Guest-Host-Communication Interface (GHCI) for Intel® Trust Domain Extensions (Intel® TDX)

344426-001US

SEPTEMBER 2020

Disclaimers

Table of Contents

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. Hence, the TD must volunteer information to access IO services, enumerate model-specific, CPU capabilities, measurement services, and provide feedback to the VMM on guest-OS-triggered actions, such as virtual-IPIs, shutdown, etc. For each operation in 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

In Section 2, the document 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 ABI between the TD and the VMM for specific operations. Section 4 describes example flows for the main 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]

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

Acronym	Full Name	Description
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 controller 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.
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 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 architecture specification	Available on request
3	Intel® Trust Domain Extensions module specification	Available on request

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 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. Details of the operation of this module are described 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 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 restrict only with regard to the TDs it manages.

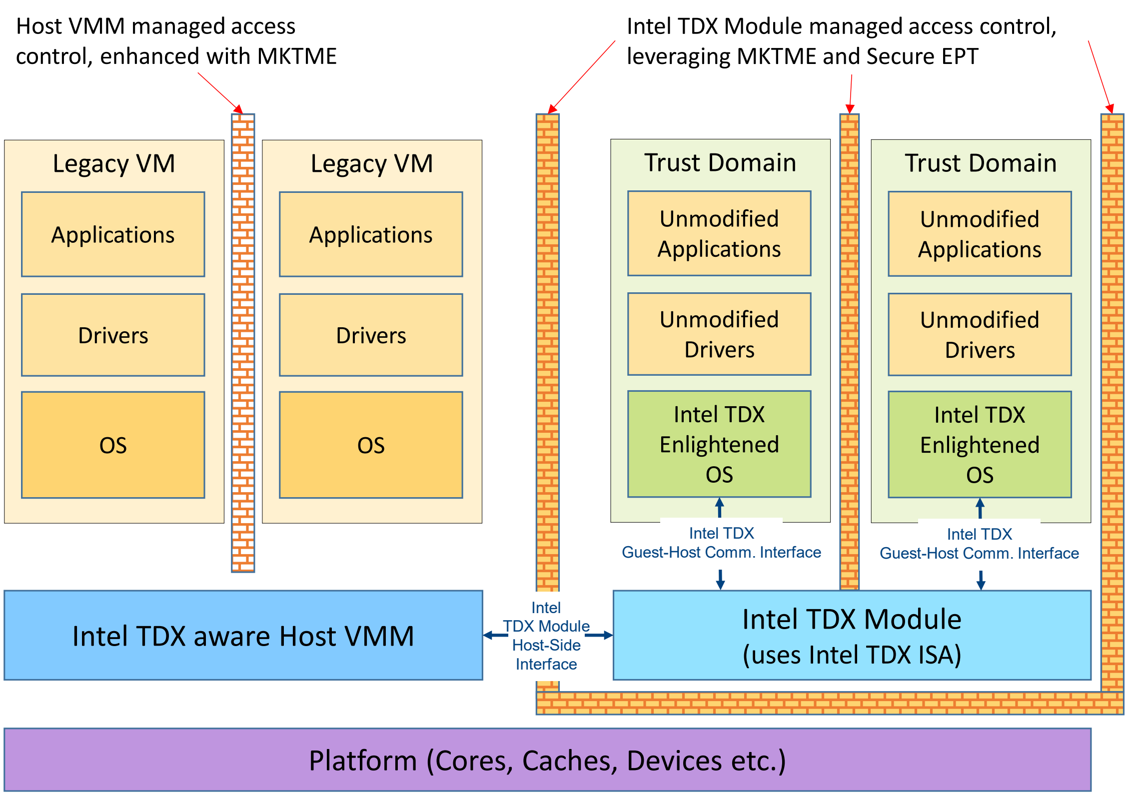

Figure 2.1: Components of Intel Trust Domain Extensions

2.2 TD-VMM-Communication Overview

TD-VMM communication can occur via asynchronous, VM exits or via synchronous (instruction), VM exits. In response to the 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 be via a TDCALL instruction [2] requesting the host VMM to provide (untrusted) services. The goal is for the VMM to receive the service request via a SEAMRET invoked by the Intel TDX module, complete the service requested, and respond to the TD via the SEAMCALL[TDH.VP.ENTER] to re-enter the TD. This document describes the mechanisms and ABI for this interaction in various scenarios expected.

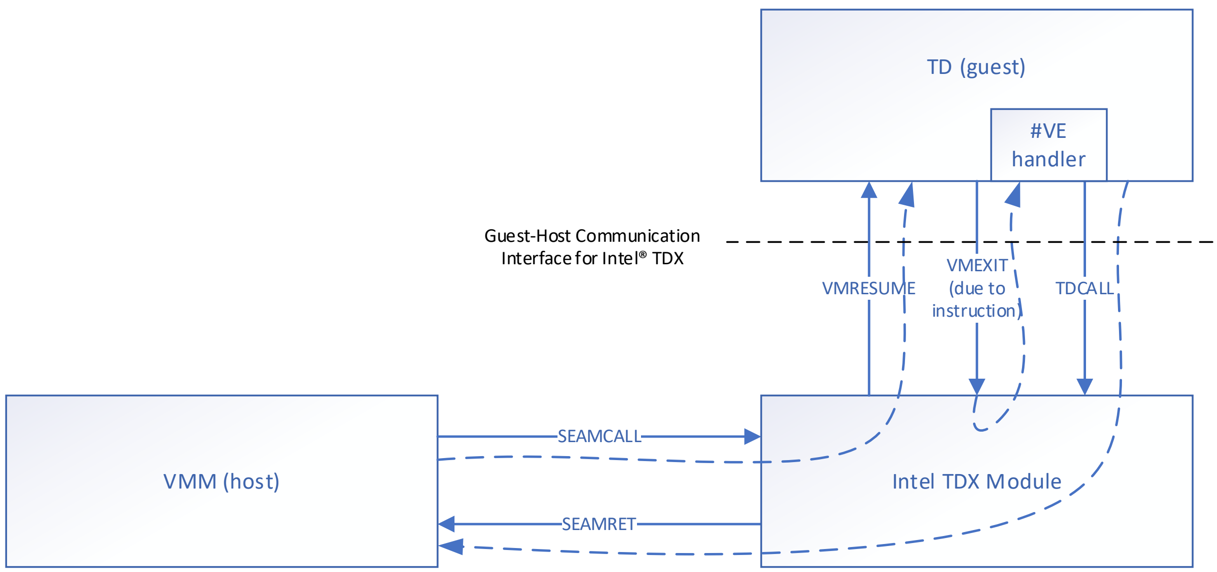

Figure 2: TD Guest-Host communication

Section 2 of this document describes the Virtualization Exception (#VE) for Intel TDX, and subsequent chapters describe the normative, TDCALL leaves intended to get the VE information as well as request services from the host VMM. There are other cases that 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, *i.e.*, IO accesses, etc. This section covers such scenarios and describes the #VE interface.

2.3.1 Virtualization-Exception Information

The **virtualization-exception-information area** (VE_INFO) is part of TDVPS and not intended to be directly accessible to the guest TD. This is done to better assure the availability and privacy of this area. Intel TDX is designed to set the TD-VMCS-virtualization-exception-information address to the physical address (and TD private HKID) of this area.

