[Public]

Dynamic Root of Trust Measurement (DRTM) Service Integration Guide

Publication # **58453** Revision: **1.00**
Issue Date: **October 2024**

[Public]

© 2024 Advanced Micro Devices, Inc. All rights reserved.

The information contained herein is for informational purposes only, and is subject to change without notice. While every precaution has been taken in the preparation of this document, it may contain technical inaccuracies, omissions and typographical errors, and AMD is under no obligation to update or otherwise correct this information. Advanced Micro Devices, Inc. makes no representations or warranties with respect to the accuracy or completeness of the contents of this document, and assumes no liability of any kind, including the implied warranties of noninfringement, merchantability or fitness for particular purposes, with respect to the operation or use of AMD hardware, software or other products described herein. No license, including implied or arising by estoppel, to any intellectual property rights is granted by this document. Terms and limitations applicable to the purchase or use of AMD's products are as set forth in a signed agreement between the parties or in AMD's Standard Terms and Conditions of Sale.

Trademarks

AMD, the AMD Arrow logo, AMD EPYC, and combinations thereof are trademarks of Advanced Micro Devices, Inc. Other product names used in this publication are for identification purposes only and may be trademarks of their respective companies.

Contents

List of Figures

Revision History

Date	Revision	Description
October 2024	1.00	Initial public release.
May 2024	0.70	Initial NDA release.

Chapter 1 Introduction

1.1 Purpose

Traditionally, the trustworthiness of the Trusted Computing Base (TCB) is authenticated statically at boot time. DRTM represents a higher level of assurance by removing the firmware and bootloader from the TCB. The AMD solution for DRTM is comprised of multiple components, as illustrated in Figure 1.

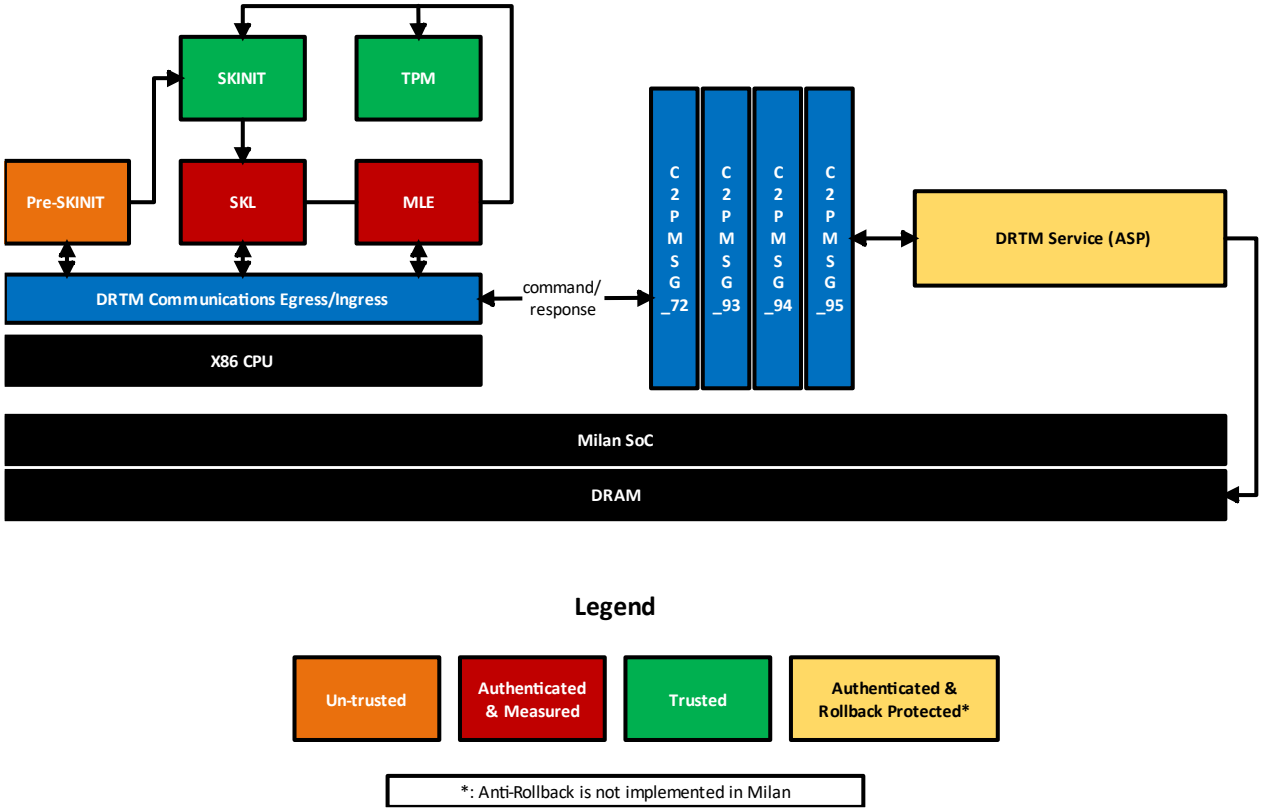


Figure 1. DRTM Secure Launch on AMD Platform

The DRTM Service is implemented in the AMD Secure Processor (ASP). Once the ASP has initialized and booted, the DRTM Service executes the commands it receives via the DRTM Communication Channel Pre-Secure Init (SKINIT) module, which runs on x86. The C2P_MSG_72 register functions as the Command/Response register. The other C2P MSG registers have their functionality defined by command (*refer to Chapter 3*).

***Note**: SKINIT is the AMD CPU instruction that reinitializes the processor, establishing a secure execution environment.*

1.2 Glossary

ASP: AMD Secure Processor

DMA: Direct Memory Access

DRTM: Dynamic Root of Trust of Measurement

DRTM INIT: DRTM Service Initialization

IVRS: I/O Virtualization Reporting Structure

MLE: Measured Launch Environment

SKINIT: Secure Init (AMD CPU instruction to reinitialize the processor establishing a secure execution environment for the next component)

SKL: Secure Kernel Loader

TEE: Trusted Execution Environment

TSME: Transparent Secure Memory Encryption

TMR: Trusted Memory Region

Chapter 2 Functional Description

The DRTM Service handles the following commands from x86. It is required for the x86 to write 0 to c2pmsg72 to start the DRTM Service Initialization in the ASP before executing the DRTM sequences as described below. The write to the c2pmsg72 will trigger a DRTM interrupt on the ASP; a value of 0 is preferred to avoid conflicts with other DRTM commands.

1. **DRTM Service Initialization**: At the initialization phase, the DRTM Service will get the anti-rollback* fuse state, the base address of the SKL module, and the DRTM Service configuration. X86 BIOS settings control the DRTM Service configuration (enabled or disabled). Please refer to Appendix B for the SKL format and limitations.

