAMETERS.
	For a firmware-backed implementation, the DMA protection for both complete and region-based DMA protection must prevent DMA accesses from Non-secure devices.
	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 may be comprised of multiple components, where an initial DCE component started by the D-CRTM loads, verifies, and measures a secondary (or 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
	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.
	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
	The DCE must verify that DLME region is in Non-secure memory and that the requirements described in section 3.14 are met.
	The DCE must verify that the DLME data fits within the DLME region.
	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.
	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.
	The DCE must extend a measurement of the DLME image into the TPM and record the measurement in the DRTM event log. 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).

	The DCE may perform digital signature verification of the DLME image if it detects a digital signature in the image.
	The DCE must initialize the DLME_DATA_HEADER. See section 3.14.
	The DCE must populate the address map in the DLME data. See section 3.14.
	The DCE must populate the protected regions in the DLME data. See section 3.14.
	The DCE may use the implementation specific region in the DLME data for implementation specific purposes. See section 3.14.
	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).
	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 4.8.2.
	Before the DLME is called, the DCE must invalidate all instruction caches.
	Before the DLME is called, any unused space in the DRTM event log region must be initialized to zeros.
	Before the DLME is called, the DCE must close locality 3.
	Before the DLME is called, locality 2 must be active.
	Before the DLME is called the DCE must set the DRTM error code to zero.
	The DCE must identify the entry point of the DLME image from the offset in the DRTM parameters. See section 3.12.
	The DCE must extend a measurement of the DLME entry point offset into the TPM and record the measurement in the DRTM event log. See PCR schema details in section 4.8.
	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.
	The DCE must transfer control to the DLME entry point.

A Normal world DCE is a DCE component that executes in the Normal world. 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 DCE component that loads and starts the Normal world DCE.

Table 31, Normal World DCE Requirements

ID	Requirement
	The DRTM implementation must determine the address and size of the Normal world DCE region from the DRTM_PARAMETERS.

	The DRTM implementation must verify that the Normal world DCE region is protected from DMA based on the protection specified in the DRTM_PARAMETERS.

In order for the DLME to defend itself, it may need a description of TCB hardware components. During a system's normal boot flow 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 may 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 very simple system, the DRTM implementation may have a static set of hashes that can be provided in the TCB hash table. If the DRTM implementation uses implementation-specific means, it may 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 may impact the system's TCB 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
	The DCE must provide a TCB hash table region or copies of TCB-critical ACPI tables in the DLME data.
	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.
	It is strongly recommended that trustworthy hashes or copies of the following ACPI tables be provided: MADT MCFG GTDT IORT TPM2

When control is transferred to the DLME the boot PE must be configured as specified in Table 33.

Table 33, Boot PE Requirements

ID	Requirement

	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
	Non-secure asynchronous exceptions must be masked on the boot PE.
	Debug exceptions must be masked on the boot PE.

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 may be an operating system specific component supplied by the same vendor that provided the DCE Preamble component, or it may 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 may be verified using digital signatures or using measurements extended into the TPM. Data may be validated to confirm it is safe to consume.

The DLME may itself be the OS kernel, or it may 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. For example, this should include 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) that detects any tampering of the DLME and establishes whether the boot flow should continue.
    - The system could use local attestation to determine trust, where a security policy could 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 could be sealed against dynamic PCRs.
    - The system could use remote attestation where it sends a TPM quote to a remote verifier that could determine whether the state measured by DRTM is as expected. For example, the verification of the quote could be a criteria for access to a secure network.
    - The system could 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 could determine this from the event log and 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 may 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 verify the available space in conjunction with the DLME region address and the DLME data offset received as arguments.
- **Protected regions**. The DMA protections requested by the DCE Preamble and put in place by the DRTM implementation are described in the protected regions in the DLME data.
    - 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 were requested, the DLME must examine the protected regions to verify they are as expected. If necessary for platform-specific reasons, the DRTM implementation may protect a subset of the regions requested.
- **Memory map**
    - Before utilizing any address space outside of the DLME region, the DLME should check that regions used are consistent with the address map provided by the DRTM implementation 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 provided by the DRTM implementation 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 will contain: 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 table 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 may 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 may 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 may refuse to boot or may 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 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 could include digital signatures or measurements.
- **TPM usage**. The DLME may make use of 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 may make measurements using locality 2.
    - If preventing additional measurements into PCRs 17 and 18 is a requirement of the system's security policy, 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), and 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 may return errors to the DCE Preamble or may enter remediation which will reset the system. Errors detected by the DLME are handled in a platform-specific manner. The DLME may 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 could 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 described in section 3.4. 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 may 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 34.

