Guest Hypervisor Communication Interface (GHCI) for Intel® Trust Domain Extensions (Intel® TDX) 2.0 extension

351995-001US (DRAFT)

May 2025

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

1 About this Document

1.1 Scope of this Document

Intel® TDX 2.0 adds Intel® TDX Connect architecture to support direct I/O device assignment. The GHCI 2.0 APIs for Intel® TDX Connect are defined in 3.2.1 TDG.VP.VMCALL <Service.TDCM>. A brief Intel® TDX Connect architecture introduction is in 4.1 TDX Connect Architecture.

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 References

Table 1-1: Technical Documents Referenced

#	Reference Document	Version & Date
1	Intel® Trust Domain Extensions module 2.0 TDX Connect architecture specification	January 2023
2	Device Attestation Model in Confidential Computing Environment	September 2022
3	Software Enabling for Intel® TDX in Support of TEE-IO	September 2022
4	SPDM specification (DSP0274), version 1.2.1	June 2022
5	Secured Message using SPDM specification (DSP0277), version 1.2.0	June 2024
6	PCI Express Base specification, version 6.2	January 2024

Unless otherwise specified, any reserved, undefined or unassigned values in enumerations or other numeric ranges are reserved for future definition by this specification.

Unless otherwise specified, field values marked as Reserved shall be written as zero, ignored when read, not modified, and not interpreted as an error if not zero.

2 TD-VMM Communication

2.1 TDCALL and SEAMCALL instruction

2.1.1 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 codes

Sub-Function Number	Sub-Function Name
0x10007	TDCM (TEE-IO Device Control and Management)

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.

Table 3-1: 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). If R12 is 1, bitmap for VMCALL codes **BIT4: <TDCM>**

3.2 TDG.VP.VMCALL <Service> (To be deprecated)

3.2.1 TDG.VP.VMCALL <Service.TDCM> (To be deprecated)

This is used to allow TDCM manage the device.

```
// {6270DA51-9A23-4B6B-81CE-DDD86970F296}
#define VMCALL_SERVICE_TDCM_GUID \
{0x6270da51, 0x9a23, 0x4b6b, 0x81, 0xce, 0xdd, 0xd8, 0x69, 0x70, 0xf2, 0x96}
```

3.2.1.1 TDG.VP.VMCALL <Service.TDCM.CheckTeeIoSupport>

This VMCALL is used by TD to check if TEE-IO is supported for the specific device.

The command data buffer layout is below: (All fields are filled by TD).

Table 3-2: TDG.VP.VMCALL< Service.TDCM.CheckTeeIoSupport >-command buffer layout

Field	Offset (Bytes)	Length (bytes)	Description
Version	0	1	0: for this data structure
Command	1	1	1: **CheckTeeIoSupport**
Reserved	2	2	Reserved
Device Identifier	4	4	Byte 0 [Bit0~2]: PCI Function Number Byte 0 [Bit3~7]: PCI Device Number BYTE 1: PCI Bus Number BYTE 2, 3: PCI Segment Number

The response data buffer layout is shown in the following table (All fields are filled by VMM).

Table 3-3: TDG.VP.VMCALL< Service.TDCM.CheckTeeIoSupport >-response buffer layout

Field	Offset (Bytes)	Length (bytes)	Description
Version	0	1	0: for this data structure
Command	1	1	1: **CheckTeeIoSupport**
Status	2	1	The TDCM status. See Table 3-16.
Reserved	3	1	Reserved

3.2.1.2 TDG.VP.VMCALL <Service.TDCM.Bind>

This VMCALL is used by TDCM to ask the VMM to bind the TDI, send TDISP LOCK_INTERFACE_REQUEST command and return the TDI Interface ID. VMM can optionally send TDISP GET_DEVICE_INTERFACE_REPORT and cache the full TDI report. This VMCALL shall be sent before <Serrvice.TDCM.StartTdi>.

The command data buffer layout is below: (All fields are filled by TD).

Table 3-4: TDG.VP.VMCALL< Service.TDCM.Bind >-command buffer layout

Field	Offset (Bytes)	Length (bytes)	Description
Version	0	1	0: for this data structure
Command	1	1	2: **Bind**
Reserved	2	2	Reserved
Device Identifier	4	4	Byte 0 [Bit0~2]: PCI Function Number Byte 0 [Bit3~7]: PCI Device Number BYTE 1: PCI Bus Number BYTE 2, 3: PCI Segment Number

The response data buffer layout is below: (All fields are filled by VMM).

Table 3-5: TDG.VP.VMCALL< Service.TDCM.Bind >-response buffer layout

Field	Offset (Bytes)	Length (bytes)	Description
Version	0	1	0: for this data structure
Command	1	1	2: **Bind**
Status	2	1	The TDCM status. See Table 3-16.
Reserved	3	1	Reserved
TDI Interface ID	4	12	The TDI's INTERFACE_ID, returned by TDX module *TDH.TDI.CREATE*.

The expected flow is as follows:

- TD calls VMM: TDG.VP.VMCALL<Service.TDCM.Bind>
- (*) VMM calls TDX-module: TDH.TDI.MT.ADD
- (*) VMM calls TDX-module: TDH.TDI.CREATE
- (*) VMM calls TDX-module: TDH.SYS.READ (LOCK_INTERFACE_FLAGS)
- VMM calls TDX-module: TDH.TDI.BIND (LockInterfaceFlags) ← VMM sends TDISP LOCK_INTERFACE_REQUEST and optionally GET_DEVICE_INTERFACE_REPORT.
- VMM signals EVENT:TDG.VP.VMCALL<Service.TDCM.Bind>
- TD calls TDX-Module: TDG.TDI.RD(INTERFACE_STATE)
- TD checks TDI state is in CONFIG_LOCKED.

(*) means skip if it is already done before.

3.2.1.3 TDG.VP.VMCALL <Service.TDCM.GetDeviceInfo>

This VMCALL is used by TDCM to get the device information including SPDM Certificate Chain in all slots, and all SPDM Measurement Transcript. After getting the initial SPDM connection and session information with zero nonce, the TD shall calculate the “DEVICE_INFO_HASH” and validate it with TDX module with TDG.TDI.VALIDATE.

If the TD supports runtime update and need freshness of the information, the TD shall input a non-zero nonce and validate the nonce returned in the device information data. Then the TD shall validate the integrity of the certificate chain and SPDM measurement transcript with a local verifier or a remote verifier. Usually, this VMCALL is used by TDCM after <Service.TDCM.Bind>.NOTE: The “DEVICE_INFO_HASH” is not recalculated during recollection. The TD shall guarantee that the SPDM device certificate is not change. Only SPDM alias certificate (also known as DICE certificate) and the SPDM measurement can be changed.

