Intel® Trust Domain Extensions (Intel® TDX) Module TDX Connect Application Binary Interface (ABI) Reference Specification

EXTERNAL DRAFT

2.1 V0.55

June 2025

Notices and Disclaimers

Intel TDX Connect Application Binary Interface (ABI) Reference

Table of Contents

Intel TDX Connect Application Binary Interface (ABI) Reference

Intel TDX Connect Application Binary Interface (ABI) Reference

Intel TDX Connect Application Binary Interface (ABI) Reference

1. About this Document

1.1. Scope of this Document

This document describes the TDX Connect related Application Binary Interface (ABI) of the Intel® Trust Domain Extensions (Intel® TDX) module, implemented using the Intel TDX Instruction Set Architecture (ISA) extensions, for confidential execution of Trust Domains in an untrusted hosted cloud environment.

This document is part of the **TDX Module Architecture Specification Set**, which includes the following documents:

Table 1.1: TDX Module Architecture Specification Set

Document Name	Reference	Description
TDX Module Base Architecture Specification	[TDX Module Base Spec]	Base TDX module architecture overview and specification, covering key management, TD lifecycle management, memory management, virtualization, measurement and attestation, service TDs, debug aspects etc.
TDX Module TD Migration Architecture Specification	[TD Migration Spec]	Architecture overview and specification for TD migration
TDX Module TD Partitioning Architecture Specification	[TD Partitioning Spec]	Architecture overview and specification for TD Partitioning
TDX Module ABI Reference Specification	[TDX Module ABI Spec]	Detailed TDX module Application Binary Interface (ABI) reference specification, covering the entire TDX module architecture except TDX connect
TDX Module TDX Connect Architecture Specification	[TDX Connect Spec]	
→ **TDX Module TDX Connect ABI Reference Specification**	[TDX Connect ABI Spec]	Detailed TDX module Application Binary Interface (ABI) reference specification, covering the TDX connect architecture
TDX Module ABI Reference Tables	[TDX Module ABI Tables]	A set of JSON format files detailing TDX module Application Binary Interface (ABI)
TDX Module ABI Incompatibilities	[TDX Module ABI Incompatibilities]	Description of the incompatibilities between TDX 1.0 and TDX 1.4/1.5 that may impact the host VMM and/or guest TDs

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.

Note: The contents of this document are accurate to the best of Intel's knowledge as of the date of publication, though Intel does not represent that such information will remain as described indefinitely in light of future research and design implementations. Intel does not commit to update this document in real time when such changes occur.

1.2. Glossary

See the [TDX Module Base Spec].

1.3. Notation

See the [TDX Module Base Spec].

1.4. References

See the [TDX Module Base Spec].

2. TDX Connect Objects and Data Model

2.1. System Level Objects and Types

2.1.1. TDX Connect System Enumerations

VMM reads the following global fields (see TDH.SYS.RD) to provide the required number of pages during the TDX Connect objects creation, as the object's internal Metadata.

Table 2.1: TDX Connect Global Fields

Global Field Code	Expected Value (current version)	Description
IOMMU_MT_PAGES_COUNT	4	Required number of pages for IOMMU metadata, see TDH.IOMMU.SETUP
SPDM_MT_PAGES_COUNT	2	Number of pages required for the SPDM Metadata (see TDH.SPDM.CREATE)
SPDM_DOE_PAGES_COUNT	1	Number of pages required for the SPDM request and response buffers (see VMM Common SPDM Interface)
SPDM_MAX_DEV_INFO_PAGES	16	The max number of pages required for the Device Attestation Info, see SPDM_DEVICE_ATTESTATION_INFO_T
IDE_MT_PAGES_COUNT	1	Number of pages required for the IDE Stream Metadata, see TDH.IDE.STREAM.CREATE
TDICS_MT_PAGES_COUNT	1	Number of pages required for the TDI Context metadata, see TDH.TDI.CREATE
LOCK_INTERFACE_FLAGS_SUPPORTED	1	TDISP Interface Lock flags (see [TDISP] supported by the TDX Module. Currently only **NO_FW_UPDATE** (bit 0) is supported.
TDISP_MAX_TDI_REPORT_PAGES	16	Number of pages required for the TDI Interface Report. Maximum TDI Report size is 64KB according to the implicit TDISP 1.0 max report size

Enumeration of TDX Connect sub-features and TDISP optional capabilities support

VMM reads the TDX Connect features support enumeration by reading the **TDX_CONNECT_FEATURES** system Field Code (see TDH.SYS.RD) in order to configure TEE-IO devices according to the capabilities of the TDX Module and the underlying platform.

TDX_CONNECT_FEATURES has the following bitmap structure:

Bit and Name	Description	Const Value

Commented [AA1]: Remove?
Commented [RI2R1]: done
Commented [AA3]: Consistency: Remove this subtitle or add subtitle for the next table
Commented [RI4R3]: done
Commented [AA5]: Why do we want to publish these values? SW shall read them from the enumeration and never use them as constants.
Commented [AA6R5]: Yilun recommended to highlight these values are only valid for the current version.
Commented [RI7R5]: Agree with both. Please review
Commented [AA8R5]: See small comments about the table text.
Commented [RI9R5]: the comments have been addressed
Commented [AA10]: Replace with (see TDH.IOMMU.SETUP)
Commented [RI11R10]: done
Commented [AA12]: Not clear if this is the maximum number, what happens if the VMM provides less pages
Commented [RI13R12]: done
Commented [AA14]: Please change to the type name SPDM_DEVICE(instead of DEV)_...
Commented [RI15R14]: done
Commented [AA18R16]: I wanted to close this comment, could you remind me to which text it was applied?
Commented [RI19R16]: LOCK_INTERFACE FLAG, make it "1", not "0x01"
Commented [AA16]: Change to 1?
Commented [RI17R16]: done
Commented [AA20]: Typo (for for)
Commented [RI21R20]: done
Commented [AA22]: Change title to "Enumeration of TDX Connect sub-features and TDSIP optional capabilities support"
Commented [RI23R22]: done
Commented [AA24]: This text has the wrong formatting and I could barely see it.
Commented [RI25R24]: check once again

0 –T_REQ_WO_PASID	TDX Module supports TDI trusted DMA requests without PASID	1
1 –T_REQ_W_PASID	TDX Module supports TDI trusted DMA requests with PASID	0
2 – T_ATS	TDX Module supports TDI trusted ATS translation requests	0
3 – RESERVED		0
4 – T_PRS	TDX Module supports TDI trusted PRS translation requests	0
5 – T_MSI	TDX Module supports TDI trusted MSI requests	0
6 – T_LINK_IDE	TDX Module supports TDI assignment using Link IDE stream	0
7 – T_SEL_IDE	TDX Module supports TDI assignment using Selective IDE stream	1
8 – T_MMIOL	TDX Module supports trusted MMIO low access	0
9 – T_MMIOH	TDX Module supports trusted MMIO high access	1
10 – T_CFG	TDX Module supports trusted CFG access	0
11 – T_DIRECT_P2P	TDX Module supports TDI direct P2P	0
12 – T_RC_P2P	TDX Module supports TDI root complex access control mediated P2P	1
63:13	Reserved	0

Commented [AA26]: This one is implicit from T_ATS, please remove unless this is coming from the TDISP spec and has a special corner case where ATS is supported without translated request.

Commented [RI27R26]: done

2.1.2. HPA_ARRAY_T Type

Commented [AA28]: Yilun reminded that he already provided the feedback that this type is too similar to another (PAGE_INFO_T) which may be seen as spec quality issue

Commented [RI29R28]: No, we already considered all existing types. There is no matching type giving us the HPA for 1 page

HPA_ARRAY_T is a 64-bit type that addresses a buffer of 1-512 4K pages:

- For a buffer of 1 page, the HPA_ARRAY_T value is the HPA of a 4K shared page,
- For a buffer of 2+ pages, the HPA_ARRAY_T has a pointer to a 4K shared auxiliary page that contains the required number (2-512) of 4K shared pages HPAs (8 bytes x 512 max entries). Bits 52:63 and 0:11 (4K aligned) of each HPA must be 0.
  The pages are not required to be contiguous, but the logical order of the pages is preserved, which is important when HPA_ARRAY_T is addresses a big output buffer.

Table 2.2: HPA_ARRAY_T

Bits	Name	Description
11:0	**RESERVED**	Reserved: must be 0
51:12	**HPA**	Bits 51:12 of the host physical address (including HKID) of a shared page
54:52	**RESERVED**	Reserved: must be 0
63:55	**LAST_IDX**	Last array Index, i.e. the number of HPAs in the array is LAST_IDX + 1.

Commented [AA30]: Bin: This is very confusing to have an array type that can be single page and sometimes is an array. Prefer to simply be an array.

Commented [RI31R30]: We resolved the HPA_ARRAY misunderstanding. This type is not similar to others

Bits	Name	Description
		0 – The array provides 1 HPA. The HPA_ARRY_T raw value is actually the HPA of a single shared 4K page: *HPA_ARRAY_T(HPA, LAST_IDX=0) == HPA*. **1-511** - the HPA field contains the HPA of an auxiliary page that contains LAST_IDX+1 HPAs (64-bit), to address the required number of shared pages.

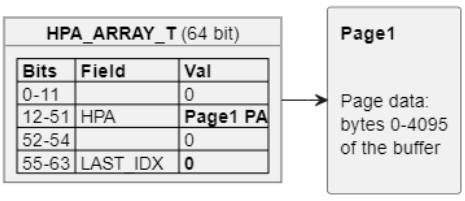
The HPA_ARRAY_T type is used:

- By the VMM as an input parameter in various TDX Connect APIs (e.g. TDH.IOMMU.SETUP, , TDH.SPDM.CREATE, , TDH.IDE.STREAM.CREATE, TDH.TDI.CREATE) to provide the enumerated required number of pages for the TDX Module internal objects' Metadata. (see: TDX Connect System Enumerations).
- By the VMM as an input parameter in various TDX Connect APIs (e.g. TDH.SPDM.CONNECT, TDH.SPDM.MNG, TDH.TDI.BIND) to provide the output buffer of the enumerated required size.
  - If *Required_Num_Of_Pages* == 1:
    - VMM simply passes the HPA of the page in the input register.
  - Else (*Required_Num_Of_Pages* > 1):
    - VMM maps an auxiliary shared 4K page and fills it with *Required_Num_Of_Pages* pages' HPAs.
    - VMM passes *HPA_ARRAY_T(HPA=aux_page_pa, LAST_IDX= Required_Num_Of_Pages - 1)* in the input register.
- By the TDX Module as an output parameter in various TDX Connect APIs (e.g. TDH.SPDM.DELETE, TDH.IDE.STREAM.DELETE, TDH.TDI.REMOVE) to return the HPA(s) of the released Metadata pages' HPAs.
  - If *Required_Num_Of_Pages* == 1:
    - TDX Module returns the HPA of the page in the output register.
    - TDX Module does not use the auxiliary page (AUX_PAGE_PA parameter).
  - Else (*Required_Num_Of_Pages* > 1):
    - TDX Module writes the HPAs of the released pages to the auxiliary page (AUX_PAGE_PA parameter.
    - TDX Module returns *HPA_ARRAY_T(HPA=AUX_PAGE_PA, LAST_IDX = Required_Num_Of_Pages - 1)* in the output register

See the respective TDX Connect API for specific input/output registers.