   **** Note**: AMD Family 19h Models 00h-0Fh do not support the anti-rollback feature.*

2. **DRTM Get Capability**: The pre-SKINIT module will call DRTM to obtain the capabilities of the DRTM Service such as the DRTM Enabled Flag, the TSME Enabled Flag, the Anti-Rollback* Fuse State, the number of TMRs supported, and the TMR memory alignment requirements.
3. **DRTM Setup TMR**: The pre-SKINIT module will call DRTM to set up the TMR to protect the memory region assigned to it from DMA attacks and unauthorized access.
4. **DRTM Launch**: The SKINIT module calls the DRTM Service in the ASP through DRTM mailbox commands. The SKL is verified in the DRTM Service. The SKL Signing Authority Key (4K Key) has a pre-defined little-endian format. Swapping Big Endian is not needed.

   The DRTM Service will perform the following tasks:

   a. Lock TPM locality 4 (to avoid Locality 3 and 4 simultaneously being open for x86).
   b. Open TPM locality 3 for ASP to Read/Extend PCR.
   c. Calculate Hash (SHA256 or SHA384) of SKL (code + header). The SKL base address is provided by SKINIT and read by DRTM Launch.
   d. Authenticate SKL signature (signature verification) using “SKL Signing Authority Key” (RSA 2048 or RSA 4096) as defined in Public Key Token Image in Fig. 3. Also, check the specific Platform Support for the algorithm used in the Appendices. For example, Appendix G for Milan
   e. Read PCR 17 (Locality 3) to get SKL Hash (SHA256) extended by pre-SKINIT code.

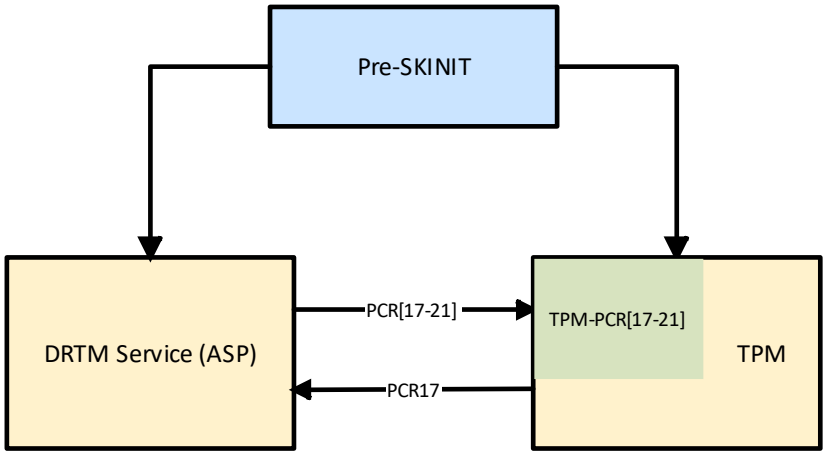


Figure 2. DRTM Launch

f. Verify the SKL by matching Hash from PCR 17 (Locality 3) with SKL Hash (SHA256)

If (SKL Authentication == FAIL):

i. Cap PCR [18-22] (Locality 3) by extending all 0xFF to PCR 18, 19, 20
ii. Remove the lock from TPM Locality 4
iii. Set “DrtmLaunchSuccess” = 0 and decline the rest of the command request in the DRTM sequence after DRTM Launch but it will still accept other DRTM commands.

Else:

i. Prepare TCG Logs for the “EV_TYPE_SL_LOAD” event for extended operation of SKL Hash.
ii. Compute Hash for SKL SPLT version and extend it to PCR 17.
iii. Perform Hash (SHA256) of “ASP Anti-Rollback Enabled Fuse State and TSME Enabled State” (two 32-bit words) and extend it to PCR 17.
iv. Prepare TCG Logs for the “EV_TYPE_TSME_RB_FUSE” event to extend the operation of the “Anti-Rollback* and TSME” Hash. The Anti-Rollback value is set to 0.
v. Write C2PMSG_93 with “Anti-Rollback state” D-word value and C2PMSG_94 with “TSME State” D-word value to be sent back to SKL at the end of the DRTM Launch command.
vi. Extend PCR 18 (Locality 3) with “SKL Signing Authority Key” Hash (SHA256).
vii. Prepare TCG Logs for the “EV_TYPE_SL_PUB_KEY” event to extend the operation of the “SKL Signing Authority Key” Hash.

viii. Remove the lock from TPM Locality 1 and 2.

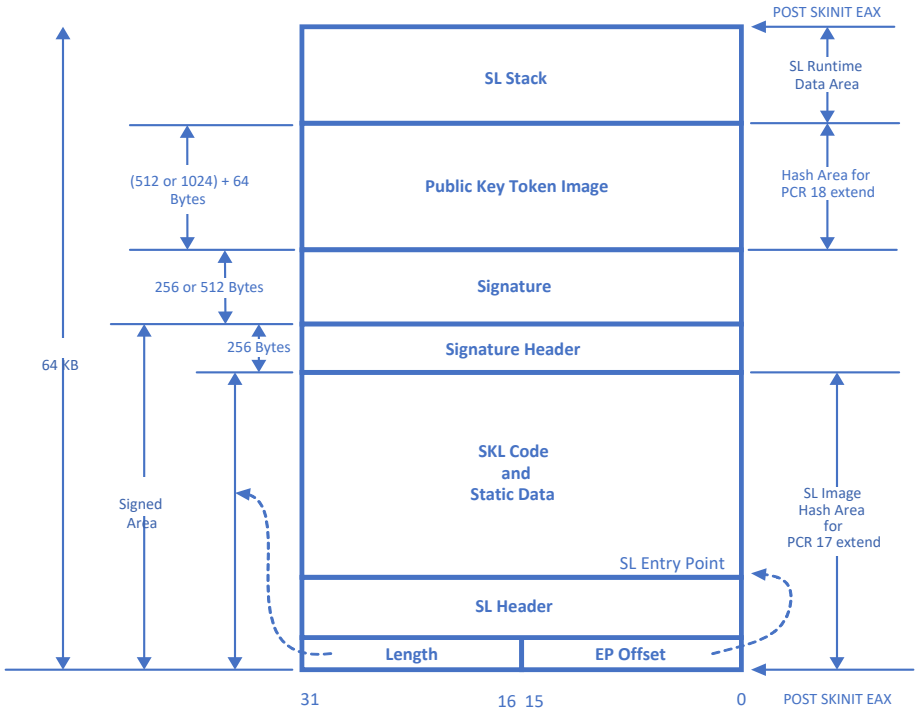


Figure 3. DRTM Authentication

5. **DRTM Extend MLE Digest**: The SKL module will call the DRTM Service with the MLE region's physical base address and size (please refer to Appendix E for MLE limitations). The DRTM Service confirms that the MLE is within the TMR range. The DRTM Service will then calculate the hash (SHA256) of the MLE and extend the hash to PCR 17 and PCR 18. In addition, it will calculate the hash and extend it to PCR 17 and PCR 18 for the marker 'SKL.'
6. **DRTM Get TMR Descriptors**: The SKL module will call the DRTM Service to get the information about all the TMRs currently set to protect DRTM memory.
7. **DRTM Allocate Shared Memory**: The SKL module calls the DRTM Service to allocate a chunk of x86 shared memory.
8. **DRTM Get TCG Logs**: The SKL module will call the DRTM Service to get the TCG Logs of all PCR extended events. The TCG Logs structure is predefined (refer to *Appendix C*). The DRTM Service will copy the TCG Logs to the DRAM region carved out by BIOS for ASP and send back the address and size of this region.
9. **DRTM Get IVRS Table Information**: The SKL will call the DRTM Service to retrieve the ACPI IVRS Table information sent earlier by the BIOS.
10. **DRTM Manage TPM Localities Access**: The MLE module will call the DRTM Service to manage the TPM localities access as per security requirements. The DRTM Service will perform the following tasks:
    a. Unlock TPM Locality 4

b. Lock TPM Locality 2

11. **DRTM Release TMR**: This releases all DRTM-specific TMR regions and reverts the DRTM security policy that was applied by the DRTM Launch. DRTM Manage TPM Localities Access (10) and DRTM Release TMR (11) should be the last two commands in the DRTM sequence. There is no ordering requirement between the last two commands.