The command data buffer layout is below: (All fields are filled by TD).

Table 3-6: TDG.VP.VMCALL< Service.TDCM.GetDeviceInfo >-command buffer layout

Field	Offset (Bytes)	Length (bytes)	Description
Version	0	1	0: for this data structure
Command	1	1	3: **GetDeviceInfo**
Reserved	2	2	Reserved
TDI Interface ID	4	12	The TDI's INTERFACE_ID, returned by TDX module *TDH.TDI.CREATE*.
Request Nonce	16	32	A nonce value to indicate this collection. The VMM returns the SPDM_DEVICE_ATTESTATION_INFO_T.
Request Flag	48	8	A 64bit flag to indicate the request to the TDX-module. This flag is only used when Request Nonce is non-0. • BIT0: Include device certificates. • BIT1: Include device measurement. • BIT2~7: Reserved to 0. • BIT8~15: Device certificate request slot mask. BIT8 means to request slot 0, BIT9 means to request slot 1, etc. • BIT16~23: Device measurement request attributes used in GET_MEASUREMENT Param1. Only RawBitStreamRequested is valid, and the reset bits are ignored. • BIT24~63: Reserved to 0.

The response data buffer layout is below: (All fields are filled by VMM).

Table 3-7: TDG.VP.VMCALL< Service.TDCM.GetDeviceInfo >-response buffer layout

Field	Offset (Bytes)	Length (bytes)	Description
Version	0	1	0: for this data structure
Command	1	1	3: **GetDeviceInfo**
Status	2	1	The TDCM status. See Table 3-16.
Reserved	3	1	Reserved
Device Information Data	4	N	The SPDM_DEVICE_ATTESTATION_INFO_T

3.2.1.4 TDG.VP.VMCALL <Service.TDCM.GetTdiReport>

This VMCALL is used by TDCM to get a full TDI report. If VMM sends GET_DEVICE_INTERFACE_REPORT before, the VMM can return the cached TDI report. If VMM has not got the TDI report before, it shall send GET_DEVICE_INTERFACE_REPORT and return the full TDI report. This VMCALL is only allowed after <Service.TDCM.Bind> and before <Service.TDCM.Unbind>. The TD shall calculate the "TDI_REPORT_HASH" and validate it with TDX module with TDG.TDI.VALIDATE.

The command data buffer layout is below: (All fields are filled by TD).

Table 3-8: TDG.VP.VMCALL< Service.TDCM.GetTdiReport >-command buffer layout

Field	Offset (Bytes)	Length (bytes)	Description
Version	0	1	0: for this data structure
Command	1	1	4: **GetTdiReport**
Reserved	2	2	Reserved
TDI Interface ID	4	12	The TDI's INTERFACE_ID, returned by TDX module *TDH.TDI.CREATE*.

The response data buffer layout is below: (All fields are filled by VMM).

Table 3-9: TDG.VP.VMCALL< Service.TDCM.GetTdiReport >-response buffer layout

Field	**Offset (Bytes)**	**Length (bytes)**	**Description**
Version	0	1	0: for this data structure
Command	1	1	4: **GetTdiReport**
Status	2	1	The TDCM status. See Table 3-16.
Reserved	3	1	Reserved
TDI Report	4	N	The full TDI report, returned from multiple GET_DEVICE_INTERFACE_REPORT commands. The TD can calculate the "TDI_REPORT_HASH" and validate it with TDX module TDG.TDI.VALIDATE. **"TDI_REPORT_HASH" == Hash (`Full TDI Report`)**

3.2.1.5 TDG.VP.VMCALL <Service.TDCM.StartTdi>

This VMCALL is used by TDCM to ask VMM to send TDISP START_INTERFACE_REQUEST command. This VMCALL is only allowed after <Service.TDCM.Bind> and before <Service.TDCM.Unbind>. This VMCALL shall be sent right after TDG.TDI.START. After this VMCALL, the TD shall use TDG.TDI.RD(INTERFACE_STATE) to verify if the TDI is changed to RUN state.

The command data buffer layout is below: (All fields are filled by TD).

Table 3-10: TDG.VP.VMCALL< Service.TDCM.StartTdi >-command buffer layout

Field	**Offset (Bytes)**	**Length (bytes)**	**Description**
Version	0	1	0: for this data structure
Command	1	1	5: **StartTdi**
Reserved	2	2	Reserved
TDI Interface ID	4	12	The TDI's INTERFACE_ID, returned by TDX module *TDH.TDI.CREATE*.

The response data buffer layout is below: (All fields are filled by VMM).

Table 3-11: TDG.VP.VMCALL< Service.TDCM.StartTdi >-response buffer layout

Field	Offset (Bytes)	Length (bytes)	Description
Version	0	1	0: for this data structure
Command	1	1	5: **StartTdi**
Status	2	1	The TDCM status. See Table 3-16.
Reserved	3	1	Reserved

The expected flow is as follows:

- TD calls TDX-Module: TDG.TDI.VALIDATE(DEVICE_INFO_HASH, TDI_REPORT_HASH)
- TD calls TDX-Module: TDG.DMAR.ACCEPT and TDG.MMIO.ACCEPT
- TD calls TDX-Module: TDG.TDI.START
- TD calls VMM: TDG.VP.VMCALL<Service.TDCM.StartTdi>
- VMM calls TDX-module: TDH.TDI.START ← VMM sends TDISP START_INTERFACE_REQUEST
- VMM signals EVENT:TDG.VP.VMCALL<Service.TDCM.StartTdi>
- TD calls TDX-Module: TDG.TDI.RD(INTERFACE_STATE)
- TD checks TDI state is in RUN.

3.2.1.6 TDG.VP.VMCALL <Service.TDCM.GetTdiState>

This VMCALL is used by TDCM to send TDISP GET_DEVICE_INTERFACE_STATE command.

The command data buffer layout is below: (All fields are filled by TD).

Table 3-12: TDG.VP.VMCALL< Service.TDCM.GetTdiState >-command buffer layout

Field	Offset (Bytes)	Length (bytes)	Description
Version	0	1	0: for this data structure
Command	1	1	6: **GetTdiState**
Reserved	2	2	Reserved
TDI Interface ID	4	12	The TDI's INTERFACE_ID, returned by TDX module *TDH.TDI.CREATE*.

The response data buffer layout is below: (All fields are filled by VMM).

Table 3-13: TDG.VP.VMCALL< Service.TDCM.GetTdiState >-response buffer layout

Field	Offset (Bytes)	Length (bytes)	Description
Version	0	1	0: for this data structure
Command	1	1	6: **GetTdiState**
Status	2	1	The TDCM status. See Table 3-16.
Reserved	3	1	Reserved

The expected flow is as follows:

- TD calls VMM: TDG.VP.VMCALL<Service.TDCM.GetTdiState>
- VMM calls TDX-module: TDH.TDI.GET.STATE 🡐 VMM sends TDISP GET_DEVICE_INTERFACE_STATE.
- VMM signals EVENT:TDG.VP.VMCALL<Service.TDCM.GetTdiState>
- TD calls TDX-Module: TDG.TDI.RD(INTERFACE_STATE)
- TD gets TDI state.

