TCG D-RTM Architecture

Document Version 1.0.0

June 17, 2013

Contact: admin@trustedcomputinggroup.org

Published

TCG

Disclaimers

- THE COPYRIGHT LICENSES SET FORTH DOES NOT REPRESENT ANY FORM OF LICENSE OR WAIVER, EXPRESS OR IMPLIED, BY ESTOPPEL OR OTHERWISE, WITH RESPECT TO PATENT RIGHTS HELD BY TCG MEMBERS (OR OTHER THIRD PARTIES) THAT MAY BE NECESSARY TO IMPLEMENT THIS SPECIFICATION OR OTHERWISE. Contact TCG Administration (admin@trustedcomputinggroup.org) for information on specification licensing rights available through TCG membership agreements.
- THIS SPECIFICATION IS PROVIDED "AS IS" WITH NO EXPRESS OR IMPLIED WARRANTIES WHATSOEVER, INCLUDING ANY WARRANTY OF MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE, ACCURACY, COMPLETENESS, OR NONINFRINGEMENT OF INTELLECTUAL PROPERTY RIGHTS, OR ANY WARRANTY OTHERWISE ARISING OUT OF ANY PROPOSAL, SPECIFICATION OR SAMPLE.
- Without limitation, TCG and its members and licensors disclaim all liability, including liability for infringement of any proprietary rights, relating to use of information in this specification and to the implementation of this specification, and TCG disclaims all liability for cost of procurement of substitute goods or services, lost profits, loss of use, loss of data or any incidental, consequential, direct, indirect, or special damages, whether under contract, tort, warranty or otherwise, arising in any way out of use or reliance upon this specification or any information herein.

Any marks and brands contained herein are the property of their respective owners.

Acknowledgements

The TCG would like acknowledge the contributions of the many individuals (listed below) and the companies who allowed them to volunteer their time to the development of this specification.

The chairs of the D-RTM working group deserve special thanks, they include Monty Wiseman, and Lee Wilson.

Special thanks are given to the editors of this specification: David Wooten, Rob Spiger, and David Robinson.

Contributors

Jacob Shin; AMD
Bill Jacobs; Cisco
Mark Scott-Nash; Intel
David Robinson; Microsoft
Dennis Mattoon; Microsoft
Eugene Myers; US Govt
Louis Hobson; HP
Graeme Proudler; HP
Geoffrey Strongin; AMD
Mukund Khatri; Dell
Varugis Kurien; Microsoft
Julian Hammersley; AMD
Shiva Dasari; IBM
Monty Wiseman; Intel
Rob Spiger; Microsoft
Dick Wilkins; Phoenix Tech
Ron Perez; AMD
Ralf Findeisen; AMD
David Challener; Johns-Hopkins
Hidenobu Ito; Fujitsu
Dalvis Desselle; HP
Steve Goodman; Lenovo
Gary Simpson; AMD
Lee Wilson; IBM
Kirk Brannock; Intel
David Wooten; Microsoft
Eric Paris; Red Hat
Amy Nelson; Dell
Mark Piwonka; HP
Conan Dailey; General Dynamics
Will Burton; CESG
Randy Springfield; Lenovo
Timothy Markey; Phoenix

Contents

1 Introduction to This Document

1.1 Overview

This specification describes the architecture and implementation examples for a Dynamic Root of Trust for Measurement (D-RTM) used for measured platform initialization without a hardware platform restart. This specification extends the TCG PC Client specification (See (1)). The term “dynamic” is used because the measured platform initialization may occur while the hardware platform is running. In contrast, the Static Root of Trust for Measurement (S-RTM) requires a platform shutdown or restart.

1.2 Document Structure

The document’s chapters describe certain aspects of a D-RTM. The document begins with a description of architectural objectives and basic architectural elements. An overview of the D-RTM environment and goals is given. Following chapters describe important aspects of the environment and what influence on a platform implementation the use and support for a D-RTM has. This is followed by a description of the D-RTM architecture in detail as well as architectural details of trust flow, logging, and error handling.

The chapters are in detail:

1. **Introduction to This Document**

   This chapter contains an overview of the document as well as terms and definitions and references used in the document.

2. **Architectural Overview**

   This chapter explains the goals for D-RTM of establishing, maintaining and attesting to a Trusted Computing Base. It describes underlying concepts at a high level to help readers understand the overall architecture.

3. **Trusted Computing Base**

   This chapter enumerates and describes hardware and software components or processes that are generally involved in establishing and maintaining a Trusted Computing Base on a system. Each description is augmented with information relevant for this specification, such as implementation best practices or potential vulnerabilities. It provides the requirements necessary for each component or process to be compliant with this specification.

4. **Structures**

   This chapter describes the syntax and meaning of data structures and their fields involved in the D-RTM. Subsequent sections define how the structures are used.

5. **System States, Transitions, and Interfaces**

   This chapter defines the system states relevant for the D-RTM process. It identifies transitions between the states. For each transition, it lists the interface used and the required system state for the transition to occur.

6. **PCR Usage**

   This chapter describes the D-RTM PCR and how they are used. It defines the Dynamic Core Root of Trust for Measurement (D-CRTM) and provides some examples of different implementations.

7. **DCE Requirements**

   This chapter describes the D-RTM Configuration Environment (DCE) and specifies its requirements.

8. **Component Setup, Configuration, Validation, and Measurement Required for the D-RTM Process**

   This chapter specifies the required configuration the DCE must achieve for each hardware or software component involved in establishing a system's Trusted Computing Base during the D-RTM process. It also specifies what validation and measurements must be performed.

9. **Event Logging and Reporting**

   This specifies the D-RTM event log and what measurements are placed in the measurement log during the D-RTM process.

1.3 References

1. **Trusted Computing Group.** TCG PC Client Specification. [Online] http://www.trustedcomputinggroup.org/developers/pc_client/specifications.

2. **ACPI.** ACPI Specification, Version 5.0. [Online] http://www.acpi.info.

3. **Trusted Computing Group.** TCG Glossary. [Online] https://www.trustedcomputinggroup.org/groups/glossary.

4. **Department of Defense.** Department of Defense Standard Department of Defense Trusted Computer System Evaluation Criteria. s.l., United States : Department of Defense, December 26, 1985. DoD 5200.28-STD, Library No. S225,711.

5. **Intel.** Extensible Firmware Interface Main Specification Version 1.10. [Online] http://www.intel.com/technology/efi.

6. X86 Calling Conventions. *Microsoft Corp.* [Online] http://msdn.microsoft.com/en-us/library/k2b2ssfy.aspx.

7. **Microsoft Corp.** Cdecl. [Online] http://msdn.microsoft.com/en-us/library/zkwh89ks.aspx.

8. —. Overview of x64 Calling Conventions. [Online] http://msdn.microsoft.com/en-us/library/ms235286.aspx.

1.4 Terms and Definitions

ACPI (Advanced Configuration and Power Interface)
An open industry specification that describes OS-directed configuration, thermal and power management. ACPI Machine Language code is usually provided as part of the BIOS and interpreted by an operating system-provided interpreter. See reference (2) for more information.

ACPI Namespace
A hierarchical tree structure in memory that contains named objects loaded from the ACPI tables. See reference (2) for more information.

AML (ACPI Machine Language)
The binary pseudo-code used to represent ACPI control methods and objects. It is compiled from ASL and is stored for use by the ACPI interpreter in the OS. See reference (2) for more information.

ASL (ACPI Source Language)
The programming language equivalent of AML. See reference (2) for more information.

BIOS (Basic Input Output System)
Performs basic system configuration and setup and starts the operating software.

BMC (Baseboard Management Controller)
A specialized controller in many servers used for management tasks (e.g. power management).

CRTM (Core Root of Trust for Measurement)
The instructions executed by the platform when it acts as the RTM (Root of Trust for Measurement). See reference (3).

DCE (Dynamic Root of Trust for Measurement Configuration Environment)
The software/firmware that executes from the instantiation of the DL Event to the transfer of control to the Dynamically Launched Measured Environment (DLME). The DCE is responsible for ensuring the platform is in a trustworthy state as defined by the CPU, chipset, and the platform manufacturer.

DCE preamble
The PM-provided code that starts the D-RTM process. The entry point to the code is called DL_Entry. It may perform some initial configuration actions that are platform specific before invoking the D-RTM CPU instruction. Some implementations of the DCE preamble may do some basic validation and return an error code for issues documented in this specification.

D-CRTM (Dynamic Core Root of Trust for Measurement)
The Core Root of Trust for the D-RTM. This is a function that is built into the Host Platform and is started by the Dynamic Launch Event (DL Event). This function is a Trusted Process. Even though the D-CRTM executes after the S-CRTM, the D-RTM's transitive trust chain will not necessarily have a trust dependency on the S-CRTM's transitive trust chain.

DLME (Dynamically Launched Measured Environment)
The software executed after the DCE- instantiated TCB is established. The DLME would nominally be supplied by an OS vendor.

DMA mapping
Controls how hardware devices access Host Platform memory; requests to access a specific physical memory address may be mapped to an alternate physical memory address. Similar to user mode processes use of virtual memory where page tables control the mapping to physical memory pages. Examples are IOMMU or VT-d.

DMA protections
Provide a mechanism to allow a Host Platform to prevent hardware devices from accessing certain Host Platform memory. Examples are a DMA exclusion scheme or DMA mapping.

DRT (D-RTM Resources Table)
An ACPI table. It holds information such as the location of the D-RTM event log and is the primary way information is passed from the DCE to the DLME. It is used before the D-RTM event to provide the location and address mode of the PM-provided DCE preamble, which starts the D-RTM process.

D-RTM (Dynamic Root of Trust for Measurement)
A platform-dependent function that initializes the state of the platform and provides a new instance of a root of trust for measurement without rebooting the platform. The initial state establishes a minimal Trusted Computing Base.

Dynamic Launch (DL)
Describes the process of starting a software environment at an arbitrary time in the runtime of a system.

Dynamic Launch Event (DL Event)
A platform dependent event that triggers the D-RTM, ending the Gap, and starting the DCE (or enters remediation). The implementation specific details of the event are encapsulated within the DCE

preamble. In many implementations the event is a privileged CPU instruction, often executed through microcode.

Dynamic Operating System Environment
The environment in which the DLME executes.

Gap
Any untrusted code that is executed after the initial S-RTM and before the Dynamic Launch Event.

Hardware attack
Any attack that requires physical access to the machine and changes to the hardware configuration of the machine, but not made using externally accessible interfaces on the platform. For example, plugging a device into a USB port is not a hardware attack.

Identity Mapped Pages
Memory pages where the virtual page addresses are mapped to the same physical page addresses.

Integrity Logging
The storage of integrity metrics in a log for later use. See reference (3).

Integrity Measurement (Metrics)
The process of obtaining metrics of platform characteristics that affect the integrity (trustworthiness) of a platform, and putting digests of those metrics in shielded locations (called Platform Configuration Registers: PCR). See reference (3).

Integrity Reporting
The process of attesting to the contents of integrity storage. See reference (3).

Legacy BIOS
Platform firmware that follows the conventions of early PC/AT-based clients and servers.

Locality
A mechanism for supporting a privilege hierarchy in the platform. See reference (3).

Manifest
Contains information, including a hash, about a collection of components. If the hash of each component is equal to its corresponding value in the manifest, then the hash of the manifest represents the combined hash of all of the components. Additionally, if the manifest is signed, the signature covers all of the components. The primary advantages of a manifest are that it links multiple components into a single, compact representation, and it allows multiple components that are either unsigned, or signed by different parties, to have a single signature.

NHP (Non-Host Platform)
A device or controller that cannot be excluded from the TCB. An NHP with immutable firmware is a *Fixed-NHP*, otherwise it is an *updateable NHP*.

OS
Operating system software that is executed after the firmware, for the purposes of this specification it includes both traditional OSes and virtualization software such as hypervisors.

PCR (Platform Configuration Register)
A shielded location containing a digest of integrity measurements. See reference (3).

PCR.Authorities

The signing authority of platform measurements made during the DCE process is placed in a PCR designated as the PCR.Authorities. The intention of this PCR is to have measurements that only change if an authority for a DCE component changes. For example, a change through a DCE update that was still signed by the platform manufacturer would not impact this PCR measurement.

PCR.Details

Details of platform measurements made during the DCE process are placed in a PCR designated as the PCR.Details. The intention of this PCR is to have measurements that change if any minutia of the D-RTM process changes. For example, a change through a DCE update would cause this measurement to change.

Peripheral

A device or controller that can be excluded from the TCB. A peripheral can be restricted from accessing the memory of the TCB through DMA protections or other means. A peripheral that is part of the TCB can be restricted from accessing TCB memory outside the peripheral, and must protect any memory within the peripheral.

Platform Manufacturer Validated Code

Validated code that is under the authority of the platform manufacturer.

Platform owner

The platform owner is used as defined by the TCG glossary (see (3)).

The platform owner is the entity responsible for the platform's security and privacy policies that is distinguished by knowledge of the owner authorization data. As an enhancement to (3), the owner is also the entity, which is authorized to update BIOS, firmware, etc., on the hardware platform. The platform owner is also the entity, which can enter the admin portion of the platform setup utility. See reference (3).

Platform user

The platform user is used as defined by the TCG glossary (see (3)).

The platform user is the entity that is making use of the TPM capabilities. See reference (3).

PM (Platform Manufacturer)

The platform manufacturer is the entity that has the final responsibility of the trust properties of the platform. It is typically the PM's logo on the platform.

Due to the complex nature of manufacturing relationships, the term OEM is not sufficient or can be misleading when addressing the manufacturer of a platform in the context of this document.

PMAM (PM Authority Module)

The authority for the platform manufacturer.

Pre-Boot Environment

The time after platform reset when the platform is executing initialization code necessary to load an operating system. This is also called the BIOS and can be either a legacy BIOS or UEFI.

Pre-Gap

The time during the S-RTM when the platform security state is completely under the control of the platform manufacturer and is trusted. In this time period, the platform manufacturer may perform actions that are dependencies for the DCE validation process. An example action is loading, measuring, and locking System Management Mode (SMM).

Remediation

The actions taken by the platform when it is not possible to perform a Dynamic Launch.

Root of Trust (component)
A component that must always behave in the expected manner because its misbehavior cannot be detected. A complete set of Roots of Trust has at least the minimum set of functions to enable a description of the platform characteristics that affect the trustworthiness of the platform.

RSM (Return from System Management Mode)
This is the instruction invoked by the System Management Interrupt (SMI) handler to return to non-SMM execution. It is the "IRET" analog for SMI.

RTM (Root of Trust for Measurement)
A computing engine capable of making inherently reliable integrity measurements. Typically this is the normal platform computing engine, controlled by the CRTM. This is the root of the chain of transitive trust.

RTR (Root of Trust for Reporting)
A computing engine capable of reliably reporting information held by the Root of Trust for Storage.

RTS (Root of Trust for Storage)
A computing engine capable of maintaining an accurate summary of values of integrity digests and the sequence of digests.

S-CRTM (Static Core Root of Trust for Measurement)
An immutable portion of the Host Platform that is required to begin at the Host Platform Reset. This is where the Host Platform begins its execution. See reference (3).

S-RTM OS Present Environment
The environment in which the post S-RTM OS executes, as defined in the PC Client Conventional BIOS specification.

Sensitive Resource
A hardware resource that must be protected in order for software to defend its Trusted Computing Base.

SM Base (System Management Base)
The per logical processor base of CPU System Management Mode (SMM) operation. This is configured by BIOS POST on S5 and S3 boot paths.

SMI (System Management Interrupt)
A non-maskable interrupt that transitions the CPU from non-System Management Mode (SMM) operation into SMM operation. It is an OS transparent interrupt that is serviced by BIOS. All context save and restore is handled either by hardware or by BIOS software as part of the handler.

SMI Entry Point (System Management Interrupt Entry Point)
This is offset from SMBASE where the SMI Handler will begin execution when it begins servicing an SMI.

SMI Handler (System Management Interrupt Handler)
The code that executes when servicing an SMI. This is normally code placed into SMRAM during BIOS POST.

SMI Initialization (System Management Interrupt Initialization)
The BIOS POST process that configures both the CPU SMBASE and SMRAM configuration and contents.

SMI Trap (System Management Interrupt Trap)

The system may implement a set of conditions that result in an SMI. For example, the chipset can be set to generate an SMI on access to an I/O port by system software. This facilitates both software interfaces to the SMI Handler and hardware work-around capabilities.

SMM (System Management Mode)

An operating mode of the CPU in which all bus cycles accessing memory are tagged with an "in SMM" indicator. The CPU may internally enable other capabilities based on this state. The chipset may also bind certain capabilities to SMM. SMM is implementation-specific to the CPU.

SMM/SMRAM Protections (System Management Mode / System Management RAM Protections)

The collection of memory controller features that preclude access to SMRAM by non-SMM entities. It includes bus cycle steering based on an "in SMM" attribute, DMA protection, and configuration locking.

SMRAM (System Management RAM)

A region of memory that is sequestered in hardware in a manner that it is only decoded by CPU cycles that are tagged with an "in SMM" indicator. SMRAM is implementation-specific to the memory controller.

Software attack

An attack that can be mounted using software only or externally accessible interfaces (e.g., boot from a different media, network interfaces). A software attack can be done remotely or locally.

S-RTM (Static Root of Trust for Measurement)

The executable component of the RTM that gains control of the Host Platform upon a Host Platform Reset. For a more detailed description, refer to the *TCG PC Client Specific Implementation Specification For Conventional BIOS*.

TCB (Trusted Computing Base)

The heart of a trusted computer system. The TCB "contains all of the elements of the system responsible for supporting the security policy and supporting the isolation of objects (code and data) on which the protection is based. The bounds of the TCB equate to the 'security perimeter' referenced in some computer security literature. In the interest of understandable and maintainable protection, a TCB should be as simple as possible consistent with the functions it has to perform. Thus, the TCB includes hardware, firmware, and software critical to protection and must be designed and implemented such that system elements excluded from it need not be trusted to maintain protection. Identification of the interface and elements of the TCB along with their correct functionality therefore forms the basis for evaluation." See reference(4).

TPM (Trusted Platform Module)

An implementation of the functions defined in the TCG Trusted Platform Module Specification; the set of Roots of Trust with shielded locations and protected capabilities. Normally includes just the RTS and the RTR. See reference (3).

Transitive trust

Also known as "inductive trust." In this process, the Root of Trust gives a trustworthy description of a second group of functions. Based on this description, an interested entity can determine the trust it is to place in this second group of functions. If the interested entity determines that the trust level of the second group of functions is acceptable, the trust boundary is extended from the Root of Trust to include the second group of functions. In this case, the process can be iterated. The second group of functions can give a trustworthy description of the third group of functions, etc. Transitive trust is used to provide a trustworthy description of platform characteristics.

Trust

Trust is the expectation that a device will behave in a particular manner for a specific purpose.

UEFI (Unified Extensible Firmware Interface)

The next generation BIOS standard. It defines a new model for the interaction between OS software and platform hardware. See reference (5).

Validation component

Platform specific code that performs the steps necessary to validate a part of the trustworthiness of a platform or a segment of code.

Validated code

Code is considered to be validated when it is assured that the code to be executed is the code that the current authority intended to be run.

2 Architectural Overview

2.1 Introduction

The Trusted Computing Base (TCB) is a fundamental concept in computer security. It is the portion of the system that is relied on to enforce the security policy of the platform. Two goals of Trusted Computing Group (TCG) technology are to be able to prove that the TCB was properly established and to allow access to sealed data only if the TCB was properly established.

For the Static Root of Trust for Measurement (S-RTM), measurements are taken during the boot process with the intent of being able to demonstrate, through Platform Configuration Registers (PCR), what software was run during the boot process. This reflects the fact that every software component in the boot path has the potential to modify the TCB of the loaded operating system (OS).

A property of the S-RTM sequence is that the evaluation of the TCB of an OS has a dependency on software/firmware that has no ongoing role in the operation of the platform. Software that is loaded, run, and discarded during the boot process must be evaluated to determine if it could or did compromise the TCB.

The purpose of the Dynamic Root of Trust for Measurement (D-RTM) is to reduce the complexity of the TCB so that evaluation of the platform state becomes more tractable.

The method of D-RTM is relatively simple. It is observed that a new chain of trust can be started if

1) all processors are placed in a known state,
2) a processor begins running measured code, and
3) the code that the processor has measured cannot be modified except as directed by the measured code running on the processor.

These conditions are met at system reset when the CPU and the rest of the chipset are forced to an initial condition by a hardware signal; system reset is the way in which the S-RTM is started. For D-RTM, the same three conditions are met but without requiring that the full system be reset.

New instructions have been added to PC processors that cause the CPU to extend the hash of a block of code to the Trusted Platform Module (TPM). Before, or during, the measurement of this code, it is protected from Direct Memory Action (DMA) access either by locking it in the CPU cache or by extending DMA protections over the code in memory. Protecting the code from DMA ensures that it can't be modified after it is measured and before it is executed. The processor then begins execution of the measured code, starting the D-RTM chain of trust.

An essential difference between the D-RTM chain and the S-RTM chain is that the D-RTM chain can start with the platform hardware already configured and with memory prepopulated with an OS. Thus, it is not necessary to rerun all of the software/firmware that was used to configure the hardware or load the OS. If the code is not run, then it is excluded from the TCB as long as any possible effect of that code on the TCB can be negated.

The way in which the D-RTM process neutralizes any possible impact of the boot code is to check to see that the hardware configuration is in a TCB-safe state. That is, it must check to see that the hardware is set up in such a way that the TCB can protect itself. This check is possible because the CPU command that started the D-RTM chain has protected the code doing this check so that no hardware settings can subvert the checks.

2.2 DMA

If it was necessary to check every piece of hardware attached to the system in order to ensure that they were in a TCB-safe mode, then D-RTM would likely be more complex than S-RTM. This level of checking would be necessary if the D-RTM TCB could not prevent DMA attacks on its code. So that this kind of checking is not necessary, the D-RTM specification requires that the system has a method of protecting

memory from DMA access. Some example methods are a simple DMA exclusion scheme or by DMA remapping as provided by an IOMMU or VT-d. Since the TCB can prevent DMA access to TCB memory, peripherals cannot change the TCB no matter how the peripherals are programmed. This effectively excludes all of the peripherals from the TCB so their state does not have to be validated. Exclusion of the peripherals from the TCB also excludes the software or firmware that may have programmed those peripherals during the boot process.

In principle, the minimal amount of checking during the Dynamic Root of Trust for Measurement Configuration Environment (DCE) would be to validate that the DMA remapping units were on and preventing all DMA access. Then the DCE could measure some portion of the OS and jump to it, turning the hardware TCB over to the OS. However, this minimal TCB would not be adequate for most modern PCs because of hardware variability and System Management Mode (SMM).

2.3 System Management Mode (SMM)

SMM is used by platform manufacturers to control portions of the hardware independent of the OS. The functions of SMM vary but some of the functions are necessary for system reliability and safety (for example, it is common for SMM to be used to prevent thermal damage to the system). SMM may have many privileges that could be used to compromise the TCB. An example of a way SMM could subvert a TCB is to modify memory. SMM has the ability to modify any portion of memory, including the memory used for the OS's TCB. This means that SMM must be included in the evaluation of the state of the TCB performed in the DCE. This specification defines the acceptable methods of validating SMM and NHPs.

2.4 Non-Host Platforms

Many systems include controllers (examples include service processors, management engines, and Baseboard Management Controllers, or BMCs) that perform critical platform functions such as power management and platform configuration. These controllers may impact the TCB (platform, DCE, or DLME) in various ways.

Many of these controllers interact with the system (i.e., access the TCB's memory) through DMA or other means, which can be restricted using DMA or similar protection. A controller that can be excluded from the TCB using DMA protection, or other means, is a *peripheral*. A controller that cannot be excluded from the TCB (e.g., can access the TCB's memory) without being subject to DMA or similar protection is called a *Non-Host Platform* or NHP.

To summarize (for the purposes of this specification):

1) A controller that can be restricted from reading or writing the platform, DCE, or DLME TCB's memory or control registers using DMA protection (or other means) is called a ***peripheral***.
2) A controller that cannot be restricted from reading or writing the platform, DCE, or DLME TCB's memory or control registers using DMA protection (or other means) is a ***Non-Host Platform*** (***NHP***).

Some NHPs have fixed firmware while others may be field programmable. Fixed firmware is loaded by the NHP's manufacturer or the platform manufacturer (PM) and cannot be changed or updated in the field (i.e. immutable). From a TCB perspective, these components are indistinguishable from hardware logic gates and are therefore part of the TCB provided by the platform. These are called ***fixed-NHP***s. Fixed-NHPs are assumed to be part of the platform's hardware without analysis by the D-RTM process. They are, therefore, outside the scope of this specification.

Some NHPs are field reprogrammable (i.e., after the controller is either manufactured or placed onto the platform by the platform manufacturer) and are called ***updatable NHP***s.

