Guest Hypervisor Communication Interface (GHCI) for Intel® Trust Domain Extensions (Intel® TDX) 1.0

344426-005US

December 2022

Disclaimers

Intel Corporation ("Intel") provides these materials as-is, with no express or implied warranties.

All products, dates, and figures specified are preliminary, based on current expectations, and are subject to change without notice. Intel does not guarantee the availability of these interfaces in any future product. Contact your Intel representative to obtain the latest Intel product specifications and roadmaps.

The products described might contain design defects or errors known as errata, which might cause the product to deviate from published specifications. Current, characterized errata are available on request.

Intel technologies might require enabled hardware, software, or service activation. Some results have been estimated or simulated. Your costs and results might vary.

No product or component can be absolutely secure.

You may not use or facilitate the use of this document in connection with any infringement or other legal analysis concerning Intel products described herein. You agree to grant Intel a non-exclusive, royalty-free license to any patent claim thereafter drafted that includes the subject matter disclosed herein.

No license (express, implied, by estoppel, or otherwise) to any intellectual-property rights is granted by this document.

This document contains information on products, services and/or processes in development. All information provided here is subject to change without notice.

Copies of documents that have an order number and are referenced in this document or other Intel literature may be obtained by calling 1-800-548-4725 or by visiting http://www.intel.com/design/literature.htm.

Table of Contents

Disclaimers i

1 About this Document 2

1.1 Scope of this Document 2

1.2 Document Organization 2

1.3 Glossary 3

1.4 References 4

2 TD-VMM Communication 6

2.1 Recap of Intel® Trust Domain Extensions (Intel® TDX) 6

2.2 TD-VMM-Communication Overview 7

2.3 Virtualization Exception (#VE) 8

2.4 TDCALL and SEAMCALL instruction 9

2.4.1 TDCALL [TDG.VP.VMCALL] leaf 9

3 TDG.VP.VMCALL Interface 13

3.1 TDG.VP.VMCALL<GetTdVmCallInfo> 13

3.2 TDG.VP.VMCALL<MapGPA> 14

3.3 TDG.VP.VMCALL<GetQuote> 16

3.4 TDG.VP.VMCALL<ReportFatalError> 19

3.5 TDG.VP.VMCALL<SetupEventNotifyInterrupt> 21

3.6 TDG.VP.VMCALL<Instruction.CPUID> 21

3.7 TDG.VP.VMCALL<#VE.RequestMMIO> 22

3.8 TDG.VP.VMCALL<Instruction.HLT> 24

3.9 TDG.VP.VMCALL<Instruction.IO> 25

3.10 TDG.VP.VMCALL<Instruction.RDMSR> 25

3.11 TDG.VP.VMCALL<Instruction.WRMSR> 26

3.12 TDG.VP.VMCALL<Instruction.WBINVD> 27

3.13 TDG.VP.VMCALL<Instruction.PCONFIG> 27

4 TD-Guest-Firmware Interfaces 29

4.1 ACPI MADT Multiprocessor Wakeup Table 29

4.2 Memory Map 29

4.3 TD Measurement 32

4.3.1 TCG-Platform-Event Log 32

4.3.2 EFI_CC_MEASUREMENT_PROTOCOL 32

4.3.3 CC-Event Log 32
4.4 STORAGE-VOLUME-KEY DATA 32
5 TD-VMM-Communication Scenarios 34
5.1 REQUESTING IPIS 34
5.2 TD-MEMORY CONVERSION AND MEMORY BALLOONING 34
5.3 PARAVIRTUALIZED IO 35
5.4 TD ATTESTATION 35

1 About this Document

1.1 Scope of this Document

Trust Domains (TDs) are used to enable confidential hosting of VM workloads that are hardware-isolated from the hosting VMM and service OS environments. The Intel® Trust Domain Extensions (Intel® TDX) architecture enables isolation of the TD-CPU context and memory from the hosting environment.

This document specifies the guest (TD) to host (VMM) communication interface that will be utilized for the paravirtualization interface between the TD and the VMM. This approach helps the Intel TDX-architecture prevent the VMM from accessing any TD runtime state. The TD must therefore volunteer information to access IO services, enumerate model-specific CPU capabilities and measurement services, and provide feedback to the VMM on guest-OS-triggered actions such as virtual-IPIs or shutdowns.

For each operation within this interface, the recommended actions are described for the host VMM (informative). The TD and the VMM are designed to use the subfunctions, which are normative and described in this document.

This document is a work in progress and is subject to change based on customer feedback and internal analysis. This document does not imply any product commitment from Intel to anything in terms of features and/or behaviors.

1.2 Document Organization

Section 2 describes a general structure/ABI of the instruction TDCALL with the TDG.VP.VMCALL leaf used for passing information to the VMM and receiving information from the VMM.

Section 3 describes the sub-leaves of TDCALL [TDG.VP.VMCALL] that define the Application Binary Interface (ABI) between the TD and the VMM for specific operations.

Section 4 describes example flows for the main scenarios.

Section 5 describes TD-VMM-Communication Scenarios.

1.3 Glossary

Table 1-1: Intel TDX Glossary

Acronym	Full Name	Description
TME	**Total Memory Encryption**	An SoC memory encryption/decryption engine used to encrypt memory contents exposed externally from the SoC using an ephemeral, platform key. Memory is decrypted using the TME when memory contents are brought into the CPU caches.
MKTME	**Multi-Key TME**	This SoC capability adds support to the TME to allow software to use separate (one or more) keys for encryption of volatile- or persistent-memory encryption. When used with Intel TDX, it can provide confidentiality via separate keys for memory-used TDs. MKTME may be used with and without Intel TDX extensions.[1]
TD	**Trust Domain**	Software operating in a CPU mode designed to exclude the host/VMM software and untrusted, platform devices from the operational TCB for confidentiality. The operational TCB for a TD includes the CPU, the TD OS, and TD applications. A TD's resources are managed by an Intel TDX-aware host VMM, but its state protection is managed by the CPU and is not accessible to the host software.
Intel TDX	**Intel TDX Architecture**	Intel-CPU-instruction-set-architecture extensions to enable host VMM to host Trust Domains.
Intel TDX module	**Intel Trust Domain Extensions module**	Intel TDX module is a CPU-measured software module that uses the instruction-set architecture for Intel TDX to help enforce security properties for hosting TDs on an Intel TDX platform. Intel TDX module exposes the Guest-Host-Communication Interface that TDs use to communicate with the Intel TDX module and the host VMM.
HKID	**Host Key ID**	When MKTME is activated, HKID is a key identifier for an encryption key used by one or more memory controllers on the platform. When Intel TDX is active, the HKID space can be partitioned into a CPU-enforced space (for TDs) and a VMM-enforced space (for legacy VMs).
-	**TD-Private Memory (Access)**	TD-Private Memory can be encrypted by the CPU using the TD-ephemeral key (or in the future with additional, TD private keys).
-	**TD-Shared Memory (Access)**	TD-Shared Memory is designed to be accessible by the TD and the host software (and/or other TDs). TD-Shared Memory uses MKTME keys managed by the VMM for encryption.
TDVPS	**TD Virtual Processor Structure**	TD per-VCPU state maintained in protected memory by the Intel TDX.

[1] In this document, the term "MKTME" means both the feature and the encryption engine itself.

Acronym	Full Name	Description
TDCS	**Trust Domain Control Structure**	Multi-page-control structure for a TD. By design, TDCS is encrypted with the TD's ephemeral, private key, its contents are not architectural, and its location in memory is known to the VMM.