3.2.1.7 TDG.VP.VMCALL <Service.TDCM.Unbind>

This VMCALL is used by TDCM to ask the VMM to unload the TDI and cleanup the corresponding resources, including sending TDISP STOP_INTERFACE_REQUEST command, cleanup MMIO and DMA pages associated with this TDI.

The command data buffer layout is below: (All fields are filled by TD).

Table 3-14: TDG.VP.VMCALL< Service.TDCM.Unbind >-command buffer layout

Field	Offset (Bytes)	Length (bytes)	Description
Version	0	1	0: for this data structure
Command	1	1	7: **Unbind**
Reserved	2	2	Reserved
Device Identifier	4	4	Byte 0 [Bit0~2]: PCI Function Number Byte 0 [Bit3~7]: PCI Device Number BYTE 1: PCI Bus Number BYTE 2, 3: PCI Segment Number

The response data buffer layout is below: (All fields are filled by VMM).

Table 3-15: TDG.VP.VMCALL< Service.TDCM.Unbind >-response buffer layout

Field	Offset (Bytes)	Length (bytes)	Description
Version	0	1	0: for this data structure
Command	1	1	7: **Unbind**
Status	2	1	The TDCM status. See Table 3-16.
Reserved	3	1	Reserved

The expected flow is as follows:

- TD calls VMM: TDG.VP.VMCALL<Service.TDCM.Unbind> 🡐 This step is to inform VMM to unload the device.
- VMM calls TDX-module: TDH.TDI.STOP 🡐 VMM sends TDISP STOP_INTERFACE_REQUEST
- VMM calls TDX-module TDH.DMAR.REMOVE clean up DMA entry.
- VMM calls TDX-module TDH.MMIO.UNMAP clean up MMIO entry.
- VMM calls TDH.TDI.REMOVE 🡐 This step check (DEVIFCS.MMIO_PAGE_CNT==0 && DEVIFCS.DMA_MAPPED_FLAG==FALSE)
- VMM signals EVENT:TDG.VP.VMCALL<Service.TDCM.Unbind>
- TD calls TDX-module: TDG.TDI.RD(INTERFACE_STATE)] 🡐 This step is to check and ensure DEVIF is disconnected successfully.

3.2.1.8 TDCM Status Code

TDCM Status reports TDCM VMCALL result.

Table 3-16: TDCM Status

Value	Name	Description
0	SUCCESS	The TDCM call is finished successfully
1	INVALID_PARAMETER	The input parameter has one or more invalid fields, such as invalid device identifier or invalid TDI interface ID.
2	UNSUPPORTED	The input parameter is unsupported, such as device is NOT TEE-IO capable.
3	OUT_OF_RESOURCE	VMM does not have enough resource to finish the TDVMCALL.
4	TDX_MODULE_ERROR	The TDX module returns error.
5	TDXIO_DEVICE_ERROR	The device returns DOE mailbox error.
6	SPDM_MESSAGE_ERROR	The device returns SPDM error.
7	IDE_KM_MESSAGE_ERROR	The device returns IDE_KM error.
8	TDISP_MESSAGE_ERROR	The device returns TDISP error.
9	INVALID_STATE	The API is called when TDI is not in invalid state, such as GetTdiReport before Bind.
0xA~0xFF	Reserved	Reserved

3.3 TDG.VP.VMCALL<TDCM>

This is used to allow TD guest TEE-IO Device Control and Management (TDCM) module to manage the device. The TDCM leaf functions are defined in the following table.

Table 3-17: TDG.VP.VMCALL<TDCM>-Input Operands

Operand	Description
R11	TDG.VP.VMCALL<TDCM> sub-function per Table 2-3.
R12	Bits [15:0] TDCM Leaf function 1: **CheckTeeIoSupport** 2: **Bind** 3: **GetDeviceInfo** 4: **GetTdiReport** 5: **StartTdi** 6: **GetTdiState** 7: **Unbind** Other: Reserved Bits [23:16] TDCM API Version – 0 for this data structure. Bits [63:24] Reserved; must be 0.
Others	Leaf-specific

Table 3-18: TDG.VP.VMCALL<TDCM>-Output Operands

Operand	Description
R10	TDG.VP.VMCALL-return code.
Others	Leaf-specific

Table 3-19: TDG.VP.VMCALL<TDCM>-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 – for example, the GPA address is invalid (beyond GPAW).
TDG.VP.VMCALL_SUBFUNC_UNSUPPORTED		This sub-function is unsupported

3.3.1 TDG.VP.VMCALL <TDCM.CheckTeeIoSupport >

This VMCALL is used by TD to check if TEE-IO is supported for the specific device.

Table 3-20: TDG.VP.VMCALL<TDCM.CheckTeeIoSupport>-Input Operands

Operand	Description
R11	TDG.VP.VMCALL<TDCM> sub-function per Table 2-1.
R12	Bits [15:0] TDCM Leaf function 1: **CheckTeeIoSupport** Bits [23:16] TDCM API Version – 0 for this data structure. Bits [63:24] Reserved; must be 0.
R13	Device Identifier: Byte 0, Bits [2:0]: PCI Function Number Byte 0, Bits [7:3]: PCI Device Number Byte 1: PCI Bus Number Byte [3:2]: PCI Segment Number

Table 3-21: TDG.VP.VMCALL<TDCM.CheckTeeIoSupport>-Output Operands

Operand	Description
R10	TDG.VP.VMCALL-return code.
R11	0: Device does not support TEE-IO. 1: Device supports TEE-IO.

3.3.2 TDG.VP.VMCALL <TDCM.Bind>

This VMCALL is used by TDCM to ask the VMM to bind the TDI, send TDISP LOCK_INTERFACE_REQUEST command and return the TDI Interface ID. VMM can optionally send TDISP GET_DEVICE_INTERFACE_REPORT and cache the full TDI report. This VMCALL shall be sent before <StartTdi>.