2.5 TCB Hardware Protection

In order for the TCB to be able to defend itself after it gets control at the end of the D-RTM process, it must have information about the hardware that is used to protect the TCB. Examples of this hardware are the CPUs, the DMA remapping units, and memory controllers. If the TCB does not control all of this essential hardware, then that hardware could be used to penetrate the TCB. The specification requires that the D-RTM process validate the Advanced Configuration and Power Interface (ACPI) tables that describe the TCB hardware so that the software TCB can have the information necessary to defend itself. Because the TCB hardware on a platform may include components the software TCB has no knowledge about (for example, a new hardware component may have been invented after the software TCB was released to market), the TCB hardware information validated during the D-RTM process contains a list of Sensitive Resources (memory addresses) to which the TCB must restrict access to prevent malicious components from subverting a TCB.

2.6 Advanced Configuration and Power Interface (ACPI)

Besides providing a description of the system hardware, ACPI provides methods for controlling various parts of the system hardware. These methods are expressed in an ACPI Machine Language (AML) in, or referenced by, the ACPI tables. On some systems, this interpreted code must access hardware that protects the TCB. This specification details how the D-RTM process will validate the tables containing the ACPI methods so that the software TCB can make an informed decision about whether an ACPI method should be able to modify TCB hardware. In some systems, such as clients, it is not necessary for any ACPI method to modify any TCB hardware. For those systems, the D-RTM provides a validated flag so that the TCB may know it can simply block all access to TCB hardware by ACPI methods.

2.7 Authorities

Another significant difference between S-RTM and D-RTM is the method of measurement. S-RTM only records measurement of the actual hashes of code and data. This means that the measurements change each time the code changes even if there is no change in the trust property of the software. D-RTM uses an additional scheme—it records the actual hashes of code and data in one PCR (PCR.Details), but it also records the hash of the “responsible party” for the code in a different PCR (PCR.Authorities). Because the authorities for a particular platform would not normally change over the lifetime of a platform, this scheme allows stability in sealing if only the PCR.Authorities is used.

Normally, the code that checks the hardware TCB (CPU, chipset, and the SMM code) would not change over the lifetime of a platform. On the rare occasion when the code does change, it still comes from the same responsible party. Keeping the authorities measurements unchanged during updates provides stability in the measurements that indicate the state of the TCB.

Because the D-RTM process-created TCB only involves a few measurements, it is relatively simple to evaluate the trustworthiness of the platform: Do you trust the CPU vendor, the chipset vendor, the platform vendor, and the operating system vendor? The PCR.Details allows someone to make more detailed evaluations, but knowing that the authorities approve of the hardware TCB settings is usually sufficient to allow the operating system to boot to a trustworthy state.

2.8 PCR Measurements

The Static Core Root of Trust for Measurement begins recording measurements into PCR at the beginning of the boot process. Code running later is unable to tamper with the earlier measurements recorded in the S-RTM PCR. This prevents code from extending the S-RTM PCR to arbitrary values.

The D-RTM process starts when the DL_Entry call executes. On some platform implementations, it is possible for software to extend the D-RTM PCR values before executing the DL Event. To prevent software from extending the D-RTM PCR to arbitrary values, the PCR begin with an initial value of negative 1 (all bits set). Only by executing the DL Event may the platform reset the D-RTM PCR.Details

and PCR.Authorities to zero. Immediately after resetting the values to zero, the D-RTM process records measurements into the PCR. Because the D-RTM process resets the PCR, if the D-RTM process is invoked multiple times during the same platform boot cycle, each invocation's measurements will be reflected in the resulting D-RTM PCR irrespective of the D-RTM PCR values before the invocation.

2.9 DCE and the DCE Preamble

The D-RTM process validation of the state of the TCB hardware occurs in the D-RTM Configuration Environment (DCE). An operating system-provided loader places operating system code (which may include a hypervisor) in memory and then invokes platform-specific code called the DCE preamble to prepare the platform for the D-RTM process. The function entry point a platform should use to invoke the preamble is specified in an ACPI table called the D-RTM Resources Table (DRT). Different hardware platforms may require specific actions before initiating the D-RTM process. The DCE preamble allows platform manufacturers to incorporate any setup work that is necessary for their platform. The DCE preamble then invokes the D-RTM process (DL Event) using an architecture-specific operation (e.g., a CPU instruction).

The DCE preamble is not significant from a security perspective. If an operating system invoked the architecture-specific CPU command to start the D-RTM process directly, the process would succeed only if all the required D-RTM validation steps were successful. Nonetheless, due to characteristics specific to the platform, executing the CPU command to start D-RTM directly without using the DCE preamble may cause the D-RTM process to fail. In summary, OSes do not need to support invoking an architecture-specific operation to start D-RTM directly and the D-RTM process must only succeed if the platform passes the validation performed by the D-RTM process.

The DCE will perform validation of the TCB hardware and TCB firmware (including SMM and relevant ACPI tables), make entries in the D-RTM event log to indicate the authorities and methods used for the validations, and extend the PCR.Details and PCR.Authorities. The DCE validation process is active in that the DCE may decide to correct anything which might cause validation to fail instead of simply detecting the failure and entering failure remediation.

This document specifies the interfaces for entry into the DCE preamble and for the DCE exit to the OS-provided code (the Dynamically Launched Measured Environment or DLME). The specification also outlines the functional requirements of the DCE, but does not mandate an implementation for a DCE. This is because the DCE varies according to processor architecture and there are many viable implementations. When possible, this document will provide suggested implementation options for the DCE in order to illustrate the expected behavior and to encourage use of implementations that are known to have the correct security properties.

2.9.1 The "Gap"

The D-RTM process changes the platform from a potentially undefined software and hardware state to a state that has a known hardware configuration and that executes measured code. For convenience, the undefined software and hardware state prior to the D-RTM process is referred to as the "Gap." This term is used to indicate that, because of the unknown properties of the code, there is a Gap in the understanding of the trust state of the platform. A system running in the Gap may be trustworthy, but the complexity of evaluation of the trust state can be nearly intractable. Once a system enters the Gap, the only way to leave is through a successful D-RTM launch or a system reset.

A system may be completely locked down with no ability to add or remove hardware components and with all code in the boot path evaluated and found to be trustworthy. This is not the typical PC. One of the great advantages of a PC is its flexibility and configurability by people with limited technical capabilities. When someone adds a device to their system, they may have implicit trust in the device without knowing whether or not the device is trustworthy. So, a system could ship with all of the components (hardware and software) completely understood by the platform manufacturer from a trust perspective, but as the system is customized (e.g., adding a peripheral) the initial trustworthiness established by the platform manufacturer may be lost and a Gap established.

D-RTM allows establishment of an attestable TCB even if the system starts in a state with completely unknown security properties—it's in the Gap. The platform manufacturer's implementation must establish the TCB and it is important that the implementation is not able to be influenced by actions that occur during the Gap.

The Gap occurs sometime after a power on reset and before the DCE. There is no simple test for determining when the Gap starts but generally upon execution of the first software/firmware component not validated by the PM, the system is considered to have entered the Gap. However, the system may be in the Gap if it is running code that is from the PM, but the PM has chosen not to evaluate the trust properties of the code.

If necessary, a platform manufacturer may set up or record information used later by the DCE to validate the platform before entering the Gap, but only if the data is unable to be altered by actions in the Gap.

What is included in the Gap depends on how the D-RTM process does platform state verification. For example, if the DCE will validate SMM by direct inspection, then the code that loads SMM may be considered to be Gap code and not in the TCB. However, if the DCE checks that SMM was validated when it was initially loaded and locked, then all of the S-RTM code that runs before SMM is locked is Pre-Gap code and becomes part of the final TCB.

2.9.2 Timeline

Figure 1 shows a nominal timeline for starting a system with D-RTM. At system reset, an S-RTM chain of trust is started and extended as the BIOS executes. This is the Pre-Gap time period and the platform security state is known completely to the platform manufacturer. The BIOS may load the System Management RAM (SMRAM) area and lock it. If the DCE is going to rely on the S-RTM to validate SMM when loaded and lock it to protect it, then the load and locking actions would need to occur prior to the system entering the Gap. If the DCE is going to independently validate or repopulate SMM, then the initial provisioning of SMRAM would not need to occur before the system enters the Gap.

In the Pre-Gap time period, the BIOS code may also sequester information critical for trust validation (e.g., a portion of the ACPI tables) by copying it to SMRAM before it is locked. The information is then available later when the DCE validates the platform.

On some implementations, the platform manufacturer will not perform any actions that are dependencies for the D-RTM process, and the Pre-Gap period will not exist.

Figure 1 shows the Gap between the Pre-Gap and the DCE preamble. The Gap may contain an arbitrary amount of code from any source and there is no upper limit on the time between startup and the start of the DCE preamble. The Gap could be limited to a small amount of BIOS code or could be a full operating system.

Code in the Gap initiates the D-RTM process by invoking the DCE preamble and passing the location and size of the DLME. The DCE preamble is started by a function call to an entry point that is described in the D-RTM Resources Table. The DCE preamble performs actions provided by the platform manufacturer to help prepare the system to execute the Dynamic Launch (DL) Event in Figure 1. If the DCE preamble determines the platform is not ready to invoke the D-RTM event, it may return an error code to the caller. Otherwise, the DCE preamble concludes by causing the DL Event. This is a CPU-specific command that starts the DCE by measuring an initial piece of code into PCR.Details and PCR.Authorities. This starts the D-CRTM chain of trust.

After the DL Event, the DCE process will ensure that actions taken in the Gap will not influence the TCB of the OS DLME code launched by the DCE.

The DCE will effectively instantiate various verification agents. Each agent will validate the trust state of a specific component (e.g., an agent provided by the CPU vendor validates the setup of the CPU and an agent provided by the PM validates SMM and the ACPI Name Space).

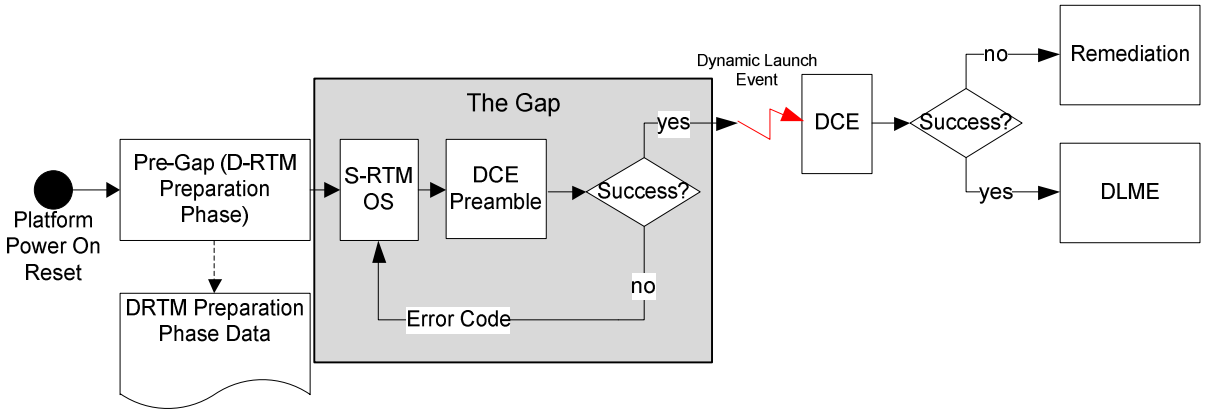

Figure 1 — System Timeline

When the DCE completes successfully, the platform's components are in a state that the chipset and platform manufacturing authorities believe to be trustworthy, as defined by this specification.

If the DCE detects an uncorrectable error in the configuration, it is expected that the DCE will cap the PCR.Authorities and PCR.Details by extending an error value into each D-RTM PCR and execute some PM-provided remediation code or reset the platform. The architecture of the DCE is such that the need for remediation is symptomatic of faulty hardware or an attack. In either case, remediation by the OS is not considered practical.

The DCE will extend detailed measurements of the DLME into PCR.Details and authority measurements for the DLME into the PCR.DLME.Authority.

It should be noted that the DCE timeline and behavior described above is for the DCE provided by the PM. A PM may provide a mechanism for a non-PM–provided DCE to be substituted and executed. This would result in the TPM having different measurements in the PCR.Authorities, and secrets that were sealed to the PM DCE PCR.Authorities values would not be accessible.

2.9.3 Conclusion

The D-RTM process puts the platform in a state that the platform manufacturer validates and that runs measured code. The PCR.Authorities and PCR.Details permit the measured code to access sealed data and prove the TCB was properly established. Based on the initial platform state and information passed to the DLME, it can continue to maintain a TCB by protecting its own memory, restricting access to Sensitive Resources, and controlling the platform configuration.

3 Trusted Computing Base

3.1 Introduction

This section defines key components that may be elements of the Trusted Computing Base (TCB) central to the D-RTM. Each component is defined and its requirements to comply with this specification are enumerated. If a platform implements the component described, the implementation SHALL conform to the requirements.

3.2 DMA (Direct Memory Access)

When devices have access to all of the system memory through DMA, then those devices are part of the hardware TCB. It is a goal of this specification that DMA devices not be included in the TCB.

To eliminate DMA from the hardware TCB, the software TCB must be able to prevent DMA devices from accessing portions of memory that are used by the TCB. The most common mechanisms are *exclusion* or *remapping*.

Exclusion uses a bitmap to prevent access to specific pages of memory. The TCB may SET or CLEAR bits in the bitmap to prevent DMA from accessing a specific page.

Remapping (e.g., an IOMMU or VT-d) uses page tables to limit access to system memory in a way that is similar to memory management of for CPUs.

3.2.1 DMA Requirements

The system shall provide a DLME-controllable mechanism for excluding DMA from selected portions of memory.

Some systems allow specific devices access to reserved portions of memory (e.g., video). If the memory allocation can be changed after system initialization, the registers used for allocation must be included in the Sensitive Resources List. Additionally, the DCE shall validate that these dedicated memory allocations do not use memory that is identified in the Sensitive Resources List as being memory available for use by the TCB.

Capabilities the PM must provide the DLME are described in the following table.

#	Capabilities the PM must provide to the DLME
1.	The platform SHALL provide the DLME the ability to turn on DMA remapping and buses without a Gap in its ability to control memory access.
2.	All devices on the platform SHALL have DMA remapping coverage.

3.3 System Management Mode

Some hardware platforms use a System Management Mode (SMM) for low-level platform hardware events and other events that need to be handled by platform firmware. Some examples are

- shutting down a system if it is in danger of overheating,
- handling chipset errors,
- memory hot adds and memory failover, and

- IO bus or IO adapter hot adds.

When a System Management Interrupt (SMI) occurs, the following takes place.

1) Normal execution stops.
2) The CPU state is saved.
3) The CPU enters a high-privilege SMM mode.
4) The SMM code associated with the SMI interrupt executes.
5) Upon completion, SMM code causes the CPU to exit SMM mode.
6) The CPU state is restored to its pre-SMM state and normal, less-privileged execution resumes.

While in System Management Mode, a CPU usually executes with an elevated level of privilege that permits it to access all memory. Virtual memory protections and DMA remapping protections do not apply to SMM code. SMM code can usually go anywhere and do anything on a CPU. SMM implementations are platform manufacturer-specific.

BIOS code early in the startup process loads SMM code into System Management RAM (SMRAM) and locks the SMRAM area so it is accessible only by a processor running in SMM. On hardware platforms with multiple processors, one processor may enter SMM while the other processors are executing normally. On some systems, SMM may be used to control very security-sensitive operations like firmware updates to the Core Root of Trust for Measurement.

While a processor is executing in SMM, if the SMM code jumps to code outside of SMRAM, it too executes with elevated privileges because the process is still in SMM. This can lead to very serious security breaches.

3.3.1 Protecting SMM

The SMM code and data residing in the SMRAM, when present, are considered to be part of the Trusted Computing Base (TCB) provided by the platform manufacturer. SMM code must defend itself at all times. DMA operations from IO devices and untrusted Non-Host Platforms (NHPs) SHALL never be allowed to update SMRAM.

The S-RTM SHALL place the SMM code in SMRAM and lock it. The DCE SHALL consider SMM to be “correct by construction.” The SMM implementation SHALL successfully maintain its code and data integrity indefinitely.

3.3.1.1 The SMM Threat

Due to the properties of SMM, running malicious code in SMM mode may permit near complete ownership of the hardware platform that is potentially undetectable and in some hardware architectures may even be able to persist across power cycles. Platform manufacturers must be extremely careful to implement appropriate safeguards for their SMM implementations. Any exploit of SMM on a hardware platform may permanently invalidate trust in the hardware platform.

3.3.1.2 SMRAM Locking Details

SMRAM SHALL be locked to stop potentially malicious code from being copied into SMRAM. SMRAM must be locked during the Pre-Gap time period while the platform manufacturer has a complete understanding of the system security state. Allowing code the platform manufacturer has not validated to execute prior to locking SMM is not permitted. Once the SMRAM area is locked (including the time after the D-RTM event) the SMM code must continue to protect itself.

3.3.1.3 SMM Overlays

An overlay is SMM code that is known to the PM prior to loading the SMM code space. When SMRAM is limited, adequate SMRAM space may not be available to load the entire SMM code base. In this case, the PM may break the code into segments that are loaded on an as-needed basis. The segments are overlaid in SMRAM when SMM code needs to execute the code in the segment. For example, the code to perform a platform manufacturer-controlled SMM update may not be permanently resident in SMM and only be loaded when an update is to be performed. The SMM segments not stored in SMRAM lack the protections provided to SMRAM when SMM is locked. SMM overlays are a case of an SMM callout. Anytime an SMM overlay is used, it must be verified beforehand.

SMM overlays execute in one of two scenarios.

1) Overlays are brought into SMRAM post D-RTM event (code incorporation).
2) Overlays are too large for the SMRAM area (hardware limitation), must reside outside of SMRAM, and be verified every time they are run, and must be protected from DMA by DMA remapping set up by post D-RTM code (code callout).

3.3.1.4 SMM Rollback

The BIOS, DCE, and SMM will typically be packaged together in an update. These components are susceptible to rollback attacks when an attacker tries to put older versions of weaker code onto a system. Platform manufacturers SHOULD prevent rollback attacks.

3.3.2 SMM Requirements

3.3.2.1 SMM Integrity

Because SMM has the ability to influence or control the state of the TCB, the platform manufacturer SHALL ensure the integrity of SMM is always maintained.

3.3.2.2 SMM Protections

The platform manufacturer SHALL ensure that the SMRAM is protected by hardware against attacks.

- The PM and the hardware supplier SHALL make sure all SMM protections exist and are enabled (i.e., cache attacks, SMM memory is locked, Flash ROM enables, etc.).
- Any code the SMM calls SHALL be trusted. All SMRAM protections SHALL be in place before entering the Gap and remain in place thereafter.
- DMA operations by peripherals cannot modify the SMRAM area.

3.3.2.3 Permitted Implementations of SMRAM Lock

An SMM lock SHALL be implemented. The SMRAM SHALL only be unlocked by a platform reset.

3.3.2.4 SMM Overlays

If SMM uses overlays, the manifest containing the hashes of all overlays must have been stored in SMRAM and recorded in PCR[0]. On each load of an overlay into SMRAM, SMM SHALL validate the overlay against the stored manifest.

3.3.2.5 SMM Callouts

SMM SHALL NOT perform callouts.

3.3.2.6 SMM Updates

SMM SHALL be updated through a secure update process (typically as part of a secure BIOS update).

After updating the SMM image in SMRAM, the platform SHALL be reset before the new image can be executed.

3.4 Non-Host Platforms

Modern platforms typically consist of one or more main CPUs that perform the majority of the computation tasks associated with the platform. The primary operating system or hypervisor and applications execute on these main CPUs, and the security properties of the platform depend primarily on the interaction between various software layers and the architectural features of the main CPUs. This primary platform is often called the Host Platform because it "hosts" or provides a foundation for one or more operating system images.

NHPs may exist within, or attached to, the Host Platform as discussed in Section 2.4 above. NHPs usually consist of a separate CPU or microcontroller that executes software or firmware that is autonomous from the primary operating system, hypervisor, and applications. In some cases SMM allows an independent software entity to use the resources of the primary CPU in ways that are independent of the main operating system or hypervisor. While SMM can be considered an NHP, it is so prevalent and commonly used that this specification treats it separately; for the purpose of this specification, SMM is not an NHP.

As mentioned in Section 2.4, if the access control mechanisms of the host's TCB can defend the TCB from the controller using the DMA protection, then that controller is considered to be a peripheral and not an NHP. If the TCB cannot defend itself from the controller using DMA protection, the controller is an NHP.

Some of these NHPs may provide information about the integrity of their field programmable firmware that may be useful for the TCB to evaluate. However, if an NHP can influence the measurement or verification process, there is little to no value in attempting to obtain the measurement or perform the verification because, by definition, the NHP is outside the protections provided by the DMA protection. Means other than DMA protection may be provided by the platform but are neither specified nor required by this specification. Further, some updatable programmable NHPs have the ability to affect memory configuration even from the CPU's point of view. DMA protection will not be sufficient here to protect the DCE's measurement or verification process from the NHP. For this reason, the verification of NHPs may be performed by the DCE but is not mandated by this specification.

If the state of an updatable NHP can be reliably reported by the DCE, then it is a DCE-verifiable NHP. If the DCE cannot reliably report on an NHP, then it is a DCE-unverifiable NHP. If a DCE-unverifiable NHP has been compromised, there is no way for the DCE to verify if any mitigations for such an NHP are in place. The categories for NHP then are: hardware, DCE-verifiable, and DCE-unverifiable. A peripheral is by definition not a NHP.

The platform's NHPs are the responsibility of the PM and trust in the platform may require implicit trust in the PM. The identity of the NHP, either called out specifically or implied, is contained within the PM's Platform Certificate. If the DCE does not measure or verify a field programmable NHP, there will be no indication from the DCE about the state of that NHP. This means that if a vulnerability or flaw is ever discovered in a field-programmable NHP, since the DCE cannot assert that the firmware has been fixed/patched, *all* systems containing that unmitigated NHP may be considered not trustworthy. While a DCE-unverifiable NHP is the responsibility of the PM, it is necessary to state general requirements in order to meet a minimum set of security properties required to create a trusted platform. One of these requirements is the security properties of the firmware update process for DCE-unverifiable NHPs. In order to keep the operation of the NHP under the control of the PM, unauthorized update of DCE-unverifiable NHPs must be prevented. For this reason, this specification requires that all updates to DCE-unverified NHPs be done using one of the industry-accepted signature schemes. This specification will not state which scheme is to be used, but the goal is to achieve a minimum security strength of 112 bits (e.g., RSA 2048 bit).

3.4.1 Requirements for Non-Host Platforms

The DCE shall measure or verify the platform's DCE-verifiable NHPs. If an NHP is not measured or verified (e.g., it's a DCE-unverifiable NHP)

- its trust is represented in the Platform Certificate, and
- its update mechanism must use a cryptographically signed scheme that has a security strength of at least 112 bits (e.g., RSA 2048).

3.5 Peripherals

Peripheral devices are I/O devices that can be controlled by the DCE, operating system, or hypervisor. The existence of I/O isolation hardware, which provides DMA protections, prevents peripherals from compromising the integrity of the TCB. The DCE process may have a role in establishing protections for the TCB against attack from peripherals but it is not concerned per se with the direct assessment of the peripheral itself.

Peripherals are not evaluated during the D-RTM process and may be added after platform manufacture by the platform owner. The addition of a peripheral is not considered a hardware attack.

3.5.1 Peripheral Requirements

- Peripherals SHALL NOT be able to compromise the D-RTM process. Security features on the Host Platform must isolate the D-RTM process from attack by peripherals (e.g., DMA access) and continue the isolation protections up to and including the launch of the DLME. (If a peripheral does not meet this requirement, it is a Non-Host Platform, not a peripheral.)
- If the hardware platform supports a Gapless S3/Resume process, when an S3/Resume cycle occurs, security mechanisms SHALL persistently protect Host Platform memory from peripherals. This MAY be implemented by retaining the security settings used to isolate peripherals across the S3/Resume cycle or MAY be implemented by resetting the security settings used to isolate peripherals during the S3/Resume cycle.
- The security isolation mechanisms (e.g. DMA protection) for peripherals must be active from Host Platform reset until completion of all actions on which the DCE has a trust dependency.

3.6 TPM

A Trusted Platform Module (TPM) is used by the D-RTM process to record integrity measurements when a DLME is successfully launched. The purpose of the D-RTM process is to establish a TCB that will be able to unseal data or attest the platform state to a remote entity. Without a TPM, neither unsealing nor attestation is possible.

There will be situations where a platform owner disables, deactivates, or does not enumerate the platform's TPM. Without an enumerated, activated, and enabled TPM, the DLME will be unable to unseal data or attest to its state.

For some implementations, the measurements of some S-RTM components, such as SMM, are important prerequisites for the DCE to validate the platform state during the D-RTM process.

3.6.1 TPM Requirements

1) The platform must have a TPM version 1.2 with revision 116 or greater to be compliant with this specification.
2) The TPM SHALL be discoverable and operational.