1.4 References

Table 1-2: Technical Documents Referenced

#	Reference Document	Version & Date
1	Intel® 64 and IA-32 Architecture Software Developer Manual	May 2020
2	Intel® Trust Domain Extensions CPU architecture specification	May 2021
3	Intel® Trust Domain Extensions module 1.0 specification	June 2022
4	Intel® Multi-key Total Memory Encryption (MK-TME) specification	April 2021
5	ACPI specification, version 6.5	August 2022
6	UEFI specification, version 2.10	August 2022

When specifying requirements or definitions, the level of commitment is specified following the convention of RFC 2119: Key words for use in RFCs to indicate Requirement Levels, as described in the following table:

Table 1-3: Requirement and Definition Commitment Levels

Keyword	Description
Must	"Must," "Required," or "Shall" means that the definition is an absolute requirement of the specification.
Must Not	"Must Not" or "Shall Not" means that the definition is an absolute prohibition of the specification.
Should	"Should" or "Recommended" means that there may exist valid reasons in particular circumstances to ignore a particular item, but the full implications must be understood and carefully weighed before choosing a different course.
Should Not	"Should Not" or the phrase "Not Recommended" means that there may exist valid reasons in particular circumstances when the particular behavior is acceptable or even useful, but the full implications should be understood and the case must be carefully weighed before implementing any behavior described with this label.
May	"May" or "Optional" means that an item is discretionary. An implementation may choose to include the item, while another may omit the same item because of various reasons.

2 TD-VMM Communication

2.1 Recap of Intel® Trust Domain Extensions (Intel® TDX)

Intel® Trust Domain Extensions (Intel® TDX) is an Intel technology that extends Virtual Machines Extensions (VMX) and Multi-Key Total Memory Encryption (MKTME) with a new kind of virtual machine guest called **Trust Domain (TD)**. A TD is designed to run in a CPU mode that protects the confidentiality of TD memory contents and the TD's CPU state from other software, including the hosting Virtual-Machine Monitor (VMM), unless explicitly shared by the TD itself.

The **Intel TDX module** uses the instruction-set architecture for Intel TDX and the MKTME engine in the SOC to help serve as an intermediary between the host VMM and the guest TDs. The operation of this module is described in detail in the Intel TDX-module specification [3]. The Intel TDX module exposes the **Guest-Host-Communication Interface (GHCI) for Intel TDX** (this specification) that TDs must use to communicate with both the Intel TDX module and the host VMM.

As shown in the diagram below, an Intel TDX-aware **host VMM** can launch and manage both guest TDs and legacy-guest VMs. The host VMM can maintain legacy functionality from the legacy VMs' perspective; the aim is for the host VMM to be restricted only with regard to the TDs it manages.

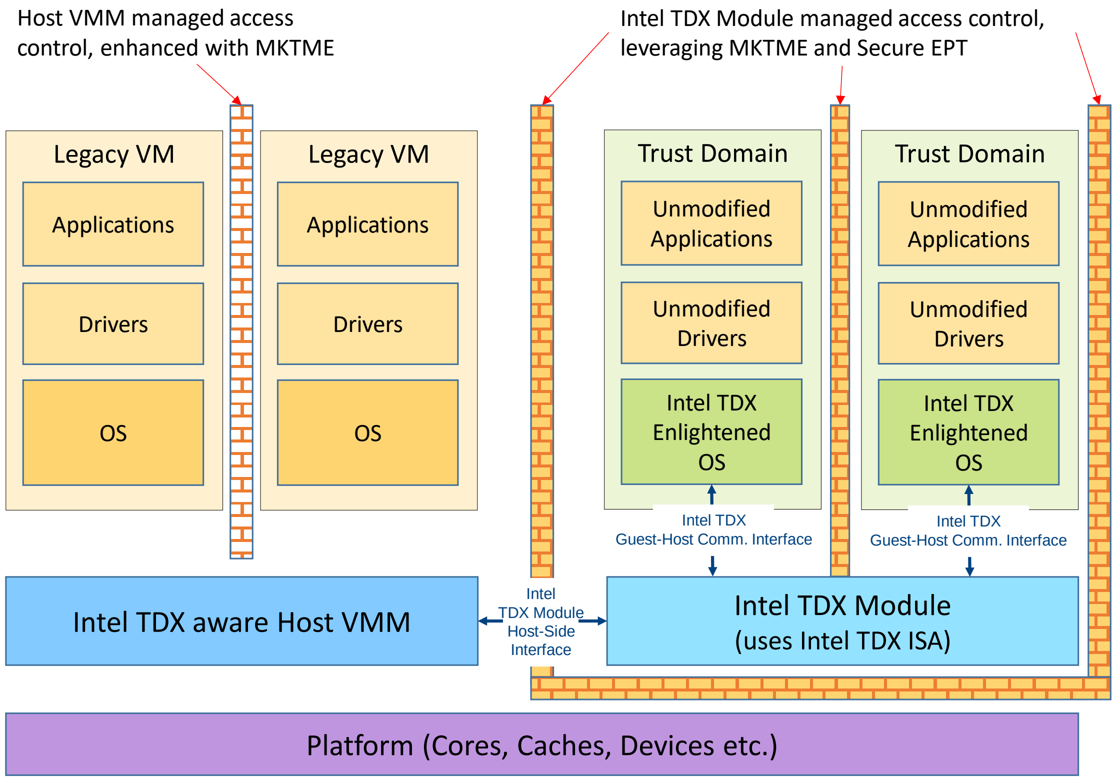

Figure 2-1: Components of Intel® Trust Domain Extensions

2.2 TD-VMM-Communication Overview

TD-VMM communication can occur via either asynchronous or synchronous (instruction) VM exits. In response to synchronous (instruction) VM exits, Intel TDX [3] is designed to generate a Virtualization Exception (#VE) [1] for instructions the TD would be disallowed to invoke. The TD-guest software may respond by using the Intel TDX-provided information directly and/or after further decoding of the instruction that caused the #VE.

The TD response must occur via a TDCALL instruction [2] requesting that the host VMM provide (untrusted) services. Ultimately, the VMM's goal is to 1) receive the service request via a SEAMRET invoked by the Intel TDX module, 2) complete the service requested, and 3) respond to the TD via SEAMCALL[TDH.VP.ENTER] to re-enter the TD.

This document describes the mechanisms and ABI for this interaction in various expected scenarios.

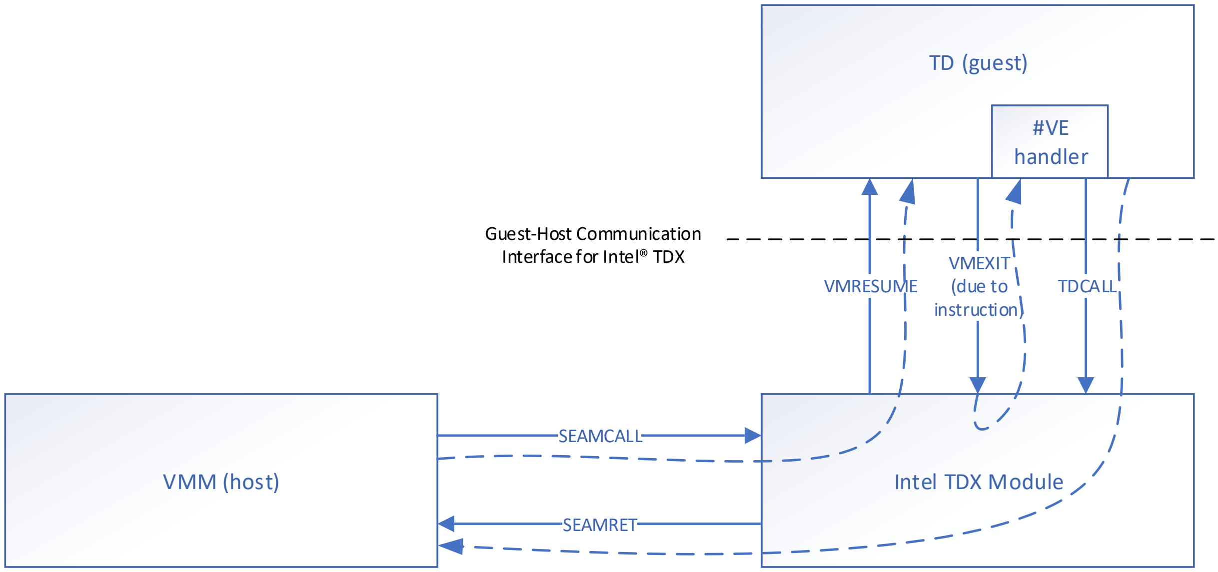

Figure 2-2: TD Guest-Host communication

Section 2 of this document describes the Virtualization Exception (#VE) for Intel TDX, and subsequent sections describe the normative TDCALL leaves intended to get the VE information and request services from the host VMM. Other scenarios may cause asynchronous VM exits to the host VMM (via SEAMRET); for those scenarios, please refer to the Intel TDX module specification [3].