Table 3-22: TDG.VP.VMCALL<TDCM.Bind>-Input Operands

Operand	Description
R11	TDG.VP.VMCALL<TDCM> sub-function per Table 2-1.
R12	Bits [15:0] TDCM Leaf function 2: **Bind** Bits [23:16] TDCM API Version – 0 for this data structure. Bits [63:24] Reserved; must be 0.
R13	Device Identifier: Byte 0, Bits [2:0]: PCI Function Number Byte 0, Bits [7:3]: PCI Device Number Byte 1: PCI Bus Number Byte [3:2]: PCI Segment Number
R14	**`DataBufferLength`** The size in bytes of the data buffer specified in R15.
R15	**`DataBufferGPA`** The shared GPA of the data buffer. See Table 3-24
RBX	**Event notification interrupt vector -** (valid values 32~255) selected by TD. 0~31: Reserved. 32~255: The interrupt vector to signal when the response is ready. The VMM should inject the interrupt vector into the TD VCPU that executed TDG.VP.VMCALL.

Table 3-23: TDG.VP.VMCALL<TDCM.Bind>-Output Operands

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

Table 3-24: <TDCM.Bind> Data Buffer format

Field	Offset (Bytes)	Length (bytes)	Description
Data Status	0	8	Byte [0] 0: Wait for VMM completion (Set by TD) 1: VMM completed successfully (Set by VMM) 2: VMM completed with error (Set by VMM) Byte [1]: If Byte [0] is 0 or 1, then reserved. If Byte [0] is 2, then TDCM Status. See Table 3-40 Byte [7:2]: reserved to 0. This field is always present.
Length	8	4	The size in bytes of the Data field, excluding the Data Status field and this Length field.
Data	12	0	When set by TD: None When set by VMM: **TDI Interface ID** (12 bytes) - The TDI's INTERFACE_ID, returned by TDX module *TDH.TDI.CREATE*.

The expected flow is as follows:

- TD calls VMM: TDG.VP.VMCALL<TDCM.Bind>
- (*) VMM calls TDX-module: TDH.TDI.MT.ADD
- (*) VMM calls TDX-module: TDH.TDI.CREATE
- (*) VMM calls TDX-module: TDH.SYS.READ (LOCK_INTERFACE_FLAGS)
- VMM calls TDX-module: TDH.TDI.BIND (LockInterfaceFlags) 🡐 VMM sends TDISP LOCK_INTERFACE_REQUEST and optionally GET_DEVICE_INTERFACE_REPORT.
- VMM signals EVENT:TDG.VP.VMCALL<TDCM.Bind>
- TD calls TDX-Module: TDG.TDI.RD(INTERFACE_STATE)
- TD checks TDI state is in CONFIG_LOCKED.

(*) means skip if it is already done before.

3.3.3 TDG.VP.VMCALL <TDCM.GetDeviceInfo>

This VMCALL is used by TDCM to get the device information including SPDM Certificate Chain in all slots, and all SPDM Measurement Transcript. After getting the initial SPDM connection and session information with zero nonce, the TD shall calculate the "DEVICE_INFO_HASH" and validate it with TDX module with TDG.TDI.VALIDATE.

If the TD supports runtime update and need freshness of the information, the TD shall input a non-zero nonce and validate the nonce returned in the device information data. Then the TD shall validate the integrity of the certificate chain and SPDM measurement transcript with a local verifier or a remote verifier. Usually, this VMCALL is used by TDCM after <TDCM.Bind>. NOTE: The "DEVICE_INFO_HASH" is not recalculated during recollection. The TD shall guarantee that the SPDM device certificate is not change. Only SPDM alias certificate (also known as DICE certificate) and the SPDM measurement can be changed.

Table 3-25: TDG.VP.VMCALL<TDCM.GetDeviceInfo>-Input Operands

Operand	Description
R11	TDG.VP.VMCALL<TDCM> sub-function per Table 2-1.
R12	Bits [15:0] TDCM Leaf function 3: **GetDeviceInfo** Bits [23:16] TDCM API Version – 0 for this data structure. Bits [63:24] Reserved; must be 0.
R13	TDI Interface ID[7:0] - The TDI's INTERFACE_ID, returned by TDX module *TDH.TDI.CREATE*.
R14	TDI Interface ID[11:8]
R15	**`DataBufferLength`** The size in bytes of the data buffer specified in RBX.
RBX	**`DataBufferGPA`** The shared GPA of the data buffer. See Table 3-27
RDI	**Event notification interrupt vector** - (valid values 32~255) selected by TD. 0~31: Reserved. 32~255: The interrupt vector to signal when the response is ready. The VMM should inject the interrupt vector into the TD VCPU that executed TDG.VP.VMCALL.

Table 3-26: TDG.VP.VMCALL<TDCM.GetDeviceInfo>-Output Operands

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

Table 3-27: <TDCM.GetDeviceInfo> Data Buffer format

Field	**Offset (Bytes)**	**Length (bytes)**	**Description**
Data Status	0	8	Byte [0] 0: Wait for VMM completion (Set by TD) 1: VMM completed successfully (Set by VMM) 2: VMM completed with error (Set by VMM) Byte [1]: If Byte [0] is 0 or 1, then reserved. If Byte [0] is 2, then TDCM Status. See Table 3-40 Byte [7:2]: reserved to 0. This field is always present.
Length	8	4	The size in bytes of the Data field, excluding the Data Status field and this Length field.
Data	12	0	When set by TD: **Request Nonce** (32 bytes) - A nonce value to indicate this collection. **Request Flag** (8 bytes) - A 64bit flag to indicate the request to the TDX-module. This flag is only used when Request Nonce is non-0. • BIT0: Include device certificates. • BIT1: Include device measurement. • BIT2~7: Reserved to 0. • BIT8~15: Device certificate request slot mask. BIT8 means to request slot 0, BIT9 means to request slot 1, etc. • BIT16~23: Device measurement request attributes used in GET_MEASUREMENT Param1. Only RawBitStreamRequested is valid, and the reset bits are ignored. • BIT24~63: Reserved to 0. When set by VMM: **Device Information** - SPDM_DEVICE_ATTESTATION_INFO_T

3.3.4 TDG.VP.VMCALL <TDCM.GetTdiReport>

This VMCALL is used by TDCM to get a full TDI report. If VMM sends GET_DEVICE_INTERFACE_REPORT before, the VMM can return the cached TDI report. If VMM has not got the TDI report before, it shall send GET_DEVICE_INTERFACE_REPORT and return the full TDI report. This VMCALL is only allowed after <TDCM.Bind> and before <TDCM.Unbind>. The TD shall calculate the "TDI_REPORT_HASH" and validate it with TDX module with TDG.TDI.VALIDATE.