3.7 TPM Interface

The TPM interface lets the Host Platform communicate with the TPM.

3.7.1 TPM Interface Requirements

The platform must be compliant with TPM Interface Specification (TIS) revision 1.21 or greater.

3.8 S-RTM

The D-RTM process requires the S-RTM to follow the PC-Client specification(1) except as otherwise explicitly stated in this specification.

3.8.1 S-RTM Requirements

A D-RTM platform SHALL implement S-RTM as documented in the TCG PC Client (1) or Server specifications.

Additional S-RTM requirements required by D-RTM are detailed in this section.

3.8.2 S-CRTM Validation

The S-CRTM is inherently trusted by definition.

3.8.3 S-RTM Update Requirements

The platform SHALL, at a minimum, use a secure update process for the S-CRTM. A system that intends to implement a platform manufacturer-controlled BIOS update must have the following:

- protected memory environment (e.g., SMM or Pre-Gap BIOS);
- processor running the platform manufacturer's validated flash code (i.e., verification and flash algorithm).

When the update occurs, the system state must allow the update code to protect the flash update process from interference from peripherals or from other processors not executing the platform manufacturer's validated update code.

The portion of the S-RTM image described above must be cryptographically signed by the PM with the PM's private key. The signing method is vendor-specific but should have a strength of function that is no less robust than the algorithm used for extending the D-CRTM measurements into PCR.Authorities.

Any mechanism (not specified here) may be used to deliver this code image to the platform and verify the code is signed by the proper authority prior to flashing the image, but it cannot degrade the security of the update.

3.8.4 S-RTM S3 Resume

On resume from S3, the S-RTM is executed and control is passed to the ACPI resume vector if present, otherwise an S5 reset occurs. The trusted OS is responsible for establishing a resume vector that calls the code necessary to trigger a D-RTM event and execute a DLME.

NOTE Malicious code could reset the "resume" vector, leading to a "denial of resume" attack failing to trigger the trusted launch of the DLME.

Before the trusted OS enters S3, it must leave in memory sufficient information to enable the DLME to re-establish the trusted OS. Common techniques such as signatures, encryption, and sealing to PCR values may be used.

3.9 Sensitive Resources

3.9.1 Sensitive Resource Definition

Sensitive Resources are defined as addresses (either memory or I/O) to which the DLME TCB must restrict access in order to protect its integrity. Examples of Sensitive Resources are DMA remapping hardware controls or memory controller settings. The Sensitive Resources must be restricted to prevent any access from occurring by elements outside the TCB, such as, peripherals using DMA access, ACPI methods not validated by the platform manufacturer, or less privileged code than the DLME TCB permits to execute.

Generally, the Sensitive Resources fall into one of these three categories:

1) Architectural elements the DLME understands and interacts with directly by accessing the Sensitive Resource. Addresses of this type would be the addresses of the DMA remapping hardware.
2) Addresses that are required to be accessed by ACPI in order to perform security sensitive functions. Addresses of this type might include a memory controller which ACPI must access in order to deal with a hot-add or RAS event. It is preferred that these operations be handled by SMM so that it is not required that the AML interpreter be included in the minimal TCB of the platform.
3) Addresses of settings that if changed could impact the ability of the TCB to defend itself. Addresses of this type might include basic configuration of the bus on which the DMA remapping hardware resides. While a hardware lock would be preferable, the TCB may be required to do the access enforcement.

3.9.2 Sensitive Resource Granularity

The smallest unit of protection allowed by this specification is a byte, which is the minimum granularity of address resolution on most computers. A write to less than a byte might require a read-modify-write and there is no assurance that the read-modify-write operation will have the correct behavior.

It is likely that the DLME TCB will need to use page-granular memory protections in order to guard the locations on this list. As a consequence, an impact on system performance might occur if a Sensitive Resource was in the same page as a non-sensitive location that was accessed often. System implementers are encouraged to place Sensitive Resources on pages that do not contain non-Sensitive Resources.

If the Sensitive Resource is at an address that can be relocated by changing a PCI or PCI Express Based Address Register (BAR), the resource can be difficult to protect because its address can be changed via the 0xCF8/0xCFC index/data pair. Using DLME TCB resources to trap every access to this often-used index/data pair would be extremely expensive in terms of performance.

Even though the PCI Express Extended Configuration Space is accessible directly through Memory Mapped Input/Output (MMIO) space, a BAR located there should not be put on the Sensitive Resources List because the BAR address is based upon the bus/device/function, and if the device is not on bus[0], then the primary and subordinate bus numbers can be changed, which also changes the BAR address.

If the index/data pair BAR is immutable (i.e., protected by hardware), the TCB does not have to protect the BAR and it does not have to be listed as a Sensitive Resource.

3.9.3 Sensitive Resources Requirements

1) Sensitive Resources SHALL exist with byte-level granularity. For example, a single bit Sensitive Resource must not be part of the same byte as a bit required for normal platform operation.
2) Sensitive Resources SHOULD be grouped onto a minimal number of pages separate from other resources. This is because the DLME TCB is likely to use page-level granularity to protect access to Sensitive Resources and each access to a protected page will incur overhead.

3) If the location of a Sensitive Resource can be relocated, the mechanism that controls the relocation SHALL be hardware-protected or a Sensitive Resource itself.
4) If the location of a Sensitive Resource is accessible in multiple ways, each mechanism for access SHALL be listed as a Sensitive Resource.
5) Operation of software outside the TCB SHALL NOT require access to a Sensitive Resource with the exception of access by the platform manufacturer-validated ACPI methods that are part of the TCB.

3.10 Memory Map

The DLME needs to know that any memory address it uses for maintaining the TCB is a conventional memory address and not a memory-mapped I/O address which is dual ported. The DCE determines the memory configuration as a by-product of validating that the memory controllers have the proper settings and passes that information to the DLME.

3.10.1 Memory Map Requirements

The platform manufacturer SHALL be able to define the hardware platform conventional memory to populate the D-RTM Resource Table with valid information.

3.11 ACPI Name Space

The Advanced Configuration and Power Interface (ACPI) specification was designed to facilitate configuration and power management of a computer system through virtualization of system hardware components for an ACPI-aware operating system. This virtualization is composed of interpreted executable code, hardware, and system feature tables that an operating system can use to configure and power-manage on a computer without having knowledge of the intimate hardware details of a particular system. This collection of code and tables is referred to as the ACPI Name Space, or just Name Space (NS) in this section of the D-RTM specification.

- The NS is supplied by the product manufacturer and is most often embedded in the product manufacturer system BIOS. The NS is not necessarily static through successive OS (ACPI-aware operating system) boots. As product manufacturer supplied hardware and features change, so necessarily does the NS. The NS is meant to live on after product manufacturer BIOS POST. It is consumed by the OS and can, and often must, be utilized to communicate to TCB Sensitive Resources. Because of this, the DCE must present to the DLME portions of a product manufacturer validated NS.
- The BIOS places the NS in system memory before control is handed to the OS boot loader. It is found by the OS by searching for an ASCII string that denotes the start of the NS. The NS is essentially a linked list of tables containing executable code and data that correspond to the hierarchical topology of the system. Because the NS is left unprotected in memory, it can be attacked by malicious software agents whose intent is to compromise the TCB.
- In order to protect the configuration of TCB Sensitive Resources, the platform must be able to communicate to the DCE unmodified versions of those sensitive portions of the NS. The actual mechanism used to protect and communicate the NS is beyond the scope of this specification, but common techniques may include: sequestration in protected memory; reconstruction from trusted code; or immutable copies.

3.11.1 ACPI Requirements

The platform manufacturer SHALL implement the D-RTM ACPI table in accordance with this specification and the ACPI 5.0, plus errata, specification.

Any ACPI table, including the D-RTM table, which is critical to the defense of the TCB, SHALL be validated by the DCE. All DCE-validated ACPI tables SHALL be placed in the Validate_Tables_List (see 4.3.1) which is made available to the DLME.

3.11.2 Protection from AML

Interpreted ACPI Machine Language (AML) is often provided by platform manufacturers to abstract implementation-specific device details away from higher-level software. The low-level access required by many AML methods has the potential to manipulate the TCB. It is essential that any ACPI interpreter take all steps necessary to protect the TCB from being compromised by AML. The details on how an ACPI interpreter can protect the TCB are beyond the scope of this specification.

3.12 Power State Transitions

All the actions documented in PC Client Specification(1) must be performed during power state transitions. The following defines additional actions that need to be performed.

- The DL_Entry (DCE preamble) and DLME_Exit code are provided by the DCE provider. This code will be run during the S0-to-S3-to-S0 (suspend/resume) transition. If this code is not validated by the DCE provider during the DCE, then the DL_Entry and DLME_Exit code will be considered to be Gap code and GapCodeOnS3Resume in DRT_Flags must be SET(1).
- If GapCodeOnS3Resume is CLEAR(0), all code in the suspend/resume path must be validated by the DCE. This code includes the DL_Entry, DLME_Exit, and S-CRTM. It does not include the code provided by the DLME. The memory containing DL_Entry and DLME_Exit, shall be included in the protected resources list so that the DLME can protect it during DLME operation.
- If GapCodeOnResume is SET, then the DLME will need to provide protections for system secrets during the suspend/resume. If the DLME cannot provide protections, the DLME may choose not to support suspend/resume.
- For any power state transitions from S0, the DLME is required to call DLME_Exit (with power state transition as parameter) to indicate the transition and allow the PM code to prepare the system for a transition.
- If the DLME_Exit and CRTM are validated by the DCE, and the CRTM reloads validated DL_Entry code on each restart, then this is logically equivalent to having the DCE check the DL_Entry code.
- CRTM code SHALL set up DMA protections during resume from S3 before any DMA hardware is enabled.

3.12.1 Power State Transition Requirements

1) DCE SHOULD validate the DL_Entry (DCE preamble) and DLME_Exit code.
2) DL_ENTRY and DLME_Exit code SHOULD be added to the Sensitive Resources list.
3) If the DCE does not validate DL_Entry, DLME_Exit, or add both to the Sensitive Resources list, the GapCodeOnS3Resume flag SHALL be SET.
4) Power state transitions from S0 to S3, S4, or S5 SHOULD call DLME_Exit to allow PM code to prepare the system for a transition.
5) TPM_SaveState SHOULD be the last command before transitioned to S3. Issuing any command to TPM after TPM_SaveState is issued could cause TPM_Startup failures on resume.
6) DLME SHALL take all measures to defend itself in case of non-Gapless resume.
7) DMA protections SHALL be set up before any DMA hardware is enabled on resume from S3.

3.13 DLME

The DCE transfers control to the DLME in a trusted state with a well-defined TCB. The processor has virtual memory (paging) enabled and the DMA remapping engine configured to prevent peripheral access to memory. A minimal DLME that only accesses data from within its measured address range (DlmeStart for DlmeLength bytes) or stack, and only executes code from within its measured address range will maintain that initial TCB.

The semantics of a DLME is beyond the scope of this specification. However, a complex DLME can choose to access resources outside its measured address range but should take steps to ensure that it does not compromise the measured address range. Any data accessed outside these initial trusted ranges must be validated before use in any manner that may affect the trusted state. Any code contained outside these initial trusted ranges should be measured or validated before execution. Any changes to the DMA remapping engine should be done in a manner that will not compromise the DLME's trusted state (e.g., prevent DMA to the DLME's code segment). Any changes to the page table should be performed through virtual addresses that have been validated to map to the physical page table pages using the set of DCE-validated mappings.

3.14 DCE Preamble

The DCE preamble is PM-provided code that starts the D-RTM process. The entry point to the code is either DL_Entry32 or DL_Entry64 (3.14 above). It may perform some initial configuration actions that are platform-specific before invoking the DL Event. This may include creating additional page table entries, rendezvousing non-boot processors, or configuring devices into a known state. Some implementations of the DCE preamble may do some basic validation and return an error code for issues documented in this specification. Because the DCE preamble is part of the Gap, the DCE cannot assume it has performed its actions when constructing the DLME's trusted environment and may need to enter remediation.

When the platform supports sleeping and resuming without entering the Gap, the DCE preamble is considered part of the TCB. When DLME_Exit is called with the intent of entering sleep, the platform may be required to take steps necessary to ensure the correctness of the DCE preamble. Alternatively, the BIOS upon resume from sleep may be required to take similar steps.

3.14.1 Preamble Requirements

A platform that supports Gapless S3 resume from sleep SHALL take steps to ensure a valid DCE preamble. The validation may be either a complete DCE preamble validation by the BIOS upon resume for sleep, or may be steps necessary to verify a DLME_Exit was performed before S3 sleep.

3.15 DLME_Exit

When the DLME either decides to leave its trusted state or transition to a lower power state (e.g., S3 sleep) it SHALL call DLME_Exit to allow the platform to either perform the steps necessary to return the platform to full (but less trusted) operation, or the steps necessary to allow a Gapless resume. When the prior DL_Entry was executed, the platform may have restricted, or allowed, certain platform capabilities that are tied to the trust state of the platform (e.g., enabling hardware virtualization only while in a trusted state). Failure to call DLME_Exit may prevent full platform operation or the ability to resume from sleep.

3.15.1 DLME_Exit Requirements

The DLME SHALL call DLME_Exit when either leaving its trusted state or entering a low power state. Failure to call DLME_Exit MAY cause a power-on platform reset.

3.16 CPU Microcode Updates

The Host Platform CPU may have microcode that is field-updatable. Microcode updates have factors that vary between vendors. Factors include the following.

- Are CPU microcode updates permitted at all?
- Does a CPU validate that microcode patches are from the CPU manufacturer before installation?
- Are CPU microcode patches cumulative or does only the most recent patch apply?
- What CPU features are field-updatable through CPU microcode patches?
    - Do CPU microcode patches have the ability to influence the CPU's reporting of the microcode patches currently installed?
    - Is the microcode for the DL Event updatable?
    - Are microcode updates able to perform the HASH_START command at Locality 4?
- For machines with multiple CPUs, can different CPU microcode patches be installed on different CPUs?
- Do CPU microcode updates installed persist across:
    - The DL Event
    - Sleep/resume

Most CPUs have the ability to install CPU microcode updates. A CPU may validate that a microcode patch was signed by the CPU manufacturer before permitting it to be installed. CPU microcode patches could be cumulative, or may discard previous CPU microcode patches during installation.

The D-RTM process should extend a measurement into the Details register that reflects installed microcode patches. In implementations where microcode patches may not persist across a DL Event and may be reloaded at some point after the event, the measurement reflecting the microcode patch SHALL be extended into the Details register at the time the microcode is loaded. In implementations where Microcode patches do persist across a DL Event, those patches SHALL be securely loaded by the platform.

Per the TCG TPM PC Client Specific TPM Interface Specification, the platform chipset is involved in asserting Locality 4 when resetting the D-RTM PCR measurements as part of the DL Event and subsequently establishing the D-RTM chain of measurements. For this reason, CPU microcode updates must not be able to assert Locality 4 without the subsequent microcode establishing the D-CRTM.

3.17 Dynamic Core Root of Trust for Measurement (D-CRTM)

3.17.1 Definition

The D-CRTM is the Core Root of Trust for the dynamically launched environment that is initiated by a DL event. As a result of the event, the processor is placed into a known good state, the dynamic PCR are reset (6.1), and the processor will execute immutable code that performs the initial measurements. The implementation of the immutable code is platform-specific, including its size and its functionality. The D-CRTM ends when the required measurements are extended. The initial measurements SHALL be extended into PCR.Details and the authority extended into PCR.Authorities; by issuing the TPM_HASH_START, TPM_HASH_DATA, and TPM_HASH_END commands to the TPM.

3.17.2 D-CRTM Requirements

- The D-CRTM SHALL be "immutable" code (as defined by the PC Client specification).
    - The D-CRTM may not be changed, or

    - The D-CRTM may be changed through a vendor-defined, secure D-CRTM update process that SHOULD have a minimum strength of function of 112 bits.
- The D-CRTM SHALL reset the D-RTM PCR to zero using the TPM_HASH_START and TPM_HASH_END commands. The process SHOULD use the TPM_HASH_DATA command.
- The TPM_HASH_START command SHALL only be performed by the D-CRTM at Locality 4.
- The D-CRTM MAY be monolithic or MAY consist of more than one component.
- If the D-CRTM is unable to execute or successfully measure code, it SHALL enter remediation.
- The D-CRTM SHALL establish its authority as either the key that signed the D-CRTM, or the hash of the D-CRTM itself.
- The D-CRTM SHALL extend a measurement of its authority into PCR.Authorities.
- The D-CRTM SHALL extend a measurement of its identity into PCR.Details.
- The D-CRTM SHALL extend measurements of the DCE into PCR.Details and PCR.Authorities (as defined in section 6.2 below).
- If the D-CRTM consists of more than one component and if components other than the initial component of the D-CRTM are updatable, the D-CRTM SHALL record the detail measurements of all subsequent components that are part of the D-CRTM before invoking the subsequent components. These detail measurements SHALL be recorded in PCR.Details.
- If any microcode patches do persist after the DL Event, the D-CRTM MAY extend a measurement of its microcode version (if modifiable) into PCR.Details.
- If the microcode patches do not persist after the DL Event, a measurement of the microcode SHALL be made to PCR.Details when they are loaded.
- Before the D-CRTM invokes a component that is not part of the D-CRTM, it SHALL extend the measurement of the component in PCR.Details.
- Before the D-CRTM invokes a component that is not part of the D-CRTM, it SHALL extend at least one measurement into PCR.Authorities. The measurement SHALL be an authority measurement of the component.
- If the D-CRTM is unable to record the measurements described above, it SHALL do one or more of the following:
    - It SHALL extend an error code into PCR.Details and PCR.Authorities.
    - It SHALL enter platform remediation.

4 Structures

4.1 Introduction

This section defines the syntax and semantics of tables and structures used for the D-RTM. Explanations of the structures and when they are used are defined elsewhere in this specification.

4.2 D-RTM Resources Table Structure

4.2.1 Description

The D-RTM Resources Table is an ACPI table that is the primary communication method from the PM-provided DCE to the DLME. The contents of the D-RTM Resources Table are validated by the DCE process before being passed to the DLME. The D-RTM Resources Table is also used to define the entry point and address mode for calling the DCE preamble.

The D-RTM Resources Table may be used both before and after the D-RTM process. The platform manufacturer must set the table values during the startup process such that the table information is correctly initialized. Platform owners should be aware that code outside the platform manufacturer's control, even if it is part of the Static Root of Trust, may modify the table values. Operating system vendors should be aware that Gap code may modify the table values. All of the contents of the D-RTM Resources Table must be validated by the DCE during the D-RTM process.

The platform manufacturer may guarantee the D-RTM Resources Table is accurate in situations other than those described above. Such situations are platform-specific.

4.2.2 Table Structure

Table 1 — D-RTM Resources Table (DRT)

Field	Byte Length	Description
Header		A standard ACPI header.
Signature	4	'DRTM.' Signature for the D-RTM resources table.
Length	4	Length, in bytes, of the entire D-RTM Resources Table.
Revision	1	For this version of this specification, this field must be set to 1.
Checksum	1	Entire table must sum to zero.
OEMID	6	OEM ID.
OEM_Table_ID	8	For the DRT, the table ID is the manufacturer model ID.
OEM_Revision	4	OEM revision of DRT table for supplied OEM Table ID.
Creator_ID	4	Vendor ID of utility that created the table. For the DSDT, RSDT, SSDT, and PSDT tables, this is the ID for the ASL Compiler.
Creator_Revision	4	Revision of utility that created the table.
DL_Entry_Base	8	Base address for DCE preamble.
DL_Entry_Length	8	Size of the DCE preamble in bytes.
DL_Entry32	4	Physical entry address of the 32-bit DCE preamble. If set to zero, the platform does not support D-RTM in 32-bit mode.
DL_Entry64	8	Physical entry address of the 64-bit DCE preamble. If set to zero, the platform does not support D-RTM in 64-bit mode.
DLME_Exit	8	Physical entry address for the DLME exit routine. If none exists, this entry is 0.
Log_Area_Start	8	Contains the physical address of the start of the system's D-RTM event log area.
Log_Area_Length	4	Length in bytes of the D-RTM event log area.
Architecture_Dependent	8	Contains the physical address of a CPU-architecture-dependent area. If none exists, this entry is zero (0).
DRT_Flags	4	D-RTM Resources Table Flags (See 4.3).
VTL_Length	4	Number of entries in Validated_Tables_List.
Validated_Tables_List	VTL_Length * 8	A list of the physical addresses of validated ACPI tables.
RL_Length	4	Number of entries in Resources_List (See 4.4.1).
Resources_List	RL_Length * sizeof(ResourceDescriptor)	A list of resource descriptors. Each entry is a ResourceDescriptor shown in Table 3.
DPS_Length	4	Number of platform-specific identifiers this hardware platform supports.
DLME_PlatformSpecificIds_Supported	16	A list of platform-specific identifiers this hardware platform supports.

4.2.3 D-RTM Resources Table Header

The D-RTM Resources Table Header area is a standard ACPI header.

4.2.4 DL_Entry_Base

The start address of the platform manufacturer's DCE preamble in physical memory.

4.2.5 DL_Entry_Length

The size of the platform manufacturer's DCE preamble. This contiguous range may also include other PM components in addition to the DCE preamble.

4.2.6 DL_Entry32

This is the 32-bit entry point for the DCE preamble. The location of DL_Entry32 is determined by the PM. See section 5.3.3 for more information.

4.2.7 DL_Entry64

This is the 64-bit entry point for the DCE preamble. The location of DL_Entry64 is determined by the PM. See section 5.3.3 for more information.

4.2.8 DLME_Exit

This is the address of the DLME_Exit routine. See section 5.3.5 for more information.

4.2.9 Log Area

The log area for the DCE is defined by the PM. This ensures that the DLME has a simple mechanism for finding the log. Since the measurements done during the DCE are known by the PM, the log can be sized as required. The area reserved for the log does not need additional space for events that occur after the DLME has started. The log is not cumulative; if multiple D-RTM events occur during a single power cycle, it is overwritten each time to represent only the most recent D-RTM event.

The log area must be physically contiguous. It must be located in a region of memory that is either ACPI reclaimable or firmware reserved. It must not be placed in memory that is identified as requiring protection in the Resources_List.

The Log_Area_Start and Log_Area_Length are placed in the D-RTM Resources Table for the use of the DLME. The caller to DL_Entry cannot move the log area used by the DCE by changing the values in the D-RTM Resources Table.

4.2.10 Architecture_Dependent

Different CPU architectures may have additional requirements in order to describe the DCE execution environment for the DCE. The additional information is referenced by the D-RTM Resources Table using the Architecture_Dependent address field and any data passed must be checked by the DCE preamble and DCE for consistency of the architecture. The structure of this data is not defined by this specification.

The DCE should perform a consistency check on any data passed using this or any other mechanism.

4.3 D-RTM Resources Table Flags (DRT_Flags)

This structure is used in the D-RTM Resources Table.

Table 2 — DRT_Flags

Flag	Bit	Description
NameSpaceInTCB	0	**SET (1):** Indicates that one or more methods in a validated ACPI table will access one or more of the Sensitive Resources listed in the Resources_List. **CLEAR(0):** Indicates that no ACPI method should access a Sensitive Resource in the Resources_List.
GapCodeOnS3Resume	1	A 1 indicates that Gap code runs on S3 resume on the platform.
GapCodeOnDLME_Exit	2	A 1 indicates that platform measurements no longer fully represent the platform state after DLME_Exit is called. For example, TPM measurements may be incomplete because an unmeasured controllable Non-Host Platform may have been restarted by DLME_Exit.
PCR_AuthoritiesChanged	3	**SET (1):** Indicates that PCR 18 is PCR.Authorities and PCR 17 is PCR.Details. **CLEAR (0):** Indicates that PCR 17 is PCR.Authorities and PCR 18 is PCR.Details.
Reserved	31:4	Reserved for future use. All reserved bits SHALL be set to zero and ignored.

4.3.1 Validated_Tables_List

The Validated_Tables_List is a list of the addresses of all ACPI tables that have been checked by the DCE.

A validated ACPI table is one for which all the table fields are accurate and for which any additional data pointed to by the table is accurate, with the sole exception of other ACPI tables for which the address is listed in the table. If a validated table directly contains AML code, the AML code must be validated. For example, for the D-RTM Resources Table to be validated, the address for DL_Entry must point to executable code that is validated to invoke the platform manufacturer's actions to start the D-RTM process. This means both the DL_Entry address value and the memory contents pointed to by the address are valid. Validation MAY include establishing correctness by construction.

The platform manufacturer must always list the D-RTM Resources Table as a validated table. For any hardware, such as the DMA remapping hardware, that the platform manufacturer expects the DLME to use, the platform manufacturer must validate the corresponding ACPI table and include it in the list of validated ACPI tables.