Table 34, Remediation Requirements

ID	Requirement
	The DRTM implementation must place an error code as defined in section 3.11 must be placed 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.
	An implementation may record implementation-defined errors using the error code 0xFF and an implementation specific data value. See section 3.11 DRTM Error Encoding.
	If a Normal world DCE is used, it must record the error code using the DRTM_SET_ERROR function. See section 3.8.
	As a final step in remediation, the DRTM implementation must initiate a system reset.
	An implementation may report DRTM errors to other error logging infrastructure that may exist on a platform.

Errors detected by the DLME are handled in a platform-specific manner. The DLME may 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 35 are applicable to all phases of DRTM.

Table 35, TPM Measurement Requirements

ID	Requirement
	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).
	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).

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.

ID	Requirement
	For the default PCR usage schema the components or items measured, the measurement order, and event type must comply with the requirements in Table 36 and Table 37 for PCR[17] and PCR[18]. See Event Types, 3.16.2, for details of how each event type is logged.

Table 36, Default PCR[17] Schema Details

Order	Component Measured	Event Type	Usage
1	DCE image	EVTYPE_ARM_DCE	Required in all implementations
2	PCR schema value in the DRTM_PARAMETERS	EVTYPE_ARM_PCR_SCHEMA	Required in all implementations

3	Secure world TCB images	EVTYPE_ARM_TZFW	Required if measured by the implementation
4	Debug/trace state	EVTYPE_ARM_DEBUG_CONFIG	Required if measured by the implementation
5	Security lifecycle state	EVTYPE_ARM_NONSECURE_CONFIG	Required in all implementations
6	Secondary DCE image(s)	EVTYPE_ARM_DCE_SECONDARY	Required if measured by the implementation
7	Separator signifying the end of DCE measurements.	EVTYPE_ARM_SEPARATOR	Required in all implementations

Table 37, Default PCR[18] Schema Details

Order	Component	Event Type	Usage
1	PCR schema in DRTM_PARAMETERS	EVTYPE_ARM_PCR_SCHEMA	Required in all implementations
2	Public key that signed DCE image(s)	EVTYPE_ARM_DCE_PUBKEY	Required in all implementations
3	DLME image	EVTYPE_ARM_DLME	Required in all implementations
4	DLME image entry point	EVTYPE_ARM_DLME_ENTRY_POINT	Required in all implementations
5	Separator signifying the end of DCE measurements.	EVTYPE_ARM_SEPARATOR	Required in all implementations

5 System Requirements

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

5.1 Processing Element (PE)

The PEs or cores referred to in this specification are the PEs that run the operating system or hypervisor, not PEs that function as devices or peripherals. Table 38 specifies the requirements for the application processor PEs to support DRTM:

Table 38, System Requirements for PEs

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

5.2 Multiple Sockets

This specification assumes that with respect to DRTM, the rules and requirements apply equally to single- and multi-socket systems. The expectation is that the boot PE and security co-processor on which the DRTM boot flow is performed have the ability to 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 described previously a key security requirement of this architecture is that the DLME and supplemental images be protected against DMA attacks during a dynamic launch.

Table 39 specifies the requirements related to SMMU and DMA capable devices:

Table 39, System Requirements for SMMUs

ID	Requirement
	All DMA capable devices in a system must be behind an SMMU. This includes both Secure and Non-secure devices. The requirement does not apply to non-host platforms.
	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 40, System Requirements for Non-host Platforms

ID	Requirement

	If a NHP can access Non-secure memory at an address chosen by or influenced by software running at a Non-secure privilege levels 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 may 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 41, System Requirements for GIC

ID	Requirement
	A GIC that implements Locality-specific Peripheral Interrupts (LPIs) should support clearing GICR_CTLR.EnableLPIs.
	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 may support an embedded hardware trace capability where captured trace data is written into system memory. Trace may 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 system's TCB.

Table 42, System Requirements for Trace

ID	Requirement
	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 can't be detected by a DRTM implementation or if trace can be enabled dynamically without a system reset the DRTM implementation will measure 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 43 specifies the lifecycle requirements.