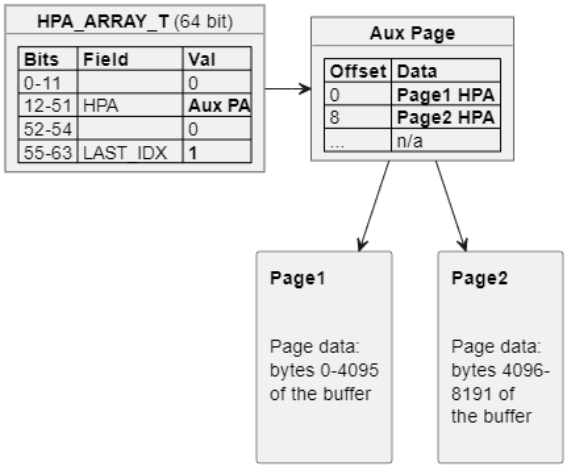
The following figure demonstrates the HPA_ARRAY_T value addressing 1 page.

Figure 2.3: One-page HPA_ARRAY_T example



The following figure demonstrates the HPA_ARRAY_T value addressing 2 pages.

Figure 2.4: Two-pages HPA_ARRAY_T example



2.2. IOMMU

2.2.1. Types and Data Structures

2.2.1.1. IOMMU_ID_T

IOMMU_ID is a handle returned by TDH.IOMMU.SETUP and used to identify the IOMMU in other APIs. .

2.3. SPDM

2.3.1. Constants

Table 2.5: SPDM Constants

Constant Name	Value	Description
MAX_SPDM_SESSIONS	256	Maximum number of SPDM sessions per IOMMU supported by the TDX module

See also the enumerated **SPDM_MT_PAGES_COUNT** and **SPDM_DOE_PAGES_COUNT** in TDX Connect System Enumerations.

2.3.2. Types and Data Structures

2.3.2.1. SPDM_ID

SPDM_ID is a handle returned by TDH.SPDM.CREATE and used to identify the SPDM session.

Interface (ABI) Reference

Commented [AA32]: Typo: handle

Commented [RI33R32]: done

Commented [AA34R32]: Please remove this sentence its confusing and incorrect.

IOMMU_ID is NOT micro architecture type because it is part of the IOMMU and IQ ABI input/output.

Commented [RI35R32]: done

Commented [AA36]: Move to system enumerations

Commented [RI37R36]: VMM already has DOE pages count enumeration. I'll better remove this one. Regarding MAX_SPDM_SESSIONS I tend to leave it a constant: it is not expected to change soon to justify the arch. change. BTW, if you still want the change, maybe we shall go for 256 SPDM sessions and 256 IDE streams per platform, not per IOMMU (as you mentioned multiple times)? Let's talk about it.

Commented [AA38R36]: We can support maximum of 256 SPDM IDs at any given time but this requires an enumeration. Regardless if it's a constant. So, I recommend remove this constant from here and add it as enumeration (lets make 256 global limit for now)

Commented [RI39R36]: We can't make it a global limit without changing the architecture and creating the global TDX Connect context. Today the SPDM context is per IOMMU. Do you want me to open a SC ticket for it?

Commented [AA40]: Typo: Handle (search for the word handler to see if there are additional typos)

Commented [RI41R40]: done

Commented [AA42]: Remove from the EAS this is very confusing and not true. If the way the handle is generated is micro-architecture you don't need to say it. SPDM ID itself is an architectural type as long as it appears in TDX ABI functions.

Commented [RI43R42]: done

2.3.2.2. SPDM_CONFIG_INFO_T

SPDM_CONFIG_INFO_T defines the input parameters structure for TDH.SPDM.CONNECT function.

Commented [AA44]: Typo: SPDM

Search for SDPM on the entire document to see if there are more similar typos.

Commented [RI45R44]: done

Table 2.11: Type Definition of SPDM_CONFIG_INFO_T

Name	SIZE (BYTES)	Description
VMM_SPDM_CAP	4	The *Flags* used in SPDM GET_CAPABILITIES message [refer to GET_CAPABILITIES request message format in the SPDM spec]
SPDM_SESSION_POLICY	1	The SPDM *SessionPolicy* used in the SPDM KEY_EXCHANGE message [refer to KEY_EXCHANGE request message format in the SPDM spec]
CERTIFICATE_SLOT_MASK	1	Device certificate request slot mask, refer to [Certificates and certificate chains in the SPDM spec]
RAW_BITSTREAM_REQUESTED	1	Boolean to dictate the data's stream format, refer to [GET_MEASUREMENTS request attributes in the SPDM spec]
RESERVED	4089	Must be zero

2.3.2.3. SPDM_MNG_PARAM_T

SPDM_MNG_PARAM_T defines the input parameters structure for TDH.SPDM.MNG function, for the Device Info Recollection operation.

Table 2.12: Type Definition of SPDM_MNG_PARAM_T

Name	SIZE (BYTES)	Description
NONCE	32	Nonce value, used in GET_MEASUREMENT in case of DEV_INFO_RECOLLECTION. Zero otherwise
Reserved	4064	Reserved, must be 0.

2.3.2.4. SPDM_DEVICE_ATTESTATION_INFO_T

SPDM_DEVICE_ATTESTATION_INFO_T defines the SPDM session Device Attestation structure, the output for TDH.SPDM.CONNECT and TDH.SPDM.MNG(Device Info Recollection) functions. The size of the structure is variable, up to SPDM_MAX_DEV_INFO_PAGES pages.

Table 2.13: Type Definition of SPDM_DEVICE_ATTESTATION_INFO_T

Name	SIZE (BYTES)	Description
HEARTBEAT_PERIOD	1	The heartbeat period returned from the device in KEY_EXCHANGE_RSP
SPDM_CAPABILITIES	4	Device's SPDM reported capabilities
TDISP_CAPABILITIES	4	The TDISP capabilities returned from the device in GET_TDISP_CAPABILITIES response
TDISP_VERSION	2	The selected TDISP version
CERTIFICATE_0_SIZE	4	Size of CERTIFICATE_0. Zero if CERT_CAP is disabled in VMM_SPDM_CAP
CERTIFICATE_0	Variable	SPDM certificate chain in slot 0
CERTIFICATE_1_SIZE	4	Size of CERTIFICATE_1. Zero if CERT_CAP is disabled in VMM_SPDM_CAP
CERTIFICATE_1	Variable	SPDM certificate chain in slot 1
…	…	Slots 2-6 of SPDM certificates
CERTIFICATE_7_SIZE	4	Size of CERTIFICATE_7. Zero if CERT_CAP is disabled in VMM_SPDM_CAP
CERTIFICATE_7	Variable	SPDM certificate chain in slot 7
MEASUREMENT_SIZE	4	Size of MEASUREMENTS. Zero if MEAS_CAP is disabled in VMM_SPDM_CAP
MEASUREMENT	variable	Concatenation of GET_MEASUREMENTS and MEASUREMENTS as defined in the SPDM spec

2.3.3. VMM Common SPDM Interface

Some TDX Connect Host Interface Functions require sending SPDM requests to device and receiving SPDM responses from the device (via DOE mailbox). The VMM Common SPDM Interface defines a generic call flow and input/output operands to be used by the host Interface Functions that involve communication with devices (see: TDH.SPDM.CONNECT, TDH.SPDM.MNG, TDI.SPDM.DISCONNECT, TDH.IDE.STREAM.KM, TDH.TDI.BIND, TDH.TDI.START and TDH.TDI.STOP).

Input Operands

SPDM_REQ_HPA_ARR –Page buffer of type HPA_ARRAY_T of size SPDM_DOE_PAGES_COUNT (see: TDX Connect System Enumerations) for the SPDM request generated by the TDX Module that the VMM will send to the device.

SPDM_RSP_HPA_ARR –Page buffer of type HPA_ARRAY_T of size SPDM_DOE_PAGES_COUNT (see: TDX Connect System Enumerations) for the SPDM response from the device (no response on the 1st Function call).

SPDM_OUT_HPA_ARR - [where applicable] Page buffer of type HPA_ARRAY_T of the length defined by the function (see: TDH.SPDM.CONNECT, TDH.SPDM.MNG, TDH.TDI.BIND), containing the operation result output, e.g. SPDM device attestation report, TDISP Interface Report.

Output Status

AVX state can be reset on output.

If the output status in RAX:

Commented [AA46]: Clarify this size field is related to the next field of CERT 0.

Also if this structure is inherited from the SPDM spec, please consider using the same structure names and refer to the SPDM spec type.

Same for all the next size fields.

Commented [RI47R46]: done clarification. The structure is defined by TPA (must be in the TPA-NRX spec).

Commented [AA48]: … generated by the TDX Module and …

Commented [RI49R48]: done

Commented [AA50]: Sentence is broken.

Commented [RI51R50]: fixed

Commented [AA52]: Not clear. You may want to indicate the ABIs names that return this object and call out that the length is defined in the ABI itself.

Also you mention below in the RAX status table that if the operation is successful the length is returned in RCX.

Commented [RI53R52]: fixed

Commented [AA54]: Not clear why AVX even appears here. Are you suggesting the ABIs that process SPDM messages can use AVX and that VMM shall save its state before using them? If so, this thing need to be called out and reviewed on TDX WG to ensure this is not breaking any interface assumption today.

Commented [RI55R54]: yes, AVX state is not preserved by the crypto library used for SPDM messages. I thought it was synced with all the parties, but let's check again

Table 2.6: VMM Common SPDM Interfaces Output Statuses

Completion Status Code	VMM Reaction
TDX_SUCCESS	Operation is completed successfully. If Function specified the result output (SPDM_OUT_HPA_ARR), RCX will contain the length of the result output in bytes.
TDX_SPDM_REQUEST	VMM is required to send the DOE+SecureSPDM enveloped request (prepared by TDX Module in SPDM_REQ_HPA_ARR) to device and recall the Function again, with the response from device in SPDM_RSP_HPA_ARR. RCX contains the request length in bytes. VMM shall not call other Functions that use the Common SPDM Interface on same SPDM session if the status is returned.
TDX_OPERAND_BUSY	VMM is required to recall the Function again with the same inputs. SPDM_REQ_HPA_ARR page content is unmodified. VMM shall not call other Functions that use the Common SPDM Interface on same SPDM session if the status is returned.
TDX_SPDM_SESSION_KEY_REQUIRE_REFRESH	The operation is completed unsuccessfully. The SPDM session requires a key refresh. The VMM must refresh the SPDM keys (see TDH.SPDM.MNG) and restart the Function
TDX_SPDM_INVALID_MESSAGE	The operation is completed unsuccessfully due to an invalid SPDM response: decryption/integrity check failure, unexpected or incorrect response.
TDX_SPDM_INVALID_STATE	The operation is completed unsuccessfully. The associated SPDM session is not in the connected state.
Other status	The operation is completed unsuccessfully. VMM shall handle the error according to the error status.