When the NameSpaceInTCB flag is SET, AML code in the validated ACPI tables SHOULD be permitted to access addresses listed as sensitive on the resources list. It is recommended the platform manufacturer minimize the number of validated ACPI tables that need to have their AML code execute with a privilege level that permits them to access the resources listed as sensitive in the Resources_List.

ACPI methods contained in a validated table MAY be permitted to access an address identified as sensitive in the Resources_List and only if the NameSpaceInTCB is SET. Other ACPI methods SHALL NOT access such sensitive resources.

Only complete ACPI tables MAY be validated. Any table that cannot be completely validated should be divided into privileged tables that may access Sensitive Resources and unprivileged tables that may not access Sensitive Resources.

The Validated_Tables_List is itself a Sensitive Resource and SHALL be protected.

4.4 Resource Descriptor Structure

Each instance of this structure defines a Sensitive Resource or a conventional system memory range. This structure is used in the D-RTM Resources Table.

Table 3 — ResourceDescriptor Structure

Field	Byte Length	Description
RegionSize	7	Size of the resource in bytes.
Type	1	Resource type and protections: **Bit 0:1** **Value** — **Indicated Resource Type** 00b — Conventional system memory in memory address space. 01b — Sensitive area in memory address space. 10b — Sensitive area in I/O address space 11b — Sensitive area in PCI configuration address space **Bit 2:6** — Reserved for future use. All reserved bits SHALL be set to zero and ignored. **Bit 7** — (Must be ignored if bits 0:1 are 00b) **Value** — **Protections** 0 — Write protected. Reads allowed. For PCI configuration and I/O addresses, the TCB may block read and write access but read access should be permitted for conventional memory. 1 — Read and write protected.
Address	8	Physical address in address space indicated by *Type*

4.4.1 Resources_List

The Resources_List is used to describe conventional memory and sensitive memory addresses that a software TCB must protect to maintain its integrity.

A location in conventional memory may be listed twice, once as a portion of conventional memory and once as a sensitive memory address. The entry indicating that this memory is sensitive must be page granular and page aligned. That is, any page in conventional memory that is to be protected as sensitive by the TCB will be protected with page granularity. A sensitive memory address that is not page granular MUST be a memory-mapped I/O address.

All of conventional memory that is available for use by the TCB must be listed in the Resources_List as conventional memory. The conventional memory is included in this list, largely for convenience of the DLME.

4.4.2 DLME_PlatformSpecificIds_Supported

A list of platform-specific identifiers (see 4.6.3.5 below) for DLME platform-specific attributes that this hardware platform supports. DLME installation utilities should look at this value to select a DLME to install that is appropriate for this hardware platform. The DCE validation will examine this field to make sure the DLME it launches is appropriate for the hardware platform.

4.5 DRTM_PUBLIC_KEY Structure

4.5.1 Description

The DRTM_PUBLIC_KEY structure is used to hold a public key that verifies a signature used in the D-RTM process. It is derived from a subset of the TPM 2.0 TPMT_PUBLIC data structure.

4.5.2 Structure Definition

Table 4 — DRTM_PUBLIC_KEY

Field	Byte Length	Description
Type	4	The TPM_ALG_ID associated with this object.
Parameters	variable	The TPMU_PUBLIC_PARAMS algorithm or structure details.
Key	variable	The TPMU_PUBLIC_ID containing the public key.

4.5.3 Structure Fields

4.5.3.1 Type

The TPM_ALG_ID associated with the public key. The possible values are defined by the TCG and published in other specifications and the value used SHALL be associated with an asymmetric algorithm.

4.5.3.2 Parameters

The algorithm-specific parameters or structure details that may be required for the public portion of the key. The TPMU_PUBLIC_PARAMS is defined by the TPM 2.0 specification and selected by the **Type** field.

4.5.3.3 Key

The public key unique data associated with the object created using the **Type** hash algorithm and parameters. The TPMU_PUBLIC_ID is defined by the TPM 2.0 specification.

4.6 DLME_DESCRIPTOR Structure

4.6.1 Description

The DLME_DESCRIPTOR structure is used to describe the DLME. The DCE uses the structure to validate and measure the DLME.

4.6.2 Structure Definition

Table 5 — DLME Descriptor

Field	Byte Length	Description
DlmeStart	4	The address in virtual memory where the DCE will begin calculating the digest of the DLME.
DlmeLength	4	The number of bytes the DCE will use in calculating the digest of the DLME.
DlmeEntryPoint	4	The virtual address of the DlmeMain entry point. The DCE jumps to this address using the platform's C calling convention with the arguments defined for DlmeMain.
StackSize	4	The number of bytes to be allocated by the Gap for the DLME.
TCGDefinedAttributes	8	Describes basic well-known properties of the DLME, some of which may be useful to determine if the DLME is appropriate for the platform.
DlmeDigest	variable	The digest of the DLME from DlmeStart for DlmeLength bytes.
DlmeSignature	variable	The signature of DLME_DESCRIPTOR, excluding the DlmeSignature field, signed by the authority for the DLME.
DlmePublicKey	Variable	The DRTM_PUBLIC_KEY used to sign the descriptor.

```
typedef structure _DLME_DESCRIPTOR {
   UINT32 DlmeStart;
   UINT32 DlmeLength;
   UINT32 DlmeEntryPoint;
   UINT32 StackSize;
   UINT64 TCGDefinedAttributes;
   TPMT_HA DlmeDigest;
   TPMT_SIGNATURE DlmeSignature;
   DRTM_PUBLIC_KEY DlmePublicKey;
} DLME_DESCRIPTOR, *PDLME_DESCRIPTOR;
```

4.6.3 Structure Field Definitions

4.6.3.1 DlmeStart

The address in virtual memory where the DCE begins calculating the digest of the DLME.

4.6.3.2 DlmeLength

The number of bytes the DCE uses in calculating the digest of the DLME.

4.6.3.3 DlmeEntryPoint

The virtual address of the DlmeMain entry point. The DCE jumps to this address using the platform's C calling convention with the arguments defined for DlmeMain.

4.6.3.4 StackSize

The number of bytes to be allocated by the Gap for the DLME.

4.6.3.5 TCGDefinedAttributes

A bitmap describing basic well-known properties of the DLME, some of which may be useful to determine whether the DLME is appropriate for the platform. The DCE will compare the properties of the DLME to the current platform when determining whether it is appropriate to launch the DLME on the current hardware platform and configuration. For example, to avoid launching a 64-bit DLME in 32-bit mode, the DCE would evaluate these flags to make sure the process mode matched the DLME implementation.

The definition of these field attributes are set by the TCG.

Table 6 — TCGDefinedAttributes

Field	Bit	Description
DlmeBit	0	The DLME is a 32-bit x86 executable.
Dlme64Bit	1	The DLME is a 64-bit x64 executable.
DlmeXorAliasDetection	2	Each 4,096-byte page of the DLME is exclusive ORed with its virtual address to enable alias detection.
Reserved	3-63	Reserved.

4.6.3.6 DlmeDigest

The digest of the DLME from DlmeStart for DlmeLength bytes.

The DlmeDigest is structured as a TPM 2.0 TPMT_HA digest.

4.6.3.7 DlmeSignature

The signature of DLME_DESCRIPTOR, excluding the DlmeSignature and DlmePublicKey fields, signed by the Authority for the DLME.

The DlmeSignature is structured as a TPM 2.0 TPMT_SIGNATURE.

4.6.3.8 DlmePublicKey

The public key used to generate the DlmeSignature.

The fields in a DRTM_PUBLIC_KEY are constructed using TPM 2.0 structures.

4.7 DRTM_ACPI_TABLES

4.7.1 Description

Pointers to several ACPI tables are passed to the DLME. Instead of passing each table as individual arguments, they are combined into this structure.

4.7.2 Structure Definition

Table 7 — DLME_ACPI_TABLES

Field	Byte Length	Description
Drtm	4	The physical address of the D-RTM Resources Table Structure.
Rsdt	4	The physical address of the RSDT ACPI Table Structure.
Dmar	4	The physical address of the DMAR ACPI Table Structure

```
typedef structure _DLME_ACPI_TABLES {
   UINT32 Drtm;
   UINT32 Rsdt;
   UINT32 Dmar;
} DLME_ACPI_TABLES, *PDLME_ACPI_TABLES;
```

4.7.3 Structure Field Definitions

4.7.3.1 Drtm

The physical address of the D-RTM Resources Table Structure.

4.7.3.2 Rsdt

The physical address of the RSDT ACPI Table Structure.

4.7.3.3 Dmar

The physical address of the DMAR ACPI Table Structure.

4.8 DLME_ARGUMENTS Structure

4.8.1 Description

The DLME has a set of arguments that are constructed by the Gap and passed through the DCE to the DLME. The contents of the arguments are opaque to the DCE and may contain any arbitrary values. The DCE does not validate the arguments and the DLME must treat the contents as untrusted.

4.8.2 Structure Definition

Table 8 — DLME_ARGUMENTS

Field	Byte Length	Description
DlmeArgsLength	4	The length of the DlmeArgs.
DlmeArgs	variable	The DLME specific argument buffer.

```
typedef structure _DLME_ARGUMENTS {
   UINT32 DlmeArgsLength;
   UINT8 DlmeArgs[0];
} DLME_ARGUMENTS, *PDLME_ARGUMENTS;
```

4.8.3 Structure Field Definitions

4.8.3.1 DlmeArgsLength

The size in bytes of the DLME arguments buffer. The DLME should perform a consistency check on any data passed.

4.8.3.2 DlmeArgs

The virtual address of a buffer constructed by the Gap to provide DLME-specific arguments.

4.9 DLME_TRANSLATION_LIST Structure

4.9.1 Description

The DLME is launched by the DCE with virtual memory (paging) enabled and as a result cannot directly access or validate the underlying page tables. The DLME_TRANSLATION_LIST is a list of virtual addresses that will be translated to physical addresses by the DCE to provide the DLME with a set of trusted address mappings. These mappings may be used to determine the virtual address to use to access an underlying physical resource—for example, the pages containing the page table itself, or a memory mapped I/O device. For efficiency, the list SHOULD be sorted in order by virtual address.

4.9.2 Structure Definition

Table 9 — DLME_TRANSLATION

Field	Byte Length	Description
VirtualAddress	8	The virtual address to be translated.
PhysicalAddress	8	The translated physical address.

```
typedef structure _DLME_TRANSLATION {
   UINT64 VirtualAddress;
   UINT64 PhysicalAddress;
} DLME_TRANSLATION, *PDLME_TRANSLATION;

typedef structure _DLME_TRANSLATION_LIST {
   UINT32 NumEntries;
   DLME_TRANSLATION List[0];
} DLME_TRANSLATION_LIST, *PDLME_TRANSLATION_LIST;
```

4.9.3 Structure Field Definitions

4.9.3.1 NumEntries

The number of DLME_TRANSLATION entries in the list.

4.9.3.2 Virtual Address

The virtual address to be translated by the DCE.

4.9.3.3 Physical Address

The translated physical address that corresponds to the virtual address. The value upon call to DL_Entry is unused by the DCE. The value upon call to the DlmeMain is a trusted translation.

5 System States, Transitions, and Interfaces

5.1 Introduction

This section contains the system state descriptions, transitions, and interfaces. A high-level summary of system states with transitions of interest numbered is in Figure 2. The interfaces are defined at the following transition points as numbered in the figure:

1) The Pre-Gap to the Gap
2) The Gap to the DCE Preamble
3) The DCE to the DLME
4) The DLME to DLME_Exit
5) The Dynamic Operating System Environment to S3
6) The Dynamic Operating System Environment to S5
7) The Dynamic Operating System Environment to the S-RTM OS Present Environment

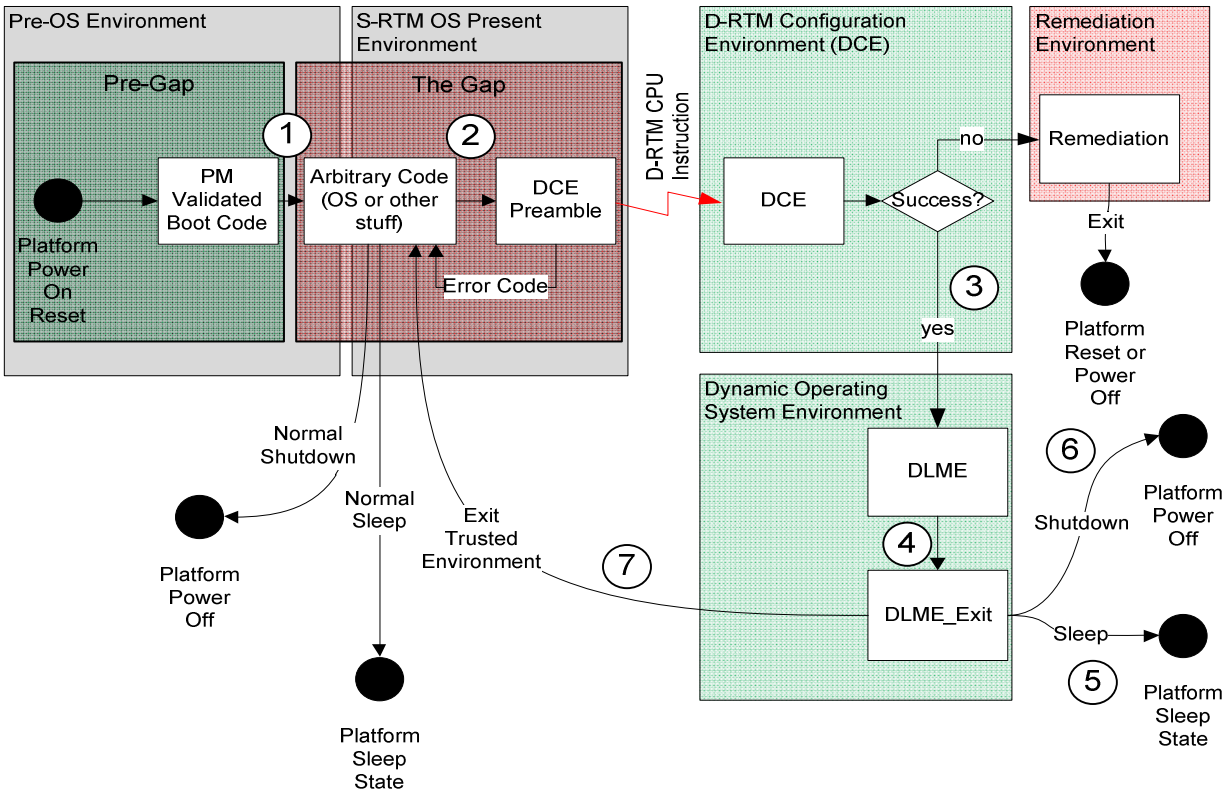

Figure 2 — System states and transitions

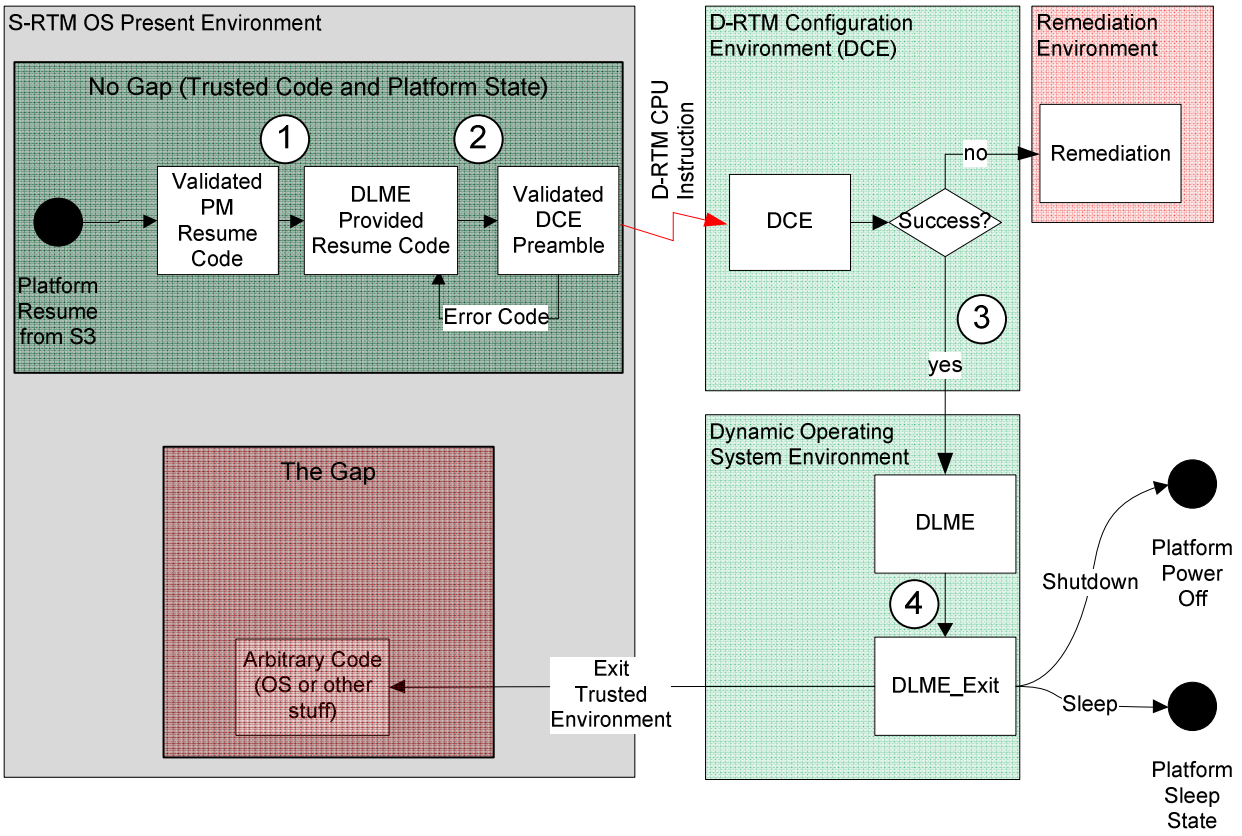

Figure 3 — Resume from sleep without Gap code on S3 resume

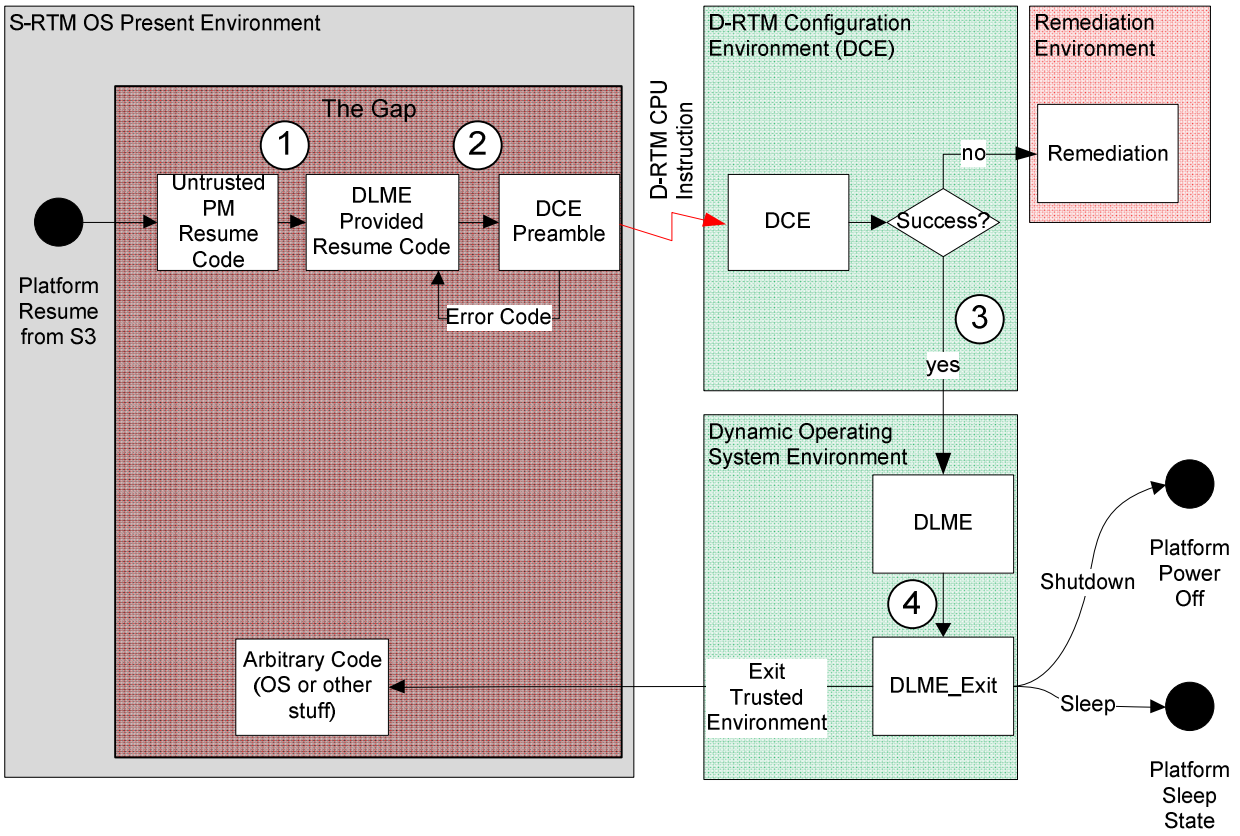

Figure 4 — Resume from sleep Gap code on S3 resume

5.2 System States

5.2.1 Pre-Gap

The Pre-Gap state definition is that only code validated by the platform manufacturer and measured in S-RTM PCR measurements has run. This state ends when any code not validated by the platform manufacturer is run. Examples of code not validated by the platform manufacturer are:

- option ROM initialization code (potentially excluding manufacturer-controlled and -validated embedded-option ROMs);
- code provided by the platform manufacturer whose security properties have not been evaluated;
- code evaluated by the platform manufacturer but whose measurements have not been recorded in the S-RTM PCR, excluding the S-CRTM.

The following examples show some different ways for a platform manufacturer to validate code.

- Validate the platform manufacturer's code on the platform in the factory and prevent updates to the code in the field.
- Validate the initial platform manufacturer's code on the platform in the factory and permit updates through a manufacturer-controlled update process. The manufacturer maintains control of the update process and only releases updates that result in code validated by the platform manufacturer code being installed on the platform.
- Validate the initial platform manufacturer's code on the platform in the factory. Have a core set of platform manufacturer code on the platform that meets the previous two conditions. The core set of platform manufacturer code validates subsequent components that run per the platform manufacturer's requirements. If the subsequent components do not validate per the platform manufacturers requirements, the components are treated as Gap code by the implementation. The subsequent components may be updated outside of the control the platform manufacturer.

In some D-RTM implementations, the Pre-Gap state may be used by the platform manufacturer to perform actions that the D-RTM process inherently trusts. An example is loading SMM code, measuring SMM code into an S-RTM PCR and locking SMRAM. The DCE may inherently trust that the Pre-Gap code performed these actions and may use the SMM code measurement to validate that the correct version of SMM code is running on the platform.

When a D-RTM implementation has dependencies on actions performed in the Pre-Gap state, it has a larger TCB for the D-RTM process than a platform implementation that does not have a Pre-Gap dependency. A smaller D-RTM TCB that does not depend on actions in the Pre-Gap state is preferable.

5.2.2 Pre-OS Environment

The Pre-OS Environment is the same Pre-OS Environment defined in the PC Client Conventional BIOS specification.

5.2.3 The Gap

The D-RTM process changes the platform from a potentially undefined software and hardware state to a state with a known hardware configuration and executing measured code. For convenience, the undefined software and hardware state prior to the D-RTM process is referred to as "the Gap". (See 2.9.1)

5.2.4 S-RTM OS Present Environment

The S-RTM OS Present Environment is the same as defined in the PC Client Conventional BIOS specification (1).

5.2.5 D-RTM Configuration Environment (DCE)

The D-RTM Configuration Environment (DCE) is manufacturer-implementation specific.

5.2.6 Remediation Environment

The Remediation Environment is manufacturer-specific and permits the platform manufacturer to optionally convey some information to the user regarding why the DCE process failed.

The platform manufacturer's implementation of the Remediation Environment SHALL only permit exiting by performing a platform reset or turning off the power.

5.2.7 Dynamic Operating System Environment

The Dynamic Operating System Environment is the environment the DLME executes in. Some system hardware may not be available unless the DLME turns it on.

5.3 System State Transitions, Requirements, and Interfaces

5.3.1 Pre-Gap Transition to the Gap

The transition from the Pre-Gap state to the Gap state is implementation-specific. It is the first invocation of non-platform manufacturer-validated code.

5.3.2 The Pre-OS Environment to the S-RTM OS Present Environment

The transition from the Pre-OS Environment to the S-RTM OS Present Environment is implementation-specific.

5.3.3 The Gap to the DCE Preamble

The Gap code initiates the D-RTM process by calling the DCE preamble through the DL_Entry32 or DL_Entry64 entry points. On a successful D-RTM operation, the DL_Entry call will not return. The DCE preamble code may be able to detect some error conditions and cause DL_Entry to return an error. Other errors that occur during DCE processing MUST NOT cause a return to the caller and WILL result in platform-specific remediation.