Section 3 of this document describes the reference/informative TDCALL[TDG.VP.VMCALL] interface sub-leaves intended to request services from the host VMM.

Section 4 describes the scenarios where TD-VMM communication interfaces described in this specification can be applied.

2.3 Virtualization Exception (#VE)

Intel TDX can cause #VE to be reported to the guest-TD software in cases of disallowed instruction execution, such as IO accesses.

The goal is for the Intel TDX module, in handling #VE delivery, to follow the architectural #VE handling for nested #VE, as described in Intel-SDM Chapter 25.5.6.3 (Delivery of Virtualization Exceptions). The TD OS should avoid instructions that may cause #VE in the #VE handler.

For detailed information about virtualization exception in Intel TDX, please refer to Intel TDX module Architecture specification.

2.4 TDCALL and SEAMCALL instruction

TDCALL is the instruction used by the guest TD software (in TDX non-root mode) to invoke guest-side TDX functions. For detailed information about the TDCALL instruction, please refer to the Intel TDX module Architecture specification.

SEAMCALL is the instruction used by the host VMM to invoke host-side TDX functions. For detailed information about the SEAMCALL instruction, please refer to the Intel TDX module Architecture specification.

2.4.1 TDCALL [TDG.VP.VMCALL] leaf

TDG.VP.VMCALL is a leaf function 0 for TDCALL. It helps invoke services from the host VMM. The input operands for this leaf are programmed as defined below:

Table 2-1: TDG.VP.VMCALL-Input Operands

Operand	Description
RAX	TDCALL instruction leaf number per Intel TDX module Specification (0 - TDG.VP.VMCALL).
RCX	A bitmap that controls which part of the guest TD GPR and XMM state is passed as-is to the VMM and back. Please refer to Intel TDX module Specification TDG.VP.VMCALL.
R10	Set to 0 indicates that TDG.VP.VMCALL leaf used in R11 is defined in this specification. All other values 0x1 to 0xFFFFFFFFFFFFFFFF indicate TDG.VP.VMCALL is vendor-specific (both R10 and R11).
R11	TDG.VP.VMCALL sub-function if R10 is 0 (see enumeration below)
RBX, RBP, RDI, RSI, R8-R10, R12-R15	See each TDG.VP.VMCALL sub-function for which registers must be used to pass values to the VMM (by setting RCX bits specified above).

Table 2-2: TDG.VP.VMCALL-Output Operands

Operand	Description
RAX	TDCALL instruction return code. Always returns Intel TDX_SUCCESS (0).
RCX	Unmodified.
R10	**TDG.VP.VMCALL sub-function return value. See table 2-6.** 0 – if no error. Non 0 – if error happens. The error code is command specific.
R11	**See each TDG.VP.VMCALL sub-function.**
R12, R13, R14, R15, RBX, RDI, RSI, R8, R9, RDX	**See each TDG.VP.VMCALL sub-function. Register used in order.**
XMM0–XMM15	If the corresponding bit in RCX is set to 1, the register value passed as-is from the host VMM's SEAMCALL (TDH.VP.ENTER) input. Otherwise, the register value is unmodified.

TDG.VP.VMCALL-Intel TDX paravirtualization sub-functions (specified in R11 when R10 is set to 0)

Table 2-3: TDG.VP.VMCALL codes

Sub-Function Number	Sub-Function Name
0x10000	GetTdVmCallInfo
0x10001	MapGPA
0x10002	GetQuote, *e.g.*, used for sending TDREPORT_STRUCT to VMM to request a TD Quote
0x10003	ReportFatalError
0x10004	SetupEventNotifyInterrupt

Table 2-4: TDG.VP.VMCALL-Instruction-execution sub-functions

Sub-Function Number Bits 15:0	Sub-Function Name
10	Instruction.CPUID
12	Instruction.HLT
30	Instruction.IO
31	Instruction.RDMSR
32	Instruction.WRMSR
48	#VE.RequestMMIO
54	Instruction.WBINVD
65	Instruction.PCONFIG

Completion-Status Codes

Table 2-5: TDCALL[TDG.VP.VMCALL]-Completion-Status Codes (Returned in RAX)

Completion-Status Code	Value	Description
TDX_SUCCESS	See Intel TDX-Architecture specification [3] for Function Completion Status Code.	TDCALL is successful
TDX_OPERAND_INVALID		Illegal leaf number
Other	See individual leaf functions.	

Table 2-6: TDCALL[TDG.VP.VMCALL]- Sub-function Completion-Status Codes (specified in R10 as output when R10 is set to 0 as input)

Completion-Status Code	Value	Description
TDG.VP.VMCALL_SUCCESS	0x0	TDCALL[TDG.VP.VMCALL] sub-function invocation was successful
TDG.VP.VMCALL_RETRY	0x1	TDCALL[TDG.VP.VMCALL] sub-function invocation must be retried
TDG.VP.VMCALL_OPERAND_INVALID	0x80000000 00000000	Invalid operand to TDG.VP.VMCALL sub-function
TDG.VP.VMCALL_GPA_INUSE	0x80000000 00000001	GPA already mapped
TDG.VP.VMCALL_ALIGN_ERROR	0x80000000 00000002	Operand (address) alignment error

3 TDG.VP.VMCALL Interface

From the perspective of the host VMM, TDCALL [TDG.VP.VMCALL] is a trap-like, VM exit into the host VMM, reported via the SEAMRET instruction flow.

By design, after the SEAMRET, the host VMM services the request specified in the parameters passed by the TD during the TDG.VP.VMCALL (that are passed via SEAMRET to the VMM), then resumes the TD via a SEAMCALL [TDH.VP.ENTER] invocation.

Refer to the Intel TDX CPU Architecture specification [2] for details of the SEAMCALL and SEAMRET instructions. This chapter describes the designed sub-functions of the TDCALL [TDG.VP.VMCALL] interface between the TD and the VMM.

3.1 TDG.VP.VMCALL<GetTdVmCallInfo>

GetTdVmCallInfo TDG.VP.VMCALL is used to help request the host VMM enumerate which TDG.VP.VMCALLs are supported. This leaf is reserved for enumerating capabilities defined in this specification. VMMs may provide alternate enumeration schemes using vendor-specific TDG.VP.VMCALL namespace, as defined in 2.4.1.

Table 3-1: TDG.VP.VMCALL< GetTdVmCallInfo>-Input Operands

Operand	Description
R11	TDG.VP.VMCALL< GetTdVmCallInfo> sub-function per Table 2-3.
R12	Leaf to enumerate TDG.VP.VMCALL functionality from this specification supported by the host. R12 must be set to 0, and successful execution of this TDG.VP.VMCALL is meant to indicate all TDG.VP.VMCALLs defined in this specification are supported by the host VMM. This register is reserved to extend TDG.VP.VMCALL enumeration in future versions.

Table 3-2: TDG.VP.VMCALL< GetTdVmCallInfo>-Output Operands

Operand	Description
R10	TDG.VP.VMCALL-instruction-return code.
R11	Leaf-specific output (when R12 is 0, will be returned as 0).
R12	Leaf-specific output (when R12 is 0, will be returned as 0).
R13	Leaf-specific output (when R12 is 0, will be returned as 0).
R14	Leaf-specific output (when R12 is 0, will be returned as 0).