Chapter 3 DRTM Interface Description

Four C2PMSG registers are used as the interface between the x86 side DRTM Modules and the ASP side DRTM Service. The interpretation of the fields in these registers depends upon the “command” field in the first register, explained in detail later in the document for each command. The first register (DRTM_C2PMSG_REG_72) will be used as the command/response register, the second register (DRTM_C2PMSG_REG_93) will be used to carry the size of buffer or Position in Hash buffer and the third and fourth registers (DRTM_C2PMSG_REG_94 and DRTM_C2PMSG_REG_95, respectively) will carry the buffer’s physical address as argument. The validity and definition of “fields” in C2PMSG registers may depend upon the “command” field in DRTM_C2PMSG_REG_72. For the specific MMIO addresses of these DRTM communication registers, please refer to Appendix G.2.

3.1 DRTM_C2PMSG_REG_72 Register Layout

***Note**: The “TmrIndex” field is only valid for the DRTM_CMD_TMR_SETUP command.*

DrtmC2PMsgReg72																															
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	Reserved			TmrIndex				Command								Status															

Definition of “`Status[15:0]`” field in **`DRTM_C2PMSG_REG_72`** register:

```
// This field will be set by DRTM Service and consumed by x86
#define DRTM_NO_ERROR                       0x00000000
#define DRTM_NOT_SUPPORTED                  0x00000001
#define DRTM_LAUNCH_ERROR                   0x00000002
#define DRTM_TMR_SETUP_FAILED_ERROR         0x00000003
#define DRTM_TMR_DESTROY_FAILED_ERROR       0x00000004
#define DRTM_GET_TCG_LOGS_FAILED            0x00000007
#define DRTM_OUT_OF_RESOURCES_ERROR         0x00000008
#define DRTM_GENERIC_ERROR                  0x00000009
#define DRTM_INVALID_SERVICE_ID_ERROR       0x0000000A
#define DRTM_MEMORY_UNALIGNED_ERROR         0x0000000B
#define DRTM_MINIMUM_SIZE_ERROR             0x0000000C
#define DRTM_GET_TMR_DESCRIPTOR_FAILED      0x0000000D
#define DRTM_EXTEND_MLE_DIGEST_FAILED       0x0000000E
#define DRTM_TMR_SETUP_NOT_ALLOWED          0x0000000F
#define DRTM_GET_IVRS_TABLE_FAILED          0x00000010
```

Definition of “`Command[23:16]`” field in **`DRTM_C2PMSG_REG_72`** register:

```
// This field will be set by x86 side DRTM Modules and read/consumed by DRTM Service
#define DRTM_CMD_GET_CAPABILITY             0x1 // Get capabilities of DRTM Service
#define DRTM_CMD_TMR_SETUP                  0x2 // Setup the TMR region
#define DRTM_CMD_TMR_RELEASE                0x3 // Release the TMR region
#define DRTM_CMD_LAUNCH                     0x4 // Launch and authenticate DRTM
#define DRTM_CMD_GET_TCG_LOGS               0x7 // Send TCG Logs to SKL
#define DRTM_CMD_TPM_LOCALITY_ACCESS        0x8 // Set TPM Localities access
#define DRTM_CMD_GET_TMR_DESCRIPTORS        0x9 // Get TMR Descriptors structures
#define DRTM_CMD_ALLOCATE_SHARED_MEMORY     0xA // Allocate Shared memory
#define DRTM_CMD_EXTEND_MLE_DIGEST          0xB // Extend MLE digest in PCR 17/18
#define DRTM_CMD_GET_IVRS_TABLE_INFO        0xC // Get IVRS Table Information
```

Definition of **"TmrIndex[27:24]"** field in **DRTM_C2PMSG_REG_72** register:

This field is set by x86 and consumed by the DRTM Service.

***Note**: The range of TmrIndex is from 0 to 7.*

Definition of the **"Ready [31]"** field in **DRTM_C2PMSG_REG_72** register:

This field will be set to 0 by Secure Loader to indicate a new command and set to 1 by the DRTM Service when the previous command finishes.

***Note**: If the "Ready" bit (Bit[31] of the DRTM_C2PMSG_REG_72 register) is set to "DRTM_RESPONSE_READY," the DRTM Service is available and ready to accept commands. Before calling the DRTM Service to execute new commands, x86 should poll the "Ready" bit to check whether it equals "DRTM_RESPONSE_READY." There should be a "pre-defined" suitable timeout in x86 up to which x86 polls for the "Ready" bit to be set before assuming that the DRTM Service is unavailable.*

Chapter 4 Interface Definitions

4.1 DRTM_CMD_GET_CAPABILITY

DRTM Modules (x86) <-> DRTM Service (ASP)

Request:

DRTM_C2PMSG_REG_72																											
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	**Reserved**			**N/A**				**Command**								**N/A**															

Command (IN) : DRTM_CMD_GET_CAPABILITY (0x01)
R(IN) : DRTM_COMMAND_READY (0x00)
DRTM_C2PMSG_REG_93 : N/A
DRTM_C2PMSG_REG_94 : N/A
DRTM_C2PMSG_REG_95 : N/A

Response:

DRTM_C2PMSG_REG_72																											
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	**Reserved**			**N/A**				**N/A**								**Status**															

Status (OUT) : One of the valid statuses defined in the "Status[15:0]" field of DRTM_C2PMSG_REG_72 reg
R(OUT) : DRTM_RESPONSE_READY

DRTM_C2PMSG_REG_93																											
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reserved																													**A**	**T**	**S**

(S) DRTM Enabled Bit (OUT) : Set to 1 if DRTM is Enabled, else set to zero
(T) TSME Enabled Bit (OUT) : Set to 1 if TSME is Enabled, else set to zero
(A) Anti-Rollback* Status Bit (OUT) : Set to 1 if Anti-Rollback Status bit is set, else set to zero

DRTM_C2PMSG_REG_94																											
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Version : <#>																															

The version number starts from 1.