5.3.3.1 Calling Sequence

5.3.3.1.1 Calling Conventions

The call to the DL_Entry functions uses the 'Cdecl'[1] calling conventions.

5.3.3.1.2 DL_Entry32 Signature

```
UINT32
DL_Entry32(PDLME_DESCRIPTOR DlmeDescriptor, PDLME_ARGUMENTS DlmeArgs,
      PDLME_TRANSLATION_LIST TranslationList, PUINT_PTR
      StackBuffer_Base);
```

[1] For 32-bit, function parameters are pushed on the stack in a right-to-left order. For 64-bit, the first 4 parameters (left-to-right order) are placed in registers RCX, RDX, R8, and R9 and stack space is reserved for the 4 parameters. Additional parameters are pushed onto the stack left-to-right.

5.3.3.1.3 DL_Entry64 Signature

```
UINT32
DL_Entry64(PDLME_DESCRIPTOR DlmeDescriptor, PDLME_ARGUMENTS DlmeArgs,
      PDLME_TRANSLATION_LIST TranslationList, PUINT_PTR
      StackBuffer_Base);
```

5.3.3.1.4 DL_Entry32 and DL_Entry64 Parameters

The return from DL_Entry32 is a 32-bit integer and the function returns only if the D-RTM process is not invoked.

DlmeDescriptor – A pointer to the DLME descriptor structure in virtual memory. The descriptor provides the digest, start, length, attributes, stack size, entry point, and signature for the DLME.

DlmeArgs – A pointer to a DLME_ARGUMENTS structure that contains a length and a buffer containing arguments for the DLME from the Gap. The structure and contents of the buffer are defined by the DLME and are treated as opaque by the DCE preamble and the DCE.

TranslationList – A pointer to a DLME_TRANSLATION_LIST structure containing its size and an array of virtual and physical address pairs for the DCE to translate.

StackBuffer_Base – A pointer to the starting virtual address of a buffer to be used for the DLME's stack. The length of the buffer is contained in the DlmeDescriptor. The contents of the buffer MAY NOT be preserved by the Dl_Entry call. Details of the calling convention and stack usage are in Section 5.3.4 below.

5.3.3.1.5 Error codes returned by DL_Entry

Table 10 — DL_Entry Error Codes

Code	Value	Description
DRTM_STATUS_FOO	42	The Dl_Entry failed to foo.

5.3.3.2 Processor State

The call to the DCE preamble SHALL be made on the Boot Strap Processor (BSP). The DCE preamble shall validate that the call is made on the BSP and return an error if not.

All Application Processors (AP) SHOULD be halted. If they are not, then the system behavior is dependent on the implementation of the DCE preamble and the DCE. The state of threads that may have been running on an AP MAY NOT be preserved. Depending on the implementation, it is likely any work being performed by APs when DL_Entry is called will be abruptly stopped. It is also possible the call will fail if all APs are not halted. The DCE preamble will perform the following steps depending on the processor state.

- A DCE preamble may check that this condition is met and return an error.
- A DCE preamble that does not check may force the condition to be met before starting the DCE.
- A DCE preamble may assume that the condition is met and proceed to the DCE. The DCE may then automatically enforce the condition.
- The DCE preamble may assume the condition is met and proceed to the DCE. If all APs are not halted, the DCE shall then fail and enter remediation.

The processor should be executing in ring 0. In some implementations, DCE entry is only possible from ring 0 and the DCE startup will fail. The platform provided DCE SHALL check the privilege state of the caller and return with an error code if it is not in ring 0.

Hardware-assisted CPU virtualization should be enabled. The DCE preamble may return an error if this condition is not met. On certain processor architectures, the DL Event will fail if CPU virtualization is not enabled. The way in which the DL Event fails is processor-architecture dependent and reporting of a failure back to the caller of the DCE preamble is not always possible.

Hardware-assisted CPU virtualization should not be active on any processor. The DCE preamble may return an error if these conditions are not met. On certain processor architectures, the DL Event will fail if the processor is executing with CPU virtualization active. The way in which the DL Event fails is processor-architecture dependent and reporting of a failure back to the caller of the DCE preamble is not always possible.

If the call is made to DL_Entry32, the processor SHALL be in 32-bit mode. The DCE preamble for DL_Entry32 will be written expecting the processor to be in this mode. If DL_Entry32 is called with the processor not in 32-bit mode, the behavior is undefined.

If the call is made to DL_Entry64, the processor SHALL be in 64-bit mode. The DCE preamble for DL_Entry64 will be written expecting the processor to be in this mode. If DL_Entry64 is called with the processor not in 64-bit mode, the behavior is undefined.

5.3.3.3 CPU Microcode Patches

The DCE preamble MAY install CPU microcode patches depending on processor specific security requirements.

5.3.3.4 DMA

It is advised that all DMA activity controlled by the caller be stopped before the call to DL_Entry. The DCE process will prohibit all DMA as part of its execution. Unexpectedly blocking DMA may have undesired side effects including the possibility that status and notifications may be lost.

The DCE preamble is not required to check that this condition is met.

If the DCE preamble is called before the operating system kernel has taken control of all hardware, some DMA may be active due to platform-provided emulation (e.g., USB keyboard and mouse). The PM-provided DCE preamble is responsible for ensuring an orderly shutdown of this DMA.

> NOTE To maintain memory, DMA, including emulation-related DMA, will be disabled when control is passed to the DLME.

5.3.3.5 Memory

Memory virtualization SHALL be enabled.

The memory map in place at the time of the call to DL_Entry32 or DL_Entry64 SHALL be a map that is consistent with the corresponding processor address mode.

The caller of DL_Entry SHALL ensure:

- The memory range starting at the DL_Entry_Base to DL_Entry_Base + DL_Entry_Length is in identity-mapped pages when it is called.
- The page tables in use when DL_Entry is called map the entire DLME_DESCRIPTOR structure pointed to by DlmeDescriptor. (DlmeDescriptor to DlmeDescriptor + sizeof(DLME_DESCRIPTOR)) The DLME_DESCRIPTOR structure does not need to be identity mapped.
- The virtual memory address range of the DLME_DESCRIPTOR structure pointed to by DlmeDescriptor is between 0 and 4 gigabytes (GB).

- The page tables in use when DL_Entry is called map the entire DLME. (DlmeDescriptor->DlmeStart to DlmeDescriptor->DlmeStart + DlmeDescriptor->DlmeLength) The DLME does not need to be identity mapped.
- The DLME is placed in virtual memory so that the virtual address for DlmeDescriptor->DlmeStart equals the actual virtual address of the DLME.
- No two virtual pages within the DlmeDescriptor, DLME, or stack buffer, may map to the same physical page.
- The virtual memory address range of the DLME is between 0 and 4 gigabytes.
- The page tables in use when DL_Entry is called map the DLME Stack Buffer. (The memory range starting at StackBuffer_Base to StackBuffer_Base + DlmeDescriptor->StackSize.) The stack buffer does not need to be identity mapped.
- The page tables in use when DL_Entry is called map the entire TranslationList data structure. The TranslationList data structure does not need to be identity mapped.

The caller of DL_Entry SHOULD ensure that the virtual and physical addresses used by DL_Entry, the DLME_DESCRIPTOR structure pointed to by DlmeDescriptor, the DLME, and the DLME Stack Buffer do not overlap. (If they do overlap, the caller should choose a different location for the DLME or the stack buffer.)

The caller of DL_Entry MAY add additional page tables to the memory map (e.g., for the DlmeArgs buffer).

5.3.3.6 DLME

Requirements:

1) The physical pages that contain the DLME SHALL be in conventional system memory.
2) The DLME SHOULD have a DlmeSignature in its DlmeDescriptor.

If a DlmeSignature is not present in the DlmeDescriptor, an error measurement will be recorded in the PCR.DLME.Authority (see 4.6 above).

If a DlmeSignature is present in the DlmeDescriptor but the signature verification fails, an error measurement will be recorded in the PCR.DLME.Authority.

5.3.3.7 Translation List

The DLME is launched into an environment where virtual memory (paging) is enabled which, due to processor architecture limitations, makes it very difficult to determine the virtual-to-physical mapping of any memory address. To assist the DLME, the caller of DL_Entry MAY provide the DCE with a list of entries that contain virtual addresses and the DCE SHALL use the DLME's page table to determine the physical addresses to which they are mapped. If the virtual address is NOT mapped by the DLME's page table, the physical addresses SHALL be set to 0xFFFFFFFFFFFFFFFF. The value of the physical address in an entry is undefined when DL_Entry is called.

One expected use case for the translation list is for the Gap to provide the DLME with the set of virtual addresses that map the pages containing the page table. The DCE will provide the actual physical addresses that those addresses map to. The DLME may then determine the physical address of the root of the page table by examining the CR3 register, and then use the translation list to determine its virtual address. The DLME may then recursively determine the physical address of each page directory entry, and then use the translation list to determine its virtual address. As a result, the DLME will have access to the full page table, allowing it to verify or extend any memory mapping.

5.3.3.8 TPM State

Requirements:

1) The TPM does not need to be owned.
2) The TPM SHOULD be activated and enabled.
3) The TPM SHOULD be at locality free.
4) The TPM SHOULD NOT be executing a command.

The DLME MAY be launched if the TPM is deactivated, disabled, or not enumerated. The DLME and platform state SHOULD be as close as possible to the same state as if the TPM was activated and enabled. The notable exception is that no measurements are recorded. This is similar to the startup process with S-RTM when the TPM is deactivated.

If the TPM is not at locality free, the DCE preamble may return an error. (Some DCE preamble implementations may change the TPM locality to free but this functionality is not required. Some implementations may enter remediation or invoke the DCE without resetting the PCR.) Refer to the TPM Interface Specification (TIS) 1.3 section 4.2.1. Per section .2.1 of the TIS, the HASH_START command requires the TPM locality to be free to successfully reset the PCR.

If the TPM is executing a command, the DCE preamble may return an error. (Some DCE preamble implementations may cancel a pending TPM command. Some implementations may enter remediation or invoke the DCE without resetting the PCR.) Refer to the TIS section 4.2.1. Per section 4.2.1 of the TIS, the TPM_HASH_START command will conditionally reset the D-RTM PCR. The DCE preamble may conservatively check all the necessary conditions are met to reset the D-RTM PCR, or may optimistically invoke the D-RTM instruction without checking, sometimes resulting in the D-RTM PCR not being reset by the TPM_HASH_START command.

5.3.3.9 Exit Boot Services

The firmware UEFI ExitBootServices interface (or equivalent) SHOULD be called before DL_Entry. Upon entering the DCE, the firmware boot services will no longer be available. Failure to properly exit boot services may result in an inconsistent software state or lost access to information such as the memory map usage. Other firmware-controlled device, platform state, or capabilities may be placed into an inconsistent state.

Some DCE implementations MAY NOT be able to complete their operations and enter remediation if boot services are not exited. As an example, the determination of the Sensitive Resources List may not be possible.

The implementation of the DCE preamble MAY exit boot services. Any resources required by the implementation to allow the preamble to exit boot services SHALL NOT require cooperation of any Gap code.

5.3.3.10 Previously Invoking D-RTM

If a prior invocation of D-RTM occurred since power-on reset, the most recent invocation should have been followed by a call to DLME_Exit before DL_Entry may be invoked again.

If a prior invocation of D-RTM occurred since power-on reset and it was not followed by a call to DLME_Exit, DL_Entry SHALL return an error code.

5.3.4 The DCE to the DLME

5.3.4.1 DLME Calling Convention

The calling convention used for DL_Entry32, DL_Entry64, and the DLME entry point will use the common C calling convention for the specific platform and architecture(6). In general, the processor's architectural state entering the DLME will be the same as existed when DL_Entry was called. Exceptions will be specified as necessary. The DL_Entry addresses specified in the D-RTM ACPI Resource Table are absolute physical addresses and must be identity-mapped to the same virtual address.

5.3.4.1.1 DlmeMain Signature

```
VOID
DlmeMain(PDLME_DESCRIPTOR DlmeDescriptor, PDLME_ARGUMENTS DlmeArgs,
      PDLME_TRANSLATION_LIST TranslationList, PDRTM_ACPI_TABLES
      DrtmAcpiTables);
```

5.3.4.1.2 DlmeMain Parameters

DlmeDescriptor – A pointer to the DLME descriptor structure in virtual memory. The descriptor provides the digest, start, length, attributes, stack size, entry point, and signature for the DLME. It MAY be the same DlmeDescriptor pointer passed to DL_Entry32 or DL_Entry64.

DlmeArgs – A pointer to a DLME_ARGUMENTS structure that contains a length and a buffer containing arguments for the DLME from the Gap. The structure and contents of the buffer are defined by the DLME and are treated as opaque by the DCE preamble and the DCE. The pointers MAY be the same as the values passed to DL_Entry32 or DL_Entry64. The contents of the buffer SHOULD be the same as the contents passed to DL_Entry32 or DL_Entry64.

TranslationList – A pointer to a DLME_TRANSLATION_LIST structure containing its size and an array of virtual and physical address pairs for the DCE to translate.

DrtmAcpiTables – A pointer to a structure containing pointers to the D-RTM and RSDT ACPI tables.

5.3.4.2 Processor State

5.3.4.2.1 x86

The widely used x86 C calling convention (cdecl), as documented in (7), and as summarized here, will be used for D-RTM. The four arguments are passed on the stack pushed in a right-to-left order. All integers are passed in as 4-byte values. All pointers are passed in as 4-byte values. The stack will be 4-byte aligned.

GDT/LDT/TSS

The GDT, LDT, and TSS at the DLME entry point will be the same as at the DL_Entry32 call.

Selectors

The selectors CS, DS, ES, SS, FS, and GS at the DLME entry point will be the same as at the DL_Entry32 call. The CS, DS, and SS selectors will reference a flat memory model descriptor with a base of zero and a limit of 4 GB.

IDT

The IDT at the DLME entry point will be the same as at the DL_Entry32 call. Interrupts will be disabled when the DLME is called.

Floating Point Registers

All floating point registers at the DLME entry point will be the same as at the DL_Entry32 call.

Integer Registers

Table 11 — x86 Integer Registers

EAX	Return Status
EBX	Preserved
ECX	Undefined
EDX	Undefined
ESI	Preserved
EDI	Preserved
EBP	Preserved On Return
ESP	Stack Pointer
EIP	Instruction Pointer

Registers EAX, ECX, and EDX are not preserved on return.

Control Registers

Table 12: x86 Control Registers

EFLAGS	Undefined
CR0	Preserved
CR2	Undefined
CR3	Preserved
CR4	Preserved
DR0-7	Undefined

Registers CR0, CR4 are generally preserved as required to maintain operating modes and state of the processor, but individual bits may be changed as required by D-RTM.

Stack

The x86 architecture defines that the stack grows downward. D-RTM is only concerned with the contents of the stack when DL_Entry32 is called and the DLME entry point is called, the state of the stack at other times is unspecified.

The value of ESP must be four byte aligned.

The caller pushes arguments on the stack right to left.

DL_Entry32 Entry Point

Table 13: DL_Entry32 Stack Frame

>ESP+14h	Preserved

ESP+10h	Argument 4
ESP+0Ch	Argument 3
ESP+08h	Argument 2
ESP+04h	Argument 1
ESP	Return Address
<ESP-04h	Undefined

DL_Entry32 is entered using a near call. DL_Entry32 must not modify the stack contents at ESP+14h or above. If DL_Entry32 returns, it will use a near return with the “RET” instruction (no operand), EAX must contain the preamble error status code, and ESP must be the value at entry.

DLME Entry Point

Table 14: x86 DLME Entry Stack Frame

>ESP+14h	Undefined
ESP+10h	Argument 4
ESP+0Ch	Argument 3
ESP+08h	Argument 2
ESP+04h	Argument 1
ESP	Undefined Return Address
<ESP-04h	Undefined

The DLME must not return to the caller and the Return Address is an undefined value and has undefined behavior if executed.

5.3.4.2.2 x64

The widely used x64 C calling convention as documented in (8), and as summarized here will be used for D-RTM. The 4 arguments are passed in registers with sufficient stack space reserved to allow the callee to store the values. All integers are passed in as 8 byte values. All pointers are passed in as 8 byte values. The stack will be 8 byte aligned.

GDT/LDT/TSS

The GDT, LDT, and TSS at the DLME entry point will be the same as at the DL_Entry64 call.

Selectors

The selectors CS, DS, ES, SS, FS, and GS at the DLME entry point will be the same as at the DL_Entry64 call. Architecturally CS, DS, ES, and SS are treated as if having a base of zero and no limit checks.

IDT

The IDT at the DLME entry point will be the same as at the DL_Entry64 call. Interrupts will be disabled when the DLME is called.

Floating Point Registers

All floating point registers at the DLME entry point will be the same as at the DL_Entry64 call.

Integer Registers

Table 15 — x64 Integer Registers

RAX	Return Status
RBX	Preserved
RCX	Argument 1
RDX	Argument 2
RSI	Preserved
RDI	Preserved
R8	Argument 3
R9	Argument 4
R10	Undefined
R11	Undefined
R12	Preserved
R13	Preserved
R14	Preserved
R15	Preserved
RBP	Preserved on return
RSP	Stack Pointer
RIP	Instruction Pointer

Arguments RCX, RDX, R8, and R9 are not preserved on return.

Registers RAX, R10, and R11 are not preserved on return.

Control Registers

Table 16 — x64 Control Registers

RFLAGS	Undefined
CR0	Preserved*
CR2	Undefined
CR3	Preserved
CR4	Preserved*
CR8	Preserved
DR0-7	Undefined

Registers CR0 and CR4 are generally preserved as required to maintain operating modes and state of the processor, but individual bits may be changed as required by D-RTM.

Stack

The x64 architecture defines that the stack grows downward. D-RTM is concerned only with the contents of the stack when DL_Entry64 is called and the DLME entry point is called. The state of the stack at other times is unspecified.

The value of RSP must be 8-byte aligned.

The caller reserves space for four 8-byte values that are typically used by the callee to back the four argument values.

DL_Entry64 Entry Point

Table 17 — DL_Entry64 Stack Frame

>RSP+28h	Preserved
RSP+20h	Uninitialized (callee may use for R9)
RSP+18h	Uninitialized (callee may use for R8)
RSP+10h	Uninitialized (callee may use for RDX)
RSP+08h	Uninitialized (callee may use for RCX)
RSP	Return Address
<RSP-08h	Undefined

DL_Entry64 is entered using a near call. DL_Entry64 must not modify the stack contents at RSP+28h or above. If DL_Entry64 returns, it will use a near return with the “RET” instruction (no operand), RAX must contain the preamble error status code, and RSP must be the value at entry.

DLME Entry Point

Table 18 — x64 DLME Entry Stack Frame

>RSP+28h	Undefined
RSP+20h	Uninitialized (callee may use for R9)
RSP+18h	Uninitialized (callee may use for R8)
RSP+10h	Uninitialized (callee may use for RDX)
RSP+08h	Uninitialized (callee may use for RCX)
RSP	Undefined Return Address
<RSP-08h	Undefined

The DLME must not return to the caller. The Return Address is an undefined value and has undefined behavior if executed.

5.3.4.3 DLME Stack

The DLME Stack memory range SHALL be the virtual memory range between the StackBuffer_Base passed to DL_Entry32 or DL_Entry64 and StackBuffer_Base + DlmeDescriptor->StackSize.

The processor stack pointer SHALL be an address within the DLME Stack memory range.

The DCE will have set up the current stack frame following the architecture-specific 'Cdecl' parameter passing convention corresponding to the processor mode the DLME is called in.

A "return" from the current stack frame has undefined behavior. Only the current stack frame within the stack buffer has a trusted value, because it was constructed by the DCE.

5.3.4.4 CPU Microcode Patches

All Host Platforms' CPUs SHOULD have the same CPU microcode patches installed. If CPUs have different microcode patches installed, it may not be possible to ensure the trustworthiness of the system.

5.3.4.5 Memory

The page tables passed to DL_Entry SHOULD be preserved when the DLME is called. However, the DCE preamble or the DCE MAY add additional entries to the page table. (For example, the DCE preamble may need to perform additional mappings (likely 1:1) in order to set up the platform-specific D-RTM event, or the DCE may want to map itself into the DLME's address space.)

The page table SHALL be 32-bit if DL_Entry32 was called or SHALL be 64-bit if DL_Entry64 was called.

5.3.4.6 Memory Aliasing

The integrity of the DLME is determined by the measurement hash over its address range and signature check based on the values specified in the *DlmeDescriptor*. This measurement is performed using the virtual address range of the DLME providing an implicit verification of the page tables. Any change to a page table entry will cause a different page to be measured resulting in a different measurement. However, it is possible that two memory pages in the DLME may have identical contents (e.g., all zero) and as a result any substitution will not be detectable. A specific threat that must be handled is the mapping of two identical virtual pages to the same underlying physical page, or memory aliasing. Such a memory alias mapping cannot be detected through measurement and could result in the modification of one virtual address resulting in the modification of another virtual address causing unexpected behavior of the DLME. The threat of memory aliasing also extends to the DLME Stack and other parameters.

A DCE SHALL detect DLME memory aliasing and SHALL extend an error value into PCR.Details and PCR.DLME.Authority.

The problem of memory aliasing can be resolved through a straightforward scan by the DCE of the range of page table entries that map the DLME and stack to determine if any physical page has more than one mapping within the DLME's virtual address range. The scan of the page table to detect memory aliasing may be a slow quadratic speed operation.

A DCE MAY perform DLME memory alias detection through scanning the page table entries for the DLME virtual address range, DLME Stack, and DLME parameters.

To enable a more efficient alias detection mechanism, the DLME may be constructed in a manner that causes each page to have a unique value. The uniqueness is generated by performing an exclusive "OR" of the address of each page with 8 bytes on that page before the DLME signature is calculated. During the measurement of the DLME by the DCE, each page is first measured and then its virtual address is exclusive "OR"ed, returning the DLME to its original state. If the page table contains an aliased page, one of the pages will not contain the correct exclusive OR of its address, resulting in an incorrect measurement. The 8 bytes used to exclusive "OR" the virtual address of the page SHALL start at an offset of 16 bytes on each page to prevent "magic numbers" or other special values from being changed.

If the DLME_XOR_ALIAS_DETECTION bit is set in the DlmeDescriptor.TCGDefinedAttributes flags, the DCE SHALL perform an exclusive OR of the address of each page after measuring the DLME.

5.3.4.6.1 Exclusive OR Memory Alias Detection

The following steps SHALL be performed for generating a DLME that uses exclusive OR memory alias detection:

1) The operating system provider constructs a DLME.
2) The *DlmeStart* and *DlmeLength* are entered in the *DlmeDescriptor*.
3) For each page in the DLME, exclusive "OR" its address:

```
for (page = DlmeStart; page < DlmeStart + DlmeLength; page += 4096)
      *(page + 0x10) ^= page;
```

4) Set the DlmeDescriptor.TCGDefinedAttributes.DlmeXorAliasDetection to 1.
5) Generate the DlmeDigest and DlmeSignature from DlmeStart for DlmeLength bytes.

The DCE when passed a DLME with a DlmeDescriptor.TCGDefinedAttributes.DlmeXorAliasDetection set, SHALL perform the following steps:

1) Measure the DLME from DlmeStart for DlmeLength bytes to calculate the digest.
2) For each page in the DLME, exclusive "OR" its address:

```
for (page = DlmeStart; page < DlmeStart + DlmeLength; page += 4096)
      *(page + 0x10) ^= page;
```

The previous steps SHALL also be performed for the DLME Stack.

5.3.4.7 Processor Startup

The DCE transfers control to the DLME with only the boot processor operating. The DLME is responsible for starting any additional processors. The mechanism to start additional processors is platform-specific and MAY be different than non-D-RTM processor startup. The DLME is responsible for any platform mechanism that allows the "hot" addition of additional processors.

5.3.4.8 D-RTM Resources Table

The information in the D-RTM Resources Table must be validated during the DCE process. This may involve the DCE modifying the contents of the D-RTM Resources Table values or other ACPI tables during the DCE process. If the DCE is unable to validate the contents of the D-RTM Resources Table, it must enter remediation. At the time the DCE invokes the DLME, the D-RTM Resources Table contents must be valid.

The address of the D-RTM Resources Table is passed as a pointer to the DLME. The address of the D-RTM Resources Table MAY be the same address as the address of the D-RTM table provided when the platform firmware measured the EV_SEPARATOR event into PCR 0 through 7. If the RSDT table is a member of the DCE-validated ACPI tables, the D-RTM Resources Table pointer entry in the RSDT table SHALL equal the same address as the D-RTM Resources Table address passed to the DLME by the DCE.

5.3.4.9 RSDT Table

The physical address of the RSDT ACPI Table is passed as a pointer to the DLME.