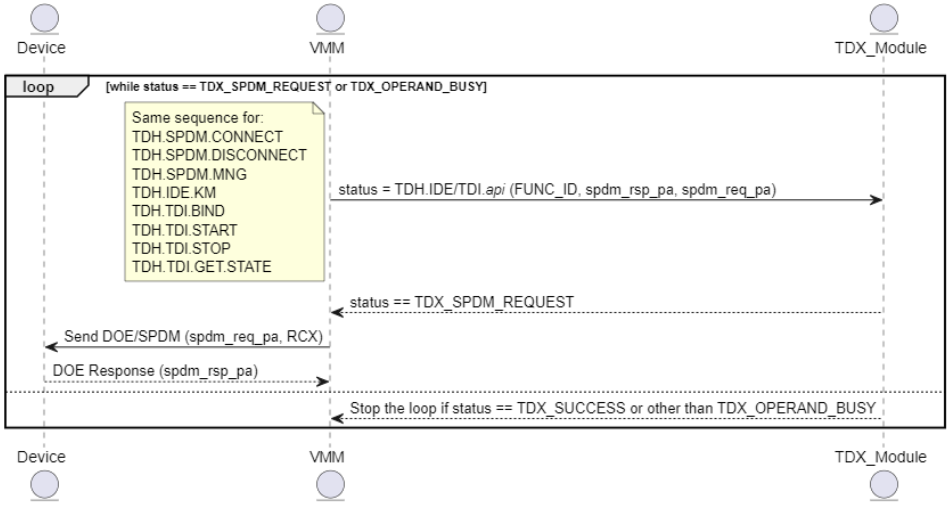
.

Call flow

The VMM calls the Function consequently (in a loop), while the API returning status is TDX_SPDM_REQUEST, or TDX_INTERRUPTED_RESUMABLE or TDX_OPERAND_BUSY, or operation completed successfully (TDX_SUCCESS) or unsuccessfully (other error status). In case of TDX_SPDM_REQUEST, the VMM sends the request to device and provide the device response as the input in the next API call.

For any other status that TDX_SPDM_REQUEST, the OnSPDM_REQ_HPA_ARR page content is unmodified on the output.

Table 2.7: VMM Common SPDM interface sequence diagram



2.4. IDE-Stream

2.4.1. Constants

Table 2.8: IDE-Stream Constants

Constant Name	Value	Description
MAX_IDE_STREAMS	256	Maximum number of IDE streams supported by TDX Connect (derived from IDE spec in PCI-SIG)
MAX_STREAM_COUNT	4	

See also the enumerated **IDE_MT_PAGES_COUNT** in TDX Connect System Enumerations.

2.5. Trusted Device Interface (TDI)

2.5.1. Types and Data Structures

2.5.1.1. TDIMT_IDX_T

Table 2.9: Trusted Device Interface Metadata TDIMT_IDX_T Type Definition

Bits	Field	Description
2:0	TDIMT_LEVEL	0 – TDIMT_L0 1 – TDIMT_L1 2 – TDIMT_L2 3 – TDIMT_ROOT Other – Reserved
31:3	RSVD	Must be 0
63:32	FUNCTION_ID	See: [PCI-SIG, TDISP] v1.0

2.5.1.2. Device Interface Type (TDI_TYPE_T)

Device Interface types enumeration

Table 2.10: TDI_TYPE_T Type Definition

Value	Name	Description
0	**TDI_PFVF**	Physical or virtual function device, identified by RID
Other	**Reserved**	

2.5.1.3. TDI State Machine

The TDX Module is maintaining the internal TDI state machine representing the TDI assignment state (note, the standard TDISP interface states colored green in the table below):

Table 2.11: TDI_SM_T Type Definition

Value	Name	Description	TDX Connect API
0	UNLOCKED	TDISP state is CONFIG_UNLOCKED.	TDH.TDI.CREATE, TDH.TDI.STOP
1	LOCKING	LOCK_INTERFACE_REQUEST issued.	TDH.TDI.BIND
2	LOCKED	TDISP state is CONFIG_LOCKED.	TDH.TDI.BIND
3	REPORT	GET_DEVICE_INTERFACE_REPORT issued and is being processed.	TDH.TDI.BIND
4	BOUND	Interface Report has been received.	TDH.TDI.BIND
5	VALIDATED	TD validated the device and report hashes.	TDG.TDI.VALIDATE
6	ACCEPTED	TD authorized starting the interface.	TDG.TDI.START

Value	Name	Description	TDX Connect API
7	STARTING	START_INTERFACE_REQUEST issued.	TDH.TDI.START
8	RUN	TDISP state is CONFIG_RUN	TDH.TDI.START
9	STOPPING	STOP_INTERFACE_REQUEST issued.	TDH.TDI.STOP
10	ERROR	TDISP state is CONFIG_ERROR	TDH.TDI.GET.STATE
Other	Reserved	n/a	n/a

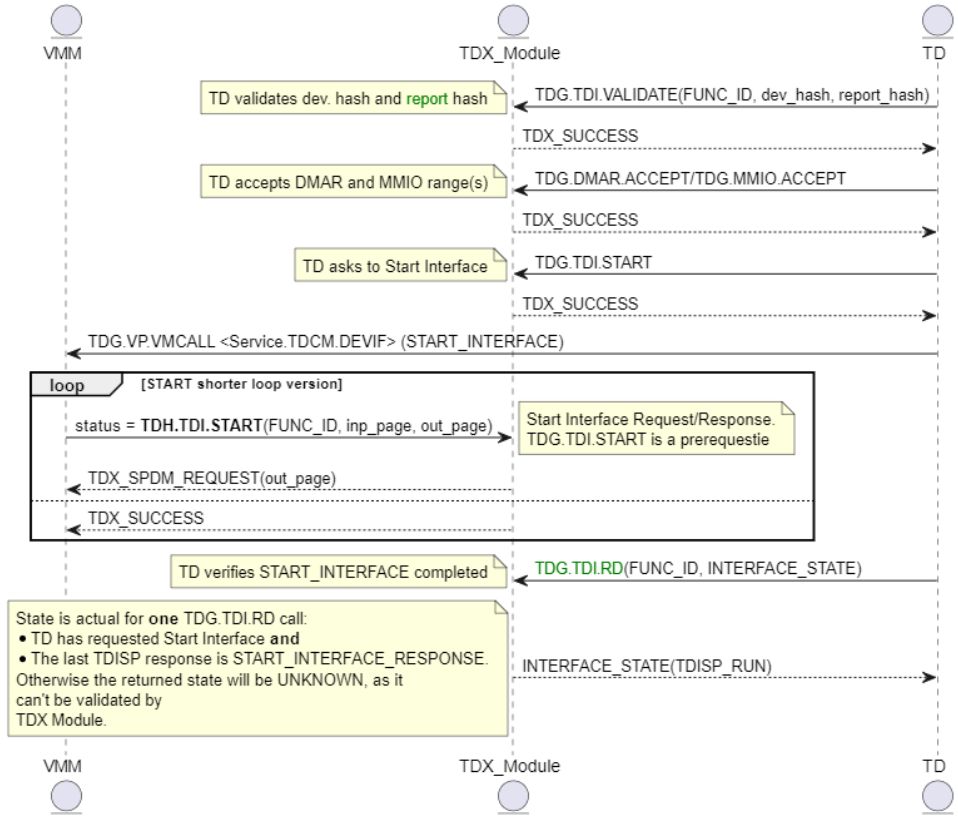
2.5.2. TDI Acceptance Flow

Commented [AA56]: Consider moving this section to somewhere else (for example the main EAS) showing the full process including the TDI bind (lock) and unbind.

Commented [RI57R56]: Agree. On the next iteration I'll hopefully have the Unbind sequence, then will move all the explanatory sub-chapters into the separate chapter. Do you want to remove them for now? Or leave in place for now?

The following sequence diagram demonstrates the TD/VMM flow for the TD TDI acceptance of the bound TDI (see TDH.TDI.BIND) and starting the interface:

Figure 2.12: Sample TDI Acceptance flow



2.6. DMA Remapping (DMAR)

2.6.1. Types and Data Structures

2.6.1.1. DMAR Level (DMAR_LVL_T)

Enumeration of DMAR page levels used by TDX Connect DMAR management functions to address specific DMAR entries

Table 2.13: DMAR_LVL_T Type Definition

Value	Name	Description
0	DMAR_PASIDTE_LVL	DMAR PASID table level
1	DMAR_PDE_LVL	DMAR PASID directory level
2	DMAR_CTE_LVL	DMAR context table level
3	DMAR_RTE_LVL	DMAR root table level

2.6.1.2. DMAR Index (DMAR_IDX_T)

DMAR entry index used by TDX Connect DMAR management functions to address specific DMAR entries

Table 2.14: DMAR_IDX_T Type Definition

Name	Bits	Description
2:0	LEVEL	See: DMAR_LVL_T
11:3	RSVD	Must be zero
31:12	RSVD	Must be zero
63:32	FUNCTION_ID	FUNCTION_ID

TDX Connect maintains DMAR entries Mapping States. A DMAR entry can be in one of the following mapping states:

Table 2.15: DMAR Entry States

DMAR State	Description
DMAR_FREE	DMAR Entry is free
DMAR_PENDING	DMAR Entry is pending TD acceptance (for PASIDTE), not mapped
DMAR_BLOCKED	DMAR Entry is blocked and can be invalidated, not mapped
DMAR_PRESENT	DMAR Entry is present and mapped

Commented [AA58]: Currently PASID is not supported, we may want to make it RSVD must be 0

Commented [RI59R58]: done

Commented [AA60]: Internal?

Commented [RI61R60]: Not really, the states are in use (and we'll explain them more with adding TDG.DMAR.RELEASE)

2.7. MMIO

2.7.1. Types and Data Structures

2.7.1.1. MMIO Metadata Level (MMIOMT_LVL_T)

Commented [AA62]: For consistency, try to use the same description style of similar types such as DMAR_LVL_T or even better, use the same style from base TDX (e.g., SEPT levels)

Commented [RI63R62]: Not sure how you mean. Please clarify

Enumeration of MMIOMT entry levels

Table 2.16: MMIOMT_LVL_T Type Definition

Value	Name	Description
0	MMIOMT_L0	Level 0 MMIOMT entry that can have type QNODE with all present bits clear (i.e. free) or DATA holding the metadata of a 4KB MMIO page
1	MMIOMT_L1	Level 1 MMIOMT entry that can have type QNODE or DATA holding the metadata of a 2MB MMIO page
2	MMIOMT_L2	Level 2 MMIOMT entry that can have type QNODE or DATA holding the metadata of a 1GB MMIO page
3	MMIOMT_L3	Level 3 MMIOMT entry of QNODE type
4	MMIOMT_L4	Level 4 MMIOMT entry of QNODE type

2.7.1.2. MMIOMT Index (MMIOMT_IDX_T)

MMIOMT entry index used by TDX Connect MMIO management functions to address specific MMIO metadata entries

Table 2.17: MMIO_IDX_T Type Definition

Name	Bits	Description
2:0	LEVEL	See: MMIOMT_LVL_T
11:4	RSVD	Must be zero
51:12	PA	MMIO HPA
63:52	RSVD	Must be zero

2.8. IO Invalidation

2.8.1. TD (L1) - Requested Invalidation – (change in progress)

Commented [AA64]: This text is important for clarity but it should be moved to general EAS because there is no data type definition here.