DRTM_C2PMSG_REG_95																											
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
TMR Count								**TMR Alignment**																							

TMR Alignment (OUT) : 0x1000 for 4k Alignment and 0x10000 for 64k Alignment
TMR Count (OUT) : Number of TMR Available for DRTM

4.2 DRTM_CMD_TMR_SETUP

Pre-SKINIT (x86) <-> DRTM Service (ASP)

Request:

DRTM_C2PMSG_REG_72																												
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	**N/A**			**TmrIndex**				**Command**								**N/A**															

Command (IN) : DRTM_CMD_TMR_SETUP (0x02)
TmrIndex (IN) : DRTM_TMR_INDEX_0 (0x00) or... DRTM_TMR_INDEX_7 (0x07) (< TmrCount)
R(IN) : DRTM_COMMAND_READY (0x00)
DRTM_C2PMSG_REG_93 : Size of the TMR region in multiples of 64 KB per unit (minimum size > TMR Alignment)
DRTM_C2PMSG_REG_94 : 32-bit lower physical address of the start of TMR region
DRTM_C2PMSG_REG_95 : 32-bit upper physical address of the start of TMR region

Response:

DRTM_C2PMSG_REG_72																												
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	**N/A**			**N/A**				**N/A**								**Status**															

Status (OUT) : One of the valid statuses defined in the "Status[15:0]" field of DRTM_C2PMSG_REG_72 reg
R(OUT) : DRTM_RESPONSE_READY
DRTM_C2PMSG_REG_93 : N/A
DRTM_C2PMSG_REG_94 : N/A
DRTM_C2PMSG_REG_95 : N/A

4.3 DRTM_CMD_LAUNCH

SKINIT (x86) <-> DRTM Service (ASP)

Request:

DRTM_C2PMSG_REG_72																												
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	**N/A**			**N/A**				**Command**								**N/A**															

Command (IN) : DRTM_CMD_LAUNCH (0x03)
R(IN) : DRTM_COMMAND_READY (0x00)

Response:

DRTM_C2PMSG_REG_72																												
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	**N/A**			**N/A**				**N/A**								**Status**															

Status (OUT) : One of the valid statuses defined in the "Status[15:0]" field of DRTM_C2PMSG_REG_72 reg R(OUT)
: DRTM_RESPONSE_READY

DRTM_C2PMSG_REG_93																												
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reserved																															**A**

(A) Anti-Rollback* fuse state Bit (OUT) : Set to 1 if Anti-Rollback fuse is set, else set to zero

DRTM_C2PMSG_REG_94																													
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reserved																															**T**

(T) Transparent Secure Memory Encryption (TSEM) enable state Bit (OUT) : Set to 1 if TSEM enable state is set, else set to zero

DRTM_C2PMSG_REG_95 : N/A

4.4 DRTM_CMD_EXTEND_MLE_DIGEST

SKL (x86) <-> DRTM Service (ASP)

Request:

DRTM_C2PMSG_REG_72																													
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	**N/A**			**N/A**				**Command**								**N/A**															

Command (IN) : DRTM_CMD_EXTEND_MLE_DIGEST (0x0B)
R(IN) : DRTM_COMMAND_READY (0X00)
DRTM_C2PMSG_REG_93 : the size of the MLE region to be hashed (bytes)
DRTM_C2PMSG_REG_94 : 32-bit lower physical address of MLE to be hashed
DRTM_C2PMSG_REG_95 : 32-bit higher physical address of MLE to be hashed

Response:

DRTM_C2PMSG_REG_72																													
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	**N/A**			**N/A**				**N/A**								**Status**															

Status (OUT) : One of the valid statuses defined in the "Status[15:0]" field of DRTM_C2PMSG_REG_72 reg
R(OUT) : DRTM_RESPONSE_READY
DRTM_C2PMSG_REG_93 : N/A
DRTM_C2PMSG_REG_94 : N/A
DRTM_C2PMSG_REG_95 : N/A

4.5 DRTM_CMD_GET_TMR_DESCRIPTORS

SKL (x86) <-> DRTM Service (ASP)

Request:

DRTM_C2PMSG_REG_72																													
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	**N/A**			**N/A**				**Command**								**N/A**															

Command (IN) : DRTM_CMD_GET_TMR_DESCRIPTORS (0x09)
R(IN) : DRTM_COMMAND_READY (0x00)
DRTM_C2PMSG_REG_93 : N/A
DRTM_C2PMSG_REG_94 : N/A
DRTM_C2PMSG_REG_95 : N/A

Response:

DRTM_C2PMSG_REG_72																													
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	N/A			N/A				N/A								Status															

Status (OUT) : One of the valid statuses defined in the "Status[15:0]" field of DRTM_C2PMSG_REG_72 reg
R(OUT) : DRTM_RESPONSE_READY
DRTM_C2PMSG_REG_93 : Size response = TMR count * size of "TMR_DESCRIPTOR_RESPONSE" structure (bytes)
DRTM_C2PMSG_REG_94 : 32-bit lower physical address of x86 share region holding TMR Descriptors
DRTM_C2PMSG_REG_95 : 32-bit higher physical address of x86 share region holding TMR Descriptors

```
typedef struct _TMR_DESCRIPTOR_RESPONSE
{
    uint32_t TmrPhyAddrLo; // 32-bit lower physical address of TMR descriptor
    uint32_t TmrPhyAddrHi; // 32-bit higher physical address of TMR descriptor
    uint32_t TmrSize;      // Size of TMR descriptor*
} TMR_DESCRIPTOR_RESPONSE;
```

***Note**: The size of the structure is 12 bytes.*

4.6 DRTM_CMD_ALLOCATE_SHARED_MEMORY

SKL (x86) <-> DRTM Service (ASP)

Request:

DRTM_C2PMSG_REG_72																													
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	N/A			N/A				Command								N/A															

Command (IN) : DRTM_CMD_ALLOCATE_SHARED_MEMORY (0x0A)
R(IN) : DRTM_COMMAND_READY (0x00)
DRTM_C2PMSG_REG_93 : Size of the buffer to be allocated (bytes)
DRTM_C2PMSG_REG_94 : N/A
DRTM_C2PMSG_REG_95 : N/A

Response:

DRTM_C2PMSG_REG_72																													
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	N/A			N/A				N/A								Status															

Status (OUT) : One of the valid statuses defined in the "Status[15:0]" field of DRTM_C2PMSG_REG_72 reg
R(OUT) : DRTM_RESPONSE_READY
DRTM_C2PMSG_REG_93 : Size of the allocated buffer (bytes)
DRTM_C2PMSG_REG_94 : 32-bit lower physical address of x86 share region buffer
DRTM_C2PMSG_REG_95 : 32-bit higher physical address of x86 share region buffer

4.7 DRTM_CMD_GET_TCG_LOGS

`SKL (x86) <-> DRTM Service (ASP)` Refer to *Appendix C* for the data structure of TCG Logs.