5.3.4.10 Platform State for DLME from the DCE

#	Platform State for DLME from the DCE	Notes

#	Platform State for DLME from the DCE	Notes
1. 5.3	The DLME SHALL receive control with access to Locality 2.	This allows it to extend PCR 17-22 and reset PCR 20-22.
2. 5.5	When the DLME is called, the entry point used SHALL be the one designated in the DLME descriptor passed to DL_Entry.	
3. 5.3	The DLME SHALL receive control with DMA remapping turned on with no access to any portion of memory.	This could be conveniently done using a null map.
4. 5.3	All interrupts except SMI SHALL be disabled when the DLME receives control. SMI MAY be enabled.	
5. 5.3	The DLME SHALL receive execution control of the BSP processor. All other processors SHALL be in a controlled state.	.
6. 5.3	The BSP processor must be executing in ring 0.	
7. 5.3	The verified microcode patches must be installed.	Some hardware implementations may remove microcode patches during the DCE. These patches must be restored.
8. 5.3 & 7	The platform must provide a PROTECT_MEMORY_ON_S3_RESUME bit whose default value is False. The bit should be protected from clearing or listed in the list of security-sensitive memory addresses. The DCE must set the bit to True before handing control to the DLME.	For backward compatibility, DMA protections are not put in place on S3 resume by default. This bit is used only to enforce DMA protections on S3 resume after a D-RTM event.
9. 9	The DCE must provide the DLME a log of what extends have occurred in PCR.Details and PCR.Authorities.	

5.3.4.11 ACPI Related

#	ACPI Related	Notes
1. 3.2	The DLME SHALL be provided a list of memory addresses that are security-sensitive in nature and need to be protected to ensure the integrity of the DLME. The list should include these classes of addresses, but it does not need to classify them based on their security properties: • Addresses the OS should not touch. • Addresses PM provided AML exclusively should touch. • Addresses of architecture elements only the OS or PM provided AML exclusively should touch. The list may include items in any address space (system memory, system I/O, PCI configuration space, etc.).	

#	ACPI Related	Notes
2. 3.2	The DLME SHALL be provided a list of all the ACPI tables containing AML code that the PM trusts to access security-sensitive memory addresses. Tables for which the LoadTable operation will be used only by the PM-provided AML code MAY also be included in this set.	This list may be empty for many client systems. It is likely it will have entries for server platforms.
3. 3.2	The DCE SHALL validate the ACPI tables containing AML code that the PM trusts to access security-sensitive memory addresses and tables for which the LoadTable operation will be used.	The DCE must validate information listed in the previous requirements.
4. 3.2	The DLME MAY be provided a list of all the ACPI tables for which the LoadTable operation may be used by the AML code the PM trusts to access security-sensitive memory addresses. Any AML code present in the tables SHALL be trusted by the PM to access security-sensitive memory addresses.	It is expected the DLME ACPI interpreter would give precedence to loading tables in this set versus other tables that may have the same name but are not listed in the set.
5. 3.2	The DCE SHALL validate the ACPI tables for which the LoadTable operation may be used by the AML code the PM trusts to access security-sensitive memory addresses.	The DCE must validate information listed in the previous requirements.
6. 3.2	Any PM-provided AML code the PM trusts to access security-sensitive memory addresses SHALL validate any additional ACPI tables (including AML code) it creates through the use of the Load operation.	This permits the “privileged” PM-provided AML code to safely add more “privileged” code to itself through the use of Load.
7. 3.2	The DCE SHALL validate certain ACPI table entries and any associated AML code used to implement them: • The mechanism to switch to ACPI mode. • The number of processors on the platform that are operational, or can be made operational, at the time the DLME is called. • All entries in the D-RTM Resources Table.	The DLME has dependencies on specific ACPI entries and must be able to trust that certain functionality provided by the platform can be accessed as part of a Trusted Computing Base.
8. 3.2 & 5.3	The DLME SHALL NOT own the ACPI Global Lock.	If the OS had the Global Lock before performing a D-RTM event, it should be freed by the DCE process.
9. 3.2	If Gap code can run on resuming from S3, a flag SHALL be set in the D-RTM Resources Table to notify the DLME.	

5.3.5 The DLME to DLME_Exit

The DLME SHALL call DLME_Exit when exiting its trusted environment or when entering an S3 or S4 power state.

5.3.5.1.1 Calling Conventions

The call to the DLME_Exit functions uses the “Cdecl” calling conventions. The mode of the processor upon entry is unspecified.

5.3.5.1.2 DLME_Exit Signature

```
VOID
DLME_Exit(UINT32 SystemPowerState);
```

5.3.5.1.3 DLME_Exit Parameters

The return from DLME_Exit is a void.

SystemPowerState – An integer corresponding to the system ACPI power “S” state that the software will be transitioned to.

5.3.5.2 DLME_Exit Call Preconditions

DLME_Exit SHALL NOT be called unless a previous successful DL_Entry call has been performed and a DLME successfully launched.

If a previous DLME_Exit has been performed, a subsequent call to DLME_Exit MAY NOT be made without a successful DL_Entry call.

The DLME_Exit SHOULD return without changing platform state if no DL_Entry has been performed.

The DLME_Exit MAY fail or enter platform-specific remediation if no DL_Entry has been performed.

5.3.5.3 Power State Transition

The system power state parameter is a mechanism to inform the platform that the software will be transitioning to a lower power state (e.g., S3, S4, or S5). The platform MAY perform platform-required specific operations to enable successful transition to a different power state. Failure to call DLME_Exit, or calling DLME_Exit with an incorrect system power state value MAY result in failure of the change in power state, failure to resume from the power state, or failure to return to a trusted state.

The DLME_Exit interface does not perform the actual system power state transition. Standard ACPI interfaces SHALL be used to initiate a power state transition.

5.3.5.4 Gapless S3

If the platform supports Gapless S3 transitions, the DLME_Exit operation MAY validate the DCE preamble code to ensure proper operation.

5.3.5.5 Continued Non-Trusted Operation

DLME_Exit SHALL be called to exit from the trusted environment even if the platform is maintaining full power state (e.g., S0). The platform MAY perform platform-specific operations that re-enable functionality previously restricted by the trusted environment. Failure to call DLME_Exit MAY result in less than full functionality. For example, DLME_Exit may re-enable certain DMA functionality that was restricted by the DCE.

5.3.5.6 Non-Host Platform

The DLME_Exit MAY perform platform-specific operations in order to correctly control Non-Host Platforms, including turning back on NHPs halted by the DCE. Specific operations may differ depending on the indicated system power state transition. See Section 8.12.

5.3.6 The S-RTM OS Present Environment to S3

The transition from an S-RTM OS Present Environment to the S3 system power state is defined in the TPM Client or Server specification.

5.3.7 S3 to the S-RTM OS Present Environment

The platform may resume from S3 sleep but may not be able to protect the platform before the dynamic operating system that was preserved in memory is restarted. This is referred to as an S3 resume with Gap code and the DRT_Flag GapCodeOnS3Resume SHALL be SET (1).

The details of how a platform performs a resume from S3 with Gap code is outside the scope of this document.

If the platform resumes from S3 with a Gap, a previous version of a dynamic operating system is not trusted and the dynamic PCR SHALL be in the default state of all bits set.

In order to transition a platform with the GapCodeOnS3Resume flag set into a dynamic trusted state, a D-RTM event must occur. The details of how to create a DLME that is able validate the memory preserved during S3 is beyond the scope of this document. However, all the requirements for launching a DLME SHALL be met.

5.3.8 The Dynamic Operating System Environment to S3

PM-provided DLME_Exit should be called for power state transition from S0 to S3 to allow PM code to prepare system for a power state transition.

5.3.9 S3 to the Dynamic Operating System Environment

5.3.9.1 Gapless S3 Resume

The DRT_Flag GapCodeOnS3Resume indicates whether the platform runs Gap code on S3 resume. If the flag GapCodeOnS3Resume is CLEAR (0), the platform must not run any Gap code resuming from S3 before control is passed to the resume vector. If DRT-flag GapCodeOnS3Resume is SET (1), the DLME should take all necessary steps to protect itself on resume. How to achieve this is outside of scope of this specification.

5.3.9.2 DMA

For a gapless S3 resume, the CRTM code must set up DMA protections before any DMA hardware is enabled on resume from S3 sleep state. PM-provided DLME_Exit and CRTM code can have a mechanism to pass the DMA protection requirement to the CRTM code.

5.3.10 S-RTM OS Present Environment to S4/S5

The transition from an S-RTM OS Present Environment to either the S4 or S5 system power state is defined in the TCG Client or Server specification.

5.3.11 The Dynamic Operating System Environment to S4/S5

The transition from S0 to S4 or S5 sleep state is defined in the TCG Client Specification. Additionally, DLME_Exit should be called with the power state transition as parameter.

6 PCR Usage

6.1 D-RTM PCR Values and PCR Reset

The D-RTM Platform Configuration Registers (PCR) are defined in the PC Client Conventional BIOS specification (1).

The S-CRTM begins recording measurements into S-RTM PCR at the beginning of the startup process. Code running later is unable to tamper with the earlier measurements recorded in the S-RTM PCR. This prevents code from extending the S-RTM PCR to arbitrary values.

The D-RTM process starts when the DL Event executes and resets the dynamic PCR to zero. On some platform implementations, it is possible for software to extend the values of the D-RTM PCR before executing the DL Event. Extends made by software prior to the DL Event are distinguishable because the initial value for the PCR is –1 (all bits set). Only by executing the DL Event will the platform reset the D-RTM PCR to zero. Immediately after resetting the values to zero, the D-CRTM records measurements into the D-RTM PCR. Because the D-RTM process resets the PCR to zero, if the D-RTM process is invoked multiple times during the same platform boot cycle, each invocation's measurements will be reflected in the resulting D-RTM PCR irrespective of the D-RTM PCR values before the invocation.

Before a platform enters S3 (sleep), the operating system instructs the TPM to save its state so it can be restored after resuming from S3. Upon resuming from S3, the platform firmware sends the TPM_Startup command to the TPM with parameters to restore the saved TPM state. The restored TPM state includes the S-RTM PCR, but does not include the D-RTM PCR. Just like a cold boot, the initial values for the D-RTM PCR after resuming from sleep are –1.

Calling DLME_Exit does not modify any PCR values. When a trusted operating system exits, it may extend values into the D-RTM PCR so unevaluated code that runs after DLME_Exit is unable to unseal data protected by D-RTM PCR values.

6.2 Detail and Authority Measurements

This specification uses two kinds of measurements. A "detail" measurement is a hash of a component that is measured. An example of a detail measurement is the S-RTM PCR measurements that are hashes of code that execute during the boot process. Due to the nature of the hash algorithm, even a trivial change in the component being measured results in the hash value changing.

An "authority" measurement is a hash of a signing authority of a component. The measurement process must first confirm the signing authority has signed the component. The measurement process then calculates the hash of some unique identifying property of the signing authority, such as the public key used to verify the signature. If the signature of the component cannot be verified, the authority measurement is undefined and a well-known error code is used for the measurement value. The intent of authority measurements is that the signer of a component may modify the component and sign the modified component. Because the new component was signed in the same way, the resulting authority measurement is identical for both the old and the new component. A single component may be signed with multiple different signature algorithms and the measuring entity may elect which signature algorithm to use to verify the component's signature.

One drawback of the authority measurement scheme is that entities that have already been signed by an authority may later be discovered to have implementation flaws, including security holes. By itself, the authority measurement scheme does not support the ability for an authority to revoke a signed component.

Stakeholders interested in knowing precisely which components executed during a D-RTM event will use the detail measurements. By exclusively using the detail measurements, a stakeholder does not have a trust dependency on the signing authority for the components involved in the D-RTM event.

Stakeholders who implicitly trust the signing authorities of components involved in the D-RTM process will use the authority measurements.

6.3 DCE Authority Measurement Special Case

The most useful authority measurement for stakeholders of this specification will be attesting that the platform manufacturer's intended DCE ran during the D-RTM process and was successful at placing the platform in a known good state and launching a measured DLME. To simplify analysis of this situation, a special well-known authority measurement of 00h has been defined, which is used instead of the actual DCE authority measurement when this condition occurs.

It should be noted the platform manufacturer's intended DCE does not consist only of the DCE signed by the platform manufacturer. It means the correct DCE component intended for the current firmware running on the platform was run to validate the platform. For example, this excludes a DCE component that may have been signed by the DCE authority, but was not intended for the current running firmware on the platform. This prevents a malicious attacker from using a DCE component from platform A, on platform B even if both platforms had DCE components produced by the same platform manufacturer and even signed with the same signing key.

6.4 DCE Component Revocation

In some circumstances the intended DCE may change over time for a specific firmware instance. For example, a platform manufacturer may release a DCE component that sometime later is determined to have a security flaw, causing a new version of the DCE component to be the intended DCE component for a firmware version already present in the field. To protect its customers from the flawed DCE component, the platform manufacturer shall include information in a firmware update (which includes any manifest update) for the platform that prevents undetected rolling back to the old firmware and old manifest. As a result, after rolling back, the old flawed DCE component on the platform will cause the special well-known authority measurement (Section 6.7) to not be recorded in the PCR.Authorities during the D-RTM process.

6.5 D-RTM PCR Defined in This Specification

The following PCR are defined in this specification:

- **PCR.Details.** Used to record detail measurements of all components involved in the D-RTM process, including components provided by the chipset manufacturer, the platform manufacturer, and the operating system vendor. This PCR SHALL be PCR[17] or PCR[18] (see 4.3 above) and shall be different from PCR.Authorities.
- **PCR.Authorities.** Used to record the authority measurements of the chipset manufacturer components and the platform manufacturer components. These measurements may include a special well-known value in the case that the intended validation steps occur for the current firmware on the hardware platform. If some components in the D-RTM process are not signed, well-known error codes will be recorded in this PCR to invalidate its measurements. This PCR SHALL be PCR[17] or PCR[18] (see 4.3 above) and shall be different from PCR.Details.
- **PCR.DLME.Authority.** Used to record the authority of the DLME launched at the end of the D-RTM process. There are advantages to separating the authority measurement of the DLME from the authority measurement for the chipset manufacturer and the platform manufacturer. As an example, a platform manufacturer may seal data to the PCR.Authorities without regard for which DLME may access the data. This PCR SHALL be PCR[19].

PCR 20 through 22 are not defined in this specification and are reserved for use by the DLME.

6.6 Detail and Authority Measurement Definitions

6.6.1 Details Measurement

The process for calculating a detail measurement for D-RTM is:

1) The measuring entity SHALL calculate the hash of the entire component.
2) The hash value is the measurement.

6.6.2 Authority Measurement

The process used by the D-CRTM for calculating an authority measurement for D-RTM is:

1) In some situations the measuring entity for an authority measurement may be incapable of evaluating a component's signature, or no signature exists. In this situation, the measuring entity SHALL calculate a detail measurement instead of an authority measurement for the component. The detail measurement is used in place of the authority measurement. The detail calculation SHALL hash the entire component including any signature information.

   NOTE This situation is discouraged because the measurement changes if any change occurs in the component even if the authority for the component sanctions the change.

2) The measuring entity SHALL examine the signature information for the component and determine the signature algorithm or algorithms used to sign the component. If the component does not contain signature information or contains improperly formatted signature information, the measurement value SHALL be 01h or the platform SHALL enter remediation. This value represents an error by the measuring entity evaluating the authority measurement.
3) If the measuring entity does not support any of the signature algorithms used to sign the component, the authority measurement cannot be evaluated by the measuring entity. The measurement value SHALL be 02h or the platform SHALL enter remediation. This value represents an error by the measuring entity evaluating the authority measurement.
4) If the measuring entity supports more than one signature algorithm used to sign the component, it SHALL choose one of the signature algorithms to use. The mechanism for choosing the algorithm SHALL be deterministic, but MAY be guided by implementation-specific policy. Once the signature algorithm is selected, the measuring entity SHALL ignore all other signature information.
5) The measuring entity SHALL verify that the signature of the component matches the signature information of the selected algorithm by cryptographic means. If the signature does not match, the measurement value SHALL be 03h or the platform SHALL enter remediation. This value indicates the measuring entity encountered a problem matching the selected signature information of the component with the component's signature.
6) The measuring entity SHALL hash a unique identifier for the authority signing key. The hash value is the authority measurement for the component.
7) If the measuring entity encounters an error or fails to successfully validate the platform (other than those listed above), the authority measurement SHALL be 04h or the platform SHALL enter remediation.
8) In the special case that the authority measurement is for the DCE and it is the DCE intended by the Pre-Gap code to validate the current running firmware on the hardware platform, the authority value SHALL be 00h. The authority measurement may encapsulate multiple DCE components based on a manifest. The authority measurement for the DCE also SHALL be recorded as a detail measurement.

Requirements for authorities when signing components:

- A given authority SHALL use a unique signing key per a signing algorithm. The same signing key shall never be used with multiple algorithms.
- Authority values 00h through FFh SHALL be reserved for use by this specification.
- The hash of the unique identifier for a signing key SHALL NOT be a number between 00h and FFh inclusive, in order to distinguish it from the reserved values.
- Components SHALL NOT have a hash with a detail measurement of 00h through FFh inclusive, in order to distinguish it from the reserved values.

6.7 The Limited Set of Known PCR.Authorities Values

Section 2.7 above provides the general motivation for authorities measurement: the reduced fragility of placing trust in who provided a system component instead of what was provided. The measurement of the DCE was a primary motivation for the creation of authorities measurements. Within a DCE there may be multiple components provided by the processor manufacturer, chipset manufacturer, or platform manufacturer. But ultimately through the process of constructing the platform, the platform manufacturer is the single authority for the platform and thus the DCE. Although each individual DCE component may be trustworthy, if the platform manufacturer does not correctly combine them during construction of the platform, trust in the platform cannot be ensured.

As briefly discussed in Section 6.6 above, there is the additional concern that the DCE that is measured and executed is the DCE that was provided by the platform manufacturer and is the DCE that matches the Pre-Gap code. The detection that the DCE was provided by the platform manufacturer is handled by the authorities measurement process, but this alone cannot handle the problem where an older DCE may be executed on a newer platform, allowing previously discovered bugs to persist. In other words, there is no mechanism for an authority to revoke a DCE without either changing the authority's signing key or requiring software to rely on details measurements.

To handle this situation, a special authorities measurement is used to indicate that the platform's Pre-Gap code and DCE matched. Each platform implementation will provide a mechanism to create a link between the Pre-Gap code and the DCE measurement. Once this linkage has been established, the special authorities measurement is sufficient to record that the Pre-Gap and DCE are under the platform's authority. Any further evaluation of the nature of that authority must rely upon the Platform Certificate.

7 DCE Requirements

7.1 Introduction

The fundamental requirement on the DCE is that it can verify that platform components (hardware and firmware) that can affect the DLME TCB are placed in a validated state before control is passed to the DLME. Additionally, the DCE must validate that the description of the TCB hardware that is given to the DLME is correct.

7.2 D-CRTM

The DL Event that starts the DCE process MUST cause a Locality 4 access to the TPM that resets PCR 17-22 inclusive to zero. The D-CRTM MUST set the initial PCR values as described in Section 6.1**Error! Reference source not found.**.

7.3 Configuration Description Validation

Information communicated to the DLME for purposes of describing the TCB hardware MUST be validated as being correct. This includes a portion of the ACPI tables and the D-RTM Resources Table. The validation methods are described in section 8.

7.4 Validation and Measurement

The DCE MUST perform the validation and measurement actions described in section 8. Whenever possible, the DCE should attempt to mitigate any validation errors that it may detect, otherwise it MUST remediate.

7.5 Remediation

Upon detecting an error that prevents successful validation, the DCE SHALL initiate a platform-specific remediation but it SHALL place the PCR.Details and PCR.Authorities into a value that does not represent a trusted state.

The first action in this remediation is to extend a capping value to both PCR.Details and PCR.Authorities.

- The capping value for PCR.Details SHALL be 01h.
- The capping value for PCR.Authorities SHALL be 04h (see 6.26.6 above).

If the TPM cannot be accessed the platform SHALL NOT execute the DLME.

8 Component Setup, Configuration, Validation, and Measurement Required for the D-RTM Process

8.1 Introduction

This section defines the normative actions required to set up, configure, validate, and measure the components required for the D-RTM process. Each section is divided into two subsections—one specifies the requirements related to the setup and configuration of that component typically performed before the D-RTM event, the second subsection specifies the requirements related to the validation and measurement of the component typically performed by the DCE or the DLME after the D-RTM event.

8.2 DMA

8.2.1 Setup and Configuration

Direct Memory Access (DMA) by peripherals has the potential to modify sensitive memory and corrupt the TCB if it is not properly controlled. The platform S-CRTM SHALL take steps to protect the S-CRTM from DMA (following all requirements in the PC Client or Server specification). This is commonly accomplished by not enabling DMA during the S-CRTM, although other techniques may be used.

When the processor performs the D-RTM event, it SHALL configure the DMA subsystem to prevent access to the DCE, DLME, and any other sensitive resources and SHALL maintain those protections through the launching of the DLME. The details of DMA configuration are implementation-dependent, but a preferred implementation to provide DMA protections is to enable DMA remapping with an empty mapping table.

8.2.2 Validation and Measurement

The DCE SHALL validate that all sensitive resources, including the DCE and DLME, are protected from DMA operations.

8.3 SMM

8.3.1 Setup and Configuration

8.3.1.1 Pre-Gap Initialization

This “verification” method relies on PM-provided code running before the Gap to install the correct SMM for the platform. After SMRAM is initialized, SMM must be locked.

If SMM integrity is provided by this method, then the DCE is using delegated validation of SMM. The requirements for this delegation are defined in Section 3.3.2.

8.3.1.2 S-CRTM Validation

The DCE can validate the S-CRTM by inspecting the code that is run on reset. If that code has not been modified, then the DCE can trust the measurements that are taken by that code.

To support delegated validation, all of the BIOS code run up to the point that SMM is locked must be checked. The check requires that the code be measured and compared to an expected value. If the check fails, the BIOS must enter remediation. The check may be done either by the BIOS or the DCE.

The DCE must check some portion of the code in all cases. Minimally, it must check the code that runs from CPU reset, to the point where the first code measurement is checked or to the point where SMM is locked, whichever comes first.

NOTE It is not strictly necessary for the BIOS to extend this checked value to the TPM because an incorrect value will cause the BIOS to go into remediation.

If the DCE will not check to the SMM lock, then BIOS must continue the checking to the point of SMM lock.

The value expected by the DCE when it checks the S-CRTM must be extended to the PCR.Authorities because it represents a measurement that, presumably, is immutable over the life of the platform and a change to this measurement represents a change that could make the other DCE measurements misrepresent the platform authorities.

NOTE It is unlikely that all the code on a platform between CPU reset and SMM lock will remain unchanged over the lifetime of the system. For this reason, it is expected that a small immutable portion of BIOS will be used as the S-CRTM and measured by the DCE with the remainder of the code being measured by the BIOS.

When the DCE validates the S-CRTM, it must also validate the first expected value of the S-CRTM.

8.3.1.3 S-CRTM Expected Value

If the S-CRTM is going to make and validate measurements, it will need to have a way of validating those measurements against an expected value. This value will likely change on each BIOS update. This value would likely not be included in the measurement of S-CRTM that the DCE extends to the PCR.Authorities.

The Chain of Trust is dependent on this first value. If an attacker could provide modified BIOS and a modified check value, the S-CRTM would not know that the system has been compromised.

In order to allow the value extended to PCR.Authorities to remain constant, the expected value is checked separately. The DCE must do an asymmetric signature check on the expected value as part of the validation process.

NOTE This expected-value check is required when the S-CRTM is not completely immutable on a platform and the DCE is delegating the validation of SMM to the BIOS (including SMM).

8.3.2 Validation and Measurement

The fundamental requirement for SMM validation is that the SMM that is running after the DCE executes is known to be an SMM that is intended for the platform on which it is running as verified by the PM. If the mechanisms described below do not achieve this requirement on a specific platform, then it is the responsibility of the PM to implement a method that achieves this requirement.

The SMM measurements must follow the following criteria:

1. The Details PCR holds explicit hashes of all code that may be used as an overlay by SMM. SMM measures the code it is going to call out to and makes sure its hash matches one of the hash values recorded in the details PCR.
2. The Details PCR would hold information:
    a) About the mechanism SMM uses to validate external code (this might be the measurement of the SMM code itself).
    b) About any external code that may be used as an overlay. It includes the specific hash of the code (possibly by using a hash of the entire signature block of the code). (In this case the DCE would not measure the code directly, only the signature blocks. At runtime, SMRAM would validate the signature block against the list of signature blocks measured into the details PCR by the DCE, and then SMM would verify that the external code matches the signature block.)

3. For the case where code is isolated, its signature was validated when it was updated. Code measurement does not go in the details PCR, only a measurement representing the mechanism for isolating the external code before calling out. This may be the measurement of the SMM code itself.
4. At the completion of the DCE, the SMM SHALL be locked by the hardware.