Commented [RI65R64]: Same as the comment above. This text will be changed by the new proposal anyway

The TDX Connect provides a TD (L1 VM) ability to request GPA Ranges IOTLB invalidation (Page-Selective-Within-Domain descriptors) and ensure the invalidations have been completed by the hardware. The invalidation procedure works as follows.

1. L1 requests the invalidation using TDG.IQ.INV.REQUEST where it provides a 4K page of up to 512 GPA Range invalidation requests. TBD: Request format
2. The TDX Module:
   a. Initializes the TD Invalidation Tracker with the snapshot of the IOMMUs associated with L1 (by the TDIs connected at the time of the request).
   b. Requests the invalidation from the VMM with TD-EXIT and the new **TDX_IOTLB_INV_REQUEST** status, providing the number of required invalidation descriptors.

3. The VMM commits the required number of descriptors per each IOMMU associated with the TDIs connected to that TD, using the existing invalidation flow with TDH.IQ.INV.REUQEST(TD_INV_REQ), and TDH.IQ.INV.PROCESS.
   > Note. The VMM chooses the IOMMUs order and the IQ pool size for a commit. The VMM may split the request into multiple commits (TDH.IQ.INV.REQUEST/PROCESS calls). The VMM needs a Wait Descriptor per a commit.

4. L1 verifies that the invalidation request is complete using TDG.IQ.INV.STATUS. After the invalidation request is complete on **all** IOMMUs, the TDX Module will return TDX_SUCESS, confirming to L1 the invalidation request is complete.

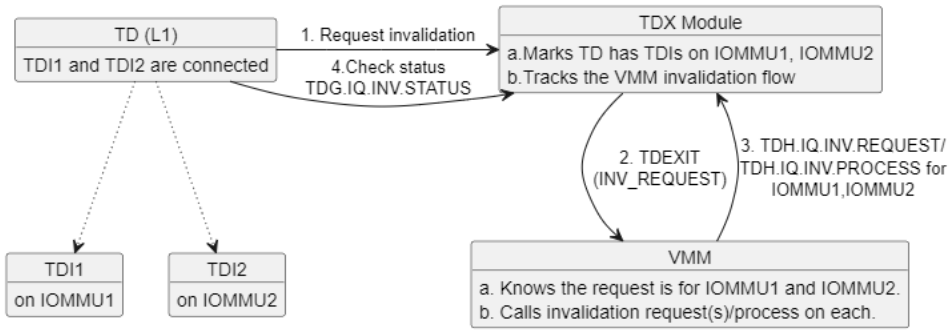
From L1 perspective: L1 requests invalidations and then polls the invalidation status TDG.IQ.INV.STATUS to ensure the invalidations are complete.

From VMM perspective: TDX_IOTLB_INV_REQUEST exit is the request for VMM to commit the required number of descriptors for each IOMMU associated with the TD.

From TDX Module perspective: TDX Module tracks the requested IOTLB invalidations' commitment and processing on each IOMMU associated with L1. After all the descriptors were completed on all IOMMUs, TDX Module returns TDX_SUCEESS on TDG.IQ.INV.STATUS.

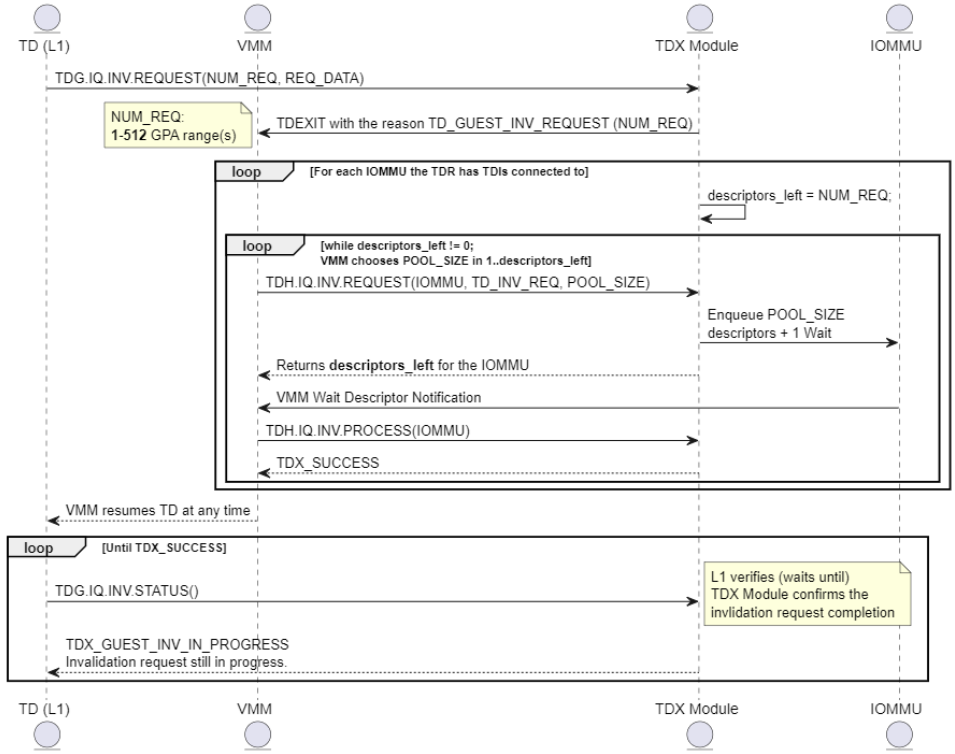
In the following example the TD (L1) has 2 TDIs connected to different IOMMUs.

Figure 2.1: L1-requested IOTLB Invalidation flow example



A generic API flow can be described by the following diagram:

Figure 2.2: TD-requested IOTLB invalidation sequence diagram



Notes:

1. VMM can choose any POOL_SIZE – the number of descriptors it allocates for the call.
2. TDX Module returns the remaining number of descriptors in the request for the IOMMU (0 - no more descriptors required).

3. TDX Connect Interface Functions

3.1. How to Read the Interface Function Definitions

Refer to the [ABI Spec].

3.2. TDX Connect Host-Side (SEAMCALL) Interface Functions

3.2.1. TDH.DMAR.ADD Leaf

Add a DMA Remapping table entry.

Table 3.1: TDH.DMAR.ADD Input Operands Definition

Operand	Name	Description		
RAX	Leaf and Version	SEAMCALL instruction leaf number and version		
		Bits	**Field**	**Description**
		15:0	Leaf Number	Selects the SEAMCALL interface function
		23:16	Version Number	Selects the SEAMCALL interface function version
		63:24	Reserved	Must be 0
RCX	DMAR_INDEX	Index into DMAR remapping table (See: DMAR_IDX_T)		
RDX	DMAR_VAL_1	1st QWORD value of the DMAR entry (corresponding to DMAR index) input.		
R8-R10	DMAR_VAL_2-4	2nd to 4th QWORD values of the DMAR entry (corresponding to DMAR index) input. Must be 0 for DMAR_RTE_LVL and DMAR_PDE_LVL		
R11-R14	DMAR_VAL_5-8	5th to 8th QWORD value of the DMAR entry (corresponding to DMAR index) input. Must be 0 for DMAR_RTE_LVL, DMAR_CTE_LVL and DMAR_PDE_LVL		

Table 3.2: TDH.DMAR.ADD Output Operands Definition

Operand	Description
RAX	SEAMCALL instruction return code
Other	Unmodified

Leaf Function Description

Note: The description below is provided at a high level. Actual details, order of checks, returned status codes, etc. may vary.

TDH.DMAR.ADD adds a DMAR table entry (DMAR_VAL_1-8, depending on the level) for the DMAR level and the device defined by DMAR_INDEX.

The function checks the following pre-conditions:

1. DMAR_INDEX is valid:
   1.1. DMAR_INDEX.FUNCTION_ID is a valid FUNCTION_ID of a device on a TDX Connect-configured IOMMU.
   1.2. DMAR_INDEX.LEVEL <= DMAR_RTE_LVL.
   1.3. DMAR_INDEX reserved fields are zero.
2. DMAR_VAL_1-8 must be 0 for the respective DMAR levels, as ruled by the Input Operand Definition.

If successful, the function does the following:

1. Walk the trusted DMAR table to locate the DMAR entry indicated by DMAR_INDEX, locking the visited DMAR pages in PAMT as Shared, and verifies that the entry is DMAR_FREE.
2. Check the DMAR value is valid per entry type (see the table below, the field details are in [VTd Spec]).

If successful, the function configures the entry's attributes according to the table below.

Table 3.3: VMM and TDX Module Configuration per DMAR Entry Level

DMAR Entry (Level)	VMM Configurable Fields	TDX Module Checks	TDX Module Configured (Fixed) Values
Root table entry (DMAR_RTE_LVL)	LCTP/UCTP	HPA is valid: • Reserved bits are 0. • Reserved HKID bits are 0. • Page is free in PAMT.	• LP/UP: 1 Reserved: 0
Context table entry (DMAR_CTE_LVL)	• PASIDDIRPTR • PDTS	HPA is valid. No PASID Directory overflow. RID_PASID is 0.	• P: 1 • Reserved: 0 DTE, PRE, RID_PRIV: 0 (no SVM support on GNR)
PASID directory entry (DMAR_PDE_LVL)	• SMPTBLPTR • FPD (Fault Processing Disable)	HPA is valid.	P: 1
PASID table entry (DMAR_PASIDTE_LVL)	• FPD • AW (must be 10b for 48 bit, or 11b for 57 bit address width) • PGTT (must be 10b)	n/a	• P: 0 (PASIDTE added as pending) • DID: Per PASID TD owner unique value (DID TDX reserved bit set) • SLPTR: PASID TD owner secure EPTP or first entry in Secure-EPT EPML5 page. • SLEE: 1 (use EPT to determine fetch permissions for DMA) • SLADE: Match TD EPTP controls. • PWSNP, PGSNP: 1 • CD: 0 • METE: 1 • EMT: 6 (WB) • PWT, PCD, PAT, SRE, ERE, FLPM, WPE, NXE, SMEP, EAFE and FLPTR: 0 Reserved: 0

5

Completion Status Codes

Table 3.4: TDH.DMAR.ADD Completion Status Codes (Returned in RAX) Definition

Completion Status Code	Description
TDX_IOMMU_INVALID_STATE	IOMMU is not in IOMMU_CONFIGURED state
TDX_OPERAND_BUSY	Operation encountered a busy operand, indicated by the lower 32 bits of the status. In many cases, this can be resolved by retrying the operation. Note the special case where the indicated operand is TLB_EPOCH. This may happen due to a conflict with TDH.MEM.TRACK or TDH.EXPORT.PAUSE. The host VMM may retry TDH.DMAR.ADD.
TDX_INTERRUPTED_RESUMABLE	Interrupt is pending, the operation can be resumed
TDX_OPERAND_INVALID	
TDX_DMAR_ENTRY_NOT_PRESENT	
TDX_SUCCESS	Operation is successful

Intel TDX Connect Application Binary Interface (ABI) Reference

3.2.2. TDH.DMAR.BLOCK Leaf

Block a DMA remapping table entry.