Request:

DRTM_C2PMSG_REG_72																															
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	**N/A**			**N/A**				**Command**								**N/A**															

Command (IN) : `DRTM_CMD_GET_TCG_LOGS (0x7)`
R(IN) : `DRTM_COMMAND_READY (0x00)`
DRTM_C2PMSG_REG_93 : N/A
DRTM_C2PMSG_REG_94 : N/A
DRTM_C2PMSG_REG_95 : N/A

Response:

DRTM_C2PMSG_REG_72																															
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	**N/A**			**N/A**				**N/A**								**Status**															

Status (OUT) : One of the valid statuses defined in the "Status[15:0]" field of DRTM_C2PMSG_REG_72 reg
R(OUT) : `DRTM_RESPONSE_READY`
DRTM_C2PMSG_REG_93 : Size of memory region containing TCG Logs (bytes)
DRTM_C2PMSG_REG_94 : 32-bit lower physical address of x86 share region holding TCG Logs
DRTM_C2PMSG_REG_95 : 32-bit upper physical address of x86 share region holding TCG Logs

4.8 DRTM_CMD_TPM_LOCALITY_ACCESS

`SKL (x86) <-> DRTM Service (ASP)`

Request:

DRTM_C2PMSG_REG_72																															
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	**N/A**			**N/A**				**Command**								**N/A**															

Command (IN) : `DRTM_CMD_TPM_LOCALITY_ACCESS (0x8)`
R(IN) : `DRTM_COMMAND_READY (0x00)`
DRTM_C2PMSG_REG_93 : N/A
DRTM_C2PMSG_REG_94 : N/A
DRTM_C2PMSG_REG_95 : N/A

Response:

DRTM_C2PMSG_REG_72																															
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	**N/A**			**N/A**				**N/A**								**Status**															

Status (OUT) : One of the valid statuses defined in the "Status[15:0]" field of DRTM_C2PMSG_REG_72 reg
R(OUT) : `DRTM_RESPONSE_READY`
DRTM_C2PMSG_REG_93 : N/A
DRTM_C2PMSG_REG_94 : N/A
DRTM_C2PMSG_REG_95 : N/A

4.9 DRTM_CMD_TMR_RELEASE

SKL (x86) <-> DRTM Service (ASP)

Request:

DRTM_C2PMSG_REG_72																													
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	N/A			N/A				Command								N/A															

Command (IN) : DRTM_CMD_TMR_RELEASE (0x03)
R(IN) : DRTM_COMMAND_READY (0x00)
DRTM_C2PMSG_REG_93 : N/A
DRTM_C2PMSG_REG_94 : N/A
DRTM_C2PMSG_REG_95 : N/A

Response:

DRTM_C2PMSG_REG_72																													
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	N/A			N/A				N/A								Status															

Status (OUT) : One of the valid statuses defined in the "Status[15:0]" field of DRTM_C2PMSG_REG_72 reg
R(OUT) : DRTM_RESPONSE_READY
DRTM_C2PMSG_REG_93 : N/A
DRTM_C2PMSG_REG_94 : N/A
DRTM_C2PMSG_REG_95 : N/A

4.10 DRTM_GET_IVRS_TABLE_INFO

SKL (x86) <-> DRTM Service (ASP)

Request:

DRTM_C2PMSG_REG_72																													
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	N/A			N/A				Command								N/A															

Command (IN) : DRTM_CMD_GET_IVRS_TABLE_INFO (0xC)
R(IN) : DRTM_COMMAND_READY (0x00)
DRTM_C2PMSG_REG_93 : N/A
DRTM_C2PMSG_REG_94 : N/A
DRTM_C2PMSG_REG_95 : N/A

Response:

DRTM_C2PMSG_REG_72																													
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	N/A			N/A				N/A								Status															

Status (OUT) : One of the valid statuses defined in the "Status[15:0]" field of DRTM_C2PMSG_REG_72 reg
R(OUT) : DRTM_RESPONSE_READY
DRTM_C2PMSG_REG_93 : Size of memory region containing IVRS Table (bytes)
DRTM_C2PMSG_REG_94 : 32-bit lower physical address of x86 share region holding IVRS Table
DRTM_C2PMSG_REG_95 : 32-bit upper physical address of x86 share region holding IVRS Table

Appendix A

A.1 DRTM Command Errors

DRTM Command	Description	Return Code (Release)
DRTM_CMD_GET_CAPABILITY	Mailbox read/write failed	DRTM_GENERIC_ERROR
DRTM_CMD_TMR_SETUP	DRTM Launched already	DRTM_TMR_SETUP_NOT_ALLOWED
	Mailbox read/write failed	DRTM_GENERIC_ERROR
	TMR setup failed	DRTM_TMR_SETUP_FAILED_ERROR
DRTM_CMD_TMR_RELEASE	TMR release command failed	DRTM_TMR_RELEASE_FAILED_ERROR
	Revert Policy Failed	DRTM_LAUNCH_ERROR
DRTM_CMD_LAUNCH	Read SKL address failed	DRTM_GENERIC_ERROR
	Check locality locking failed	DRTM_LAUNCH_ERROR
	Locality locking failed	
	SKL range is disabled	
	SKL is not 64KB aligned	
	SKL within the TMR range failed	
	Shared memory mapping failed	
	SKL size overflow	
	SKL size mismatch	
	SHA384 hash SKL failed	
	SKL hash 256 failed for PCR17	
	SKL signature authentication failed	
	SKL hash failed or updated TCG Log for the same failed	
	PCR17 read operation failed	
	SKL hash not matched with PCR17	
	SPLTVersion hash failed	
	Extend PCR17 failed	
	FAR, TSME hash failed	
	Signing key hash failed	
	Extend PCR18 failed	
	Update TCG Log for SPL failed	
	Locality unlocking failed	
	Apply Sec Pol failed	
	Clear SLB DMA protection bit failed	

DRTM Command	Description	Return Code (Release)
	Check SPL state	
	SPL is enabled and in an untrusted state	
DRTM_CMD_GET_TCG_LOGS	DRTM has not launched yet	DRTM_NOT_SUPPORTED
	Shared memory mapping failed	DRTM_GET_TCG_LOGS_FAILED
DRTM_CMD_TPM_LOCALITY_ACCESS	DRTM has not launched yet	DRTM_NOT_SUPPORTED
	Locality locking failed	DRTM_GENERIC_ERROR
DRTM_CMD_GET_TMR_DESCRIPTORS	DRTM has not launched yet	DRTM_NOT_SUPPORTED
	Shared memory mapping failed	DRTM_GET_TMR_DESCRIPTOR_FAILED
DRTM_CMD_ALLOCATE_SHARED_MEMORY	DRTM has not launched yet	DRTM_OUT_OF_RESOURCES_ERROR
	Mailbox read/write failed	DRTM_GENERIC_ERROR
DRTM_CMD_EXTEND_MLE_DIGEST	DRTM has not launched yet	DRTM_NOT_SUPPORTED
	Shared memory mapping failed	DRTM_SHARED_MEMORY_MAPPING_FAILED
	Generic error for extending MLE	DRTM_EXTEND_MLE_DIGEST_FAILED
	MLE is not TMR is not protected	
	MLE digest authentication failed	
	Extend PCR17 failed	
	Extend PCR18 failed	
	MLE in the TMR range failed	
DRTM_CMD_GET_IVRS_TABLE_INFO	Shared memory mapping failed	DRTM_GET_IVRS_TABLE_FAILED
	Invalid IVRS table size	
	Invalid IVRS table buffer information	