Table: Virtualization-Exception-Information Area (VE_INFO), based on [Intel SDM, Vol. 3, Table 25-1]

Section	Field	Offset (Bytes)	Size (Bytes)	Description
Architectural	**EXIT_REASON**	0	4	The value that would have been saved into the VMCS as an exit reason had a VM exit occurred instead of the virtualization exception
	VALID	4	4	0 indicates that VE_INFO has no valid contents. The Intel TDX module will not update VE_INFO if VALID is not 0. After updating VE_INFO, the Intel TDX module writes 0xFFFFFFFF to the VALID field.
	EXIT_QUALIFICATION	8	8	The value that would have been saved into the VMCS as an exit qualification had a VM exit occurred instead of the virtualization exception
	GLA	16	8	The value that would have been saved into the VMCS as a guest-linear address had a VM exit occurred instead of the virtualization exception
	GPA	24	8	The value that would have been saved into the VMCS as a guest-physical address had a VM exit occurred instead of the virtualization exception
	EPTP_INDEX	32	2	The current value of the EPTP-index, VM-execution control
Non-Architectural	**INSTRUCTION_LENGTH**	Non-arch.	4	The 32-bit value that would have been saved into the VMCS as VM-exit-instruction length had a legacy-VM exit occurred instead of the virtualization exception

Section	Field	Offset (Bytes)	Size (Bytes)	Description
	INSTRUCTION_ INFORMATION	Non-arch.	4	The 32-bit value that would have been saved into the VMCS as VM-exit-instruction information had a legacy-VM exit occurred instead of the virtualization exception

VE_INFO's architectural section's format is defined in the [Intel SDM] and accordingly meant to be used directly by the CPU when the CPU injects a #VE (see 2.3.2 below). VE_INFO can also be used for #VE injected by Intel TDX. Some VE_INFO fields are applicable only for some exit reasons.

VE_INFO.VALID is designed to be initialized to 0 and then set to 0xFFFFFFFF when a #VE is injected to the guest TD. When handling a #VE, the guest TD aims to retrieve the #VE information using the **TDCALL (TDG.VP.VEINFO.GET)** function. **TDG.VP.VEINFO.GET** verifies that VE_INFO.VALID is 0xFFFFFFFF. After reading the information, TDG.VP.VEINFO.GET sets VE_INFO.VALID to 0.

The goal is for #VE delivery by Intel TDX module 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 (See section 2.3.4-6) in the #VE handler.

2.3.2 #VE Injected by the CPU due to EPT Violations

By design, #VE is enabled unconditionally for Intel TDX-non-root operation, and the Intel TDX module sets the TD-VMCS-**EPT-violation #VE** VM-execution control to 1.

For shared-memory accesses (*i.e.*, when GPA.SHARED == 1), as with legacy VMX, the VMM can choose which pages are eligible for #VE mutation based on the value of the Shared EPTE bit 63.

For private-memory accesses (GPA.SHARED == 0), an EPT Violation on a non-pending entry is intended to always cause a TD Exit. The Intel TDX module is designed to set the Suppress VE bit (63) for all active, Secure-EPT entries and clear it for pending entries.

2.3.3 #VE Injected by Intel TDX (Overview)

#VE may be injected by Intel TDX as a result of guest-TD execution of unconditionally disallowed instructions (see list below), conditionally disallowed MSR accesses, or CPUID virtualization – see Intel TDX-non-root mode of operation [3] for details. In such cases, the guest-TD-VM is designed to exit to the Intel TDX module, which helps do the following:

1. First check VE_INFO.VALID to make sure VE_INFO does not contain information that has not been read yet using TDCALL[TDG.VP.VEINFO.GET].
2. If VE_INFO.VALID is 0, copy the exit reason, exit qualification, and other fields from the TD VMCS to VE_INFO and inject a #VE to the guest TD.
3. If VE_INFO.VALID is not 0, inject a #DF (0) to the guest TD to indicate a #VE overrun.

2.3.3.1 #VE Injected due to disallowed instructions

Intel TDX helps block certain instructions from executing in Intel TDX-non-root mode. By design, execution of those instructions results in an exception (#VE) to the guest TD.

The following is a designated list of disallowed instructions that can cause #VE when invoked in the TD guest:

- String I/O (INS, OUTS), IN, OUT
- HLT
- MONITOR, MWAIT
- WBINVD, INVD
- VMCALL

The following is a designated list of disallowed instructions that can cause #UD when invoked in the TD guest:

- All VMX instructions: INVEPT, INVVPID, VMCLEAR, VMFUNC, VMLAUNCH, VMPTRLD, VMPTRST, VMREAD, VMRESUME, VMWRITE, VMXOFF, VMXON
- ENCLS, ENCLV
- GETSEC
- RSM
- ENQCMD

The following is a designated list of disallowed instructions that can cause #VE or #UD when invoked in the TD guest, depending on features enabled by the host VMM for the TD guest:

- PCONFIG

2.3.3.2 #VE Injected due to MSR accesses

From the guest TD's point of view, MSRs can be divided to the following categories:

- MSRs that are context-switched on TD VM entry and TD VM exit. Guest-TD access to such MSRs may be full, partial, or none. *See Appendix for details on these MSRs*.
- MSRs that are not context-switched, but guest-TD access is read-only.
- MSRs that are not context-switched and are inaccessible to the guest TD.

MSR behavior is designed to be fixed for any version of Intel TDX, and the host VMM has no API to modify guest-TD-MSR configuration, *e.g.*, the host VMM cannot set a certain MSR to TD exit on write. Instead, guest-TD-access violations to MSRs can cause a #VE (or a #GP). As designed, a guest TD that wishes to access an MSR not allowed by Intel TDX (and that causes a #VE) must do so via explicit requests from the host VMM using TDCALL (TDG.VP.VMCALL).

Intel TDX helps enforce this by setting the MSR bitmaps (which are part of TDCS) to fixed values. Any disallowed access can cause a #VE to the TD guest. In some cases, the Intel TDX module sets the MSR bitmap for exit on WRMSR and performs access checks *e.g.*, in the case of IA32_XSS, the guest-TD software is meant to only be allowed to set bits are set in TDCS.XFAM (Extended-Features-Available Mask: can indicate the extended user and system features available for the TD).

2.3.3.3 #VE Injected due to CPUID invocation

CPUID leaves and sub-leaves faulting behavior are classified by Intel TDX as follows:

Faulting Guest-TD execution of CPUID with this leaf or sub-leaf access can unconditionally raise #VE.

Non-Faulting Guest-TD execution of CPUID with this leaf or sub-leaf access can return values (in guest EAX/EBX/ECX/EDX).

The guest TD may use the TDCALL (TDG.VP. CPUIDVE.SET) to toggle on or off the **faulting** (unconditional injection of #VE) on all CPUID leaves and sub-leaves, when CPUID is invoked in supervisor mode (CPL == 0) and/or user mode (CPL > 0). This enables the TD OS to control CPUID as seen by drivers or by user-level code. To support the control, the VCPU-scope, TDVPS-control structure is designed to hold flags, initially set to no #VE inject, that control that operation.