Table 3.5: TDH.DMAR.BLOCK Input Operands Definition

Operand	Name	Description		
RAX	Leaf and Version	SEAMCALL instruction leaf number and version		
		Bits	**Field**	**Description**
		15:0	Leaf Number	Selects the SEAMCALL interface function
		23:16	Version Number	Selects the SEAMCALL interface function version
		63:24	Reserved	Must be 0
RCX	DMAR_INDEX	Index into DMAR remapping table (See: DMAR_IDX_T)		

Table 3.6: TDH.DMAR.BLOCK Output Operands Definition

Operand	Description
RAX	SEAMCALL instruction return code
Other	Unmodified

Leaf Function Description

Note: The description below is provided at a high level. Actual details, order of checks, returned status codes, etc. may vary.

TDH.DMAR.BLOCK is used by the VMM to block a DMA remapping entry, as the 1st step of DMAR removing sequence allowing the entry invalidation.

Concurrency Note: This function may run concurrently to a TD guest TDG.DMAR.ACCEPT operation. From the host VMM perspective the race does not exist, and the PASID table entry will always be blocked after this operation. From guest TD perspective, the accept may compete successfully or fail with TDX_DMAR_INVALID_MAPPING_STATE error.

The function checks the following pre-conditions:

1. DMAR_INDEX is valid:
   1.1. DMAR_INDEX.FUNCTION_ID is a valid FUNCTION_ID of a device on a TDX Connect-configured IOMMU. DMAR_INDEX.LEVEL <= DMAR_RTE_LVL.
   1.2. DMAR_INDEX reserved fields are zero.
2. Walk the trusted DMAR to the DMAR_INDEX entry with locking the visited DMAR pages in PAMT as Shared is successful.
3. The DMAR entry state is DMAR_PRESENT or DMAR_PENDING.
4. For PASIDTE DMAR entry, TDI MMIO pages must be unmapped first. The VMM is not allowed to block PASIDTE without TD's awareness.

If successful, the function sets the DMAR entry state to DMAR_BLOCKED state.

Completion Status Codes

Table 3.7: TDH.DMAR.BLOCK Completion Status Codes (Returned in RAX) Definition

Completion Status Code	Description
TDX_OPERAND_INVALID	
TDX_OPERAND_BUSY	
TDX_IOMMU_INVALID_STATE	IOMMU is not in IOMMU_CONFIGURED state
TDX_DMAR_ENTRY_NOT_PRESENT	
TDX_DMAR_INVALID_MAPPING_STATE	Only PRESENT or PENDING (for PASIDTE) DMAR states can be blocked. All child nodes must be free to block the entry
TDX_MMIO_STILL_MAPPED	PASIDTE DMAR can be blocked only if the TDI's MMIO pages were unmapped.
TDX_SUCCESS	Operation is successful

3.2.3. TDH.DMAR.RD Leaf

Read a trusted DMAR entry.

Table 3.8: TDH.DMAR.RD Input Operands Definition

Operand	Name	Description		
RAX	Leaf and Version	SEAMCALL instruction leaf number and version		
		Bits	**Field**	**Description**
		15:0	Leaf Number	Selects the SEAMCALL interface function
		23:16	Version Number	Selects the SEAMCALL interface function version
		63:24	Reserved	Must be 0
RCX	DMAR_INDEX	Index into DMAR remapping table (See: DMAR_IDX_T)		

Table 3.9: TDH.DMAR.RD Output Operands Definition

Operand	Description
RAX	SEAMCALL instruction return code
RDX	DMAR state info (see: DMAR_STATE_INFO_T)
R8	1st QWORD value of the DMAR entry (corresponding to DMAR index) input.
R9-R11	2nd to 4th QWORD values of the DMAR entry (corresponding to DMAR index) input. Unmodified for DMAR_RTE_LVL and DMAR_PDE_LVL
R12-R15	5th to 8th QWORD value of the DMAR entry (corresponding to DMAR index) input. Unmodified for DMAR_RTE_LVL, DMAR_CTE_LVL and DMAR_PDE_LVL
Other	Unmodified

Leaf Function Description

Note: The description below is provided at a high level. Actual details, order of checks, returned status codes, etc. may vary.

TDH.DMAR.RD is used by the VMM is used by the VMM to read entry in IOMMU's DMA Remapping structure.

The function checks the following pre-conditions:

1. DMAR_INDEX is valid:
   1.1. DMAR_INDEX.FUNCTION_ID is a valid FUNCTION_ID of a device on a TDX Connect-configured IOMMU.
   1.2. DMAR_INDEX.LEVEL <= DMAR_RTE_LVL.
   1.3. DMAR_INDEX reserved fields are zero.

If successful, the function reads the requested DMAR entry content and returns it in RDX-R15 registers.

Completion Status Codes

Table 3.10: TDH.DMAR.RD Completion Status Codes (Returned in RAX) Definition

Completion Status Code	Description
TDX_OPERAND_INVALID	
TDX_OPERAND_BUSY	
TDX_DMAR_ENTRY_NOT_PRESENT	The DMAR entry is not present
TDX_IOMMU_INVALID_STATE	IOMMU is not in IOMMU_CONFIGURED state
TDX_SUCCESS	Operation is successful

3.2.4. TDH.DMAR.REMOVE Leaf

Remove a trusted DMAR entry.

Table 3.11: TDH.DMAR.REMOVE Input Operands Definition

Operand	Name	Description		
RAX	Leaf and Version	SEAMCALL instruction leaf number and version		
		Bits	**Field**	**Description**
		15:0	Leaf Number	Selects the SEAMCALL interface function
		23:16	Version Number	Selects the SEAMCALL interface function version
		63:24	Reserved	Must be 0
RCX	DMAR_INDEX	Index into DMAR remapping table (See: DMAR_IDX_T)		

Table 3.12: TDH.DMAR.REMOVE Output Operands Definition

Operand	Description
RAX	SEAMCALL instruction return code
RCX	Physical address of first page of the DMAR table removed (0 for PASIDTE)
Other	Unmodified

Leaf Function Description

Note: The description below is provided at a high level. Actual details, order of checks, returned status codes, etc. may vary.

TDH.DMAR.REMOVE is used by the VMM to remove an entry from the IOMMU's DMA Remapping structure.

The function checks the following pre-conditions:

1. DMAR_INDEX is valid:
   1.1. DMAR_INDEX.FUNCTION_ID is a valid FUNCTION_ID of a device on a TDX Connect-configured IOMMU. DMAR_INDEX.LEVEL <= DMAR_RTE_LVL.
   1.2. DMAR_INDEX reserved fields are zero.
2. The DMAR entry state is DMAR_BLOCKED.
3. The DMAR entry Invalidation is DMAR_INV_DONE.
4. For DMAR_RTE_LVL, DMAR_CTE_LVL and DMAR_PDE_LVL, mapping state of all child DMAR table entries must be DMAR_FREE.
5. For PASIDTE entry IOTLB tracking is done.

The function then does the following:

1. For DMAR_RTE_LVL, DMAR_CTE_LVL and DMAR_PDE_LVL: Release the child DMAR table page by setting PT_NDA in PAMT.
2. Set the DMAR entry state to DMAR_FREE.

Completion Status Codes

Table 3.13: TDH.DMAR.REMOVE Completion Status Codes (Returned in RAX) Definition

Completion Status Code	Description
TDX_OPERAND_INVALID	
TDX_OPERAND_BUSY	Operation encountered a busy operand, indicated by the lower 32 bits of the status. In many cases, this can be resolved by retrying the operation. Note the special case where the indicated operand is TLB_EPOCH. This may happen due to a conflict with TDH.MEM.TRACK. The host VMM may retry TDH.DMAR.REMOVE.
TDX_INTERRUPTED_RESUMABLE	Interrupt is pending, the operation can be resumed
TDX_IOMMU_INVALID_STATE	IOMMU is not in IOMMU_CONFIGURED state
TDX_DMAR_ENTRY_NOT_PRESENT	
TDX_DMAR_INVALID_MAPPING_STATE	DMAR entry must be BLOCKED to be removed
TDX_DMAR_INVALID_INV_STATE	DMAR entry invalidation must complete before removal
TDX_IOMMU_IOTLB_TRACKING_NOT_DONE	Outstanding IOTLB tracking and invalidation must be processed before PASIDTE removal.
TDX_SUCCESS	Operation is successful

Intel TDX Connect Application Binary Interface (ABI) Reference

3.2.5. TDH.IDE.STREAM.BLOCK Leaf

Block host RP IDE stream.

Table 3.14: TDH.IDE.STREAM.BLOCK Input Operands Definition

Operand	Name	Description		
RAX	Leaf and Version	SEAMCALL instruction leaf number and version		
		Bits	**Field**	**Description**
		15:0	Leaf Number	Selects the SEAMCALL interface function
		23:16	Version Number	Selects the SEAMCALL interface function version
		63:24	Reserved	Must be 0
RCX	SPDM_ID	SPDM Session Id, see SPDM_ID_T		
RDX	STREAM_ID	Bits **0-7**: IDE-Stream identifier Bits **8-63**: Reserved. Must be zero		

Table 3.15: TDH.IDE.STREAM.BLOCK Output Operands Definition

Operand	Description
RAX	SEAMCALL instruction return code
Other	Unmodified

Leaf Function Description

Note: The description below is provided at a high level. Actual details, order of checks, returned status codes, etc. may vary.

TDH.IDE.STREAM.BLOCK is used by the host VMM to block (disable) an IDE-Stream by clearing the IDE-ECAP and Key Config Bar of the indicated IDE-Stream ID and moving it to IDE non-secure state as defined by the IDE spec.

The function checks the following pre-conditions:

1. SPDM_ID must be valid:
   1.1. SPDM_ID reserved bits are 0.
   1.2. SPDM_ID is on a TDX Connect-configured IOMMU .
2. STREAM_ID is valid:
   2.1. STREAM_ID < MAX_IDE_STREAMS.
   2.2. Check the IDE-Stream is configured and not already blocked.

If successful. the function then blocks the IDE stream, disables the stream in the IDE Stream Control register, clears the stream keys from the Key Config Bar.

Completion Status Codes

Table 3.16: TDH.IDE.STREAM.BLOCK Completion Status Codes (Returned in RAX) Definition

Completion Status Code	Description
TDX_OPERAND_INVALID	
TDX_OPERAND_BUSY	Operation encountered a busy operand, indicated by the lower 32 bits of the status. In many cases, this can be resolved by retrying the operation.
TDX_IOMMU_INVALID_STATE	IOMMU is not in IOMMU_CONFIGURED state
TDX_STREAM_NOT_CONFIGURED	IDE-Stream is not configured
TDX_IDE_STREAM_BLOCKED	IDE-Stream is already blocked
TDX_SUCCESS	Operation is successful

3.2.6. TDH.IDE.STREAM.CREATE Leaf

Create an IDE stream.