Table 3-3: TDG.VP.VMCALL< GetTdVmCallInfo> Status Codes

Error Code	Value	Description
TDG.VP.VMCALL_SUCCESS	See values in Table 2-6	TDG.VP.VMCALL is successful. The TD is free to use the GPA as a shared, memory page.

3.2 TDG.VP.VMCALL<MapGPA>

MapGPA TDG.VP.VMCALL is used to help request the host VMM to map a GPA range as private or shared-memory mappings. This API may also be used to convert page mappings from private to shared. The GPA range passed in this operation can indicate if the mapping is requested for a shared or private memory via the GPA.Shared bit in the start address.

For example, to exchange data with the VMM, the TD may use this TDG.VP.VMCALL to request that a GPA range be mapped as a shared memory (for example, for a paravirtualized IO) via the shared EPT. If the GPA (range) was already mapped as an active, private page, the host VMM may remove the private page from the TD by following the “Removing TD Private Pages” sequence in the Intel TDX-module specification [3] to safely block the mapping(s), flush the TLB and cache, and remove the mapping(s). The VMM can then map the specified GPA (range) in the shared-EPT structure and allow the TD to access the page(s) as a shared GPA (range).

If the Start GPA specified is a private GPA (GPA.S bit is clear), this MapGPA TDG.VP.VMCALL can be used to help request the host VMM map the specific, private page(s) (which mapping may involve converting the backing-physical page from a shared page to a private page). As intended in this case, the VMM must unmap the GPA from the shared-EPT region and invalidate the TLB and caches for the TD VCPUs, to help ensure that no stale mappings exist. Similarly, if the shared page was assigned to any device for use with IO, then the VMM should invalidate the IOTLB and purge any pending IO transactions (e.g. by queuing a wait descriptor) to remove stale mappings on the IO side. Then, the content of all data caches should be flushed.

The aim is for the VMM to then follow the sequence specified in “Dynamically Adding TD Private Pages during TD Run Time” in the Intel TDX-module specification [3] to use TDH.MEM.PAGE.AUG to add the GPA(s) to the TD as pending, private mapping(s) in the secure-EPT. When the VMM responds to this TDG.VP.VMCALL with success, the goal is for the TD to execute TDCALL[TDG.MEM.PAGE.ACCEPT] to complete the process to make the page(s) usable as a private GPA inside the TD.

Upon MapGPA from shared to private, the VMM needs to check if the page is mapped by the IOMMU page table. If direct I/O is already enabled and the page is mapped, MapGPA should fail. This is equivalent to removing a page from a (legacy) guest with direct I/O enabled; the pages need to be pinned there. If the VMM provides a virtual IOMMU (vIOMMU) or cooperative IOMMU (coIOMMU), then the guest can indicate that it is not using that memory for DMA. In that case, MapGPA can: 1) Check that page is not pinned by vIOMMU; 2) Check that the page is not mapped in physical IOMMU. If 1 and 2 succeed, then unmap and remap it.

Table 3-4: TDG.VP.VMCALL<MapGPA>-Input Operands

Operand	Description
R11	TDG.VP.VMCALL<MapGPA> sub function per Table 2-3.
R12	4KB-aligned Start GPA of address range (Shared bit may be set or clear to indicate if a shared- or private-page mapping is desired). Shared-bit position is indicated by the GPA width [Guest-Physical-Address-Width-execution control is initialized by the host VMM for the TD during TDH.VP.INIT].
R13	Size of GPA region to be mapped (must be a multiple of 4KB).

Table 3-5: TDG.VP.VMCALL<MapGPA> Output Operands

Operand	Description
R10	TDG.VP.VMCALL-instruction-return code.
R11	GPA at which MapGPA failed (see failure or retry reason in TDG.VP.VMCALL specific status code.

Table 3-6: TDG.VP.VMCALL<MapGPA>-Status Codes

Error Code	Value	Description
TDG.VP.VMCALL_SUCCESS	See values in Table 2-6	TDG.VP.VMCALL is successful. The TD is free to use the GPA (range) specified.
TDG.VP.VMCALL_RETRY		TD must retry this operation for the pages in the region starting at the GPA specified in R11.
TDG.VP.VMCALL_INVALID_OPERAND		Invalid operand – for example, the GPA address is invalid (beyond GPAW).
TDG.VP.VMCALL_GPA_INUSE		GPA is already in use by the TD, for e.g., GPA used for hosting memory dedicated for IO. R11 specifies which GPA in the range specified was in use.
TDG.VP.VMCALL_ALIGN_ERROR		Alignment error for size or Start GPA.

3.3 TDG.VP.VMCALL<GetQuote>

GetQuote TDG.VP.VMCALL is a doorbell-like interface used to help send a message to the host VMM to queue operations that tend to be long-running operations. GetQuote is designed to invoke a request to generate a TD-Quote signing by a service hosting TD-Quoting Enclave operating in the host environment for a TD Report passed as a parameter by the TD. TDREPORT_STRUCT is a memory operand intended to be sent via the GetQuote TDG.VP.VMCALL to indicate the asynchronous service requested.

For the GetQuote operation, the goal is for the TDREPORT_STRUCT to be received by the TD via a prior TDCALL[TDG.MR.REPORT] in a buffer and placed in a shared-GPA space passed to the VMM as an operand in the GetQuote TDG.VP.VMCALL. In the case of this operation, the VMM can access the TDREPORT_STRUCT, queue the operation for a service hosting TD-Quoting enclave, and, when completed, return the Quote via the same, shared-memory area.

For the TD to invoke the TDG.VP.VMCALL<GetQuote>, the host VMM can signal the event completion to the TD OS via a notification interrupt the host VMM injects into the TD (using the Event-notification vector registered via the SetupEventNotifyInterrupt TDG.VP.VMCALL).

Table 3-7: TDG.VP.VMCALL< GetQuote >-Input Operands

Operand	Description
R11	TDG.VP.VMCALL< GetQuote > sub-function per Table 2-3.
R12	Shared GPA as input – the memory contains a TDREPORT_STRUCT. The same buffer is used as output – the memory contains a TD Quote.
R13	Size of shared GPA. The size must be 4KB-aligned.

Table 3-8: TDG.VP.VMCALL< GetQuote >-Output Operands

Operand	Description
R10	TDG.VP.VMCALL return code.

Table 3-9: TDG.VP.VMCALL< GetQuote >-Status Codes

Error Code	Value	Description
TDG.VP.VMCALL_SUCCESS	See values in Table 2-6	TDG.VP.VMCALL is successfully received by the VMM. This status does not mean that the TD Quote is generated or returned. The caller shall wait for event-notification to evaluate the output buffer to know if the TD Quote is generated successfully.
TDG.VP.VMCALL_RETRY		The TD should retry the operation.
TDG.VP.VMCALL_INVALID_OPERAND		Invalid operand – for example, the GPA may be mapped as a private page.

Table 3-10: TDG.VP.VMCALL<GetQuote> - format of shared GPA

Field	Offset (Bytes)	Length (bytes)	Description
Version	0	8	Version for this structure. It must be 1. This field is always filled by TD.
Status Code	8	8	Status Code updated by VMM before the VMM returns the VMCALL or interrupts the TD. See Table 3-11. This field is always filled by VMM.
Input Length	16	4	The length for data from TD as input. It must be equal to or smaller than the size of shared GPA (R13) – 24. This field is always filled by TD.
Output Length	20	4	The length for data from VMM as input. It must be equal or smaller than size of shared GPA (R13) – 24. This field is always filled by VMM.
Data	24	Size of shared GPA - 24	On input, the data filled by TD with input length. The data should include TDREPORT_STRUCT. TD should zeroize the remaining buffer to avoid information leak if size of shared GPA (R13) > Input Length. On output, the data filled by VMM with output length. The data should include TD Quote.