8.3.2.1 Delegated Validation of SMM

The DCE may delegate the validation of SMM to another trusted component that may be either the Pre-Gap code or the platform manufacturer-controlled BIOS update process.

> NOTE If SMM hardware locking is not available on the platform, then delegation of validation is not allowed.

For Pre-Gap validation, an asymmetric signature validation on the SMM code must be performed before that code is loaded. This validation must ensure that

- the code was provided by the PM,
- the code is appropriate for the current platform, and
- the code complies with the anti-rollback policy of the platform.

If the Pre-Gap validation fails, the BIOS must enter remediation and not allow a normal startup until the problem is corrected.

If the SMM is maintained in protected storage (e.g., flash memory), locked by hardware when that code is loaded, and the validation of the SMM is done by the platform manufacturer-controlled BIOS update process, then the same checks listed above must be performed before the update is allowed.

8.3.2.2 Direct Inspection Validation

Direct inspection of the SMRAM code/data requires that the DCE verify that the current contents of SMRAM contain code that was signed by the PM and that the data is in an internally consistent state.

The details of this checking are beyond the scope of this document but the problem is recognized to be difficult. As a consequence, it is expected that few implementations will use this method.

8.4 Non-Host Platform

8.4.1 Setup and Configuration

The DCE SHALL validate the integrity of any updatable NHPs. Three classes of NHPs exist in systems:

- **Immutable**. NHPs that have hardcoded firmware that cannot be modified in the field and are thus not updatable. They are considered to be benign and the DCE validates them by construction.
- **Simple**. NHPs that can be modified in the field, but the DCE can directly access their code to measure and verify.
- **Complex**. NHPs that are very sophisticated, typically arising in servers, and are often full systems running an operating system.

Simple and complex NHPs are updatable and SHALL be validated by the DCE.

A DCE may validate a simple NHP by:

- measuring the NHP without quiescing it if the DCE is running in memory that cannot be modified by the NHP, or

- measuring the NHP after quiescing it if the DCE is running in memory that can be modified by the NHP.

A DCE may validate a complex NHP by:

- measuring the NHP after quiescing the NHP, but only if the NHP cannot be dynamically modified (e.g., code loaded from network or remote disk) after measurement,
- remotely attesting and validating the NHP security properties, but only if the NHP cannot modify the DCE, or
- using information that is bound to the platform's certificate. Platform manufacturers who implement complex NHPs should design them as *very* high-security devices because they cannot be measured by the DCE and can only be implicitly trusted.

Systems containing NHPs that cannot be directly measured or cannot be put into a required quiesced state have weak trust properties and are not recommended.

8.4.2 Validation and Measurement

The DCE SHALL verify the integrity of any updatable NHPs on the platform and the verification process SHALL be isolated from any influence by the updatable NHPs.

- If the DCE verifies the integrity of a DCE-verifiable NHP, the hardware must be designed such that the DCE is able to verify that the DCE is isolated from the influence of the DCE-verifiable NHP. This may be expressed in the Platform Certificate (e.g., the hardware is designed to meet this requirement).
- If the DCE cannot verify an updatable NHP (per requirement #1 above), then the PM must provide information about the updatable NHP in information that is bound to the platform's certificate.

8.5 Peripherals

8.5.1 Setup and Configuration

The DCE does not need to perform any setup or configuration actions for peripherals.

8.5.2 Validation and Measurement

The DCE SHALL NOT extend peripherals measurements.

8.6 TPM

8.6.1 Setup and Configuration

The DCE does not need to perform any setup or configuration actions for the TPM.

8.6.2 Validation and Measurement

If the DCE needs to validate PCR[0] measurements and the TPM is deactivated, the platform should enter remediation or cap the PCR. Validation of PCR[0] (or any other PCR) by the DCE may not be required on some platform implementations, thus the TPM being deactivated may not cause remediation.

8.7 DCE Measurement

The initial measurement made by the CPU while executing the DL Event SHALL be extended in PCR 17. As with any event recorded to a PCR, a corresponding event log entry SHOULD be made.

8.8 Platform S-RTM

8.8.1 Setup and Configuration

The platform SHALL set up and configure the S-RTM as specified in the client or server implementation specification.(1)

8.8.2 Validation and Measurement

The platform SHALL validate the S-RTM and generate the measurements as specified in the client or server implementation specification.(1)

If the S-RTM implements an S3 resume without a Gap, it SHALL then CLEAR the DRT_Flag GapCodeOnS3Resume, otherwise it SHALL be SET.

8.9 Sensitive Resources

8.9.1 Setup and Configuration

The DCE MAY relocate Sensitive Resources that are able to be relocated.

8.9.2 Validation and Measurement

- All Sensitive Resources on the hardware platform that the DLME should protect to defend itself SHALL be listed in the D-RTM Resource Table when control is passed to the DLME from the DCE.
- If the location of a Sensitive Resource is able to be relocated, the DCE SHALL validate that the resource is located at the location provided in the list of Sensitive Resources when control is passed to the DLME.

8.10 Memory Map

8.10.1 Setup and Configuration

The platform manufacturer SHALL set up the memory map in a manner that enables the determination of whether memory is conventional.

The platform manufacturer SHALL configure the D-RTM Resource Table with a valid list of conventional memory.

8.10.2 Validation and Measurement

The DCE SHALL have a mechanism to validate that the D-RTM Resource Table is protected and is the Resource Table defined by the platform. The mechanism is platform-dependent.

The Resource Table is not measured.

8.11 ACPI Name Space

8.11.1 Setup and Configuration

A platform that implements an ACPI Name Space SHALL set up and configure it as specified in the appropriate standards.

8.11.2 Validation and Measurement

The platform SHALL have a mechanism to validate any portion of the ACPI Name Space that is required by the DLME to maintain its TCB, and provide a protected copy to the DCE. The mechanism is platform-dependent.

The ACPI Name Space is not measured.

8.12 Power State Transitions

8.12.1 Setup and Configuration

Prior to entering an S3 or S4 low power state, the DLME SHALL call DL_Exit to allow the platform to perform any actions necessary to properly transition into the low power state.

8.12.2 Validation and Measurement

If the DLME fails to call DL_Exit, the platform MAY fail to correctly resume from S3 or S4, resulting in either platform remediation or reset.

8.13 DLME

8.13.1 Setup and Configuration

Prior to calling the DLME, the DCE SHALL:

1. Validate and measure the DLME per Section 8.13.2, including any anti-aliasing checks.
2. Place the processor in the proper state per Section 5.3.4.10.
3. Put the processor into an address mode (32-bit or 64-bit) that corresponds with the address mode attribute in DlmeDescriptor->Attributes.
4. Enable virtual memory.
5. Initialize the DLME Stack frame and stack pointer.
6. Construct the D-RTM Resource Table and ACPI table list per Section 5.3.4.11.
7. Initialize the DLME parameters per Section 5.3.4.1.

8.13.2 Validation

8.13.2.1 Definitions

The DLME virtual memory range is DlmeDescriptor->DlmeStart to DlmeDescriptor->DlmeStart + DlmeDescriptor->DlmeLength.

The DLME_DESCRIPTOR structure virtual memory range is DlmeDescriptor to DlmeDescriptor + sizeof(DLME_DESCRIPTOR).

The DLME Stack virtual memory range is StackBuffer_Base to StackBuffer_Base + StackBuffer_Base + DlmeDescriptor->StackSize.

8.13.2.2 DlmeDescriptor Validation

If any of the validation criteria below fail, the platform SHALL enter remediation.

1. DlmeDescriptor->DlmeStart SHALL be less than 4 gigabytes.

2. DlmeDescriptor->DlmeStart + DlmeDescriptor->DlmeLength SHALL be less than 4 gigabytes.
3. StackBuffer_Base SHALL be less than 4 gigabytes.
4. StackBuffer_Base + DlmeDescriptor->StackSize SHALL be less than 4 gigabytes.
5. DlmeDescriptor->DlmeEntryPoint SHALL be equal to or greater than DlmeDescriptor->Start.
6. DlmeDescriptor->DlmeEntryPoint SHALL be less than DlmeDescriptor->DlmeStart + DlmeDescriptor->DlmeLength.
7. The DCE SHALL understand the DlmeDescriptor->Attributes. (Different attributes' definitions will be defined over time. After release, a DCE will not understand new attribute definitions.)
8. The DCE SHALL validate that the DlmeDescriptor->Attributes correspond with the manner by which the DCE will invoke the DLME (e.g., if the attributes denote a 32-bit DLME, the DCE SHALL invoke the DLME with the processor in 32-bit mode).
9. The DLME_DESCRIPTOR structure virtual memory range SHALL not overlap the DLME virtual memory range.
10. The DLME_DESCRIPTOR structure virtual memory range SHALL not overlap the DLME Stack virtual memory range.
11. The DCE SHALL validate that it supports the signature algorithm specified by DlmeDescriptor->DlmeSignature.

8.13.2.3 Memory Validation

If any of the validation criteria below fail, the platform SHALL enter remediation.

1. The memory map in place at the time of calling the DLME SHALL be a map that is consistent with the corresponding processor address mode for the DlmeDescriptor->Attributes value.
2. The memory map page tables SHALL be conventional system memory.
3. The memory map page tables SHALL be on physical address pages between 0 and 4 gigabytes.
4. The memory map page tables SHALL be on virtual address pages between 0 and 4 gigabytes.
5. The memory map in place at the time of calling the DLME SHALL contain mappings for the DLME virtual memory range, the DLME_DESCRIPTOR structure virtual memory range, and the DLME Stack virtual memory range.
6. The physical address pages used for the DLME virtual memory range, the DLME_DESCRIPTOR structure virtual memory range, and the DLME Stack virtual memory range SHALL not contain addresses listed in the Sensitive Resources table of the D-RTM ACPI table.
7. The physical address pages used for the DLME virtual memory range, the DLME_DESCRIPTOR structure virtual memory range, and the DLME Stack virtual memory range SHALL be conventional system memory between 0 and 4 gigabytes.
8. The physical address pages used for the DLME Stack SHALL be in conventional system memory between 0 and 4 gigabytes and/or firmware reserved memory allocated by the DCE between 0 and 4 gigabytes.
9. The DLME virtual memory range, the DLME_DESCRIPTOR structure virtual memory range, and the DLME Stack virtual memory range SHALL be between 0 and 4 gigabytes.
10. The physical address pages used for the DLME virtual address range and the DLME_DESCRIPTOR structure virtual address range SHALL NOT be any of the physical pages used for the DLME Stack virtual memory range.
11. Each page for the DLME virtual address range, the DLME_DESCRIPTOR structure virtual address range, and the DLME Stack virtual address range SHALL consist of unique physical

pages. No physical page may be aliased to two virtual pages. (For example, no single physical page used for the DLME Stack virtual memory range may be used for more than one virtual page.)

8.13.3 Measurement

1. The DCE SHALL measure the hash of the DlmeDescriptor virtual memory range contents concatenated with the DLME virtual memory range contents into PCR.Details.
2. The DCE SHALL measure the hash of the DlmePublicKey field of the DlmeDescriptor into the PCR.DLME.Authority.
3. If a DlmeSignature is not present in the DlmeDescriptor, an error measurement SHALL be recorded in the PCR.DLME.Authority.
4. If a DlmeSignature is present in the DlmeDescriptor but the signature verification fails, an error measurement SHALL be recorded in the PCR.DLME.Authority.
5. If a DlmeSignature is present in the DlmeDescriptor but the DCE lacks the capability to verify the signature, an error measurement SHALL be recorded in the PCR.DLME.Authority.

8.14 CPU and Microcode Updates

8.14.1 Setup and Configuration

The DCE MAY install CPU microcode patches.

8.14.2 Validation

1. The DCE SHALL validate that all Host Platform CPUs have the same microcode patches installed.
2. The DCE SHALL validate that any Host Platform CPU microcode patches installed are those intended by the platform manufacturer.
3. If the DCE installs CPU microcode patches, it SHALL install the same patches on all Host Platform CPUs.

8.14.3 Measurement

1. The D-CRTM SHALL measure the active microcode patches installed on the Host Platform CPU in PCR.Details. Different CPU microcode patches SHALL result in different measurements.
2. The DCE SHALL measure any CPU microcode patches it installs in the Host Platform CPU into PCR.Details. Different CPU microcode patches SHALL result in different measurements.
3. The PCR.Authorities SHALL reflect whether the Host Platform CPU microcode updates were validated by the platform manufacturer.
4. If CPU microcode patches must be installed for the platform manufacturer to validate the trustworthiness of the CPU and CPU microcode patches are discarded during S3/Resume, the platform manufacturer SHALL indicate the platform runs Gap code on S3/Resume per the GapCodeOnS3Resume flag in the DRT_Flags in the D-RTM Resources Table Structure.

9 Event Logging and Reporting

9.1 DCE Log

9.1.1 Introduction

The purpose of the DCE is to place the TCB hardware in a known state and then measure and start a software component (the DLME) provided by an OS vendor. The DCE places integrity measurements about itself and the DLME into PCR.

This section documents the structure of the measurement log that a remote verifier of a DLME could use to understand the individual measurements that resulted in the PCR values present at the start of DLME execution.

The measurement log may also be helpful for determining why a platform entered remediation. This use is platform-specific because the measurement log is only required to exist when the DCE calls the DLME.

9.1.2 Log Format

The D-RTM measurement log format SHALL have the same structure and format as the PC Client or Server specification used for the platform's S-RTM measurements. In other words, a platform will use either the format of either a Legacy BIOS or UEFI measurement log and either the TPM 1.2 or TPM 2.0 event structures that correspond to the platform firmware implementation and TPM version present.

9.1.3 ACPI Table Usage and Measurement Log Location and Lifetime

The DCE extends measurements into PCR and logs the meaning of the measurements into the D-RTM measurement log.

The firmware is required to define the memory associated with the D-RTM measurement log and reports this memory as "firmware reserved" memory. This allows the platform manufacturer to place the D-RTM measurement log location at a fixed location in order to simplify their implementation.

Platform manufactures should be careful to allocate enough space for the event log to hold all events. The DL_Entry code may verify that ample space is reserved for the D-RTM measurement log. If not enough space is reserved, it MAY return an error code to the caller of DL_Entry.

If the platform enters remediation it is possible, but not guaranteed, that the DCE may be able to provide the D-RTM measurement log and specify its location in the D-RTM Resources Table.

If the measurement event data exceeds the space reserved for the D-RTM event log, the DCE process SHALL extend a zero measurement into the Details and Authority PCR and MAY enter platform manufacturer remediation. The remediation process MAY reconfigure the platform to allocate more reserved space for the D-RTM measurement log in the future.

The system's firmware SHALL use the D-RTM Resources Table to specify the D-RTM measurement log location and size to the DLME. The information in the D-RTM Resources Table is not guaranteed to be valid until the DLME begins execution.

If the platform enters remediation, the D-RTM measurement log MAY be available. Its location MAY be specified in the D-RTM Resources Table.

9.1.4 Measurement Log Location in Memory

The DCE SHALL use identity-mapped pages to place the measurement log into the DLME's virtual address space. The DCE SHALL verify that the virtual memory of the log does not overlap the DLME, the descriptor block, or the DLME stack. If memory does overlap, the DCE SHALL NOT generate a measurement log and MAY enter remediation.

The location of the measurement log and its length are defined in the D-RTM Resources Table Log_Area_Start and Log_Area_Length fields. The details of those fields are defined in section 4.2.9.

9.1.5 Measurement Log Initialization

The DCE SHOULD initialize the measurement log address range to zeros before adding events.

Before the DLME is called, any unused space in the log SHALL be initialized to zeros.

9.1.6 Scope of PCR measurements in the log

The purpose of the D-RTM measurement log is to provide a mechanism for the DCE to communicate integrity measurements to the DLME for the most recent D-RTM event. Measurements from prior D-RTM events SHALL NOT be present. Measurements made by the DLME are outside the scope of this specification.

Resets of PCR are not recorded in the measurement log.

The D-RTM measurement log SHALL only contain event data about extensions to PCR.Details, PCR.Authorities, and PCR.DLME.Authority, or other informative events. The log SHALL only contain events that occurred after the most recent reset of PCR.Details, PCR.Authorities, and PCR.DLME.Authority.

9.1.7 Measured Events

Measurement	Description	Measured with Log Entry	Required?	PCR
PCR Resets	Resetting of PCR.Details, PCR.Authorities, and PCR.DLME.Authority	No	Yes	None
DCE	Measurement of the DCE	Yes	Yes	Details
DCE Authority	Authority for the DCE	Yes	Yes	Authorities
Sensitive Resources List	Resources_List in the DRT	Yes	Yes	Details
DLME Signature	The hash of the DlmePublicKey	Yes	Yes	DLME_Authority
DLME_Descriptor	The hash of the DLME_DESCRIPTOR	Yes	Yes	Details

The DCE may be implemented using multiple code segments, multiple data segments, multiple configuration data, or multiple authorities. If this is the case, the DCE or DCE Authority SHALL contain a measurement and log entry for each component of the DCE.

The DCE SHALL measure the Resources_List contained in the DRT into PCR.Details.

A D-RTM Example Implementations

A.1 Disclaimer

This specification is intended to be a vendor neutral way of doing a D-RTM system launch. A lot of questions arise as to how to actually implement the D-RTM. This appendix is informative – not normative. It is provided to help implementers understand how to map the D-RTM specification into existing hardware. The two examples given address how to map the D-RTM into two X86 architectures. Other implementations into other architectures are possible, but not addressed here.

A.2 Taxonomy of a D-RTM Capable BIOS

Two examples are given of how a D-RTM capable BIOS might be constructed. The examples address the two X86 processor architectures that are capable of launching with a D-RTM capable BIOS today. Other examples are possible.

A.2.1 A Few Comments on D-RTM Instructions, DL Event, and D-CRTM

The normative portion of the D-RTM specification describes the first portion of the DCE as the execution of DL Event followed by the D-CRTM. When a vendor implements the D-RTM architecture with the currently available CPU architectures, the combined “DL Event plus D-CRTM” component gets mapped into a combination of:

- CPU instruction (processor hardware)
- CPU Microcode (low level code embedded in the processor)
- X86 Processor Instructions

The implementation is done differently in the two X86 architectures given as examples in this appendix but both can be designed to be D-RTM compliant as the two examples given demonstrate.

The following figure shows how the implementations map into the D-RTM architecture at the D-RTM instruction level.

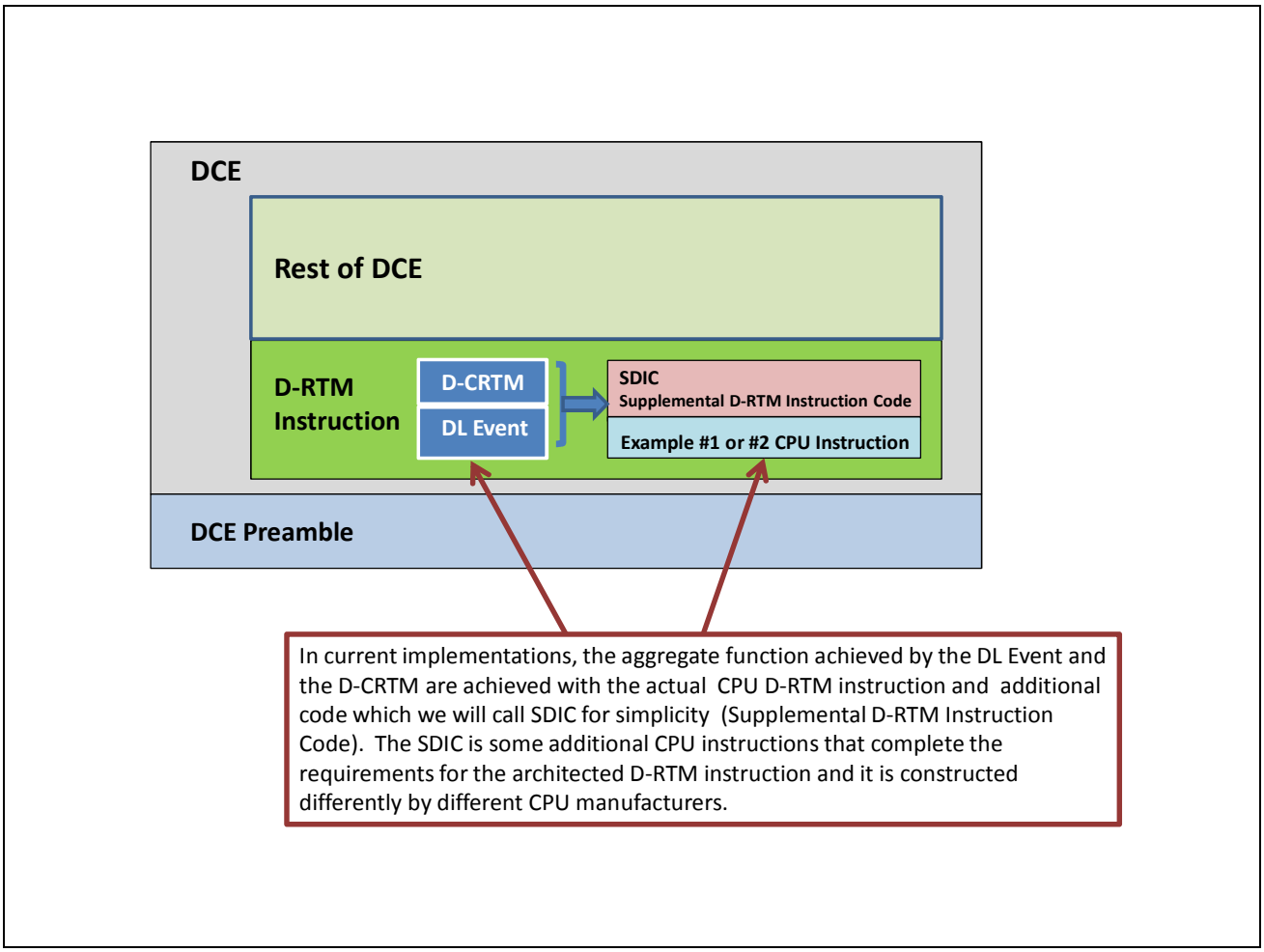

Figure 3 — Implementing the Requirements for the Architected D-RTM instruction with Current Architectures

A.2.2 Initial Common Steps Taken by Both D-RTM Example Implementations

The D-RTM capable platform BIOS is started as follows for both examples (see sections 5.3.1 through 5.3.3):

1) The platform is turned on and the S-RTM begins to execute starting with the CRTM.
2) The CRTM validates and launches the validated portion of the S-RTM. During the CRTM and validated S-RTM phase, the platform does all security operations that must be completed prior to entering the gap.
3) The validated S-RTM code launches the gap code. Many functions may run in the gap in addition to required D-RTM functions. The required D-RTM functions that run in the gap are:
   a) The DLME loader starts and loads a DLME into memory.
   b) The DLME loader locates the D-RTM ACPI table and determines the location of the D-RTM Preamble.
   c) The DLME loader calls the D-RTM Preamble passing the DLME descriptor.
   d) The D-RTM preamble does some platform-specific preparation work and loads the SDIC (Supplemental D-RTM Instruction Code) component into memory.
   e) The D-RTM preamble stores the DLME descriptor in a place where the DCE knows to find it.

A.2.3 D-RTM Implementation Example #1

The characteristics of the first example are:

- DCE Module #1, DCE Module #2, etc. are separately signed.
- The SDIC (Supplemental D-RTM Instruction Code) is a separate chunk of code not packaged with DCE module #1 and, if signed, signed separately from DCE module #1.
- The PCR.Authorities register is PCR[17].
- The PCR.Details register is PCR[18].