Table 3.17: TDH.IDE.STREAM.CREATE Input Operands Definition

<table>
<tr><th>Operand</th><th>Name</th><th colspan="3">Description</th></tr>
<tr><td rowspan="4">RAX</td><td rowspan="4">Leaf and Version</td><td colspan="3">SEAMCALL instruction leaf number and version</td></tr>
<tr><td>Bits</td><td>Field</td><td>Description</td></tr>
<tr><td>15:0</td><td>Leaf Number</td><td>Selects the SEAMCALL interface function</td></tr>
<tr><td>23:16</td><td>Version Number</td><td>Selects the SEAMCALL interface function version</td></tr>
<tr><td>63:24</td><td>Reserved</td><td>Must be 0</td></tr>
<tr><td>RCX</td><td>STREAM_DF_TYPE</td><td colspan="3">Root Port D/F and Stream Type. STREAM_DF_TYPE_T:
<table><tr><th>Name</th><th>Bits</th><th>Description</th></tr><tr><td>RP_DF</td><td>7:0</td><td>Root Port D/F</td></tr><tr><td>STREAM_TYPE</td><td>8</td><td>IDE-Stream type (0: LINK_IDE, 1: SEL_IDE)</td></tr><tr><td>RSVD</td><td>63:9</td><td>Must be zero</td></tr></table></td></tr>
<tr><td>RDX</td><td>SPDM_ID</td><td colspan="3">SPDM Session Id, see SPDM_ID_T</td></tr>
<tr><td>R8</td><td>STREAM_MT_HPA_ARR</td><td colspan="3">Page buffer of type HPA_ARRAY_T, with IDE_MT_PAGES_COUNT free page(s) for hosting TDX module IDE Stream Metadata. See: TDX Connect System Enumerations.</td></tr>
<tr><td>R9</td><td>STREAM_CONTROL</td><td colspan="3">Link/Selective IDE Stream Control Register (see [PCI-IDE])</td></tr>
<tr><td>R10</td><td>RID_ASSOC1</td><td colspan="3">RID association register 1 (see [PCI-IDE])</td></tr>
<tr><td>R11</td><td>RID_ASSOC2</td><td colspan="3">RID association register 2 (see [PCI-IDE])</td></tr>
<tr><td>R12</td><td>ADDR_ASSOC1</td><td colspan="3">Address association register 1 (see [PCI-IDE])</td></tr>
<tr><td>R13</td><td>ADDR_ASSOC2</td><td colspan="3">Address association register 2 (see [PCI-IDE])</td></tr>
<tr><td>R14</td><td>ADDR_ASSOC3</td><td colspan="3">Address association register 3 (see [PCI-IDE])</td></tr>
</table>

Table 3.18: TDH.IDE.STREAM.CREATE Output Operands Definition

Operand	Description
RAX	SEAMCALL instruction return code
RCX	Bits **0:7** - IDE stream ID assigned to the stream. Bits **8:63** are 0
RDX	Bits **0:7** - IDE_ID – ID of the RP IDE configuration register block. The value can be used for a VMM debugging purpose. Bits **8:63** are 0
Other	Unmodified

Leaf Function Description

Note: The description below is provided at a high level. Actual details, order of checks, returned status codes, etc. may vary.

TDH.IDE.STREAM.CREATE is used by the VMM to create IDE-Steam object and configure it as a Link or a Selective IDE stream in the root port IDE-ECAP registers and in the stream Key Config BAR (Note that TDX Connect devices can only use a Selective IDE Stream).

For the TDX Connect devices, the VMM creates a Selective IDE stream, if the Device Policy (returned by TDH.SPDM.CREATE) enumerates the IDE link security (TDX_DEVICE_POLICY_T. LINK_SECURITY == 0). Otherwise, the VMM shall skip creation of IDE stream and move on to the TDIs assignment.

The function checks the following pre-conditions:

1. SPDM_ID is a valid ID of a connected SPDM session.
2. STREAM_DF_TYPE.RP_DF is a valid Root Port of the Configured IOMMU and the Reserved field is zero.
3. The Root Port's IDE ECAP support the required Stream type (Link/Selective) and if there is a free IDE configuration register block on the RP.
4. STREAM_CONTROL is valid:
   - 4.1. Reserved bits are 0.
   - 4.2. ENABLE must be 0.
   - 4.3. Algorithm must be AES_GCM_256_96B_MAC.
   - 4.4. CFG_SEL_IDE must be 0.
   - 4.5. For Selective IDE stream: TX_AGGR_MODE_NPR, TX_AGGR_MODE_PR and TX_AGGR_MODE_CPL must be 0.
5. RID_ASSOC1 and RID_ASSOC2 operands are correct:
   - 5.1. For a Link IDE streams these operands must be 0.
   - 5.2. For Selective IDE streams:
     - 5.2.1. Reserved bits are 0.
     - 5.2.2. RID_ASSOC2. VALID must be 1.
     - 5.2.3. RID range must be a valid RID base/limit range within the IOMMU configured range.
6. Check the ADDR_ASSOC1-3 operands:
   - 6.1. For link IDE streams these operands must be 0
   - 6.2. For Selective IDE streams:
     - 6.2.1. Reserved bits are 0.
     - 6.2.2. ADDR_ASSOC1. VALID must be 1.
     - 6.2.3. Addresses must be valid: no bits beyond MAX_PA-1 and no HKID set.
     - 6.2.4. Address range must be a valid MMIO range within IOMMU configured range.
7. Check that the new IDE Stream RID and Address Association ranges do not overlap with other configured streams.
8. STREAM_MT_HPA_ARR is valid: IDE_MT_PAGES_COUNT page(s) must be correct and free (PT_NDA in PAMT).

If all checks above passed, the function configures the stream in the root port IDE-ECAP registers and in the stream Key Config BAR.

Completion Status Codes

Table 3.19: TDH.IDE.STREAM.CREATE Completion Status Codes (Returned in RAX) Definition

Completion Status Code	Description
TDX_OPERAND_INVALID	
TDX_OPERAND_BUSY	Operation encountered a busy operand, indicated by the lower 32 bits of the status. In many cases, this can be resolved by retrying the operation.
TDX_IOMMU_INVALID_STATE	IOMMU is not in IOMMU_CONFIGURED state
TDX_SPDM_INVALID_STATE	SPDM session must be valid and connected
TDX_IDE_STREAM_NOT_SUPPORTED	IDE-Stream is not supported by the indicated RP
TDX_IDE_STREAM_RID_OVERLAP	Selective IDE-Stream RID association range requested configuration overlaps with another IDE-Stream
TDX_IDE_STREAM_ADDRESS_OVERLAP	Selective IDE-Stream Address range requested configuration overlaps with another IDE-Stream
TDX_NO_FREE_IDE_KEYS	All IDE key slots for the current IOMMU are occupied. The error can happen if the VMM creates a stream during the IDE Key Refresh operation with other streams on the RP. The VMM shall finish all IDE Key Refresh operations and then repeat the call.
TDX_SUCCESS	Operation is successful

3.2.7. TDH.IDE.STREAM.DELETE Leaf

Delete an IDE Stream.

Table 3.20: TDH.IDE.STREAM.DELETE Input Operands Definition

Operand	Name	Description		
RAX	Leaf and Number	SEAMCALL instruction leaf number and version		
		Bits	**Field**	**Description**
		15:0	Leaf Number	Selects the SEAMCALL interface function
		23:16	Version Number	Selects the SEAMCALL interface function version
		63:24	Reserved	Must be 0
RCX	SPDM_ID	SPDM Session Id, see SPDM_ID_T		
RDX	STREAM_ID	Bits **0-7**: IDE-Stream identifier Bits **8-63**: Reserved. Must be zero		
R8	AUX_PAGE_PA	HPA of a shared 4K auxiliary page: • If **IDE_MT_PAGES_COUNT == 1:** TDX Module will not use the auxiliary page, although will check its validity for forward compatibility. The HPA of the released IDE Metadata page will be in the output register. • If **IDE_MT_PAGES_COUNT > 1:** TDX Module uses the auxiliary page for HPA_ARRAY_T object with the released IDE Metadata pages (acquired in TDH.IDE.STREAM.CREATE). The object is returned in the output register, see: HPA_ARRAY_T.		

Table 3.21: TDH.IDE.STREAM.DELETE Output Operands Definition

Operand	Description
RAX	SEAMCALL instruction return code
RCX	HPA_ARRAY_T type containing the released IDE Metadata pages: • If IDE_MT_PAGES_COUNT == 1: the value is the HPA of the released page. • If IDE_MT_PAGES_COUNT > 1: the value is *HPA_ARRAY_T(HPA == AUX_PAGE_PA, LAST_IDX == IDE_MT_PAGES_COUNT - 1)*. The VMM needs to map the AUX_PAGE_PA page to read the HPAs of the released pages.
Other	Unmodified

Leaf Function Description

Note: The description below is provided at a high level. Actual details, order of checks, returned status codes, etc. may vary.

TDH.IDE.STREAM.DELETE is used by the host VMM to remove a blocked IDE-Stream and release its Metadata page(s).

The function checks the following pre-conditions:

1. SPDM_ID is a valid ID of a connected SPDM session.

2. Check STREAM_ID operand:
   2.1. STREAM_ID < MAX_IDE_STREAMS
   2.2. Check that the IDE-Stream is Configured and Blocked.
   2.3. Check that there are no TDIs associated to this IDE-Stream.

If all checks passed, deletes the stream and returns the released Metadata page(s) to VMM.

Completion Status Codes

Table 3.22: TDH.IDE.STREAM.DELETE Completion Status Codes (Returned in RAX) Definition

Completion Status Code	Description
TDX_OPERAND_INVALID	
TDX_OPERAND_BUSY	Operation encountered a busy operand, indicated by the lower 32 bits of the status. In many cases, this can be resolved by retrying the operation.
TDX_IOMMU_INVALID_STATE	IOMMU is not in IOMMU_CONFIGURED state
TDX_STREAM_NOT_CONFIGURED	IDE-Stream is not configured
TDX_IDE_STREAM_NOT_BLOCKED	IDE-Stream is not blocked
TDX_IDE_STREAM_HAS_DEVICE_INTERFACES	IDE-Stream has associated TDIs. VMM must remove the TDIs before deleting the stream.
TDX_SUCCESS	Operation is successful

3.2.8. TDH.IDE.STREAM.KM Leaf

Setups and programs the keys for the IDE Stream.

Table 3.23: TDH.IDE.STREAM.KM Input Operands Definition

Operand	Description		
RAX	SEAMCALL instruction leaf number and version		
	Bits	**Field**	**Description**
	15:0	Leaf Number	Selects the SEAMCALL interface function
	23:16	Version Number	Selects the SEAMCALL interface function version
	63:24	Reserved	Must be 0
RCX	SPDM_ID		SPDM Session Id, see SPDM_ID_T
RDX	STREAM_ID		IDE-Stream identifier
R8	OP_TYPE		Operation type with the IDE stream: **0 – SETUP** : Setup the IDE stream. Programs the given Stream ID Tx/Rx keys for all 3 sub-streams of the IDE stream. **1 – REFRESH** : Refresh the IDE Key. Programs new Tx/Rx keys for all 3 sub-streams of the IDE stream. **2 – STOP** : Stop the stream. Removes the Tx/RX keys from the stream. Other values are reserved.
R9	SPDM_RSP_HPA_ARR		Page buffer of type HPA_ARRAY_T, with SPDM_DOE_PAGES_COUNT free page(s) for the SPDM Response message (Input)
R10	SPDM_REQ_HPA_ARR		Page buffer of type HPA_ARRAY_T, with SPDM_DOE_PAGES_COUNT free page(s) for the SPDM Request message (Output)

Table 3.24: TDH.IDE.STREAM.KM Output Operands Definition

Operand	Description
RAX	SEAMCALL instruction return code
RCX	If RAX returns TDX_SPDM_REQUEST – Generated IDE_KM output message (DOE+Secure SPDM enveloped) length in bytes. For other return codes – 0.
Other	Unmodified

Leaf Function Description

Note: The description below is provided at a high level. Actual details, order of checks, returned status codes, etc. may vary.