The bit fields of CPUID-return values, in guest EAX/EBX/ECX/EDX, of **non-faulting** leaf or sub-leaf are classified by Intel TDX architecture using the following virtualization types:

Table: CPUID-Field Virtualization for Non-Faulting, CPUID Leaves or Sub-Leaves

CPUID Field Virtualization	Description	Comments
Configurable	Bits fields for which the host VMM configures the value seen by the guest TD. Configuration is done on TDH.MNG.INIT.	There are several ways for the VMM to configure such bit fields. See Table below for details.
Allowable	Bits fields for which the host VMM configures, such that the guest TD either sees the field's native value or a value of 0. Configuration is done on TDH.MNG.INIT.	If the bit enumerates a CPU feature and the feature is natively supported, then the feature is designed to be either allowed or effectively deprecated for the guest TD. There are several ways for the VMM to configure such bit fields. See Table below for details.

"Configurable" and "Allowable" CPUID fields are specified by the host VMM at guest-TD-initialization time (using TDH.MNG.INIT) using the TD_PARAMS input structure of TDH.MNG.INIT and are further classified as follows:

Table: Configuration of "Configurable" and "Allowable" CPUID-Bit Fields on TDH.MNG.INIT

Configuration Mode	Description	TD_PARAMS Field
Directly	Bit fields configurable directly (CONFIG_DIRECT) or allowable directly (ALLOW_DIRECT) based on a configuration table	CPUID_CONFIG
XFAM	Bit fields configurable based on the guest TD's XFAM.	XFAM

Configuration Mode	Description	TD_PARAMS Field
	XFAM control of extended features' virtualization is described in Intel TDX-module architecture specification [3]	
ATTRIBUTES	Bit fields configurable based on the guest TD's ATTRIBUTES.	ATTRIBUTES

A list of directly configurable CPUID leaves and sub-leaves is enumerated by SEAMCALL(TDH.SYS.INFO) as described in Intel TDX-module specification [3].

At guest-TD-initialization time (TDH.MNG.INIT API function), the intent is for the host VMM to provide values for directly and indirectly configurable and allowable CPUID-bit fields as part of the TD_PARAMS-input structure.

TDH.MNG.INIT is designed to perform the following actions:

1. Verify the validity of the CPUID configuration (TD_PARAMS.CPUID_CONFIG)
2. Compute values for all CPUID leaves and sub-leaves that contain directly configurable and allowable CPUID-bit fields, based on the native-CPUID values and the TD_PARAMS-input structure, and store such computed values in TDCS.
3. CPUID-return values for bit fields configurable or allowable by XFAM or ATTRIBUTES are computed on VM exit from the guest TD

At guest TD runtime, Intel TDX is designed to handle VM exits from the guest TD due to CPUID execution as follows:

4. If the TDVPS flag for the current, guest CPL (supervisor or user) indicates an unconditional #VE, inject a #VE to the guest TD.
5. If the requested CPUID leaf or sub-leaf is defined as faulting, inject a #VE to the guest TD.
6. Emulate the CPUID instruction. Start with the return values pre-computed on guest-TD initialization, then calculate fields configured by XFAM or ATTRIBUTES and Dynamic fields, and return the values of the requested CPUID leaf and sub-leaf from the pre-computed set of CPUID values of the guest TD.

2.4 TDCALL instruction

This section describes the common functionality of TDCALL. Leaf functions are described in the following sections.

Table: TDCALL-Input Operands

Operand	Description
RAX	Leaf number. See enumeration below.
Other	See individual, TDCALL-leaf functions.

Table 2: TDCALL-Output Operands

<table>
<tr><th>Operand</th><th>Description</th></tr>
<tr><td>RAX</td><td>Instruction return code, indicating the outcome of execution of the instruction.
Intel TDX-Interface Functions' Completion Status in RAX @ Lowest-Details Level
<table>
<tr><th>RAX</th><th>Meaning</th><th>Description</th></tr>
<tr><td>0</td><td>Success</td><td>Function completed successfully</td></tr>
<tr><td>> 0
(0x00000000_00000001 – 0x7FFFFFFF_FFFFFFFF)</td><td>Informational / Warning</td><td>Function completed successfully but with some informational or warning code.</td></tr>
<tr><td>< 0
(0x80000000_00000000 – 0xFFFFFFFF_FFFFFFFF)</td><td>Error</td><td>Function aborted due to some error</td></tr>
</table>
</td></tr>
<tr><td>Other</td><td>See individual TDCALL leaf functions</td></tr>
</table>

The following TDCALL leaves are defined.

Opcode	Leaf	Leaf Num.	Description
TDCALL	TDG.VP.VMCALL	0	Performs a TD Exit to the host VMM to pass/receive information.
TDCALL	TDG.VP.INFO	1	Get TD-execution-environment information.
TDCALL	TDG.MR.RTMR.EXTEND	2	Extend a TDCS.RTMR-measurement register.
TDCALL	TDG.VP.VEINFO.GET	3	Get Virtualization-Exception Information for the recent, #VE exception. (See below)
TDCALL	TDG.MR.REPORT	4	Create a TDREPORT_STRUCT structure that contains the measurements/configuration information of the guest TD that called the function, measurements/configuration information of the Intel TDX module and a REPORTMACSTRUCT.
TDCALL	TDG.VP.CPUIDVE.SET	5	Controls unconditional #VE on CPUID execution by the guest TD.
TDCALL	TDG.MEM.PAGE.ACCEPT	6	Accept a pending, private page and initialize the page to 0 using the TD-ephemeral-private key.

Instruction Description

TDCALL is detailed in the Intel TDX-Architecture specification [1]. This document describes how TDCALL [TDG.VP.VMCALL] leaf functions are to be used by the TD.

On VM exit, Intel TDX module is designed to perform the following checks:

1. If the CPU mode is not 64b (IA32_EFER.LMA == 1), inject a #GP(0) fault to the guest TD;
2. If the leaf number in RAX is not supported, inject a #GP(0) fault to the guest TD.

If all checks pass, Intel TDX module is designed to call the leaf function according to the **leaf number in RAX**.

Memory addresses passed via registers are specified as GPA. Intel TDX-module accesses are meant to be subject to normal, GPA-to-HPA-address-translation rules. Whether the flow uses a Private or a Shared semantics can be determined by the SHARED bit of the GPA.

To simplify the flow logic, operands of guest-side flows are intended to be contained within a single, 4KB page.

Completion-Status Codes

Table: TDCALL-Completion-Status Codes (Returned in RAX)

Completion-Status Code	Description
TDX_SUCCESS	TDCALL is successful
TDX_OPERAND_INVALID	Illegal leaf number
Other	See individual leaf functions

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:

TDG.VP.VMCALL-Input Operands

Operand	**Description**
RAX	TDCALL instruction leaf number (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. A bit value of 0 indicates that the corresponding register is saved by Intel TDX module, *i.e.*, scrubbed to 0 before SEAMRET to the host VMM, and restored by Intel TDX module on the following TDENTER. A bit value of 1 indicates that the corresponding register is passed as-is to the host VMM, and, on the following TDENTER, the register value is used as input from the host VMM and passed as-is to the guest TD. The value of RCX itself is always passed to the host VMM. Bits Name Description 15:0 GPR Mask Controls the transfer of GPR values: Bit 0: RAX (must be 0) Bit 1: RCX (must be 0) Bit 2: RBX Bit 3: RDX Bit 4: RSP (must be 0) Bit 5: RBP Bit 6: RSI Bit 7: RDI Bit 8: R8 Bit 9: R9 Bit 10: R10 (must be 1) Bit 11: R11 (must be 1) Bits 12:15: R12-R15 Bits 31:16: XMM Mask - Controls the transfer of XMM0 – XMM15 registers Bits 63:32: Reserved, always 0
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)

TDG.VP.VMCALL-Output Operands

Operand	Description
RAX	TDCALL instruction return code. Always returns Intel TDX_SUCCESS (0).
RCX	Unmodified
R10	**TDG.VP.VMCALL return value (STATUS_CODE)** 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 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

TDG.VP.VMCALL-Instruction-execution sub-functions

Sub-Function Number Bits 15:0	Sub-Function Name
10	Instruction.CPUID

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

Note that some instructions that unconditionally cause #VE (such as WBINVD, MONITOR, MWAIT) do not have corresponding TDCALL [TDG.VP.VMCALL <Instruction>] leaves, since the TD has been designed with no deterministic way to confirm the result of those operations performed by the host VMM. In those cases, the goal is for the TD #VE handler to increment the RIP appropriately based on the VE information provided via TDCALL [TDG.VP.VEINFO.GET].