Table 3-11: TDG.VP.VMCALL<GetQuote> - GetQuote Status Code

Error Code	Value	Description
GET_QUOTE_SUCCESS	0x0	TDG.VP.VMCALL<GetQuote> is successfully completed.
GET_QUOTE_IN_FLIGHT	0xFFFFFFFF_FFFFFFFF	TDG.VP.VMCALL<GetQuote> is under processing. The shared GPA isn't ready for TD to consume.
GET_QUOTE_ERROR	0x80000000_00000000	Error without specifying any reason.
GET_QUOTE_SERVICE_UNAVAILABLE	0x80000000_00000001	Quoting service isn't available.

The following is a typical execution flow:

1) Guest TD sets up notification vector via TDG.VP.VMCALL<SetupEventNotifyInterrupt>
2) Guest TD allocates share GPA and initializes it, and then issues TDG.VP.VMCALL<GetQuote>. The data field should include TDREPORT_STRUCT.
3) VMM receives TDG.VP.VMCALL<GetQuote> request. It checks input operands, fields in shared GPA.
4) If the input operands and fields in shared GPA are good, the VMM updates status code in shared GPA to GET_QUOTE_IN_FLIGHT and queues the request. Then, TDG.VP.VMCALL<GetQuote> is returned to TD. The VMM processes the request in background.
5) The VMM sends the data field from the TD in shared GPA to a service hosting TD-Quoting Enclave and receives response message from it.
6) The VMM stores the data field in the received message in shared GPA and updates output length and status code in shared GPA.
7) The VMM notifies the TD with an interruption vector specified by TDG.VP.VMCALL<SetupEventNotifyInterrupt>.
8) The guest TD is interrupted. It checks if the GetQuote status code fields in shared GPA are not GET_QUOTE_IN_FLIGHT. If GetQuote status code is GET_QUOTE_SUCCESS, the data field includes the TD Quote.

TDG.VP.VMCALL<GetQuote> API allows one TD to issue multiple requests. This is implementation specific as to how many concurrent requests are allowed. The TD should be able to handle TDG.VP.VMCALL_RETRY if it chooses to issue multiple requests simultaneously.

3.4 TDG.VP.VMCALL<ReportFatalError>

The FatalError TDG.VP.VMCALL can inform the host VMM that the TD has experienced a fatal-error state, and let the VMM access debugging information. The output returned by the TDG.VP.VMCALL by the host VMM for Debug and Production versions of the platform may be different.

This TDG.VP.VMCALL is intended to be used by the TD OS during early boot (in guest-firmware execution, for example) where some instructions like IN/OUT may be avoided to prevent causing a #VE. It may be also used by the TD guest post-boot when it detects an error (e.g., a security violation) and the TD wants to stop reliably with information exposed to the host via the TD-specific error code (and additional information as a zero-terminated string via the shared memory 4KB region).

Table 3-12: TDG.VP.VMCALL< ReportFatalError >-Input Operands

<table>
<tr><th>Operand</th><th colspan="3">Description</th></tr>
<tr><td>R11</td><td colspan="3">TDG.VP.VMCALL< ReportFatalError > sub-function per Table 2-3</td></tr>
<tr><td rowspan="4">R12</td><td>Bits</td><td>Name</td><td>Description</td></tr>
<tr><td>31:0</td><td>TD-specific error code</td><td>TD-specific error code
Panic – 0x0.
Values – 0x1 to 0xFFFFFFFF reserved.</td></tr>
<tr><td>62:32</td><td>TD-specific extended error code</td><td>TD-specific extended error code.
TD software defined.</td></tr>
<tr><td>63</td><td>GPA Valid</td><td>Set if the TD specified additional information in the GPA parameter (R13).</td></tr>
<tr><td>R13</td><td colspan="3">4KB-aligned GPA where additional error data is shared by the TD. The VMM must validate that this GPA has the Shared bit set. In other words, that a shared-mapping is used, and that this is a valid mapping for the TD. This shared memory region is expected to hold a zero-terminated string.
Shared-bit position is indicated by the GPA width [Guest-Physical-Address-Width-execution control is initialized by the host VMM for the TD during TDH.VP.INIT].</td></tr>
<tr><td>R14, R15, RBX, RDI, RSI, R8, R9, RDX</td><td colspan="3">Optional error data in registers if the corresponding bit is set in RCX bitmap.
The order is in the input register order (R14 is first, R15 second, etc).
The information is byte sequence, LSB is filled first. Typically, ASCII code(0x20-0x7e) is filled.</td></tr>
</table>

Table 3-13: TDG.VP.VMCALL<ReportFatalError>-Output Operands

Operand	Description
R10	TDG.VP.VMCALL-return code.

Table 3-14: TDG.VP.VMCALL< ReportFatalError >-Status Codes

Error Code	Value	Description
TDG.VP.VMCALL_SUCCESS	See values in Table 2-6	TDG.VP.VMCALL is successful.

3.5 TDG.VP.VMCALL<SetupEventNotifyInterrupt>

The guest TD may request that the host VMM specify which interrupt vector to use as an event-notify vector. This is designed as an untrusted operation; thus, the TD OS should be designed not to use the event notification for trusted operations. Example of an operation that can use the event notify is the host VMM signaling a device removal to the TD, in response to which a TD may unload a device driver.

The host VMM should use SEAMCALL [TDWRVPS] leaf to inject an interrupt at the requested-interrupt vector into the TD VCPU that executed TDG.VP.VMCALL <SetupEventNotifyInterrupt> via the posted-interrupt descriptor. See Intel TDX-module specification [3] for TD-interrupt handling.

Table 3-15: TDG.VP.VMCALL< SetupEventNotifyInterrupt>-Input Operands

Operand	Description
R11	TDG.VP.VMCALL<Setup Event Notify Interrupt> sub-function per Table 2-3.
R12	Interrupt vector (valid values 32:255) selected by TD.

Table 3-16: TDG.VP.VMCALL< SetupEventNotifyInterrupt >-Output Operands

Operand	Description
R10	TDG.VP.VMCALL-return code.

Table 3-17: TDG.VP.VMCALL< SetupEventNotifyInterrupt >-Status Codes

Error Code	Value	Description
TDG.VP.VMCALL_SUCCESS	See values in Table 2-6.	TDG.VP.VMCALL is successful.
TDG.VP.VMCALL_INVALID_OPERAND		Invalid operand.

3.6 TDG.VP.VMCALL<Instruction.CPUID>

Instruction.CPUID TDG.VP.VMCALL is designed to enable the TD-guest to request the VMM to emulate CPUID operation, especially for non-architectural, CPUID leaves.

Table 3-18: TDG.VP.VMCALL<Instruction.CPUID>-Input Operands

Operand	Description
R11	TDG.VP.VMCALL<Instruction.CPUID>-Instruction-execution sub-functions per Table 2-3.
R12	EAX
R13	ECX

Table 3-19: TDG.VP.VMCALL<Instruction.CPUID>-Output Operands

Operand	Description
R10	TDG.VP.VMCALL-return code.
R12	EAX
R13	EBX
R14	ECX
R15	EDX

Table 3-20: TDG.VP.VMCALL<Instruction.CPUID>-Status Codes

Error Code	Value	Description
TDG.VP.VMCALL_SUCCESS	See values in Table 2-6	TDG.VP.VMCALL is successful.

3.7 TDG.VP.VMCALL<#VE.RequestMMIO>

This TDG.VP.VMCALL is used to help request the VMM perform emulated-MMIO-access operation. The VMM may emulate MMIO space in shared-GPA space. The VMM can induce a #VE on these shared-GPA accesses by mapping shared GPAs with the suppress-VE bit cleared in the EPT Entries corresponding to these mappings.

In response to the #VE, the TD can use the TDCALL[TDG.VP.VEINFO.GET] to get the Virtualization-Exception-Information Fields (See Intel TDX-module specification) and validate that the #VE exit reason is 48 (EPT violation causing #VE). After the TD software decodes the instruction causing the #VE locally and validating the accessed region and source of access, the TD may choose to use this TDG.VP.VMCALL to request MMIO read/write operations.