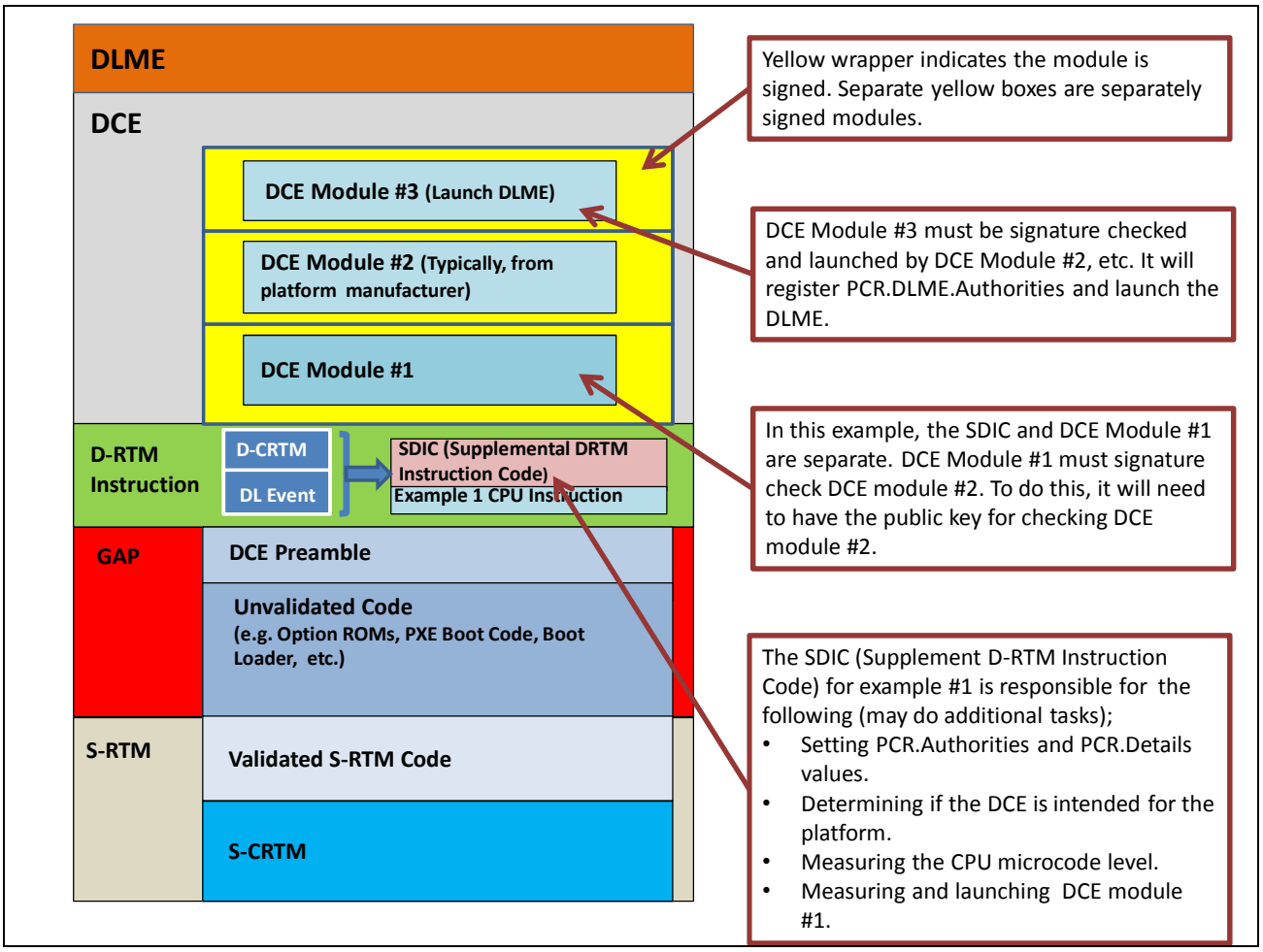

Figure 4 — The D-RTM Firmware Stack (Example 1)

See steps 1 through 3 given previously in the "Initial Common Steps Taken by Both D-RTM Example Implementations" section:

4) The D-RTM Preamble executes the D-RTM CPU instruction, passing the start address and size of the SDIC (Note: The gap ends when the D-RTM CPU instruction executes.). The D-RTM instruction (for example #1) does the following operations:
   a) Puts the chipset in a known state (e.g., stops all other processors from executing code, disables interrupts, etc.).
   b) Issues a TPM_HASH_START command to reset the D-RTM PCR.

c) Successively measures the SDIC component by sending a word at a time to the TPM using TPM_HASH_DATA.
d) Sends a TPM_HASH_END command to the TPM to finalize the SDIC component measurement and records the hash computed in PCR[17], which is the authorities measurement.

NOTE: This is an example of a detail measurement being placed into the PCR.Authorities. The measurement serves a dual purpose as both an authority measurement and a detail measurement.

e) Invokes the SDIC component.

5) The SDIC component locates and performs signature verification of DCE Module #1. It enters remediation if the DCE Module #1 fails its signature check.
6) The SDIC calculates the detail measurement of DCE Module #1.
7) The SDIC component records the detail measurement of DCE Module #1 in PCR[18], which is the PCR.Details.
8) If DCE Module #1 is the intended first DCE component for this platform, then the SDIC component extends 00h into PCR.Authorities. The SDIC component invokes DCE module #1.
9) If DCE Module #1 is not the intended DCE validation component for this platform, the SDIC component records the authority measurement for DCE Module #1 (eg. The hash of the public key that was used to sign DCE module #1) into PCR.Authorities. The SDIC component invokes DCE Module #1.
10) Jump to step 11 in the “Final Common Steps Taken by Both D-RTM Example Implementations” section below.

A.2.4 D-RTM Implementation Example #2

The characteristics of the second example are:

- DCE Module #1 and the SDIC are packaged in one signed module. This module is executed by the D-RTM instruction.
- DCE Module #2, DCE Module #3, etc. are separately signed.
- The PCR.Authorities register is PCR[18].
- The PCR.Details register is PCR[17].

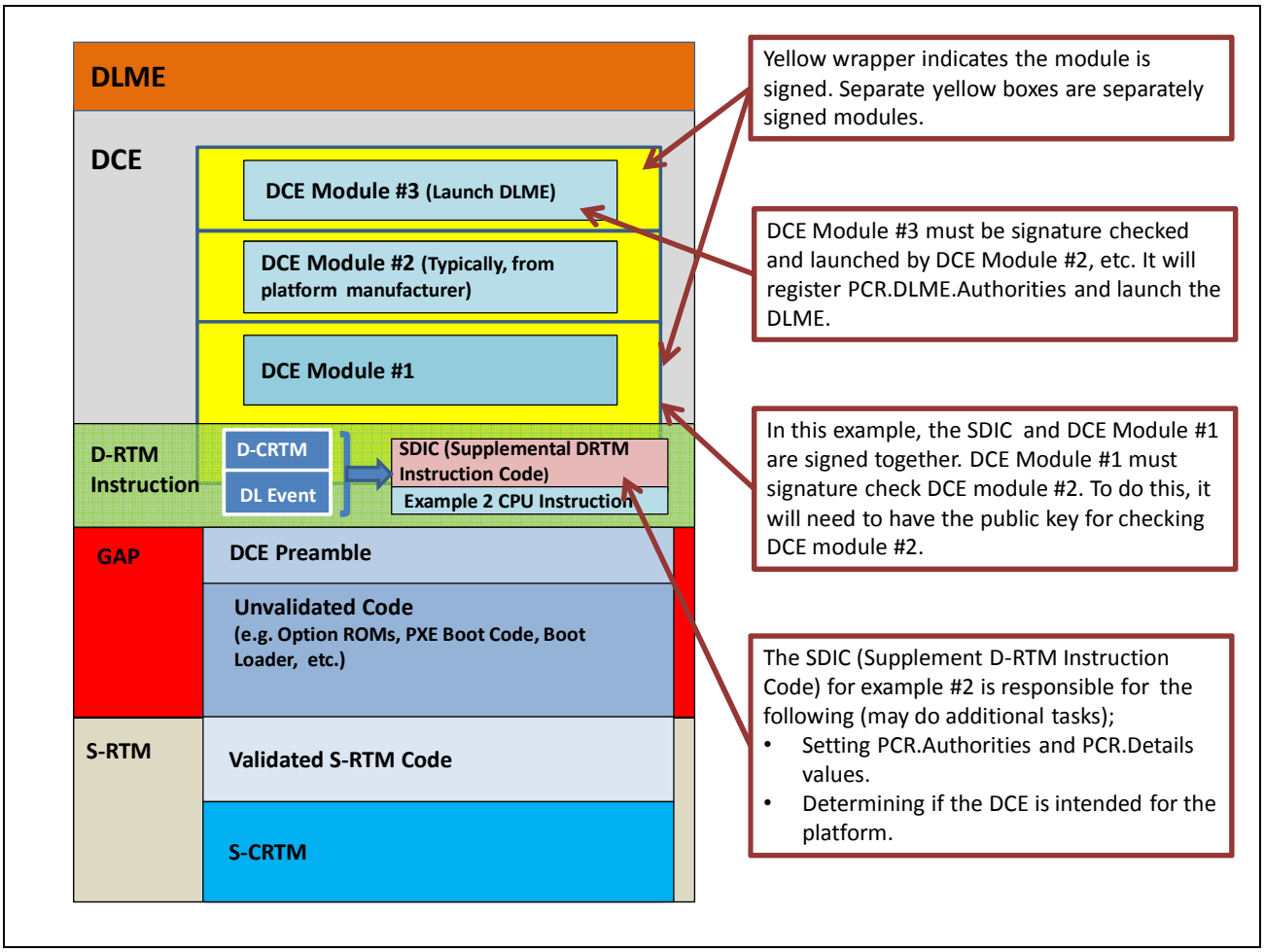

Figure 5 — The D-RTM Firmware Stack (Example 2)

See steps 1 through 3 given previously in the "Initial Common Steps Taken by Both D-RTM Example Implementations" section:

4) The D-RTM preamble executes the D-RTM CPU instruction, passing the start address and size of the signed combined module (containing both the SDIC and DCE Module #1) as operands.
5) The D-RTM CPU instruction takes the following actions:
   a) It puts the chipset in a known state (e.g., stops all other processors from executing code, disables interrupts, and so on).
   b) It loads code passed to it via operands mentioned in the previous step. This code is provided by the chipset manufacturer and moved from a fixed location in memory into a location protected from modification by peripherals and Non-Host Platforms.
   c) It issues a TPM_HASH_START command to reset the D-RTM PCR.
   d) It hashes the loaded module (containing both the SDIC and DCE Module #1) and successively measures bytes of the hash of it by sending a byte at a time to the TPM using TPM_HASH_DATA.
   e) It sends a TPM_HASH_END command to the TPM to finalize this hash measurement and records the resulting hash value in PCR.Details.
   f) It verifies the signature of the combined code module (SDIC and DCE Module #1) to confirm whether it is signed by the chipset manufacturer; if the signature verification fails, the platform enters remediation and resets.
   g) It invokes the combined code module (starting with the SDIC component).

6) The SDIC confirms that DCE Module #1 is intended for the hardware platform.
7) If DCE Module #1 is intended for the platform, the SDIC extends 00h into PCR.Authorities.
8) If DCE Module #1 is not intended for this platform, the SDIC records the authority measurement for the chipset manufacturer in PCR.Authorities.

   Note: The detail measurement for the combined SDIC and DCE Module #1 has already been hash and hash extended into PCR.Details so this step does not have to be performed at this point for example #2.

9) Jump to step 11 in the "Final Common Steps Taken by Both D-RTM Example Implementations" section below.

A.2.5 Final Common Steps Taken by Both D-RTM Example Implementations

The D-RTM capable platform BIOS is ended as follows for both examples:

11) The SDIC module invokes DCE Module #1 which validates the state of the chipset.
12) DCE Module #1 validates DCE Module #2 extending PCR.Authorities if an authority change has occurred. It also puts a details measurement of DCE Module #2 into PCR.Details.
13) DCE Module #3 records the detail measurement of the DLME into PCR.Details.
14) The DCE Module #3 records the authority measurement of the DLME into PCR.DLME.Authorities (PCR[19]).
15) DCE Module #3 invokes the DLME.
16) The DLME runs.

A.2.6 Steps Taken to Exit the Measured DLME Environment

When the DLME needs to exit the measured environment (after it has been booted and running) it will do the following:

1) The DLME calls DLME_Exit and caps PCR.Details, PCR.Authorities, and PCR.DLME.Authority (e.g. Typically you cap a PCR by extending zero).
2) There may be additional chipset vendor specific details that must be attended to in exiting the measured DLME environment.

A.2.7 What Should PCR.Authorities Be Set To?

When an OS is installed on a platform, it is the responsibility of the installer to insure the platform is in a correct trustworthy state as the installing OS will accept the platform authority that is presented to it at install as correct.

For example #1, the PCR.Authorities register is established as follows:

1) The D-RTM instruction will hash the SDIC and hash extend the computed value into PCR.Authorities.
2) The SDIC will extend the hash of the SDIC into PCR.Details.
3) The SDIC will signature check the DCE module #1 and if it is signed correctly as prescribed by the platform manufacturer, it will hash extend '00'x into PCR.Authorities.
4) As long as each DCE module that follows is signed as expected by the platform manufacturer, there is no need to do further updates to PCR.Authorities. If an unexpected DCE module signature or module occurs, then the hash of the public key that module was signed with is extended into PCR.Authorities.

For example #2, PCR.Authorities register is established as follows:

1) The D-RTM instruction will signature check the module containing the SDIC and DCE Module #1. It will hash extend the computed value into PCR.Details. If the signature check fails, it will abort the D-RTM process without setting PCR.Authorities.
2) If the D-RTM instruction signature check described in step #1 passes, the SDIC will extend '00'x PCR.Authorities and begin running DCE module #1.
3) As long as each DCE module that follows is signed as expected by the platform manufacturer, there is no need to do further updates to PCR.Authorities. If an unexpected DCE module signature or module occurs, then the hash of the public key that module was signed with is extended into PCR.Authorities.

A.3 S-CRTM To D-RTM Preamble

The functional flow of the system from initial power on reset through the calling of the D-RTM preamble is represented by the following flowchart (see sections 5.3.1 through 5.3.3).

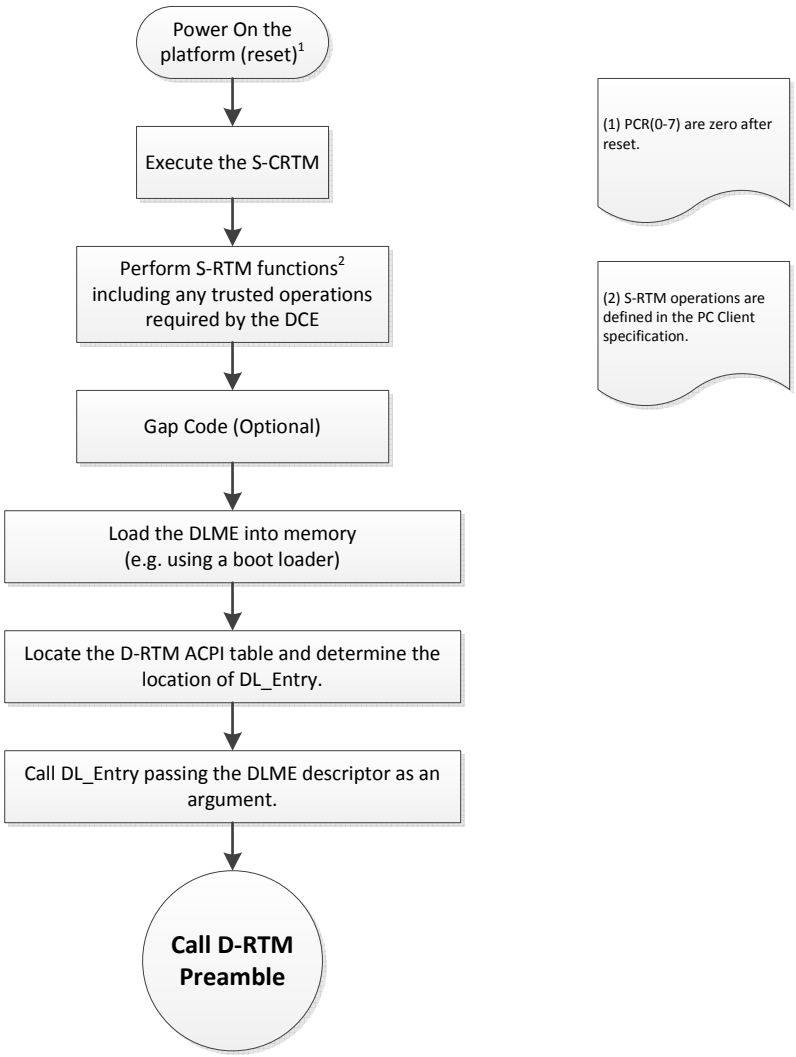

Figure 6 — S-CRTM To D-RTM Preamble

A.4 D-RTM Preamble and Instruction

The D-RTM Preamble is the final piece of code run before the processors D-RTM instruction is executed. The functions of the low level CPU instructions for examples 1 and 2 are given in this flowchart. When combined with the processor specific SDIC, these instructions create the function required by the DL Event and the D-CRTM for the two examples given.

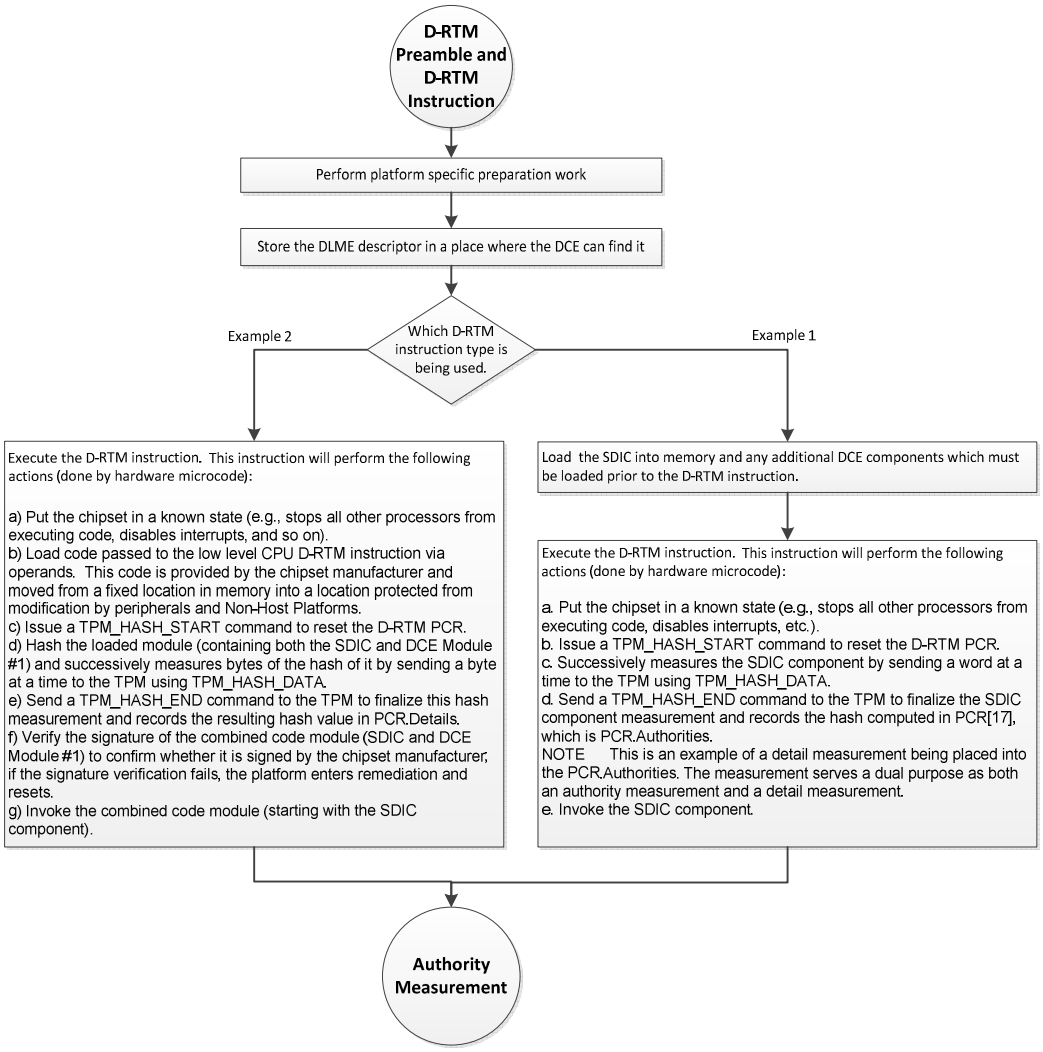

Figure 7 — D-RTM Preamble and Instruction

A.5 Authority Measurement

The following flowchart describes functional flow of an Authority measurement (see section 6.6.2). Authority measurements are computed for each code module before it is executed. Most notably the D-CRTM follows this flow to compute the initial PCR.Authorities and the DCE when computing the PCR.DLME.Authority. An implementation that cannot extend error values shall enter remediation.

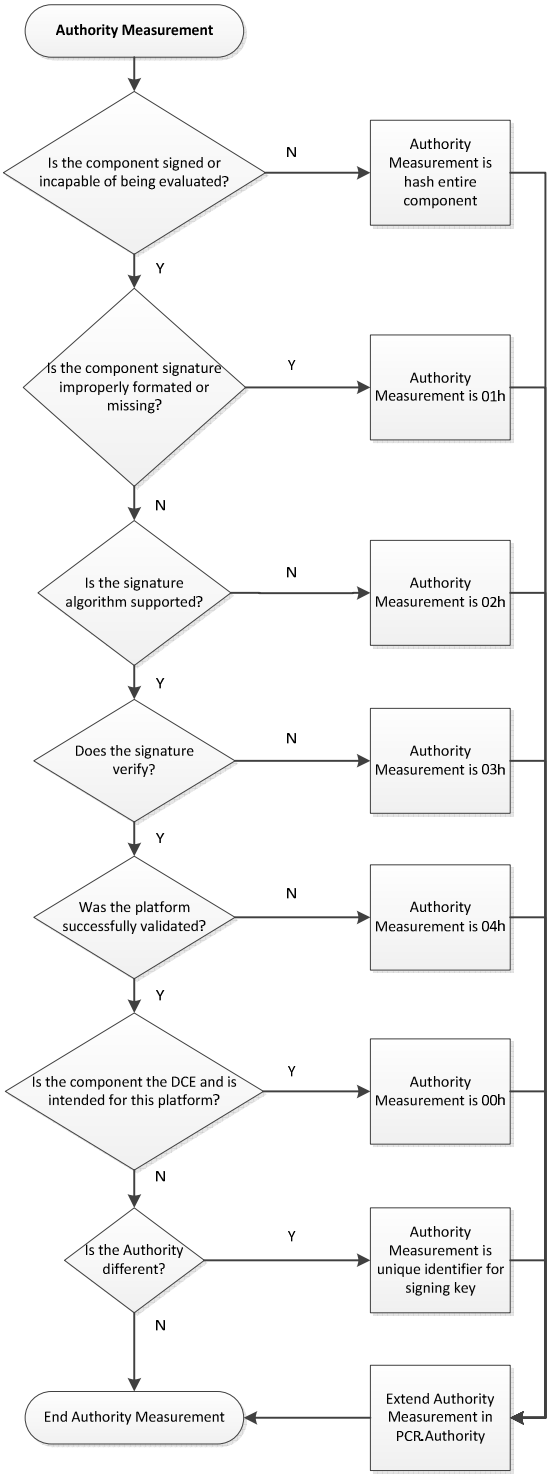

Figure 8 — Authority Measurement

A.6 Additional DCE Module Execution

The DCE may consist of more than one signed code module. The D-RTM Specification does not place restrictions on the number of signed modules used to build a DCE. An implementation may choose to

have one module from the chipset/processor vendor, one module from the platform vendor, or other modules. Each module may come from different authorities. Each module's details must be measured and extended in PCR.Details and any authority changes must be extended into PCR.Authorities.

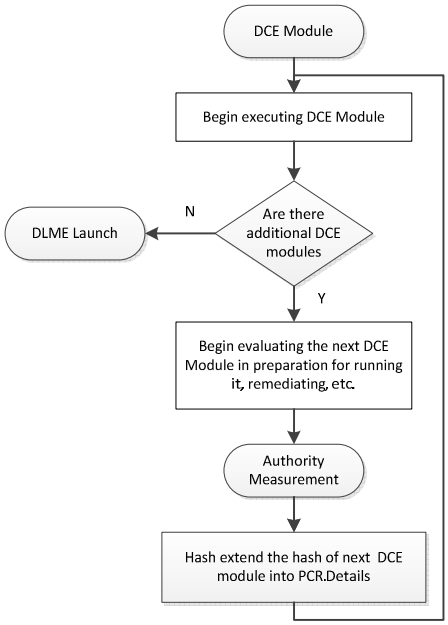

Figure 9 — DCE Module Execution

A.7 DLME Launch

The DCE will launch the DLME as described in section 5.3.4, measuring the details and authority into the PCR.Details and PCR.Detaiils.Authority respectively.

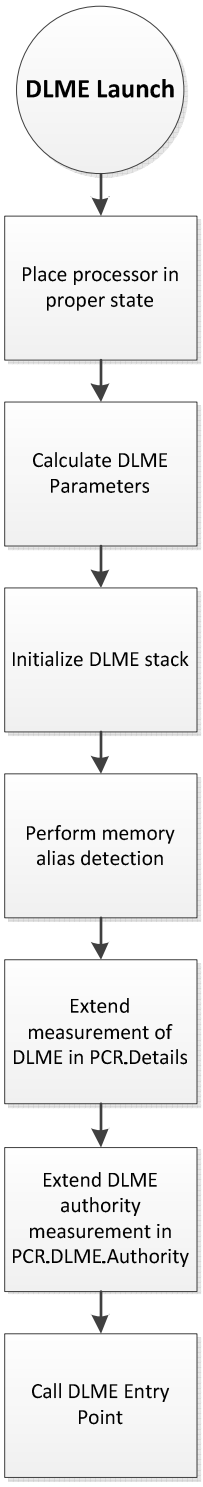

Figure 10 — DLME Launch