Completion-Status Codes

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

Completion-Status Code	Value	Description
TDX_SUCCESS	0x0	TDCALL is successful
TDX_OPERAND_INVALID	0x80000000 00000000	Illegal leaf number
Other	See individual leaf functions	

2.4.2 TDCALL [TDG.VP.INFO] leaf

Means to help get guest-TD-execution-environment information not provided by CPUID.

Table: TDG.VP.INFO-Input Operands

Operand	Description
RAX	TDCALL-instruction-leaf number 1

Table: TDG.VP.INFO-Output Operands

<table>
<tr><th>Operand</th><th colspan="3">Description</th></tr>
<tr><td>RAX</td><td colspan="3">TDCALL-instruction-return code, returns TDX_SUCCESS (0).</td></tr>
<tr><td rowspan="3">RCX</td><td>Bits</td><td>Name</td><td>Description</td></tr>
<tr><td>5:0</td><td>GPAW</td><td>The effective GPA width (in bits) for this TD (do not confuse with MAXPA). SHARED bit is at GPA bit GPAW-1.
Only GPAW values 48 and 52 are possible.</td></tr>
<tr><td>63:6</td><td>RESERVED</td><td>Reserved, always 0</td></tr>
<tr><td>RDX</td><td colspan="3">The TD's ATTRIBUTES (provided as input to TDINIT)</td></tr>
<tr><td rowspan="3">R8</td><td>Bits</td><td>Name</td><td>Description</td></tr>
<tr><td>31:0</td><td>NUM_VCPUS</td><td>Number of Virtual CPUs that are usable, i.e., either active or ready</td></tr>
<tr><td>63:32</td><td>MAX_VCPUS</td><td>TD's maximum number of Virtual CPUs (provided as input to TDINIT)</td></tr>
<tr><td>R9</td><td colspan="3">Reserved for enumerating future, Intel TDX capabilities, etc. Set to 0.</td></tr>
<tr><td>R10</td><td colspan="3">Reserved for enumerating future, Intel TDX capabilities, etc. Set to 0.</td></tr>
<tr><td>R11</td><td colspan="3">Reserved for enumerating future, Intel TDX capabilities, etc. Set to 0.</td></tr>
<tr><td>Other</td><td colspan="3">Unmodified</td></tr>
</table>

Leaf-Function Description

TDG.VP.INFO is designed to provide the TD-guest software with execution-environment information, beyond information provided by CPUID.

Completion-Status Codes

Table: TDG.VP.INFO-Completion-Status Codes (Returned in RAX)

Completion-Status Code	Value	Description
TDX_SUCCESS	0x0	TDG.VP.INFO is successful

2.4.3 TDCALL [TDG.MR.RTMR.EXTEND] leaf

This TDCALL is designed to extend a TDCS.RTMR-measurement register.

Table: TDG.MR.RTMR.EXTEND-Input Operands

Operand	Description
RAX	TDCALL-instruction leaf number 2
RCX	64B-aligned, guest-physical address of 48B-extension data
RDX	Index of the measurement register to be extended

Table: TDG.MR.RTMR.EXTEND-Output Operands

Operand	Description
RAX	TDCALL-instruction-return code; see Table 2: TDCALL-Output Operands
Other	Unmodified

Leaf-Function Description

This function helps extend one of the specified, RTMR-measurement registers in the TD's TDCS with the provided, extension data in memory. Extension is enabled by calculating SHA384 hash over a 96B-buffer composed as follows: Bytes 0 through 47 contain the current, RTMR value; Bytes 48 through 95 contain the extension data.

Completion-Status Codes

Table: TDG.MR.RTMR.EXTEND-Completion-Status Codes (Returned in RAX)

Completion-Status Code		Description
TDX_OPERAND_INVALID	0x80000000 00000000	E.g. invalid RTMR index or mis-aligned GPA provided for extension data
TDX_SUCCESS	0x0	TDG.MR.RTMR.EXTEND is successful

2.4.4 TDCALL [TDG.VP.VEINFO.GET] leaf

Means to help get Virtualization-Exception Information for the recent, #VE exception.

Table: TDG.VP.VEINFO.GET-Input Operands

Operand	Description
RAX	TDCALL-instruction-leaf number 3

Table: TDG.VP.VEINFO.GET-Output Operands

<table>
<tr><th>Operand</th><th colspan="3">Description</th></tr>
<tr><td>RAX</td><td colspan="3">TDCALL-instruction-return code – See Table 2: TDCALL-Output Operands</td></tr>
<tr><td rowspan="3">RCX</td><td>Bits</td><td>Name</td><td>Description</td></tr>
<tr><td>31:0</td><td>Exit Reason</td><td>The 32-bit value that would have been saved into the VMCS as an exit reason had a VM exit occurred instead of the virtualization exception.</td></tr>
<tr><td>63:32</td><td>Reserved</td><td>Reserved, always 0</td></tr>
<tr><td>RDX</td><td colspan="3">Exit Qualification, the 64-bit value that would have been saved into the VMCS as an exit qualification had a legacy VM exit occurred instead of the virtualization exception</td></tr>
<tr><td>R8</td><td colspan="3">Guest-Linear Address, the 64-bit value that would have been saved into the VMCS as a guest-linear address, had a legacy-VM exit occurred instead of the virtualization exception</td></tr>
<tr><td>R9</td><td colspan="3">Guest-Physical Address, the 64-bit value that would have been saved into the VMCS as a guest-physical address, had a legacy VM exit occurred instead of the virtualization exception</td></tr>
<tr><td rowspan="3">R10</td><td>Bits</td><td>Name</td><td>Description</td></tr>
<tr><td>31:0</td><td>VM-exit-instruction length</td><td>The 32-bit value that would have been saved into the VMCS as VM-exit-instruction length, had a legacy VM exit occurred instead of the virtualization exception</td></tr>
<tr><td>63:32</td><td>VM-exit-instruction information</td><td>The 32-bit value that would have been saved into the VMCS as VM-exit-instruction information, had a legacy VM exit occurred instead of the virtualization exception</td></tr>
<tr><td>Other</td><td colspan="3">Unmodified</td></tr>
</table>

Leaf-Function Description

TDG.VP.VEINFO.GET returns the virtualization-exception information corresponding to a #VE exception that was previously delivered to the guest TD.