Table 43, System Requirements for Lifecycle

ID	Requirement
	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 co-processor other than the PEs that initiated DRTM.
- Firmware implementation in the Secure world.

Table 44 specifies the requirements for the TPM implementation:

Table 44, System Requirements for TPMs

ID	Requirement
	A TPM implementation must be compliant with the TPM Library Specification Family 2.0 [8].
	A discrete TPM 2.0 chip must be certified by TCG.
	The system must support hardware-based enforcement to mediate access to the dynamic localities (localities 1-4) of the TPM. Implementation note: For a firmware-backed implementation of DRTM, a common approach to protecting the dynamic localities would be to make a discrete TPM chip a Secure device and accessible only at Secure privilege levels. This would allow a Secure TPM service running at SEL0 to provide an interface such as a CRB interface to the Normal world. The TPM service would mediate accesses to all localities and thus enforce access to the dynamic localities.
	The platform interface to access the TPM must support the dynamic localities (localities 1-4) of the TPM.
	The implementation must support closing localities 2 and 3. See section 5.6.2.
	A system-level reset must reset the TPM.

It is beyond the scope of this specifications 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 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 additional extend operations into PCRs 17 and 18 following the dynamic launch. The DCE will always close locality 3. If preventing additional measurements into PCRs 17 and 18 is a requirement of the system's security policy, the DLME may close locality 2 using the DRTM_CLOSE_LOCALITY function.

Table 45 specifies requirements for the TPM implementation with respect to closing localities:

Table 45, Requirements for Closing TPM Localities

ID	Requirement
	When a locality is closed: • The TPM must ignore writes to the TPM_ACCESS_x register (FIFO interface) • The TPM must ignore writes to the TPM_LOC_CTRL_x register (CRB interface) • 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".
	Following a system reset, the TPM must treat the dynamic localities 1, 2, and 3 as closed.
	The TPM must provide a means for the D-CRTM to open localities 1, 2, and 3 as part of the dynamic launch sequence.
	The TPM implementation must support closing locality 3. See the DRTM_CLOSE_LOCALITY function, section 3.6.
	The TPM implementation must support closing locality 2. See the DRTM_CLOSE_LOCALITY function, section 3.6.
	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 described in section 5.6.1 access to the TPM implementation must be mediated, and it is expected that the mediation layer that clients use to access the TPM will be responsible for enforcing access to closed localities.

5.7 ACPI

This specification assumes the platform uses an ACPI-based system description. Table 46 specifies the requirements for ACPI:

Table 46, System Requirements for ACPI

ID	Requirement
	The system must implement a system description based on ACPI that is compliant with the definition in the Base Boot Requirements specification [10].
	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 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 be protected against DMA by devices during the dynamic launch. The DLME may 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 may 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 47, System Requirements for DMA Protection

ID	Requirement
	A platform must support at least one type of DMA protection specified in this architecture.
	For a firmware-backed implementation the DMA protection (complete or region-based) must prevent DMA accesses from Non-secure devices.
	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 may 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 48, Platform Configuration Requirements

ID	Requirement
	It must not be possible for components operating at Non-secure privilege levels to change the physical memory map of a system. This 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 such that a single memory location may be accessed through multiple physical addresses.

Table 49, Physical Memory Aliasing

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

5.10 Firmware

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

Table 50, Firmware Integrity Requirments

ID	Requirement
	For a firmware-backed DRTM implementation the integrity of the D-CRTM must be established by the system using measurements or digital signatures.
	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 51, Requirements to Mitigate Platform Reset Attacks

ID	Requirement
	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 may 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 will persist 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 prior to transferring control to the DLME.

Table 52, System Requirements for DRTM Launch Errors

ID	Requirement
	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.
	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 53, SMCCC and PSCI Requirements

ID	Requirement
	System firmware must implement PSCI 1.0 or later.
	System firmware should implement the MEM_PROTECT function specified in PSCI 1.1.
	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 may contain Secure services that execute on PEs at Secure privilege levels. Secure services are considered part of the system's TCB, and thus it is critical for the architecture of a system to comprehend possible threats to DRTM posed by Secure services that may run asynchronously and access Non-secure memory.