Table 3-28: TDG.VP.VMCALL<TDCM.GetTdiReport>-Input Operands

Operand	Description
R11	TDG.VP.VMCALL<TDCM> sub-function per Table 2-1.
R12	Bits [15:0] TDCM Leaf function 4: **GetTdiReport** Bits [23:16] TDCM API Version – 0 for this data structure. Bits [63:24] Reserved; must be 0.
R13	TDI Interface ID[7:0] - The TDI's INTERFACE_ID, returned by TDX module *TDH.TDI.CREATE*.
R14	TDI Interface ID[11:8]
R15	**`DataBufferLength`** The size in bytes of the data buffer specified in RBX.
RBX	**`DataBufferGPA`** The shared GPA of the data buffer. See Table 3-30
RDI	**Event notification interrupt vector** - (valid values 32~255) selected by TD. 0~31: Reserved. 32~255: The interrupt vector to signal when the response is ready. The VMM should inject the interrupt vector into the TD VCPU that executed TDG.VP.VMCALL.

Table 3-29: TDG.VP.VMCALL<TDCM.GetTdiReport>-Output Operands

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

Table 3-30: <TDCM.GetTdiReport> Data Buffer format

Field	**Offset (Bytes)**	**Length (bytes)**	**Description**
Data Status	0	8	Byte [0] 0: Wait for VMM completion (Set by TD) 1: VMM completed successfully (Set by VMM) 2: VMM completed with error (Set by VMM) Byte [1]: If Byte [0] is 0 or 1, then reserved. If Byte [0] is 2, then TDCM Status. See Table 3-40 Byte [7:2]: reserved to 0. This field is always present.
Length	8	4	The size in bytes of the Data field, excluding the Data Status field and this Length field.
Data	12	0	When set by TD: None When set by VMM: **TDI Report -** The full TDI report, returned from multiple GET_DEVICE_INTERFACE_REPORT commands. The TD can calculate the “TDI_REPORT_HASH” and validate it with TDX module TDG.TDI.VALIDATE. **“TDI_REPORT_HASH” == Hash (`Full TDI Report`).**

3.3.5 TDG.VP.VMCALL <TDCM.StartTdi>

This VMCALL is used by TDCM to ask VMM to send TDISP START_INTERFACE_REQUEST command. This VMCALL is only allowed after <TDCM.Bind> and before <TDCM.Unbind>. This VMCALL shall be sent right after TDG.TDI.START. After this VMCALL, the TD shall use TDG.TDI.RD(INTERFACE_STATE) to verify if the TDI is changed to RUN state.

Table 3-31: TDG.VP.VMCALL<TDCM.StartTdi>-Input Operands

Operand	Description
R11	TDG.VP.VMCALL<TDCM> sub-function per Table 2-1.
R12	Bits [15:0] TDCM Leaf function 5: **StartTdi** Bits [23:16] TDCM API Version – 0 for this data structure. Bits [63:24] Reserved; must be 0.
R13	TDI Interface ID[7:0] - The TDI's INTERFACE_ID, returned by TDX module *TDH.TDI.CREATE*.
R14	TDI Interface ID[11:8]
R15	**`DataBufferLength`** The size in bytes of the data buffer specified in RBX.
RBX	**`DataBufferGPA`** The shared GPA of the data buffer. See Table 3-33
RDI	**Event notification interrupt vector** - (valid values 32~255) selected by TD. 0~31: Reserved. 32~255: The interrupt vector to signal when the response is ready. The VMM should inject the interrupt vector into the TD VCPU that executed TDG.VP.VMCALL.

Table 3-32: TDG.VP.VMCALL<TDCM.StartTdi>-Output Operands

Operand	Description
R10	TDG.VP.VMCALL-return code.
R11	TDCM Status. See Table 3-40

Table 3-33: <TDCM.StartTdi> Data Buffer format

Field	Offset (Bytes)	Length (bytes)	Description
Data Status	0	8	Byte [0] 0: Wait for VMM completion (Set by TD) 1: VMM completed successfully (Set by VMM) 2: VMM completed with error (Set by VMM) Byte [1]: If Byte [0] is 0 or 1, then reserved. If Byte [0] is 2, then TDCM Status. See Table 3-40 Byte [7:2]: reserved to 0. This field is always present.
Length	8	4	The size in bytes of the Data field, excluding the Data Status field and this Length field.
Data	12	0	When set by TD: None When set by VMM: None

The expected flow is as follows:

- TD calls TDX-Module: TDG.TDI.VALIDATE(DEVICE_INFO_HASH, TDI_REPORT_HASH)
- TD calls TDX-Module: TDG.DMAR.ACCEPT and TDG.MMIO.ACCEPT
- TD calls TDX-Module: TDG.TDI.START
- TD calls VMM: TDG.VP.VMCALL<TDCM.StartTdi>
- VMM calls TDX-module: TDH.TDI.START 🡐 VMM sends TDISP START_INTERFACE_REQUEST
- VMM signals EVENT:TDG.VP.VMCALL<TDCM.StartTdi>
- TD calls TDX-Module: TDG.TDI.RD(INTERFACE_STATE)
- TD checks TDI state is in RUN.

3.3.6 TDG.VP.VMCALL <TDCM.GetTdiState>

This VMCALL is used by TDCM to send TDISP GET_DEVICE_INTERFACE_STATE command.

Table 3-34: TDG.VP.VMCALL<TDCM.GetTdiState>-Input Operands

Operand	Description
R11	TDG.VP.VMCALL<TDCM> sub-function per Table 2-1.
R12	Bits [15:0] TDCM Leaf function 6: **GetTdiState** Bits [23:16] TDCM API Version – 0 for this data structure. Bits [63:24] Reserved; must be 0.
R13	TDI Interface ID[7:0] - The TDI's INTERFACE_ID, returned by TDX module *TDH.TDI.CREATE*.
R14	TDI Interface ID[11:8]
R15	**DataBufferLength** The size in bytes of the data buffer specified in RBX.
RBX	**DataBufferGPA** The shared GPA of the data buffer. See Table 3-36
RDI	**Event notification interrupt vector** - (valid values 32~255) selected by TD. 0~31: Reserved. 32~255: The interrupt vector to signal when the response is ready. The VMM should inject the interrupt vector into the TD VCPU that executed TDG.VP.VMCALL.

Table 3-35: TDG.VP.VMCALL<TDCM.GetTdiState>-Output Operands

Operand	Description
R10	TDG.VP.VMCALL-return code.
R11	TDCM Status. See Table 3-40

Table 3-36: <TDCM.GetTdiState> Data Buffer format

Field	Offset (Bytes)	Length (bytes)	Description
Data Status	0	8	Byte [0] 0: Wait for VMM completion (Set by TD) 1: VMM completed successfully (Set by VMM) 2: VMM completed with error (Set by VMM) Byte [1]: If Byte [0] is 0 or 1, then reserved. If Byte [0] is 2, then TDCM Status. See Table 3-40 Byte [7:2]: reserved to 0. This field is always present.
Length	8	4	The size in bytes of the Data field, excluding the Data Status field and this Length field.
Data	12	0	When set by TD: None When set by VMM: None

The expected flow is as follows:

- TD calls VMM: TDG.VP.VMCALL<TDCM.GetTdiState>
- VMM calls TDX-module: TDH.TDI.GET.STATE ← VMM sends TDISP GET_DEVICE_INTERFACE_STATE.
- VMM signals EVENT:TDG.VP.VMCALL<TDCM.GetTdiState>
- TD calls TDX-Module: TDG.TDI.RD(INTERFACE_STATE)
- TD gets TDI state.