Completion-Status Codes

Table: TDG.VP.VEINFO.GET-Completion-Status Codes (Returned in RAX)

Completion-Status Code	Value	Description
TDX_NO_VE_INFO	0x80000000 00000000	There is no Virtualization-Exception information.
TDX_SUCCESS	0x0	TDG.VP.VEINFO.GET is successful

2.4.5 TDCALL [TDG.MR.REPORT] leaf

This TDCALL is designed to create a TDREPORT_STRUCT structure that contains the measurements/configuration information of the guest TD that called the function, measurements/configuration information of the Intel TDX module and a REPORTMACSTRUCT.

Table: TDG.MR.REPORT-Input Operands

Operand	Description
RAX	TDCALL-instruction-leaf number 4
RCX	1024B-aligned, guest-physical address where the report structure will be created
RDX	64B-aligned, guest-physical address of additional REPORTDATA to be signed
R8	Report sub-type (Must be 0)

Table: TDG.MR.REPORT-Output Operands

Operand	Description
RAX	TDCALL-instruction-return code – See Table 2: TDCALL-Output Operands
Other	Unmodified

Leaf-Function Description

This function is designed to create a TDREPORT_STRUCT structure that contains the measurements/configuration information of the guest TD that called the function, measurements/configuration information of the Intel TDX module and a REPORTMACSTRUCT. The REPORTMACSTRUCT is designed integrity-protected with a MAC and containing the hash of the measurements and configuration as well as additional REPORTDATA (64-byte value) provided by the TD software during the TDCALL. See Intel TDX-module-architecture specification [3] for details on the TDREPORT_STRUCT received via this TDCALL. This structure is meant to be used to generate a Quote for the TD for remote attestation. The aim is for the quote to be generated via a TDG.VP.VMCALL [GetQuote] described in a later section in this document.

The function is not designed to do the following:

1. Assemble a report-type structure based on the report sub-type provided in R8
2. Assemble the output report's TDG.VP.INFO fields from the TDCS-reported fields (ATTRIBUTES, XFAM, MRTD, MRCONFIGID, MROWNER, MROWNERCONFIG, and RTMRs)
3. Calculate a SHA384 hash over TDG.VP.INFO
4. Execute SEAMOPS [SEAMREPORT] to generate the MAC on the report based on the input REPORTDATA (RDX), the TDG.VP.INFO hash calculated above, and the report-type structure
5. Write the output report to memory at the GPA specified in RCX

Completion-Status Codes

Table: TDG.MR.REPORT-Completion-Status Codes (Returned in RAX)

Completion-Status Code	Value	Description
TDX_OPERAND_BUSY	0x1	The Intel TDX module may return a busy indicator if it cannot satisfy the TDCALL
TDX_OPERAND_INVALID	0x80000000 00000000	Invalid operands specified to the TDCALL – for example, the GPA specified may not be mapped to the TD
TDX_SUCCESS	0x0	TDG.MR.REPORT is successful

2.4.6 TDCALL [TDG.VP.CPUIDVE.SET] leaf

This TDCALL is designed to control unconditional #VE on CPUID execution by the guest TD.

Table: TDG.VP.CPUIDVE.SET Input Operands

<table>
<tr><th>Operand</th><th colspan="3">Description</th></tr>
<tr><td>RAX</td><td colspan="3">TDCALL-instruction-leaf number 5</td></tr>
<tr><td rowspan="5">RCX</td><td colspan="3">Control whether CPUID executed by the guest TD will cause #VE unconditionally</td></tr>
<tr><td>Bits</td><td>Name</td><td>Description</td></tr>
<tr><td>0</td><td>SUPERVISOR</td><td>Flag that a CPUID executed by the guest TD when CPL is 0 will cause a #VE unconditionally</td></tr>
<tr><td>1</td><td>USER</td><td>Flag that a CPUID executed by the guest TD when CPL > 0 will cause a #VE unconditionally</td></tr>
<tr><td>63:2</td><td>RESERVED</td><td>Reserved, must be 0</td></tr>
</table>

Table: TDG.VP.CPUIDVE.SET-Output Operands

Operand	Description
RAX	TDCALL-instruction-return code – See Table 2: TDCALL-Output Operands
Other	Unmodified

Leaf-Function Description

This function helps control whether execution of CPUID by the guest TD will unconditionally result in a #VE when running in supervisor mode and/or in user mode.

Completion-Status Codes

Table: TDG.VP.CPUIDVE.SET-Completion-Status Codes (Returned in RAX)

Completion-Status Code	Value	Description
TDX_OPERAND_INVALID	0x80000000 00000000	
TDX_SUCCESS	0x0	TDG.VP.CPUIDVE.SET is successful

2.4.7 TDCALL [TDG.MEM.PAGE.ACCEPT] leaf

This TDCALL is designed to accept a pending, private page and initialize the page to 0 using the TD-ephemeral-private key.

Table: TDG.MEM.PAGE.ACCEPT-Input Operands

Operand	Description
RAX	TDCALL-instruction-leaf-number 6
RCX	Guest-physical address of the private page to accept
RDX	GPA page size value 0 = 4KB, 1= 2MB, 3 = 1GB

Table: TDG.MEM.PAGE.ACCEPT-Output Operands

Operand	Description
RAX	TDCALL-instruction-return code – See Table 2: TDCALL-Output Operands
Other	Unmodified

Leaf-Function Description

Designed to accept a pending, private-TD page previously added into the TD-GPA-address space by the host VMM via TDH.MEM.PAGE.AUG (See Intel TDX-module-Architecture specification for memory-management details) into the TD. As part of this flow, Intel TDX-module initializes the page to 0.

Completion-Status Codes

Table: TDG.MEM.PAGE.ACCEPT-Completion-Status Codes (Returned in RAX)

Completion-Status Code	Value	Description
TDX_OPERAND_INVALID	0x80000000 00000000	Invalid operand passed in – for example, the GPA may not be assigned to the TD by the host VMM or be in a pending state
TDX_PAGE_ALREADY_ACCEPTED	0x80000000 00000001	GPA was not in already in a valid state in the secure-EPT structure. See [3] for details on the secure-EPT management.
TDX_PAGE_SIZE_INVALID	0x80000000 00000002	GPA mapping was found to be smaller than pending S-EPT mapping for GPA
TDX_SUCCESS	0x0	TDG.MEM.PAGE.ACCEPT is successful

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) and resumes the TD via a SEAMCALL [TDH.VP.ENTER] invocation. Refer to the Intel TDX-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.

TDG.VP.VMCALL< GetTdVmCallInfo>-Input Operands

Operand	Description
R11	TDG.VP.VMCALL< GetTdVmCallInfo> sub-function per Table 2
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.

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)

TDG.VP.VMCALL< GetTdVmCallInfo> Status Codes

Error Code	Value	Description
TDG.VP.VMCALL_SUCCESS	0x0	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 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 is designed to be able to 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 no stale mappings and cache contents exist. 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.

TDG.VP.VMCALL<MapGPA>-Input Operands

Operand	Description
R11	TDG.VP.VMCALL<MapGPA> sub function per Table 2
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)

TDG.VP.VMCALL<MapGPA> Output Operands

Operand	Description
R10	TDG.VP.VMCALL-instruction-return code.
R11	GPA at which MapGPA failed

TDG.VP.VMCALL<MapGPA>-Status Codes

Error Code	Value	Description
TDG.VP.VMCALL_SUCCESS	0x0	TDG.VP.VMCALL is successful. The TD is free to use the GPA (range) specified
TDG.VP.VMCALL_INVALID_OPERAND	0x80000000 00000000	Invalid operand – for example, the GPA may be already mapped as a shared page.