Appendix B

B.1 SKL Signature Header

```
typedef struct tdFW_SIG_HEADER
{
    uint8_t     Nonce[16];          // [0x00] Unique image id
    uint32_t    HeaderVersion;      // [0x10] Version of the header
    uint32_t    SizeFWSigned;       // [0x14] Signed Fw Size in bytes
    uint32_t    EncOption;          // [0x18] 0 - Not encrypted, 1 - encrypted
    uint32_t    EncAlgID;           // [0x1C] Encryption algorithm id
    uint8_t     EncParameters[16];  // [0x20] Encryption Parameters
    uint32_t    SigOption;          // [0x30] 0 - not signed 1 - signed
    uint32_t    SigAlgID;           // [0x34] Signature algorithm ID
    uint8_t     SigParameters[16];  // [0x38] Signature parameter
    uint32_t    CompOption;         // [0x48] Compression option
    uint32_t    SecPatchLevel;      // [0x4C] Security patch level
    uint32_t    UnCompImageSize;    // [0x50] Uncompressed Image Size
    uint32_t    CompImageSize;      // [0x54] compressed Image Size
    uint8_t     CompParameters[8];  // [0x58] Compression Parameters
    uint32_t    ImageVersion;       // [0x60] Off Chip Firmware Version
    uint32_t    APUFamilyID;        // [0x64] APU Family ID or SoC ID
    uint32_t    FirmwareLoadAddr;   // [0x68] Firmware Load address (default 0)
    uint32_t    SizeImage;          // [0x6C] FW size with signature
    uint32_t    SizeFWUnSigned;     // [0x70] Size of Un-signed portion of the FW
    uint32_t    FirmwareSplitAddr;  // [0x74] Offset of Nwd OS
    uint32_t    SigFlags;           // [0x78] Flags for FW signing options, perm, etc
    uint8_t     FwType;             // [0x7C] FwType
    uint8_t     SubType;            // [0x7D] SubType identifies FW
    uint8_t     Reserved1[2];       // [0x7E] *** RESERVED ***
    uint8_t     EncKey[16];         // [0x80] Encryption Key (Wrapped MEK)
    uint8_t     SigningInfo[16];    // [0x90] Signing tool specific information
    uint8_t     Padd[96];           // [0xA0] *** RESERVED ***

} FW_SIG_HEADER; // Total 256 bytes
```

B.2 SKL Public Key Token Image

```
typedef struct KEY_HEADER {
    uint32_t  VersionID;                // Version ID
    uint8_t   KeyID[16];                // Key ID
    uint8_t   CertKeyID[16];            // Certifying key ID
    uint32_t  KeyUsageFlag;             // Key usage flag
    uint8_t   PlatformVendorID;         // Platform Vendor ID
    uint8_t   PlatformModelAndKeyRevID; // Platform Model ID [7-4], BIOS Key ID [3-0]
    uint8_t   Reserved[14];             // Reserved
    uint32_t  PublicExponentSize;       // Public exponent size in bits
    uint32_t  ModulusSize;              // Modulus size in bits
} KEY_HEADER;

typedef struct {
     KEY_HEADER  Header;                        // The common part of all keys
     uint8_t         PublicExponent[256];   // Public exponent little-endian
     uint8_t         Modulus[256];          // Modulus of the ASP little-endian
} KEY_IMAGE_2048;

typedef struct {
     KEY_HEADER  Header;                        // The common part of all keys
     uint8_t         PublicExponent[512];   // Public exponent little-endian
     uint8_t         Modulus[512];          // Modulus of the ASP little-endian
} KEY_IMAGE_4096;
```

The ASP uses Public Exponent Size, Modulus Size, Public Exponent, and Modulus to verify the SKL signature. The other parameters are not used for other applications.

B.3 SKL Header

ASP does not have a dependency on the AMD_SL_HDR_V1_EXT portion of the AMD_SL_HDR_V1 structure.

```
typedef struct
{
  GUID   Guid;                      //GUID of AMDSL  (16 Bytes)
  uint32_t Reserved;
  uint32_t TotalLength;             //Total Length include Signature,Pubkey Raw data,
Os BL need to set the OSSL parameter at SLB + TotalLength
} AMD_SL_HDR_V1_EXT;

typedef struct
{
  uint16_t EntryPointOffset;        //Offset point to the Entry of SL, required by APM
V2
```

```
  uint16_t  AmdSlLength;                  //Size of SL static data, required by APM V2,
SKINIT instruction will use this field to calculate the HASH of SL in order to
authenticate the SL.
  AMD_SL_HDR_V1_EXT ExtHdr;               //Extended Header
  uint16_t SocFlagStructOffset;           //Offset of SOCFLAG Structure from start of
AMD_SL_HDR_V1
} AMD_SL_HDR_V1;
```

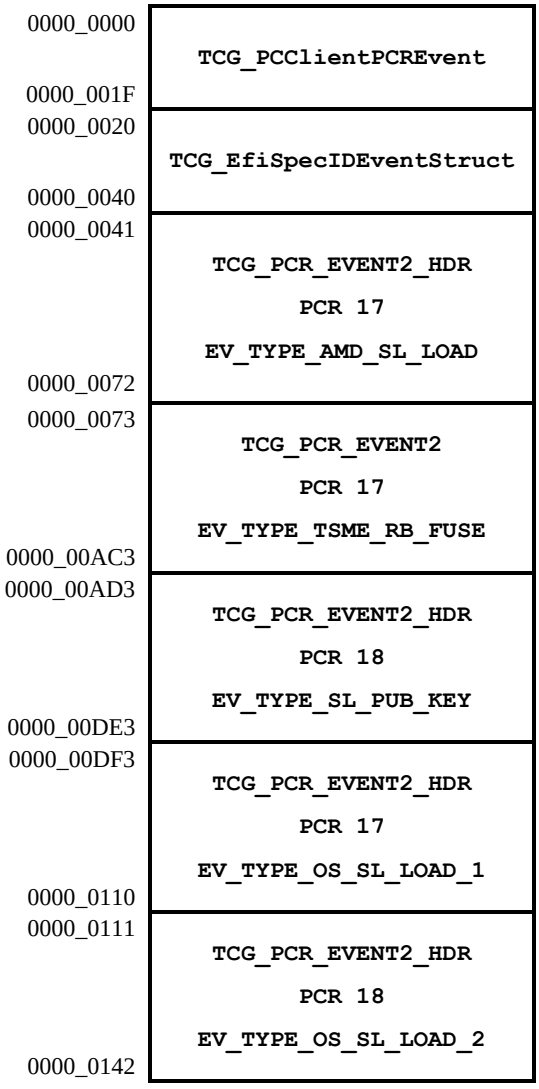
B.4 SKL Size Limitation

SKL has a 64 K bytes limitation. The SKL executable can be less than 64K bytes, but the image needs to be padded to 64K bytes.