3.3.7 TDG.VP.VMCALL <TDCM.Unbind>

This VMCALL is used by TDCM to ask the VMM to unload the TDI and cleanup the corresponding resources, including sending TDISP STOP_INTERFACE_REQUEST command, cleanup MMIO and DMA pages associated with this TDI.

Table 3-37: TDG.VP.VMCALL<TDCM.Unbind>-Input Operands

Operand	Description
R11	TDG.VP.VMCALL<TDCM> sub-function per Table 2-1.
R12	Bits [15:0] TDCM Leaf function 7: **Unbind** Bits [23:16] TDCM API Version – 0 for this data structure. Bits [63:24] Reserved; must be 0.
R13	Device Identifier: Byte 0, Bits [2:0]: PCI Function Number Byte 0, Bits [7:3]: PCI Device Number Byte 1: PCI Bus Number Byte [3:2]: PCI Segment Number
R14	**`DataBufferLength`** The size in bytes of the data buffer specified in R15.
R15	**`DataBufferGPA`** The shared GPA of the data buffer. See Table 3-39
RBX	**Event notification interrupt vector -** (valid values 32~255) selected by TD. 0~31: Reserved. 32~255: The interrupt vector to signal when the response is ready. The VMM should inject the interrupt vector into the TD VCPU that executed TDG.VP.VMCALL.

Table 3-38: TDG.VP.VMCALL<TDCM.Unbind>-Output Operands

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

Table 3-39: <TDCM.Unbind> Data Buffer format

Field	Offset (Bytes)	Length (bytes)	Description
Data Status	0	8	Byte [0] 0: Wait for VMM completion (Set by TD) 1: VMM completed successfully (Set by VMM) 2: VMM completed with error (Set by VMM) Byte [1]: If Byte [0] is 0 or 1, then reserved. If Byte [0] is 2, then TDCM Status. See Table 3-40 Byte [7:2]: reserved to 0. This field is always present.
Length	8	4	The size in bytes of the Data field, excluding the Data Status field and this Length field.
Data	12	0	When set by TD: None When set by VMM: None

The expected flow is as follows:

- TD calls VMM: TDG.VP.VMCALL<TDCM.Unbind> 🡐 This step is to inform VMM to unload the device.
- VMM calls TDX-module: TDH.TDI.STOP 🡐 VMM sends TDISP STOP_INTERFACE_REQUEST
- VMM calls TDX-module TDH.DMAR.REMOVE clean up DMA entry.
- VMM calls TDX-module TDH.MMIO.UNMAP clean up MMIO entry.
- VMM calls TDH.TDI.REMOVE 🡐 This step check (DEVIFCS.MMIO_PAGE_CNT==0 && DEVIFCS.DMA_MAPPED_FLAG==FALSE)
- VMM signals EVENT:TDG.VP.VMCALL<TDCM.Unbind>
- TD calls TDX-module: TDG.TDI.RD(INTERFACE_STATE)] 🡐 This step is to check and ensure DEVIF is disconnected successfully.

3.3.8 TDCM Status Code

TDCM Status reports TDCM VMCALL result from VMM.

Table 3-40: TDCM Status

Value	Name	Description
10	TDX_MODULE_ERROR	The TDX module returns error.
11	TDXIO_DEVICE_ERROR	The device returns DOE mailbox error.
12	SPDM_MESSAGE_ERROR	The device returns SPDM error.
13	IDE_KM_MESSAGE_ERROR	The device returns IDE_KM error.
14	TDISP_MESSAGE_ERROR	The device returns TDISP error.
15	INVALID_STATE	The API is called when TDI is not in invalid state, such as GetTdiReport before Bind.
Other value	Reserved	Reserved

4 TD-VMM-Communication Scenarios

4.1 TDX Connect Architecture

TDX 1.0 and 1.5 only support software-based IO virtualization. The untrusted VMM exposes a virtual device to a TD using shared (untrusted) synthetic IO and para-virtualized device interfaces managed by the VMM. The VMM, the device, and the IO channel are out of TCB.

The synthetic IOs have two limitations:

1) The communication between a TD and a physical device must be done through shared memory bounce-buffers. In order to help prevent attack from VMM, the data in the bounce buffer needs to be encrypted back and forth to private memory buffers inside the TD. This will bring performance overhead.
2) Because the device is untrusted, the TD cannot offload the computation to the device accelerator.

The goal of TDX Connect is to support direct assignment if IO devices to TDs increasing its IO performance and enabling new secure IO use cases. Figure 5-4 shows the TDX Connect component.