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 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 asysnchronous service requested. For the GetQuote operation, the goal is the TDREPORT_STRUCT be received by the TD via a prior TDCALL[TDG.MR.REPORT] in a 1024-byte 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 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).

TDG.VP.VMCALL< GetQuote >-Input Operands

Operand	Description
R11	TDG.VP.VMCALL< GetQuote > sub-function per Table 2
R12	Shared 4KB GPA as input – the memory contains a TDREPORT_STRUCT.

TDG.VP.VMCALL< GetQuote >-Output Operands

Operand	Description
R10	TDG.VP.VMCALL return code.

TDG.VP.VMCALL< GetQuote >-Status Codes

Error Code	Value	Description
TDG.VP.VMCALL_SUCCESS	0x0	TDG.VP.VMCALL is successful
TDG.VP.VMCALL_INVALID_OPERAND	0x80000000 00000000	Invalid operand – for example, the GPA may be mapped as a private page.
TDG.VP.VMCALL_TDREPORT_FAILED	0x80000000 00000001	Quoting service indicates the TDREPORT has failed. TD may generate a new TDREPORT_STRUCT (via TDCALL[TDG.MR.REPORT]) and retry the operation.

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 debug 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 help prevent causing a #VE.

TDG.VP.VMCALL< ReportFatalError >-Input Operands

Operand	Description
R11	TDG.VP.VMCALL< ReportFatalError > sub-function per Table 2
R12	Error Code

TDG.VP.VMCALL<FatalError>-Output Operands

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

TDG.VP.VMCALL< ReportFatalError >-Status Codes

Error Code	Value	Description
TDG.VP.VMCALL_SUCCESS	0x0	TDG.VP.VMCALL is successful

3.5 TDG.VP.VMCALL<SetupEventNotifyInterrupt>

The guest TD may request 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 to not 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 via the posted-interrupt descriptor. See Intel TDX-module specification [3] for TD-interrupt handling.

TDG.VP.VMCALL< SetupEventNotifyInterrupt>-Input Operands

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

TDG.VP.VMCALL< SetupEventNotifyInterrupt >-Output Operands

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

TDG.VP.VMCALL< SetupEventNotifyInterrupt >-Status Codes

Error Code	Value	Description
TDG.VP.VMCALL_SUCCESS	0x0	TDG.VP.VMCALL is successful
TDG.VP.VMCALL_INVALID_OPERAND	0x80000000 00000000	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.

TDG.VP.VMCALL<Instruction.CPUID>-Input Operands

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

TDG.VP.VMCALL<Instruction.CPUID>-Output Operands

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

TDG.VP.VMCALL<Instruction.CPUID>-Status Codes

Error Code	Value	Description
TDG.VP.VMCALL_SUCCESS	0x0	TDG.VP.VMCALL is successful
TDG.VP.VMCALL_INVALID_OPERAND	0x80000000 00000000	Invalid CPUID requested

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 2.3.1) 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.

TDG.VP.VMCALL<RequestMMIO>-Input Operands

Operand	Description
R11	TDG.VP.VMCALL<RequestMMIO> sub-function per Table 2
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.

TDG.VP.VMCALL<Instruction.MMIO>-Output Operands

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

TDG.VP.VMCALL<Instruction.MMIO>-Status Codes

Error Code	Value	Description
TDG.VP.VMCALL_SUCCESS	0x0	TDG.VP.VMCALL is successful
TDG.VP.VMCALL_INVALID_OPERAND	0x80000000 00000000	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.

TDG.VP.VMCALL<Instruction.HLT>-Input Operands

Operand	Description
R11	TDG.VP.VMCALL<Instruction.HLT>-Instruction-execution sub-functions per Table 2

TDG.VP.VMCALL<Instruction.HLT>-Output Operands

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

TDG.VP.VMCALL<Instruction.HLT>-Status Codes

Error Code	Value	Description
TDG.VP.VMCALL_SUCCESS	0x0	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.

TDG.VP.VMCALL<Instruction.IO>-Input Operands

Operand	Description
R11	TDG.VP.VMCALL<Instruction.IO>-Instruction-execution sub-functions per Table 2
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.

TDG.VP.VMCALL<Instruction.IO>-Output Operands

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

TDG.VP.VMCALL<Instruction.IO>-Status Codes

Error Code	Value	Description
TDG.VP.VMCALL_SUCCESS	0x0	TDG.VP.VMCALL is successful
TDG.VP.VMCALL_INVALID_OPERAND	0x80000000 00000000	Invalid-IO-Port access

3.10 TDG.VP.VMCALL<Instruction.RDMSR>

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

TDG.VP.VMCALL<Instruction.RDMSR>-Input Operands

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

TDG.VP.VMCALL<Instruction.RDMSR>-Output Operands

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

TDG.VP.VMCALL<Instruction.RDMSR>-Status Codes

Error Code	Value	Description
TDG.VP.VMCALL_SUCCESS	0x0	TDG.VP.VMCALL is successful
TDG.VP.VMCALL_INVALID_OPERAND	0x80000000 00000000	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.

TDG.VP.VMCALL<Instruction.WRMSR>-Input Operands

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

TDG.VP.VMCALL<Instruction.WRMSR>-Output Operands

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

TDG.VP.VMCALL<Instruction.WRMSR>-Status Codes

Error Code	Value	Description
TDG.VP.VMCALL_SUCCESS	0x0	TDG.VP.VMCALL is successful
TDG.VP.VMCALL_INVALID_OPERAND	0x80000000 00000000	Invalid MSR rd/wr requested or access-denied

3.12 TDG.VP.VMCALL<Instruction.PCONFIG>

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

TDG.VP.VMCALL<Instruction.PCONFIG>-Input Operands

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

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.

TDG.VP.VMCALL<Instruction.PCONFIG>-Status Codes

Error Code	Value	Description
TDG.VP.VMCALL_SUCCESS	0x0	TDG.VP.VMCALL is successful
TDG.VP.VMCALL_INVALID_OPERAND	0x80000000 00000000	If PCONFIG-operation requested is invalid

4 TD-Guest-Firmware Interfaces

4.1 ACPI-MADT-AP-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.

During system boot, the guest firmware is designed to put the guest-application processors in a state to check a shared mailbox that is a 4K-aligned, 4K-size, memory block allocated by the firmware in the ACPI Nvs memory. The guest firmware is not designed to be able to modify the mailbox location when the guest firmware transfers the control to the guest-OS loader. The layout of the mailbox breaks down into two 2KB sections: a guest-OS section and a guest-firmware section. The guest-OS section is designed to only be written by guest OS and read by the guest firmware, except the command field. As designed, the guest application processor must clear the command to Noop(0) to acknowledge when the command is received. The guest firmware helps cache the content in the mailbox which content might be used later before clearing the command, such as WakeupVector. The goal is that, only after the command is changed to Noop(0), can the guest OS send the next command. The guest-firmware section should be considered read-only to the guest OS and only be written to by the guest firmware. All data communication between the guest OS and guest firmware should be in little-endian format.