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

Table 54, Secure Services

ID	Requirement
	If a Secure service may be invoked asynchronously and may access Non-secure memory at addresses chosen by or influenced by software running at a 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 (alpha)

The content in this chapter is considered alpha quality and is subject to change.

6.1 Overview

In a hardware-backed implementation of DRTM, the D-CRTM is located in a co-processor separate from the Arm v8-A PE that initiated the dynamic launch.

Beginning the dynamic launch in the co-processor offers an even higher level of assurance that the measurements made during the dynamic launch have integrity:

- The D-CRTM is in a protected and isolated environment that cannot be tampered with even if other Secure firmware in the system may have been compromised.
- System hardware enforces that only the D-CRTM and co-processor have access to locality 4 in the TPM. This ensures that only the D-CRTM can initiate a dynamic launch, reset the dynamic PCRs, and make the initial measurement of the DCE.
- The chain of trust rooted in the D-CRTM on the co-processor extends to the DCE and DLME which run on the boot PE. All firmware components that execute in the DRTM boot flow are measured and verified using digital signatures.

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 co-processor. See Figure 10 for an example of a hardware-backed implementation.

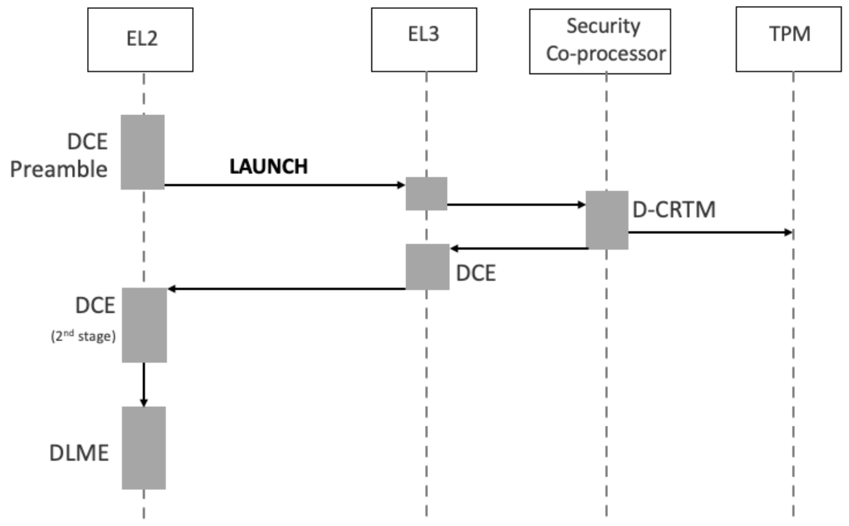


Figure 10, Hardware-backed Implementation Example

6.2 D-CRTM Requirements

In a hardware-backed implementation, the D-CRTM is located in a co-processor 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 proxies the dynamic launch request to the co-processor where the D-CRTM is located. See section 6.4.2 for further details about the co-processor that hosts the D-CRTM.

The table below describes requirements for the DRTM firmware that transitions the dynamic launch to the D-CRTM in the co-processor.

Table 55, DRTM Firmware Requirements for Transition to D-CRTM

ID	Requirement
	The DRTM implementation must verify that the DCE preamble used PSCI CPU_OFF to turn off all PEs except the boot PE and return an error to the caller of DRTM_DYNAMIC_LAUNCH if this is not the case.
	A DRTM implementation may require the boot PE to be a specific PE in the topology of the SoC. In this case the firmware must return an error if the PE used to initiate the dynamic launch is not the correct one.
	The DRTM implementation must pass the DRTM parameters to the D-CRTM by an implementation specific method.
	The DRTM implementation must start execution of the D-CRTM by an implementation specific method.
	If the DRTM implementation encounters an error, it must return the error to the caller or enter remediation. See section 4.7.

The table below specifies the requirements for the D-CRTM in a hardware-backed implementation.

Table 56, Hardware-backed D-CRTM Requirements

ID	Requirement
	The D-CRTM must hold all PEs in a reset state where all of the logic on which the PE executes is reset, including the integrated debug functionality. This is referred to as a *Cold reset* in the Armv8-A Architecture [1].
	The D-CRTM must verify no TPM localities are active. If this is not the case it must enter remediation.
	The D-CRTM must reset the dynamic PCRs using TPM_HASH_START (or equivalent) at locality 4.
	The D-CRTM must prepare a tamper-resistant memory space for the DCE to execute from. This must include protection against Secure devices that are DMA capable.
	The D-CRTM must load the DCE image and must enforce verification of the cryptographic signature on the DCE.
	The D-CRTM must extend a measurement of the DCE image into the TPM and record the measurement in the event log. See PCR schema details in section 4.8.