Appendix C

C.1 TCG Log Format

- The **TCG Logs** are a collection of all executed **PCR Events**.
- A **PCR Event** is an incident that occurs when any given PCR value is changed, extended, or reset.
- Please refer to the **PCR Event Log Type** table (*Appendix D*) for a list of DRTM-specific events.



Appendix D

D.1 PCR Event Log Type

Event Type	Value	Digest and Event Data	PCR	Comments
EV_TYPE_BASE (EB)	0x8000	N/A	N/A	Base value for Event type
EV_TYPE_SL_LOAD	0x8001	*Digest = SHA_256(SKL Image) EventDataSize = 0 EventData = Empty Buffer*	17	SKINIT: ucode loading SKL
EV_TYPE_AMD_ASP_FW_SPLT	0x8002	*Digest = SHA_256(SPLTVersion) EventDataSize = 4 EventData = SPLTVersion*	17	Security Patch Level Table version
EV_TYPE_TSME_RB_FUSE	0x8003	*Digest = SHA_256(RB_FUSE || TSME) EventDataSize = 8 EventData = RB_FUSE || TSME*	17	Event extended by the DRTM service ***Note:*** *|| -> concatenation RB_FUSE -> 32-bit Anti-Rollback Fuse state TSME -> 32-bit where bit 0 represents enable state*
EV_TYPE_SL_PUB_KEY	0x8004	*Digest = SHA_256(SKL Public key token) EventDataSize = 0 EventData = Empty Buffer*	18	Event extended by DRTM service
EV_TYPE_SL_SVN	0x8005	*Digest = SHA_256(SPLVersion) EventDataSize = 4 EventData = SPLVersion*	18	SKL Security Patch Level version
EV_TYPE_OS_SL_LOAD_1	0x8006	*Digest = SHA_256(OS SL Image) EventDataSize = 0 EventData = Empty Buffer*	17/18	Event extended by SKL
EV_TYPE_AMD_SL_SEPARATOR	0x8007	*Digest = SHA_256(Marker) EventDataSize = 5 EventData = "SKL"*	17/18	Marker Event to signify the end of SKL events in TCG Logs

D.2 PCR Event Structure Definition

```
typedef UINT32                TCG_PCRINDEX;
typedef UINT32                TCG_EVENTTYPE;
typedef UINT16                TPM_ALG_ID;
typedef TPM_ALG_ID               TPMI_ALG_HASH;

#define MAX_PCR_COUNT            2
#define MAX_EVENT_COUNT          3
#define MAX_DIGEST_COUNT            1

#define TPM_ALG_SHA1                    (TPM_ALG_ID)(0x0004)
#define TPM_ALG_SHA256                  (TPM_ALG_ID)(0x000B)
#define TPM_ALG_SHA384                  (TPM_ALG_ID)(0x000C)
#define TPM_ALG_SHA512                  (TPM_ALG_ID)(0x000D)

#define EV_NO_ACTION             ((TCG_EVENTTYPE)0x00000003)
#define EV_TYPE_BASE             ((TCG_EVENTTYPE)0x80000000)
#define EV_TYPE_SL_LOAD             (EV_TYPE_BASE + 1)
#define EV_TYPE_TSME_RB_FUSE                (EV_TYPE_BASE + 2)
#define EV_TYPE_SL_PUB_KEY              (EV_TYPE_BASE + 3)
#define EV_TYPE_OS_SL_LOAD_1                (EV_TYPE_BASE + 4)

///
/// SHA-1 digest size in bytes.
///
#define SHA1_DIGEST_SIZE            20

///
/// SHA-256 digest size in bytes
///
#define SHA256_DIGEST_SIZE          32

///
/// SHA-384 digest size in bytes
///
#define SHA384_DIGEST_SIZE          48

///
/// SHA-512 digest size in bytes
///
#define SHA512_DIGEST_SIZE          64
```

```
// Table 209 – Defines for SM3_256 Hash Values
#define SM3_256_DIGEST_SIZE          32

// Table 66 – TPMU_HA Union
typedef union
{
  BYTE                      sha1[SHA1_DIGEST_SIZE];
  BYTE                      sha256[SHA256_DIGEST_SIZE];
  BYTE                      sm3_256[SM3_256_DIGEST_SIZE];
  BYTE                      sha384[SHA384_DIGEST_SIZE];
  BYTE                      sha512[SHA512_DIGEST_SIZE];
} TPMU_HA;

typedef struct
{
    TPMI_ALG_HASH      hashAlg;            // ID of hashing algorithm
    TPMU_HA                digest;      // Digest value in little-endian
} TPMT_HA;

typedef struct tdTPML_DIGEST_VALUES
{
   UINT32              count;          // number of digests in one event
   TPMT_HA          digests[MAX_DIGEST_COUNT];  // Array of digests
} TPML_DIGEST_VALUES;

//
// Crypto Agile Log Entry Format
//
typedef struct tdTCG_PCR_EVENT2
{
   TCG_PCRINDEX            PCRIndex;
   TCG_EVENTTYPE           EventType;
   TPML_DIGEST_VALUES      Digest;
   UINT32                  EventSize;
   UINT8                   Event[1];
} TCG_PCR_EVENT2;

//
// TCG PCR Event2 Header
// Follow TCG EFI Protocol Spec 5.2 Crypto Agile Log Entry Format
//
typedef struct tdTCG_PCR_EVENT2_HDR
{
```

```
    TCG_PCRINDEX            PCRIndex;
    TCG_EVENTTYPE           EventType;
    TPML_DIGEST_VALUES      Digests;
    UINT32                  EventSize;
} TCG_PCR_EVENT2_HDR;

//
// Log Header Entry Data
//
typedef struct
{
    //
    // TCG defined hashing algorithm ID.
    //
    UINT16                  algorithmId;
    //
    // The size of the digest for the respective hashing algorithm.
    //
    UINT16                  digestSize;
} TCG_EfiSpecIdEventAlgorithmSize;

#define TCG_EfiSpecIDEventStruct_SIGNATURE_03 "Spec ID Event03"

#define TCG_EfiSpecIDEventStruct_SPEC_VERSION_MAJOR_TPM2   2
#define TCG_EfiSpecIDEventStruct_SPEC_VERSION_MINOR_TPM2   0
#define TCG_EfiSpecIDEventStruct_SPEC_ERRATA_TPM2          0

typedef struct
{
    UINT8               signature[16];
    //
    // The value for the Platform Class.
    // The enumeration is defined in the TCG ACPI Specification Client Common Header.
    UINT32              platformClass;
    //
    // The TCG EFI Platform Specification minor version number this BIOS supports.
    // Any BIOS supporting version (1.22) MUST set this value to 02h.
    // Any BIOS supporting version (2.0) SHALL set this value to 0x00.
    UINT8               specVersionMinor;
    //
    // The TCG EFI Platform Specification major version number this BIOS supports.
    // Any BIOS supporting version (1.22) MUST set this value to 01h.
    // Any BIOS supporting version (2.0) SHALL set this value to 0x02.
```

```
    UINT8               specVersionMajor;
    //
    // The TCG EFI Platform Specification errata for this specification this BIOS
supports.
    // Any BIOS supporting version and errata (1.22) MUST set this value to 02h.
    // Any BIOS supporting version and errata (2.0) SHALL set this value to 0x00.
    //
    UINT8               specErrata;
    //
    // Specifies the size of the UINTN fields used in various data structures used in
this specification.
    // 0x01 indicates UINT32 and 0x02 indicates UINT64.
    //
    UINT8               uintnSize;
    //
    // This field is added in "Spec ID Event03".
    // The number of hashing algorithms used in this event log (except the first event).
    // All events in this event log use all hashing algorithms defined here.
    //
    UINT32              numberOfAlgorithms;
    //
    // This field is added in "Spec ID Event03".
    // An array of size numberOfAlgorithms of value pairs.
    //
    TCG_EfiSpecIdEventAlgorithmSize digestSize[MAX_DIGEST_COUNT];
    //
    // Size in bytes of the VendorInfo field.
    // Maximum value SHALL be FFh bytes.
    //
    UINT8               vendorInfoSize;
    //
    // Provided for use by the BIOS implementer.
    // The value might be used, for example, to provide more detailed information about
the specific BIOS, such as BIOS revision numbers, etc.
    // The values within this field are not standardized and are implementer-specific.
    // Platform-specific or -unique information SHALL NOT be provided in this field.
    //
    //UINT8               vendorInfo[4];
} TCG_EfiSpecIDEventStruct;

typedef struct tdTCG_PCClientPCREvent
{
    UINT32          pcrIndex;
```

```
    UINT32          eventType;
    BYTE          digest[20];
    UINT32        eventDataSize;
    BYTE          event[33]; // sizeof (TCG_EfiSpecIDEventStruct)
} TCG_PCClientPCREvent;
```