The VMM may emulate the access based on the inputs provided by the TD. However, note that, like other TDG.VP.VMCALLs, this TDCALL is designed as an untrusted operation and to be used for untrusted IO with other cryptographic protection for the TD data provided by the TD itself.

Table 3-21: TDG.VP.VMCALL<#VE.RequestMMIO>-Input Operands

Operand	Description
R11	TDG.VP.VMCALL<#VE.RequestMMIO> sub-function per Table 2-3.
R12	Size of access. 1=1byte, 2=2bytes, 4=4bytes, 8=8bytes. All rest value = reserved.
R13	Direction. 0=Read, 1=Write. All rest value = reserved.
R14	MMIO Address
R15	Data to write, if R13 is 1.

Table 3-22: TDG.VP.VMCALL<#VE.RequestMMIO>-Output Operands

Operand	Description
R10	TDG.VP.VMCALL-return code.
R11	Data to read, if R13 is 0.

Table 3-23: TDG.VP.VMCALL<#VE.RequestMMIO>-Status Codes

Error Code	Value	Description
TDG.VP.VMCALL_SUCCESS	See values in Table 2-6	TDG.VP.VMCALL is successful.
TDG.VP.VMCALL_INVALID_OPERAND		If invalid operands provided by the TD, *e.g.*, MMIO address.

3.8 TDG.VP.VMCALL<Instruction.HLT>

Instruction.HLT TDG.VP.VMCALL is used to help perform HLT operation. The TD guest informs the VMM regarding the TD's interrupt (blocked) status via this interface.

Table 3-24: TDG.VP.VMCALL<Instruction.HLT>-Input Operands

Operand	Description
R11	TDG.VP.VMCALL<Instruction.HLT>-Instruction-execution sub-functions per Table 2-3.
R12	Interrupt Blocked Flag. The TD is expected to clear this flag if RFLAGS.IF == 1 or the TDCALL instruction (that invoked TDG.VP.TDVMCALL(Instruction.HLT)) immediately follows an STI instruction, otherwise this flag should be set.

Table 3-25: TDG.VP.VMCALL<Instruction.HLT>-Output Operands

Operand	Description
R10	TDG.VP.VMCALL-return code.

Table 3-26: TDG.VP.VMCALL<Instruction.HLT>-Status Codes

Error Code	Value	Description
TDG.VP.VMCALL_SUCCESS	See values in Table 2-6	TDG.VP.VMCALL is successful.

3.9 TDG.VP.VMCALL<Instruction.IO>

Instruction.IO TDG.VP.VMCALL is used to help request the VMM perform IO operations.

Table 3-27: TDG.VP.VMCALL<Instruction.IO>-Input Operands

Operand	Description
R11	TDG.VP.VMCALL<Instruction.IO>-Instruction-execution sub-functions per Table 2-3.
R12	Size of access. 1=1byte, 2=2bytes, 4=4bytes. All rest value = reserved.
R13	Direction. 0=Read, 1=Write. All rest value = reserved.
R14	Port number
R15	Data to write, if R13 is 1.

Table 3-28: TDG.VP.VMCALL<Instruction.IO>-Output Operands

Operand	Description
R10	TDG.VP.VMCALL-return code.
R11	Data to read, if R13 is 0.

Table 3-29: TDG.VP.VMCALL<Instruction.IO>-Status Codes

Error Code	Value	Description
TDG.VP.VMCALL_SUCCESS	See values in Table 2-6	TDG.VP.VMCALL is successful.
TDG.VP.VMCALL_INVALID_OPERAND		Invalid-IO-Port access

3.10 TDG.VP.VMCALL<Instruction.RDMSR>

Instruction.RDMSR TDG.VP.VMCALL is used to help perform RDMSR operation.

Table 3-30: TDG.VP.VMCALL<Instruction.RDMSR>-Input Operands

Operand	Description
R11	TDG.VP.VMCALL<Instruction.RDMSR> Instruction execution sub-functions per Table 2-3.
R12	MSR Index

Table 3-31: TDG.VP.VMCALL<Instruction.RDMSR>-Output Operands

Operand	Description
R10	TDG.VP.VMCALL-return code.
R11	MSR Value

Table 3-32: TDG.VP.VMCALL<Instruction.RDMSR>-Status Codes

Error Code	Value	Description
TDG.VP.VMCALL_SUCCESS	See values in Table 2-6	TDG.VP.VMCALL is successful.
TDG.VP.VMCALL_INVALID_OPERAND		Invalid MSR rd/wr requested or access-denied.

3.11 TDG.VP.VMCALL<Instruction.WRMSR>

Instruction.WRMSR TDG.VP.VMCALL is used to help perform WRMSR operation.

Table 3-33: TDG.VP.VMCALL<Instruction.WRMSR>-Input Operands

Operand	Description
R11	TDG.VP.VMCALL<Instruction.WRMSR>-Instruction-execution sub-functions per Table 2-3.
R12	MSR Index
R13	MSR Value

Table 3-34: TDG.VP.VMCALL<Instruction.WRMSR>-Output Operands

Operand	Description
R10	TDG.VP.VMCALL-return code.

Table 3-35: TDG.VP.VMCALL<Instruction.WRMSR>-Status Codes

Error Code	Value	Description
TDG.VP.VMCALL_SUCCESS	See values in Table 2-6	TDG.VP.VMCALL is successful.
TDG.VP.VMCALL_INVALID_OPERAND		Invalid MSR rd/wr requested or access-denied.

3.12 TDG.VP.VMCALL<Instruction.WBINVD>

Instruction.WBINVD TDG.VP.VMCALL is used to help perform WBINVD operation.

Table 3-36: TDG.VP.VMCALL<Instruction.WBINVD>-Input Operands

Operand	Description
R11	TDG.VP.VMCALL<Instruction.WBINVD>-Instruction-execution sub-functions per Table 2-3.
R12	0: WBINVD 1: WBNOINVD Others: Reserved

Table 3-37: TDG.VP.VMCALL<Instruction.WBINVD>-Output Operands

Operand	Description
R10	TDG.VP.VMCALL-return code.

Table 3-38: TDG.VP.VMCALL<Instruction.WRMSR>-Status Codes

Error Code	Value	Description
TDG.VP.VMCALL_SUCCESS	See values in Table 2-6	TDG.VP.VMCALL is successful.
TDG.VP.VMCALL_INVALID_OPERAND		Invalid R12 value requested.

3.13 TDG.VP.VMCALL<Instruction.PCONFIG>

Instruction.VMCALL PCONFIG is used to help perform Instruction-PCONFIG operation.

Table 3-39: TDG.VP.VMCALL<Instruction.PCONFIG>-Input Operands

Operand	Description
R11	TDG.VP.VMCALL<Instruction.PCONFIG> sub-function per Table 2-3.
R12	PCONFIG-Leaf function requested.
R13, R14, R15	Leaf-specific purpose (See PCONFIG ISA definition in MKTME spec. [4]

Table 3-40: TDG.VP.VMCALL<Instruction.PCONFIG>-Output Operands

Operand	Description
R10	TDG.VP.VMCALL-return code.
R11	VMM-Vendor Specific
R12, R13, R14, R15, RBX, RDI, RSI, R8, R9, RDX	VMM-Vendor Specific
XMM0–XMM15	If RCX bit 1 is set, the XMM content is set by VMM host when executing SEAMCALL(TDENTER). Otherwise, the XMM content is unmodified.

Table 3-41: TDG.VP.VMCALL<Instruction.PCONFIG>-Status Codes

Error Code	Value	Description
TDG.VP.VMCALL_SUCCESS	See values in Table 2-6	TDG.VP.VMCALL is successful.
TDG.VP.VMCALL_INVALID_OPERAND		If PCONFIG-operation requested is invalid.