The guest-OS section is devised to contain command, flags, APIC ID, and a wakeup address. After the guest OS detects the processor number from the MADT table, the guest OS may prepare the wakeup routine, fill the wakeup-address field in the mailbox, indicate which guest-virtual processor needs to be woken up in the APID-ID field, and send the wakeup command. Once a guest-application processor detects the wakeup command and its own APIC ID, the guest application processor is designed to jump to the guest-OS-provided, wakeup address and ignore the command if the APIC ID does not match its own.

For each TD-guest, application processor, the mailbox can be used once for the wakeup command. After the guest-application processor takes the action according to the command, the intention is the mailbox will no longer be checked by the guest-application processor. Other guest processors can continue using the mailbox for the next command.

Table 4: MPWakeup Structure

Field	Byte Length	Byte Offset	Description
Type	1	0	0x10 Multiprocessor-Wakeup structure
Length	1	1	16
MailBoxVersion	2	2	Version of the mailbox. 0 for this version.
Reserved	4	4	Must be 0.

MailBoxAddress	8	8	Physical address of the mailbox. It must be in ACPI Nvs. It must be 4K bytes aligned.

Table 5: MP-Wakeup Mailbox

Field	Byte Length	Byte Offset	Description
Command	2	0	0: Noop - no operation. 1: Wakeup – jump to the wakeup vector. 2~0xFFFF: Reserved
Reserved	2	2	Must be 0.
ApicId	4	4	The processor's local, APIC ID. The application processor will check if the ApicId field matches its own APIC ID. The application processor will ignore the command in case of APIC-ID mismatch.
WakeupVector	8	8	The wakeup address for application processor(s). For Intel processor, the execution environment is: Interrupts must be disabled. RFLAGES.IF set to 0. Long mode enabled. Paging mode is enabled and physical memory for waking vector is identity mapped (virtual address equals physical address). Waking vector must be contained within one physical page. Selectors are set to flat and otherwise not used.
ReservedForOs	2032	16	Reserved for OS use.
ReservedForFirmware	2048	2048	Reserved for firmware use.

4.2 UEFI Memory Map

The memory in the TD guest-environment can be:

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

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

3) Uninitialized memory - TDH.MEM.PAGE.AUG by VMM and not accepted by TDVF yet.

4) Memory-mapped IO (MMIO) - accessed via TDVMCALL<#VE.RequestMMIO> only.

If a TD-memory region is private memory, the TD owner shall have the final, UEFI-memory map report the region with `EFI_MEMORY_CPU_CRYPTO` attribute.

If a TD-memory region is shared memory, the TD owner shall have the final, UEFI-memory map report the region in `EfiBootServicesData` without `EFI_MEMORY_CPU_CRYPTO` attribute.

If a TD-memory region is uninitialized memory and requires TDACCEPTPAGE in the TD guest OS, then the TD owner shall have the final, UEFI-memory map report this region in `EfiReservedMemoryType` without `EFI_MEMORY_CPU_CRYPTO` attribute.

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 is designed to be reported in ACPI ASL code via memory-resource descriptors.

Table 6: TDVF-memory map for OS

UEFI Memory Type	Memory Attribute	Usage	TD-Memory Type	OS Action
EfiReservedMemoryTpe	CPU_CRYPTO	Firmware-Reserved region, such as flash.	Private	Reserved.
EfiReservedMemoryTpe	Without CPU_CRYPTO	Uninitialized Memory	Uninitialized	Use after accepted, private pages via TDCALL [TDG.MEM.PAGE. ACCEPT]
EfiLoaderCode	CPU_CRYPTO	UEFI-Loader Code	Private	Use after EBS.

EfiLoaderData	CPU_CRYPTO	UEFI-Loader Data	Private	Use after EBS.
EfiBootServicesCode	CPU_CRYPTO	UEFI-Boot-Service Code	Private	Use after EBS.
EfiBootServicesData	CPU_CRYPTO	UEFI-Boot-Service Data	Private	Use after EBS.
EfiBootServicesData	**Without CPU_CRYPTO**	UEFI-Boot-Service Data (Shared Memory) VMM-shared buffer.	Shared	Use after EBS and converting to private page. ============== TDCALL[TDG.VP.VMCALL] <MapGPA> TDCALL[TDG.MEM.PAGE.ACCEPT]
EfiRuntimeServicesCode	CPU_CRYPTO | RUNTIME	UEFI-Runtime-Service Code	Private	Map-virtual address. Reserved.
EfiRuntimeServicesData	CPU_CRYPTO | RUNTIME	UEFI-Runtime-Service Data	Private	Map-virtual address. Reserved.
EfiConventionalMemory	CPU_CRYPTO	Freed memory (Private)	Private	Use directly.
EfiACPIReclaimMemory	CPU_CRYPTO	ACPI table.	Private	Use after copy ACPI table.
EfiACPIMemoryNVS	CPU_CRYPTO	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.	Shared	Access via TDCALL[TDG.VP.VMCALL] <#VE.RequestMMIO>
EfiPersistentMemory		For NVDIMM only. Unsupported in this version.		Reserved.

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 TD-measurement register.

Table 7: TD-Event-Log-PCR-Index Interpretation

TD-Register Index	TDX-measurement register
0	MRTD
1	RTMR[0]
2	RTMR[1]
3	RTMR[2]
4	RTMR[3]

4.3.2 EFI_TD_PROTOCOL

If TD-Guest Firmware supports measurement, the TD Guest Firmware is designed to produce EFI_TD_PROTOCOL with new GUID **TD_TD_PROTOCOL_GUID** to report event log and provide hash capability.

**#define TD_TD_PROTOCOL_GUID **

{0x96751a3d, 0x72f4, 0x41a6, {0xa7, 0x94, 0xed, 0x5d, 0xe, 0x67, 0xae, 0x6b}}

4.3.3 TD-Event Log

TDVF may set up an ACPI table to pass the event-log information. The event log created by the TD owner contains the hashes to reconstruct the MRTD and RTMR registers.

Table 8: Intel TDX-Event-Log, ACPI Table

Field	Byte Length	Byte Offset	Description
Header			
Signature	4	0	‘TDEL’ Signature.
Length	4	4	Length, in bytes, of the entire Table

Revision	1	8	1
Checksum	1	9	Entire table must sum to zero.
OEMID	6	10	Standard ACPI header
OEM Table ID	8	16	Standard ACPI header
OEM Revision	4	24	Standard ACPI header
Creator ID	4	28	Standard ACPI header
Creator Revision	4	32	Standard ACPI header
Reserved	4	36	Reserved. Must be 0.
Log-Area-Minimum Length (LAML)	8	40	Identifies the minimum length (in bytes) of the system's pre-boot-TD-event-log area
Log-Area-Start Address (LASA)	8	48	Contains the 64-bit-physical address of the start of the system's pre-boot-TD-event-log area in QWORD format. Note: The log area ranges from address LASA to LASA+(LAML-1).

4.4 Storage-Volume-Key Data

In 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 ACPI table (proposed below).

Table 9: Storage-Volume-Key-Location-ACPI Table

Field	Byte Length	Byte Offset	Description
Header			
Signature	4	0	'SVKL' Signature.
Length	4	4	Length, in bytes, of the entire Table
Revision	1	8	1
Checksum	1	9	Entire table must sum to zero.