Appendix E

E.1 MLE Size and Limitations

The MLE needs to be 64M bytes aligned in memory and limited to 64 M bytes.

Appendix F

F.1 Locality Usage and Restrictions

Locality	Accessible To
0	Non-trusted software/firmware
1	Trusted Operating System/Hypervisor software
2	SKL, MLE
3	PSP DRTM Service
4	SKINIT Instruction

No software will be allowed to access Locality 4. The Locking and Unlocking of Locality 4 translate into access privileges for SKINIT instruction to TPM HASH START, TPM HASH END, and TPM DATA at Locality 4.

Appendix G Milan/Genoa Platform Support

G.1 Signing and Verification Parameters

Hash	SHA-384
Algorithm	RSA-PSS 4096
Mask Generation Function	PKCS1_MGF1
Trailer Field	0xBC (Standard)
Salt Length	48 bytes. (hLen, Hash Length of SHA-384)
Modulus	512 bytes, Little Endian from Public Key Token Image, Appendix B.
Modulus Size	Size is given in bits from Public Key Token Token Image, Appendix B.
Exponents	512 bytes, Little Endian from Public Key Token Image, Appendix B.
Exponent Size	Size is given in bits from Public Key Token Token Image, Appendix B.

G.2 Platform Addresses

The C2PMSG mailboxes are considered non-PCI resources outside of the PCI enumeration process. This address range is reserved and should not be assigned by the PCI enumeration process.

All these addresses should be available in the PPR (Processor Programmer Reference)

Name	SMN Address
NBCFG0x00000000 (IOHC)	0x13B00000
PSP_BASE_ADDR_LO	(NBCFG0x00000000 + 0x102E0) = 13B102E0
PSP_BASE_ADDR_HI	(NBCFG0x00000000 + 0x102E4) = 13B102E4

After reading the SMN address for PSP_BASE_ADDR_LO and PSP_BASE_ADDR_HI, extract the MMIO base from this map.

IOHCMISC[0...3]x000002E0 (IOHC::PSP_BASE_ADDR_LO)	
Read-write. Reset: 0000_0000h.	
PSP MMIO base address	
_nbio[3:0]_aliasSMN; IOHCMISC[3:0]x000002E0; IOHCMISC[3:0]=13[E:B]1_0000h	
Bits	**Description** JIRA
31:20	**PSP_BASE_ADDR_LO**. Read-write. Reset: 000h. PSP private MMIO base address bits 31:20.
19:9	Reserved.
8	**PSP_MMIO_LOCK**. Read-write. Reset: 0. Locks the PSP private MMIO address range and enable until the next warm reset.
7:1	Reserved.
0	**PSP_MMIO_EN**. Read-write. Reset: 0. PSP private MMIO enable.

IOHCMISC[0...3]x000002E4 (IOHC::PSP_BASE_ADDR_HI)	
Read-write. Reset: 0000_0000h.	
PSP private MMIO base address.	
_nbio[3:0]_aliasSMN; IOHCMISC[3:0]x000002E4; IOHCMISC[3:0]=13[E:B]1_0000h	
Bits	**Description** JIRA
31:16	Reserved.
15:0	**PSP_BASE_ADDR_HI**. Read-write. Reset: 0000h. PSP private MMIO base address bits 47:32.

PSP_BASE_ADDR = (PSP_BASE_ADDR_HI[15:0] << 32) | PSP_BASE_ADDR_LO [31:20]

Mailbox	PPR Nomenclature	Offset from PSP_BASE_ADDR
C2PMSG_72	MP::MP0_C2PMSG_72	0x10A20
C2PMSG_93	MP::MP0_C2PMSG_93	0x10A74
C2PMSG_94	MP::MP0_C2PMSG_94	0x10A78
C2PMSG_95	MP::MP0_C2PMSG_95	0x10A7C

Appendix H Implementation Differences across the Datacenter Product Portfolio

H.1 SMM Supervisor

SMM Supervisor is not supported in Server Implementation.

The SMM Supervisor Signed image is not installed and authenticated. The SMM Supervisor in another system is responsible for preventing the SMI Handler from accessing system resources owned by the OS/Hypervisor. The pre-SKINIT software installs the SMM Supervisor at Boot time, and the DRTM service authenticates it.

H.2 ABI

List of DRTM Commands Not Supported in Sever

Command	Command ID
DRTM_CMD_PROCESS_SMM	0x5
DRTM_CMD_GET_SMM_POLICY	0x6
DRTM_CMD_SMM_DISABLE	0xD
DRTM_CMD_SMM_ENABLE	0xE

The Server's DRTM sequence terminates with these two commands: DRTM_CMD_RELEASE (Command ID = 3) and DRTM_CMD_TPMM_LOCALITY_ACCESS (Command ID=8), there is no order restriction for these two commands. Unless the DRTM sequence is restarted, all the commands that depend on DRTM_LAUNCH are not processed.