4 TD-Guest-Firmware Interfaces

4.1 ACPI MADT Multiprocessor Wakeup Table

The guest firmware is designed to publish a multiprocessor-wakeup structure to let the guest-bootstrap processor wake up guest-application processors with a mailbox. The mailbox is memory that the guest firmware can reserve so each guest virtual processor can have the guest OS send a message to them.

For detailed information about the Multiprocessor Wakeup Table, please refer to ACPI 6.4 specification.

4.2 Memory Map

The memory in the TD guest-environment can be:

1) Private memory – SEAMCALL[TDH.MEM.PAGE.ADD] by VMM or TDCALL [TDG.MEM.PAGE.ACCEPT] by TDVF with S-bit clear in page table.

2) Shared memory – SEAMCALL[TDH.MEM.PAGE.ADD] by VMM or TDCALL [TDG.MEM.PAGE.ACCEPT] by TDVF with S-bit set in page table.

3) Unaccepted memory – SEAMCALL[TDH.MEM.PAGE.AUG] by VMM and not accepted by TDVF yet.

4) Memory-Mapped I/O (MMIO) - Shared memory accessed via TDVF via TDVMCALL<#VE.RequestMMIO>.

If a TD-memory region is private memory, the TD owner shall have the final UEFI-memory map report the region with **`EfiReservedMemoryType`**, **`EfiLoaderCode`**, **`EfiLoaderData`**, **`EfiBootServiceCode`**, **`EfiBootServiceData`**, **`EfiRuntimeServiceCode`**, **`EfiRuntimeServiceData`**, **`EfiConventionalMemory`**, **`EfiACPIReclaimMemory`**, **`EfiACPIMemoryNVS`**.

If a TD-memory region is shared memory, the TD owner shall convert it to private memory before transfer to OS kernel.

If a TD-memory region is unaccepted memory and requires TDCALL [TDG.MEM.PAGE.ACCEPT] in the TD guest OS, then the TD owner shall have the final UEFI-memory map report this region with **`EfiUnacceptedMemoryType`**. Please refer to UEFI 2.9 specification.

If a memory region is MMIO, it is designed to only be accessed via TDVMCALL<#VE.RequestMMIO> and not via direct memory read or write. Accordingly, as designed, there is no need to report this region in UEFI-memory map, because no RUNTIME attribute is required. The full MMIO regions are designed to be reported in ACPI ASL code via memory-resource descriptors.

Table 4-1: TDVF-memory map for OS

UEFI Memory Type	Usage	TD-Memory Type	OS Action
EfiReservedMemoryTpe	Firmware-Reserved region, such as flash.	Private	Reserved.
EfiLoaderCode	UEFI-Loader Code	Private	Use after EBS.
EfiLoaderData	UEFI-Loader Data	Private	Use after EBS.
EfiBootServicesCode	UEFI-Boot-Service Code	Private	Use after EBS.
EfiBootServicesData	UEFI-Boot-Service Data	Private	Use after EBS.
EfiRuntimeServicesCode	UEFI-Runtime-Service Code	Private	Map-virtual address. Reserved.
EfiRuntimeServicesData	UEFI-Runtime-Service Data	Private	Map-virtual address. Reserved.
EfiConventionalMemory	Freed memory (Private)	Private	Use directly.
EfiACPIReclaimMemory	ACPI table.	Private	Use after copy ACPI table.
EfiACPIMemoryNVS	Firmware Reserved for ACPI, such as the memory used in ACPI OpRegion	Private	Reserved.
EfiMemoryMappedIO	No need to report the MMIO region, as no RUNTIME-virtual address is required for TD. The full MMIO should be reported in ACPI-ASL code.	N/A	N/A
EfiUnacceptedMemoryType	UEFI-Boot-Service Data (Shared Memory) VMM-shared buffer.	Unaccepted	Use after EBS and converting to private page. ============== TDCALL[TDG.VP.VMCALL] <MapGPA> TDCALL[TDG.MEM.PAGE.ACCEPT]

For a non-UEFI system, the memory map can be reported via E820 table. Please refer to ACPI 6.5 specification.

If a TD-memory region is private memory, the TD Shim shall have the final memory map report the region with **`AddressRangeMemory`**, **`AddressRangeReserved`**, **`AddressRangeACPI`**, or **`AddressRangeNVS`**.

If a TD-memory region is shared memory, the TD Shim shall convert it to private memory before transfer to OS kernel.

If a TD-memory region is unaccepted memory and requires TDCALL [TDG.MEM.PAGE.ACCEPT] in the TD guest OS, then the TD Shim shall have the final memory map report this region with **`AddressRangeUnaccepted`**.

If a memory region is MMIO, it is designed to only be accessed via TDVMCALL<#VE.RequestMMIO> and not via direct memory read or write. Accordingly, as designed, there is no need to report this region in the final memory map.

Table 4-2: TDVF E820 memory map for OS

E820 Memory Type	Usage	TD-Memory Type	OS Action
AddressRangeMemory	Usable by OS.	Private	Use directly
AddressRangeReserved	Firmware-Reserved region, such as flash.	Private	Reserved.
AddressRangeACPI	ACPI table.	Private	Use after copy ACPI table.
AddressRangeNVS	Firmware Reserved for ACPI, such as the memory used in ACPI OpRegion	Private	Reserved.
AddressRangeUnaccepted	Allocated by VMM, but not accepted by TD guest yet.	Unaccepted	Use after convert to private page.

4.3 TD Measurement

4.3.1 TCG-Platform-Event Log

If TD-Guest Firmware supports measurement and an event is created, TD-Guest Firmware is designed to report the event log with the same data structure in TCG-Platform-Firmware-Profile specification with **EFI_TCG2_EVENT_LOG_FORMAT_TCG_2** format.

The index created by the TD-Guest Firmware in the event log should be the index for the confidential computing (CC) measurement register.

Table 4-3: CC-Event-Log-PCR-Index Interpretation for TDX

CC Measurement Register Index	**TDX-measurement register**
0	MRTD
1	RTMR[0]
2	RTMR[1]
3	RTMR[2]
4	RTMR[3]

4.3.2 EFI_CC_MEASUREMENT_PROTOCOL

If TD-Guest Firmware supports measurement, the TD Guest Firmware is designed to produce EFI_CC_MEASUREMENT_PROTOCOL with new GUID **EFI_CC_MEASUREMENT_PROTOCOL_GUID** to report event log and provide hash capability. Please refer to UEFI 2.10 specification.

4.3.3 CC-Event Log

TD-Guest Firmware may set up a CCEL ACPI table to pass the event-log information. The event log created by the TD owner contains the hashes to reconstruct the confidential computing (CC) measurement registers. Please refer to ACPI 6.5 specification.

4.4 Storage-Volume-Key Data

In the TD-execution environment, the storage volume will typically be an encrypted volume. In that case, by design, the TD-Guest Firmware will need to support quote generation and attestation to be able to fetch a set of storage-volume key(s) from a remote-key server during boot and pass the key to the guest kernel. Also by design, the key is stored in the memory, and the information of the key is passed from TD-Guest Firmware via an SVKL ACPI table. Please refer to ACPI 6.5 specification.

5 TD-VMM-Communication Scenarios

5.1 Requesting IPIs

Various TD-VMM-communication scenarios require the TD to request that the host generate IPIs to TD VCPUs – for example, synchronizing, guest-TD-kernel-managed-IA-page-table updates. This operation is supported via the TDCALL [TDG.VP.VMCALL <Instruction.WRMSR>] to the x2APIC ICR MSR.

To perform a cross-VCPU IPI, the guest-TD ILP is designed to request an operation from the host VMM using this TDCALL (TDG.VP.VMCALL <Instruction.WRMSR>). The VMM can then inject an interrupt into the guest TD's RLPs using the posted-interrupt mechanism. This is an untrusted operation; thus, the TD OS must track its completion and not rely on the host VMM to faithfully deliver IPIs to all the TD VCPUs.