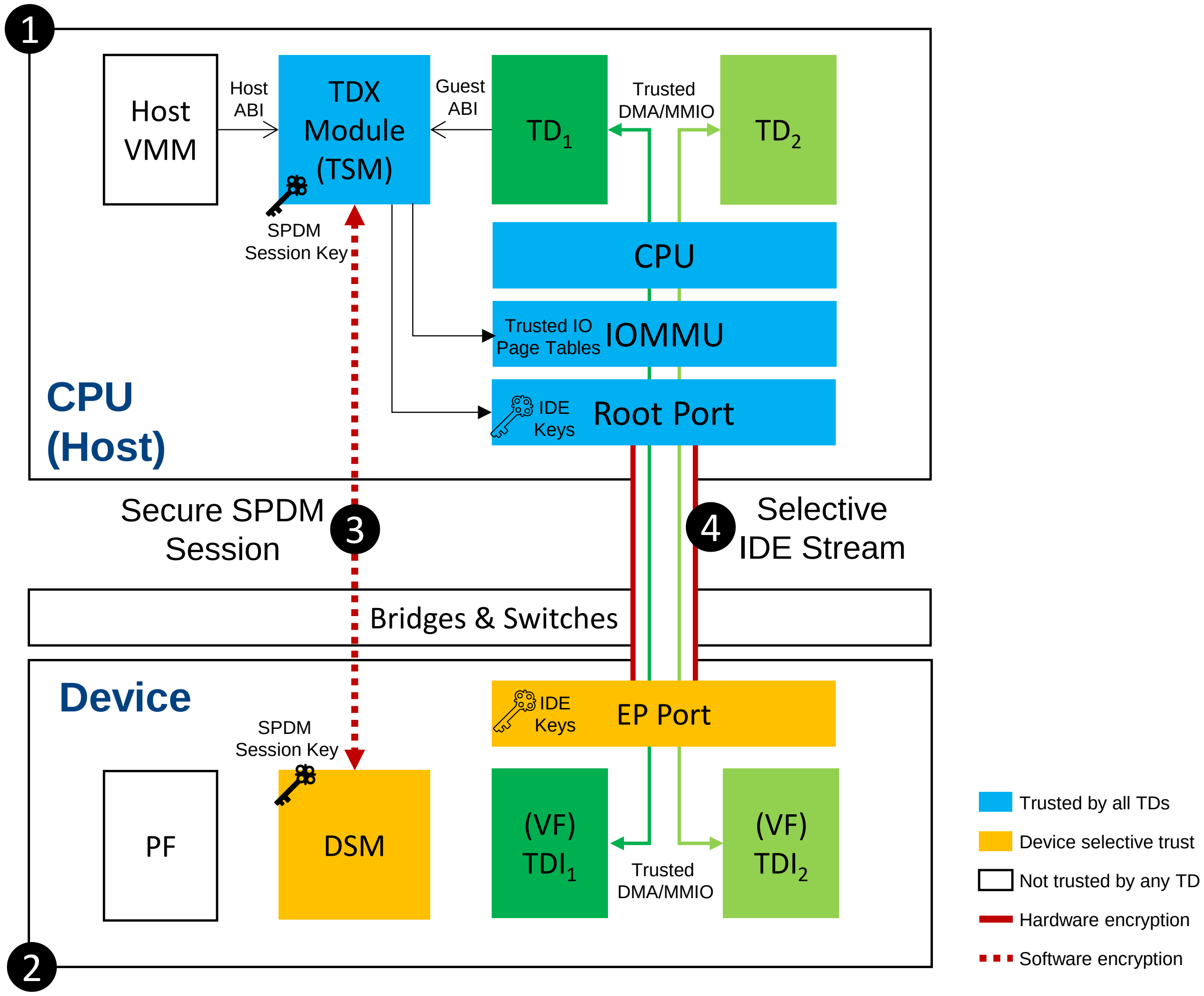

Figure 4-1: TDX Connect Component

4.1.1 TDX Connect Component

4.1.1.1 TDX Connect Capable Device

A TDX Connect capable device needs support below industry standard:

- DMTF **Secure Protocol and Data Model (SPDM)** specification. SPDM is used for device authentication, measurement collection and secure session. With SPDM session, the Trusted Execution Environment (TEE) Security Manager (TSM) can establish a secure software link with the device security manager (DSM).
- PCI-SIG PCI Express Base specification – **Data Object Exchange (DOE)**. DOE is a mechanism to allow the host software send or receive SPDM message.
- PCI-SIG PCI Express Base specification – **Component Measurement and Authentication (CMA)**. With CMA, the host can collect the device identity and measurement via standard SPDM command.
- PCI-SIG PCI Express Base specification – **Integrity and Data Encryption (IDE)**. IDE provides confidentiality, integrity, and replay protection for PCIE Transaction Layer Packets (TLPs) Transmitted and Received between two Ports. IDE provides a secure hardware link between host PCIE Root Port and device PCIE endpoint. The IDE hardware link is protected by an IDE key. The IDE key is transmitted from the host via IDE key management (IDE_KM) protocol, which runs in an SPDM session.
- PCI-SIG **Trusted Execution Environment (TEE) Device Interface (TDI) Security Protocol (TDISP)**. A TEE Virtual Machine (TVM) can use TDISP to attach and detach a TDI in a trusted manner. The TDISP message is transmitted in an SPDM session.

For more detail on these technologies, please refer to the corresponding specifications.

4.1.1.2 TDX Connect Capable Silicon

A TDX Connect capable silicon need support below features:

- PCI-SIG PCI Express Base specification – **Integrity and Data Encryption (IDE)**. The PCIE or CXL root port need support IDE for link encryption.
- **Trusted IO Memory Management Unit (IOMMU)**. This IOMMU engine should be partitioned to trust world and untrusted world. The TDX module owes the trusted IOMMU register, which manages the DMA for the trusted device for the TD. The VMM must be TDX module API to access the IOMMU engine.
- **Trusted DMA**. The TDX Connect silicon allows the trusted device DMA to a TD private memory. It helps prevent a DMA attack, such as remapping and alias attack, device ID spoofing, man-in-the-middle attack, confused deputy attack, etc.
- **Trusted MMIO**. The TDX Connect silicon allows the TD to access the trusted device MMIO. It helps prevent an MMIO attack, such as untrusted software access, untrusted device access, remap/swizzle attack, region overlap attack, etc.

4.1.1.3 TDX Module

TDX module acts as a TEE security manager (TSM) for a TD, which is a TEE virtual machine (TVM). It enforces all security policies, such as SPDM secure session, IDE setup, trusted IOMMU manager, etc.

The relationship between device, interface, SPDM and IDE is as follows.

- There is **one SPDM session** between TDX module and each **physical device**.
- Each **physical device** link can be protected by **one or more selective IDE streams**.
- A **physical device** may host **one or more device interfaces**.
- A **device interface** is bound to the **default IDE stream** between the SOC and device.

Please refer to TDX Module v2.0 architecture specification for more detail.

4.1.1.4 Virtual Machine Monitor

Host VMM manages all system resources, including managing TDX Connect device. In TDX Connect architecture, multiple components may be involved in the device communication. The role and responsibility are below:

1) **VMM is the physical device manager**. VMM allocates system resource for the device, such as assigning MMIO base address register, setting up IOMMU table for the device, locking the device, etc. Finally, VMM need assign the device interface to a TD.
2) **TD is the device interface owner**. TD should verify the device identity via certificate and device version via measurement and accept the device interface. TD should also verify the device resource and policy setting by VMM and accept the configuration. Only after the device is verified and accepted, the TD can start the device interface.
3) **TDX-module is a device security state maintainer**. It helps ensure that the device security state is not tampered by the VMM when TD uses the device interface. For example, TDX-Module maintains the trusted-IOMMU configuration, maintains the SPDM session for IDE_KM and TDISP and maintains the hardware IDE session configuration.

4.1.2 TDX Connect Software Flow Lifecycle

4.1.2.1 TDX Connect Setup Sequence

Figure 5-4 shows the high-level setup sequence:

1) **SOC initialization and IOMMU setup**. VMM launches the TDX-module and sets up IOMMU.
2) **Device SPDM session start**. VMM invokes TDX-module to start SPDM session and gets the device information such as certificate and measurement.

3) **Device IDE start**. VMM allocates IDE stream and uses IDE_KM to set up IDE session with help of TDX-module.

4) **Device Interface Start**. VMM configures the device resource such as MMIO and DMA, then assigns the deice interface to a TD. TD verifies the device, accepts the device interface, then uses TDISP to start the device interface.