	The D-CRTM must extend a measurement of the public key that signed the DCE image into the TPM and record the measurement in the event log. See PCR schema details in section 4.8.
	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. 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.
	The D-CRTM may detect and measure the security lifecycle state of the platform into the TPM. 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.
	The D-CRTM must release the boot PE from reset and cause execution to begin at the DCE entry point.
	If the D-CRTM encounters an error, it must enter remediation. See section 4.7.
	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].
	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 DCE Requirements

The table below defines requirements specific to the DCE in a hardware-backed implementation.

Table 57, Hardware-backed DCE Requirements

ID	Requirement
	DCE components that execute at Secure privilege levels (Secure EL0, Secure EL1, Secure EL2, EL3) 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.
	For a hardware-backed implementation, any firmware components in the Secure world TCB (apart from the DCE) that execute during the dynamic launch 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.
	The DCE must ensure that DRTM operation on the boot PE cannot be pre-empted by Non-secure asynchronous exceptions unless explicitly enabled by the DLME. Examples include interrupts, Serror exceptions, and SDEI events.
	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) This ensure that the GIC makes no memory accesses during the dynamic launch.

6.4 System Requirements

6.4.1 TPM

The base requirements for a TPM implementation are described in section 5.6. The table below specifies additional TPM requirements for a hardware-backed implementation of DRTM.

Table 58, TPM Requirements in a Hardware-backed Implementation

ID	Requirement
	The TPM 2.0 must be a discrete chip or hardware enclave implementation.
	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.
	If the TPM implementation is a discrete chip, it must not be possible for any component in the system, except for the co-processor, to bypass the system's locality enforcement mechanism and directly access the bus to which the TPM is connected.
	A mechanism must be provided to close localities 2 and 3.

6.4.2 Co-Processor

In a hardware-backed implementation, the DRTM process is rooted in a D-CRTM that executes on a security co-processor 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.

The table below specifies the requirements for the DRTM security co-processor:

Table 59, Co-processor Requirements

ID	Requirement
	The co-processor must be a compute element isolated from all other hardware agents, including the application PEs. The co-processor must not share caches with the application PEs.
	The co-processor must be tamper-resistant. The specific tamper-resistant properties required will depend on the system's threat model.
	The co-processor must host the D-CRTM code that starts the dynamic launch.

	The co-processor must provide an interface that allows the DRTM firmware to make function requests and get return codes for DRTM functions handled by the co-processor.
	The co-processor must support a verified boot process to ensure the integrity of the D-CRTM.
	The co-processor must support a secure firmware update process that ensures that only authorized updates to the D-CRTM are allowed.
	The co-processor must have rollback protection for firmware, so the D-CRTM can be protected.
	The co-processor must be able to prevent external debug of the co-processor itself.
	The co-processor must be able to detect if external debug mechanisms are enabled.
	The co-processor must be able to issue a Cold reset (as defined in the Armv8-A Architecture [1]) to the Arm v8-A PEs on which DRTM is being performed.
	The co-processor must be able to access the memory containing the DCE image.
	The co-processor must be able to cause the boot PE to execute the DCE entry point.
	The co-processor must be capable of recording a persistent error code if an error occurs during the dynamic launch before entering remediation.
	The co-processor must be capable of initiating a system-level reset if an error occurs during the dynamic launch in order to complete remediation.

The table below specifies additional requirements for a hardware-backed implementation for how the security co-processor relates to the TPM:

Table 60, Requirements for TPMs in a Hardware-backed Implementation

ID	Requirement
	The co-processor must have exclusive access to access locality 4 of the discrete TPM.

6.4.3 DMA Protection

Table 61 specifies DMA protection requirements for a hardware-backed implementation.

Table 61, DMA Protection Requirements

ID	Requirement
	For a hardware-backed implementation, the DMA protection (complete or region-based) must prevent DMA accesses from both Non-secure and Secure devices.

6.5 DRTM Functions

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 co-processor 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.