5.2 TD-memory conversion and memory ballooning

Recall that, by design, guest-physical memory used by a TD is encrypted with a TD-private key or with a VMM-managed key based on the GPA-shared bit (GPA [47 or 51] based on GPAW). A TD OS may operate on a fixed, private-GPA space configured by the host VMM. Typically, the OS manages a physical-page-frame database for state of (guest) physical-memory allocations.

Instead of expanding these PFN databases for large swaths of shared-GPA space, the TD OS can manage an attribute for the state of physical memory to indicate whether it is encrypted with the TD-private key or a VMM key. The TD-guest OS can then use TDG.VP.VMCALL(MapGPA) so that, within the fixed-GPA map, the TD OS can request that the host VMM map Shared-IO memory aliased as shared memory in that GPA space. So in this case, the OS can select a page of the private-GPA space and make a TDG.VP.VMCALL(MapGPA(GPA) with GPA.S=1) to map that GPA using the S=1 alias.

The VMM can then TDH.MEM.RANGE.BLOCK, TDH.MEM.TRACK, and TDH.MEM.SEPT.REMOVE the affected GPA from the S-EPT mapping; and the VMM can then re-claim the page using direct-memory stores and map the alias-shared GPA for the TD OS in the shared EPT (managed by the VMM).

At a later point, the TD OS may need to use the GPA as a private page via the same TDG.VP.VMCALL(MapGPA) with the GPA specified as a private GPA (GPA.S=0) – the intent is for this to allow the host VMM to unlink the page from the shared EPT and then perform a TDH.MEM.PAGE.AUG to set up a pending-EPT mapping for the private GPA. The successful completion of the TDG.VP.VMCALL flow can be used by the TD guest to TDG.MEM.PAGE.ACCEPT to re-initialize the page using the TD-private key and mark the S-EPT mapping as active.

5.3 Paravirtualized IO

The TD guest can use paravirtualized-IO interfaces (for example, using virtio API in KVM) exposed by the host VMM to use physical and virtual devices on the host platform that are managed by the VMM. For this scenario, Virtualized IO is typically enumerated over emulated PCIe (port I/O or MMIO). The TD drivers can help ensure that the data passed via memory referenced in emulated-MMIO accesses are placed in the TD's shared-GPA-memory space.

Paravirtualized drivers could pre-allocate a primary-shared buffer during initialization. Subsequently, drivers can allocate a portion of the shared-GPA-space buffer for each individual transfer and reclaim the buffer after a specific transfer is completed. In this scenario, the primary buffer can expand and shrink as needed.

Shared buffers can be deallocated during driver-stack tear-down. This scenario is optimal, as allocating shared buffer can involve at least one TDG.VP.VMCALL (for mapping shared page) and TDCALL[TDG.MEM.PAGE.ACCEPT] for mapping back as a private-TD page, as described in Section 4.3.

The guest TD may employ VMM functions for IO to participate in the emulation of MMIO accesses from legacy-device drivers. To support this scenario, if the TD OS opts-in, the host VMM can host the emulated-device-MMIO space in shared-GPA space of the TD OS. Legacy-device-driver accesses to the emulated region can cause EPT violations that can be mutated to the TD-#VE handler, which can then support emulation of the MMIO.

The enlightened-TD-OS-#VE handler can emulate the access causing the #VE by decoding the instruction (within the TD) and invoking the Instruction.IO functions hosted by the VMM using TDCALL [TDG.VP.VMCALL <#VE.RequestMMIO>]. From that point on, like the paravirtualized I/O model, the TD software should ensure that data buffers passed via memory referenced by parameters in function TDG.VP.VMCALL are placed in the TD's shared -GPA space.

5.4 TD attestation

Goals of TD Attestation are to enable the TD OS to request a TDREPORT that contains version information about the Intel TDX module, measurement of the TD, along with a TD-specified nonce. By design, the TDREPORT is locally MAC'd and used to generate a quote for the TD via a quoting enclave (QE). The remote verifier can verify the quote to help verify the trustworthiness of the TD.

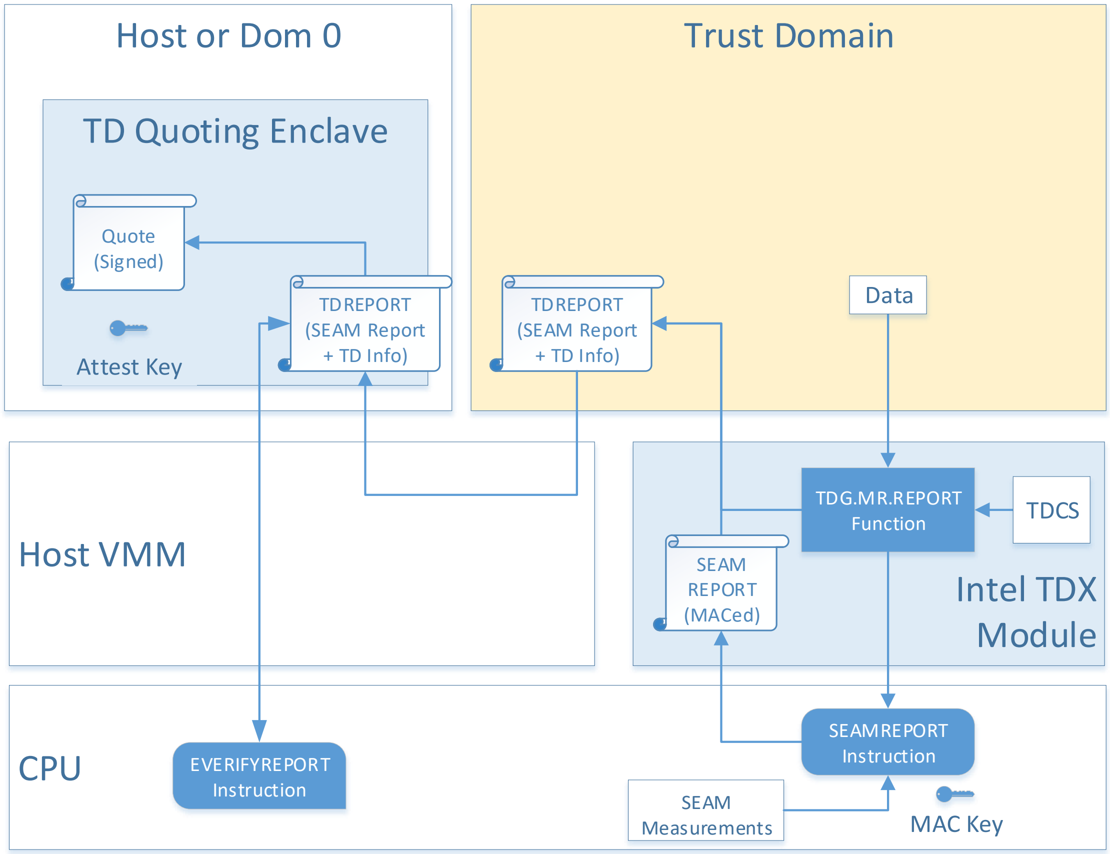

Figure 5-1: TD-Attestation flow

1. Guest-TD software invokes the TDCALL(TDG.MR.REPORT)-API function.
2. The Intel TDX module uses the SEAMOPS[SEAMREPORT] instruction to create a MAC'd TDREPORT_STRUCT with the Intel TDX-module measurements from CPU and TD measurements from the TDCS.
3. Guest-TD software uses the TDCALL(TDG.VP.VMCALL) interface to request the TDREPORT_STRUCT be converted into a Quote.
4. The TD-Quoting enclave uses ENCL[EVERIFYREPORT2] to verify the TDREPORT_STRUCT. This allows the Quoting Enclave to help verify the report without requiring direct access to the CPU's HMAC key. Once the integrity of the TDREPORT_STRUCT has been verified, the TD-Quoting Enclave signs the TDREPORT_STRUCT body with an ECDSA-384-signing key.
5. The Quote can then be used by TD software to perform a remote-attestation protocol with a verifying-remote party.

~~ End of document ~~