TDH.IDE.STREAM.KM is used by the VMM to request a Key Management operation on the stream:

- SETUP the IDE stream (program the stream keys).
- REFRESH the IDE stream keys

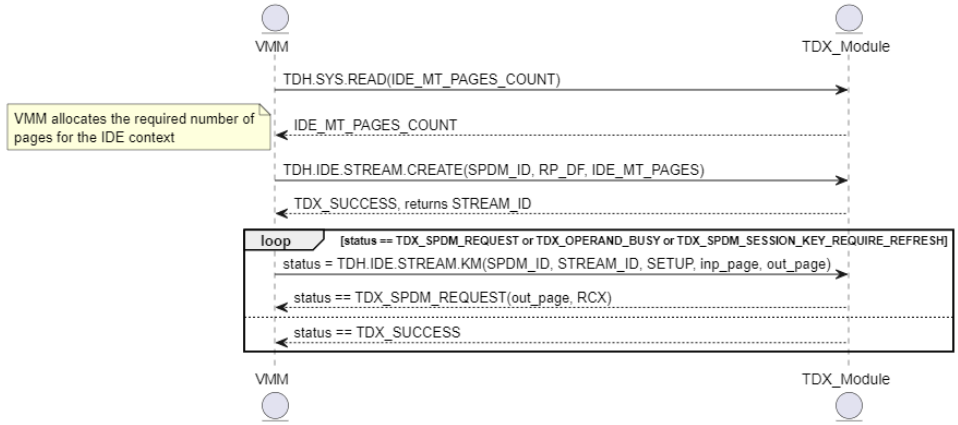
- STOP the IDE stream (issue the KEY_STOP sequence). Note, once a stream has been stopped, it cannot be setup (programmed) again. This STOP operation is terminal and used for a graceful IDE stream stop before the stream block and deletion.

The function creates a series of IDE_KM protocol requests (as defined by the [PCI-SIG IDE] extension) enveloped in a secure SPDM message, and expects from the VMM to provide the device responses. See VMM Common SPDM Interface chapter for more details.

At the end of the Setup or Key Refresh operation, the stream keys have been programmed, acknowledged and activated for 3 sub-streams of the IDE stream and in both Tx/Rx directions. A random key is generated for each sub-stream and direction.

The following example shows a typical VMM flow to create and setup an IDE stream (program its keys):

Table 3.25: Sample flow for the IDE Stream creation and Setup



The function checks the following pre-conditions:

1. SPDM_ID is a valid ID of a connected SPDM session.
2. STREAM_ID is valid:
   2.1. STREAM_ID < MAX_IDE_STREAMS.
   2.2. Check the IDE-Stream is configured and not blocked.
3. Validate the OP_TYPE:
   3.1. Is one of the listed values (SETUP, REFRESH, STOP).
   3.2. Another IDE operation is not active. Nested IDE operations are not allowed.
   3.3. Stream is not already enabled and not in a Stopped state on a SETUP or REFRESH.
   3.4. There are no active TDIs in case of a STOP.

If successful, the function performs the operation required.

Completion Status Codes

Table 3.26: TDH.IDE.STREAM.KM Completion Status Codes (Returned in RAX) Definition

Completion Status Code	Description
TDX_SUCCESS	Operation is successful
TDX_OPERAND_BUSY	Operation encountered a busy operand, indicated by the lower 32 bits of the status. In many cases, this can be resolved by retrying the operation.
TDX_OPERAND_INVALID	
TDX_IOMMU_INVALID_STATE	IOMMU is not in IOMMU_CONFIGURED state
TDX_IDE_STREAM_NOT_CONFIGURED	IDE-Stream must be configured
TDX_IDE_STREAM_BLOCKED	IDE-Stream must not be blocked
TDX_SPDM_ENTRY_NOT_PRESENT	The used SPDM entry ID is not configured in the SPDM directory
TDX_IDE_STREAM_HAS_DEVICE_INTERFACES	STOP is forbidden, IDE stream should not be stopped when there are active TDIs bound to it
TDX_SPDM_INVALID_STATE	SPDM session for this IDE-Stream is not connected or this is a VMM pre-configured IDE-Stream
TDX_INTERRUPTED_RESUMABLE	In case an interrupt occurred while waiting for the keyset to move to a ready state
TDX_RND_NO_ENTROPY	IDE key generation failed due to lack of entropy (VMM should retry the call)
TDX_SPDM_SESSION_KEY_REQUIRE_REFRESH	The SPDM session requires a key refresh
TDX_SPDM_INVALID_MESSAGE	Invalid SPDM message or decryption/integrity check failure
TDX_SPDM_REQUEST	TDX Module's request from the VMM to send the request in OUT_PA with the given length

3.2.9. TDH.IQ.INV.REQUEST Leaf

Enqueue the trusted IQ descriptors for specified I/O invalidation type.

Intel TDX Connect Application Binary Interface (ABI) Reference

Table 3.27: TDH.IQ.INV.REQUEST Input Operands Definition

<table>
<tr><th>Operand</th><th>Name</th><th colspan="4">Description</th></tr>
<tr><td rowspan="4">RAX</td><td rowspan="4">Leaf and Version</td><td colspan="4">SEAMCALL instruction leaf number and version</td></tr>
<tr><td>Bits</td><td colspan="2">Field</td><td>Description</td></tr>
<tr><td>15:0</td><td colspan="2">Leaf Number</td><td>Selects the SEAMCALL interface function</td></tr>
<tr><td>23:16</td><td colspan="2">Version Number</td><td>Selects the SEAMCALL interface function version</td></tr>
<tr><td>63:24</td><td colspan="2">Reserved</td><td>Must be 0</td></tr>
<tr><td>RCX</td><td>IOMMU_ID</td><td colspan="4">IOMMU identifier</td></tr>
<tr><td rowspan="9">RDX</td><td rowspan="9">INV_REQ_TYPE</td><td colspan="4">Bits 0-31: Requested invalidation type</td></tr>
<tr><td>Value</td><td colspan="2">Name</td><td>Description</td></tr>
<tr><td>0</td><td colspan="2">IOTLB_INV_REQ</td><td>IOTLB invalidation</td></tr>
<tr><td>1</td><td colspan="2">RTE_INV_REQ</td><td>Root Entry Invalidation</td></tr>
<tr><td>2</td><td colspan="2">CTE_INV_REQ</td><td>Context Entry Invalidation</td></tr>
<tr><td>3</td><td colspan="2">PDE_INV_REQ</td><td>PASID Directory Invalidation</td></tr>
<tr><td>4</td><td colspan="2">PASIDTE_INV_REQ</td><td>PASID Cache invalidation</td></tr>
<tr><td>5</td><td colspan="2">TD_INV_REQ</td><td>TD Invalidation request</td></tr>
<tr><td>Other</td><td colspan="2">Reserved</td><td></td></tr>
<tr><td></td><td></td><td colspan="4">Bits 32-63: Invalidation Request Data. Depends on the requested invalidation type:
TD_INV_REQ:
• Bits 0:15 - POOL_SIZE number of descriptors VMM commits in this call for the TD invalidation request
• Bits 16:31 – Reserved, 0

All other requests – Reserved, 0.</td></tr>
<tr><td rowspan="3">R8</td><td rowspan="3">INV_TARGET</td><td colspan="4">Invalidation Target</td></tr>
<tr><td colspan="2">INV_REQ_TYPE</td><td colspan="2">Description</td></tr>
<tr><td colspan="2">CTE_INV_REQ
PASIDTE_INV_REQ
RTE_INV_REQ
PDE_INV_REQ</td><td colspan="2">RID/PASID:
Bits 0-15: Requestor ID (BDF)
Bits 16-31: Reserved, must be 0
Bits 32-51: PASID
Bits 52-63: Reserved, must be 0</td></tr>
<tr><td colspan="2">IOTLB_INV_REQ
TD_INV_REQ</td><td colspan="2">TDR page physical address of the TD</td></tr>
<tr><td>R9</td><td>WAIT_INV_DSC_1</td><td colspan="4">Wait invalidation descriptor bits 63:0</td></tr>
<tr><td>R10</td><td>WAIT_INV_DSC_2</td><td colspan="4">Wait invalidation descriptor bits 127:64</td></tr>
<tr><td>R11</td><td>WAIT_INV_DSC_3</td><td colspan="4">Wait invalidation descriptor bits 191:128. Reserved, must be 0</td></tr>
<tr><td>R12</td><td>WAIT_INV_DSC_4</td><td colspan="4">Wait invalidation descriptor bits 255:192. Reserved, must be 0</td></tr>
</table>

Table 3.28: TDH.IQ.INV.REQUEST Output Operands Definition

Operand	Description
RAX	SEAMCALL instruction return code
RCX	For TD_INV_REQ only: Remaining number of requested descriptors for this IOMMU For other requests: 0.
Other	Unmodified

Leaf Function Description

Note: The description below is provided at a high level. Actual details, order of checks, returned status codes, etc. may vary.

TDH.IQ.INV.REQUEST enqueues the IQ descriptors for specified I/O invalidation type INV_REQ_TYPE in the IOMMU invalidation queue.

The following descriptors are queued by this function based on the type of invalidation requested:

RTE_INV_REQ Queues an invalidation wait descriptor. Invoking this invalidation requires one free entry in the invalidation context queue.

CTE_INV_REQ Queues a context-cache invalidation descriptor and an invalidation wait descriptor. Invoking this invalidation requires two free entries in the invalidation queue. The context cache invalidation descriptor is queued with granularity set to global invalidation. The function mask for invalidation is set to 0, i.e., no bits of the SID field are masked for invalidation.

PDE_INV_REQ Queues an invalidation wait descriptor. Invoking this invalidation requires one free entry in the invalidation context queue.

PASIDTE_INV_REQ Queues a PASID-cache invalidation descriptor, IOTLB invalidation descriptor and an invalidation wait descriptor. Invoking this invalidation requires three free entries in the invalidation queue. The PASID-cache invalidation descriptor is queued with granularity set to PASID-selective-within-domain. The IOTLB invalidation descriptor is queued with granularity set to domain-selective and the DR/DW bits set to 1.

IOTLB_INV_REQ Queues an IOTLB invalidation descriptor and invalidation wait descriptor. Invoking this invalidation request requires two free entries in the invalidation queue. The IOTLB invalidation descriptor is queued with granularity set to domain-selective and the DR/DW bits set to 1.