4.1.2.2 TDX Connect Communication Maintenance

At runtime, the VMM need maintain the communication with the device:

5) **Device SPDM session maintenance**. If an SPDM session requires HEARTBEAT or KEY_UPDATE, the VMM invokes TDX-module to perform the HEARTBEAT or KEY_UPDATE action to keep SPDM session alive.
6) **Device Information Data recollection**. If device supports runtime update, the TD may recollect the device information such as certificate and measurement again. VMM need invoke the TDX-module to perform such recollection action and return the device information data to TD.

4.1.2.3 TDX Connect Teardown Sequence

The high-level graceful teardown sequence is:

7) **Device Interface Stop**. The TD or VMM uses TDISP to stop the device interface. The device interface resource can be freed as well, such as MMIO and DMA.
8) **Device IDE Stop**. VMM used IDK_KM to stop the IDE session with help of TDX-module and removes IDE stream.
9) **Device SPDM session Stop**. After all interfaces are stopped, VMM invokes TDX-module to stop SPDM session.
10) **SOC IOMMU reset**. Once all SPDM sessions are stopped, VMM can reset IOMMU state.

For the device management detail, please refer to TDX Connect Architecture specification: Device Setup and TDI Assignment Life-Cycle.

4.1.3 TDX Connect Device Attestation

Once a TD, acting as TVM, finds a device interface on the platform, the TD need verify the device before accepting and using the device. The TDX-module, acting as TSM, collects the device evidence (including device certificate and device measurement) and presents to the TD in a secure way. The TEE device verifier in the TD needs to verify the device evidence based on the endorsement (such as a root certificate), reference manifest (such as device certificate and reference measurement), and a predefined policy from TVM owner. Figure 5-6 shows an example.

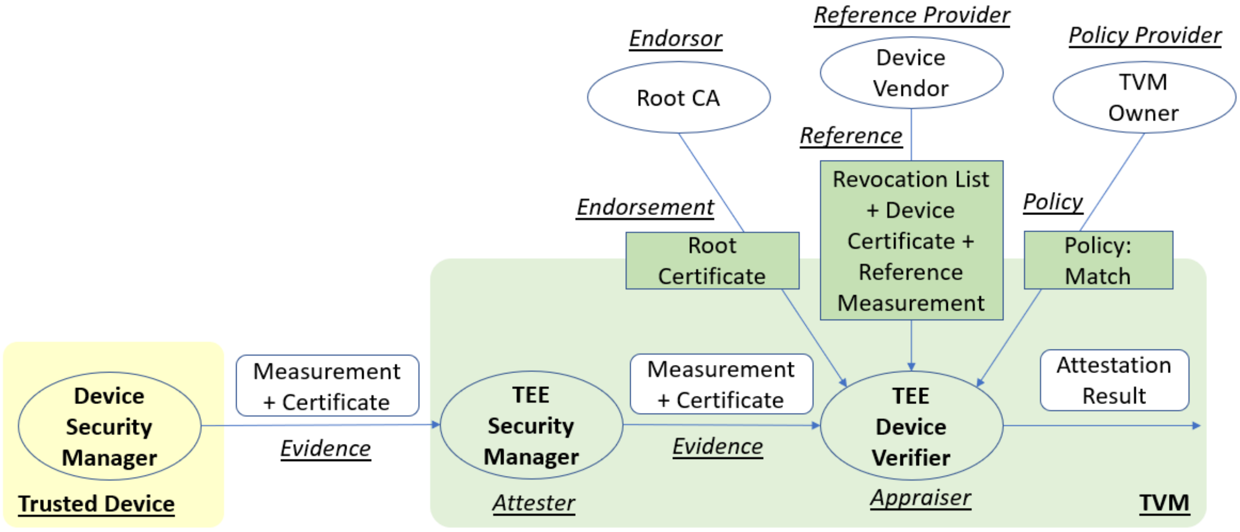

Figure 4-2: TDX Connect Device Attestation Flow Example

Please refer to the white paper *"Device Attestation Model in Confidential Computing Environment"* for more information.

4.1.4 TDX Connect Device Runtime Update

A device may want to perform an update when it is connected to a TD. Usually, a VMM should detach the device from TD, perform the update, and re-attach the device to a TD. However, it may bring some latency concern if the TD is using the device, such as network interface card. If a device supports runtime update without reset, then the TDX Connect architecture can support to keep the device attaching to the TD when the device performs runtime update.

There are two policies to be set up for runtime update capability.

4.1.4.1 SPDM 1.2 Session Policy - TerminationPolicy

According to the SPDM 1.2 specification, TerminationPolicy specifies behavior of the Requester when the Responder completes a runtime code or configuration update that affects the hardware or firmware measurement of the Responder. If not set, the Responder shall terminate the session when the runtime update has taken effect. If set, the Responder shall decide whether to terminate or continue with the session based on its own policy.

The VMM shall input the TerminationPolicy as SPDM Policy Data to the TDX-module. The TDX-module shall include the policy in DEVICE_INFO_DATA for the TD. Finally, the TD should verify the TerminationPolicy in the DEVICE_INFO_DATA to decide if the TD accepts this setting.

The VMM should check the TD manifest to determine what TerminationPolicy should be used to avoid unnecessary device rejection by TD.

4.1.4.2 TDISP LOCK_INTERFACE – NO_FW_UPDATE

According to TDISP specification, NO_FW_UPDATE controls whether firmware update is allowed.

When set, this bit indicates that when this TDI is in CONFIG_LOCKED or RUN state, the device must not accept firmware updates. This option allows certain TVM to opt-out of further firmware updates to the device once the TVM starts using the TDI. A VMM must detach from the TEEs all TDI that are locked with NO_FW_UPDATE=1 from the TVM, and move them to CONFIG_UNLOCKED state to perform a firmware update.

The VMM shall select the policy and construct the LOCK_INTERFACE command.

The NO_FW_UPDATE flag is reported in TDISP DEVICE_INTERFACE_REPORT response. The TD can verify the REPORT to decide whether the TD accepts the flag.

The VMM should check the TD manifest to determine if NO_FW_UPDATE flag should be used to avoid unnecessary device rejection by TD.

4.1.4.3 Device Runtime Update Requirement

If the device supports runtime update without session termination, the device shall report secure version number (SVN) of all mutable firmware in the device. The device shall only allow updates to the firmware with equal or higher SVN. Downgrading to lower SVN shall be forbidden.

4.1.4.4 Device Re-attestation

When the device is runtime updated, the TD may need to reevaluate the device certificate and measurement. The TD may use TDG.VP.VMCALL<Service.TDCM.GetDeviceInfo> with a nonce to get the fresh device information from VMM. Then the TD needs to verify the device again.

~~ End of document ~~