OEMID	6	10	Standard ACPI header
OEM Table ID	8	16	Standard ACPI header
OEM Revision	4	24	Standard ACPI header
Creator ID	4	28	Standard ACPI header
Creator Revision	4	32	Standard ACPI header
Key Count (C)	4	36	The count of key structure
Key Structure	16 * C	40	The key structure

Table 10: Storage-Volume-Key Structure

Field	Byte Length	Byte Offset	Description
Key Type	2	0	The type of the key. 0: the main storage volume key 1~0xFFFF: reserved.
Key Format	2	2	The format of the key. 0: raw binary. 1~0xFFFF: reserved.
Key Size	4	4	The size of the key in bytes.
Key Address	8	8	The guest-physical address (GPA) of the key. The address must be in ACPI-Reserved Memory.

5 TD-VMM-Communication Scenarios

5.1 Requesting IPIs

Various. TD-VMM-communication scenarios require the TD to request 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. Such a TD-guest OS can use TDG.VP.VMCALL(MapGPA) so that, within this fixed-GPA map, the TD OS can request 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 desire 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 primaryshared 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 that 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 <Instruction.MMIO>]. From that point on, like the previous paravirtualized IO model, the TD software must ensure that the data buffers passed via memory referenced in parameters that are passed 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.

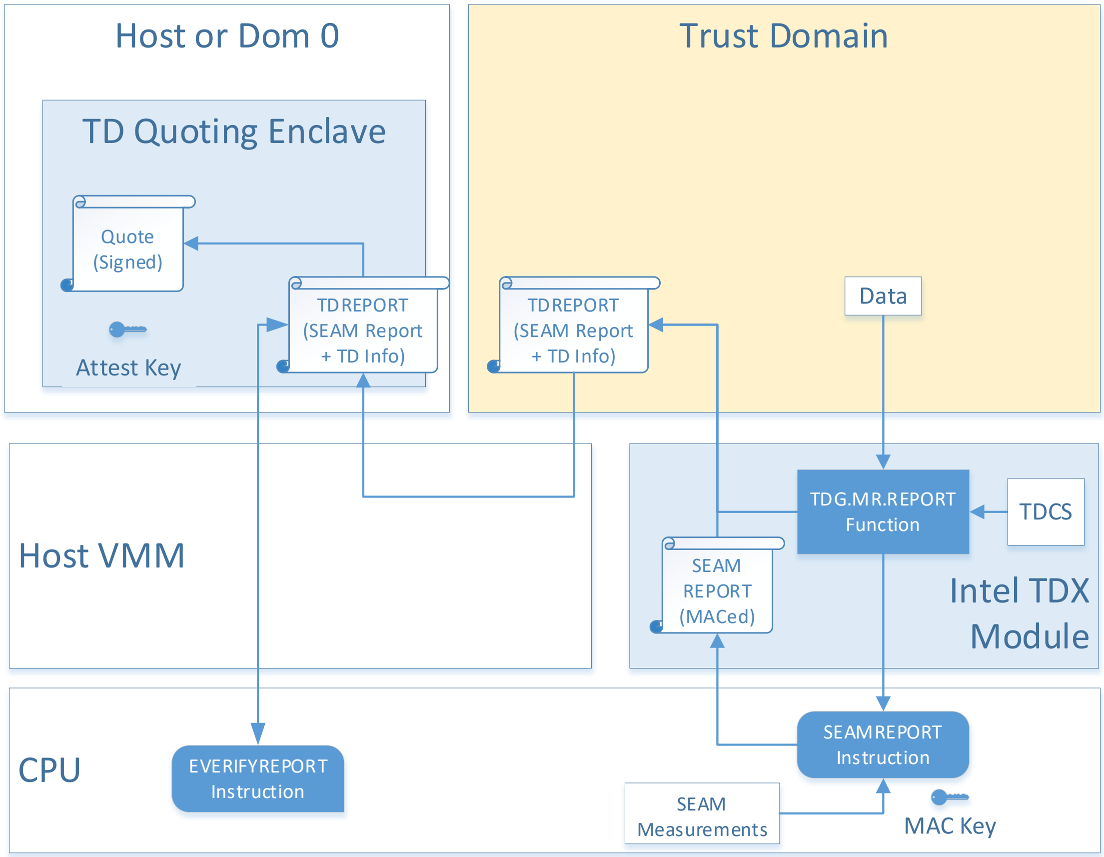

Figure 3: TD-Attestation flow

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.

6 Appendix – TD-Guest-MSR accesses

MSRs listed here are either context switched during TD-VM Enter/Exit and/or the guest can read/write (as specified below) - some are qualified accesses based on if the feature is exposed via CPUID exposed to the TD guest or features enabled via XFAM during TD initialization.

MSR Architectural Name	Access and qualification
IA32_TIME_STAMP_COUNTER	Read-only
IA32_SPEC_CTRL	Read-write
IA32_PRED_CMD	Read-write
IA32_PMCx	Based on TD PERFMON attribute
IA32_UMWAIT_CONTROL	Based on virtualized CPUID(7,0).ECX[5])
IA32_ARCH_CAPABILITIES	Read-write
IA32_FLUSH_CMD	Read-write
IA32_SYSENTER_CS	Read-write
IA32_SYSENTER_ESP	Read-write
IA32_SYSENTER_EIP	Read-write
IA32_PERFEVTSELx	Based on TD PERFMON attribute
IA32_MISC_ENABLE	Read-only Based on TD PERFMON attribute
MSR_OFFCORE_RSPx	Based on TD PERFMON attribute
IA32_DEBUGCTL	Read-write for debug TD with restrictions (See Intel TDX Module FAS)
IA32_PAT	Read-write
IA32_FIXED_CTRx	Based on TD PERFMON attribute
IA32_PERF_METRICS	Based on TD PERFMON attribute
IA32_PERF_CAPABILITIES	Read-only based on TD PERFMON attribute || Based on XFAM[8]
IA32_FIXED_CTR_CTRL	Based on TD PERFMON attribute
IA32_PERF_GLOBAL_*	Based on TD PERFMON attribute
IA32_PEBS_ENABLE	Based on TD PERFMON attribute
IA32_A_PMCx	Based on TD PERFMON attribute
IA32_RTIT_*	Based on XFAM[8]
IA32_DS_AREA	Read-write
IA32_U_CET	Based on (XFAM[11] | XFAM[12])
IA32_S_CET	Based on (XFAM[11] | XFAM[12])
IA32_PL*_SSP	Based on TD XFAM[11] | XFAM[12])
IA32_INTERRUPT_SSP_TABLE_ADDR	Based on TD XFAM[11] | XFAM[12])
IA32_PKRS	Based on TD PKS ATTRIBUTE
IA32_X2APIC_TPR	Read-write
IA32_X2APIC_PPR	Read-write
IA32_X2APIC_EOI	Read-write
IA32_X2APIC_ISRx	Read-write
IA32_X2APIC_TMRx	Read-write
IA32_X2APIC_IRRx	Read-write
IA32_X2APIC_SELF_IPI	Read-write
IA32_DEBUG_INTERFACE	Read-only
IA32_XSS	Read-only
IA32_EFER	Read-only
IA32_STAR	Read-write
IA32_LSTAR	Read-write
IA32_FMASK	Read-write

IA32_FSBASE	Read-write
IA32_GSBASE	Read-write
IA32_KERNEL_GS_BASE	Read-write
IA32_TSC_AUX	Read-write

By design, any MSRs **not** listed here are disallowed to be read/written to and generate a #VE or #GP on RD/WRMSR (based on whether the MSR can be enabled for the feature via CPUID exposed to the TD guest or features enabled via XFAM during TD initialization).