TD_INV_REQ Queues POOL_SIZE number of IOTLB invalidation descriptors from TD (L1) request for the given IOMMU. TDX Module queues The IOTLB invalidation descriptors with DID of TD. This invalidation request requires POOL_SIZE + 1 free entries in the invalidation queue.

The VMM is required to provide an invalidation wait descriptor with the request. The VMM must specify the fence flag (FN) as 1 for this descriptor. The page-request drain may be specified as 1. The wait descriptor queued by the VMM may set the IF to request an interrupt to be generated on completion of the invalidation. The VMM may set the status write (SW) bit to 1 and provide a status address and data. The status address specified must be a shared PA.

TDH.IQ.INV.REQUEST checks the following pre-conditions:

1. IOMMU_ID is valid and the IOMMU is in the Configured state.
2. INV_REQ_TYPE must be valid and trusted IQ must have enough space per INV_REQ_TYPE.
3. INV_TARGET must be valid per INV_REQ_TYPE.
4. For RTE, CTE, PDE, PASIDTE invalidation requests, the DMAR entry state is blocked (see TDH.DMAR.BLOCK).
5. For IOTLB and TD invalidation requests, TD keys are configured, and the TD is initialized for the provided TDR.
6. For IOTLB_INV_REQ request, IOTLB invalidation is required for the TD and IOMMU and not already in progress.
7. For TD_INV_REQ request, TD request is progress, and invalidations are required for the IOMMU.

Intel TDX Connect Application Binary Interface (ABI) Reference

8. Check that WAIT_INV_DSC_1 is valid:
   8.1. Type must be 0x05.
   8.2. Reserved fields must be 0.
   8.3. PD must be 0 if IOMMU ECAP_REG.PSD is 0.
   8.4. FN must be 1.
9. Check that WAIT_INV_DSC_2 (STATUS_ADDRESS) is a valid, 4B aligned and shared HPA address.
10. Check that WAIT_INV_DSC_3 and WAIT_INV_DSC_4 are equal to 0.

If successful, the function queues the invalidation descriptors into the IOMMU trusted IQ.

Completion Status Codes

Table 3.29: TDH.IQ.INV.REQUEST Completion Status Codes (Returned in RAX) Definition

Completion Status Code	Description
TDX_OPERAND_INVALID	
TDX_OPERAND_BUSY	
TDX_OPERAND_ADDR_RANGE_ERROR	
TDX_OPERAND_PAGE_METADATA_INCORRECT	
TDX_IOMMU_IQ_QUEUE_FULL	IOMMU trusted IQ does not have free space for the requested invalidation type
TDX_IOMMU_IOTLB_TRACKING_NOT_REQUIRED	IOTLB invalidation for the requested IOMMU and TD is not required
TDX_GUEST_INV_NOT_REQUIRED	There are no uncommitted TD IOTLB descriptors for this IOMMU, or no active TD request.
TDX_IOMMU_IOTLB_TRACKING_NOT_DONE	IOTLB tracking for the requested IOMMU and TD is in-progress
TDX_SUCCESS	Operation is successful

Intel TDX Connect Application Binary Interface (ABI) Reference

3.2.10. TDH.IQ.INV.PROCESS Leaf

Tracks and process the completed trusted IOMMU invalidations.

Table 3.30: TDH.IQ.INV.PROCESS Input Operands Definition

Operand	Name	Description		
RAX	Leaf and Version	SEAMCALL instruction leaf number and version		
		Bits	**Field**	**Description**
		15:0	Leaf Number	Selects the SEAMCALL interface function
		23:16	Version Number	Selects the SEAMCALL interface function version
		63:24	Reserved	Must be 0
RCX	IOMMU_ID	IOMMU identifier		

Table 3.31: TDH.IQ.INV.PROCESS Output Operands Definition

Operand	Description
RAX	SEAMCALL instruction return code
Other	Unmodified

Leaf Function Description

Note: The description below is provided at a high level. Actual details, order of checks, returned status codes, etc. may vary.

TDH.IQ.INV.PROCESS tracks completed invalidations. In each invocation the module may process 0 or more invalidation completions.

TDH.IQ.INV.PROCESS checks the following pre-conditions:

1. IOMMU_ID is valid and the IOMMU is in the Configured state.

If successful, the function updates the internal TDX Module invalidation context/tracker and marks the completed entries.

Completion Status Codes

Table 3.32: TDH.IQ.INV.PROCESS Completion Status Codes (Returned in RAX) Definition

Completion Status Code	Description
TDX_OPERAND_INVALID	
TDX_OPERAND_BUSY	
TDX_INTERRUPTED_RESUMABLE	
TDX_SUCCESS	Operation is successful

3.2.11. TDH.IOMMU.SETUP Leaf

Setup the I/O stack to work in the secure (TDX) Mode.

Table 3.33: TDH.IOMMU.SETUP Input Operands Definition

Operand	Description		
RAX	SEAMCALL instruction leaf number and version		
	Bits	**Field**	**Description**
	15:0	Leaf Number	Selects the SEAMCALL interface function
	23:16	Version Number	Selects the SEAMCALL interface function version
	63:24	Reserved	Must be 0
RCX	IOMMU_VTBAR		The IOMMU VTBAR address
RDX	IOMMU_MT		HPA of the IOMMU metadata, when called on IOMMU_INIT state, otherwise ignored.

Table 3.34: TDH.IOMMU.SETUP Output Operands Definition

Operand	Description
RAX	SEAMCALL instruction return code
RCX	IOMMU_ID_T
Other	Unmodified

Input Parameters

IOMMU_MT metadata is a 4K page that has the following structure:

Table 3.35: Structure of IOMMU_MT Parameter

Offset	Size	Description	TDX Module Checks
0	8	HPA and size of the 1st contiguous buffer page. Bits 0-11: NUM_PAGES Bits 12-: HPA The buffer size is NUM_IQ_PAGES pages	NUM_PAGES must not exceed 128 pages and must be a power of 2. The NUM_PAGES contiguous pages must be free (PT_NDA)
8	8	HPA and size of the 2nd contiguous buffer page. Bits 0-11: NUM_PAGES Bits 12-: HPA The buffer size is NUM_IQ_PAGES pages	The NUM_PAGES pages must be equal to the NUM_PAGES of the 1st contiguous buffer. The NUM_PAGES contiguous pages must be free (PT_NDA)
16	8	HPA1 of a page for IOMMU metadata	Page must be free (PT_NDA)
24	8	HPA2 of a page for IOMMU metadata	
…		VMM shall provide **IOMMU_MT_PAGES_COUNT** HPAs of PT_NDA pages for IOMMU metadata.	

		The rest of the page from (IOMMU_MT_PAGE_COUNT * 8 + 16) is reserved and must be zero.	

Leaf Function Description

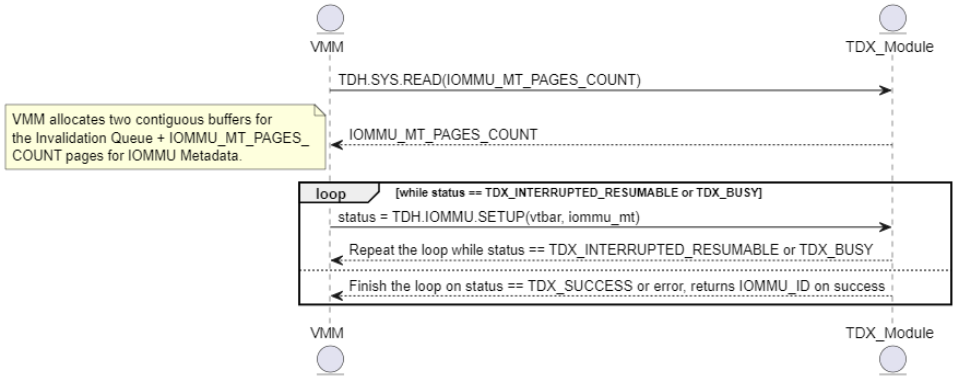
Note: The description below is provided at a high level. Actual details, order of checks, returned status codes, etc. may vary.

TDH.IOMMU.SETUP Setups the IOMMU(VTBAR), HIOP, KCBAR, PCIe Root Ports and other I/O hardware IPs to work in the Secure TDX Mode.

The IOMMU and Root Ports hardware configuration can be time consuming, therefore the VMM is required to call this function repeatedly, in case it returns TDX_INTERRUPTED_RESUMABLE or TDX_BUSY statuses.

The VMM shall read the IOMMU_MT_PAGES value, prepare the required number of pages for the IOMMU metadata and the invalidation contiguous buffers, then call the function, as demonstrated in the flow below:

Table 3.36: TDH.IOMMU.SETUP flow



TDH.IOMMU.SETUP checks the following pre-conditions:

1. There is the IOMMU with the VTBAR address IOMMU_VTBAR, and the IOMMU is not in Configured state.
2. IOMMU_MT is a valid shared HPA and the content is correct (see the TDX Module Checks column in the IOMMU_MT structure definition above).
3. VTBAR is in the valid state. Fail with TDX_IOMMU_INVALID_STATE in case of an invalid VTBAR state:
    a. ECRSP_REG.IP == 0 (No enhanced command in progress).
    b. ESTS0_REG.TMS == 0 (IOMMU not in TDX mode).
    c. GSTS.RTPS == 1 (VMM's RTADDR is latched).
    d. GSTS.QIES == 1 (VMM's queued invalidations enabled).
    e. GSTS.TES == 1 (DMA translations enabled).
    f. PMEN.EPM == 0 (DMA access to PRM disabled).
    g. PMEN.PRS == 0 (PRM status disabled).
    h. RTADDR_REG.TTM == 01b (DMA translation is in scalable mode).
4. No pre-configured VMM IDE-Streams exist in the KCBAR.

If successful, the function configures the hardware and returns TDX_SUCCESS upon the configuration is complete and the IOMMU is in the Configured state.

IOMMU and Root Ports hardware configuration can be time consuming. The function checks for pending interrupts and may return TDX_INTERRUPTED_RESUMABLE. In this case the VMM must repeat TDH.IOMMU.SETUP call while TDX_INTERRUPTED_RESUMABLE or TDX_BUSY are returned.

Completion Status Codes

Table 3.37: TDH.IOMMU.SETUP Completion Status Codes (Returned in RAX) Definition

Completion Status Code	Description
TDX_OPERAND_BUSY	
TDX_OPERAND_INVALID	
TDX_SUCCESS	
TDX_IOMMU_INVALID_STATE	
TDX_IOMMU_RP_INVALID_CONFIGURATION	Invalid RP registers configuration
TDX_IOMMU_ECMD_ERROR	IOMMU TDX Mode programming error
TDX_INTERRUPTED_RESUMABLE	Interrupt is pending or IOMMU ECMD command timeout. VMM shall repeat the call

3.2.12. TDH.IOMMU.CLEAR Leaf

Releases the Root Ports and the IOMMU to VMM.

Table 3.38: TDH.IOMMU.CLEAR Input Operands Definition

Operand	Description		
RAX	SEAMCALL instruction leaf number and version		
	Bits	**Field**	**Description**
	15:0	Leaf Number	Selects the SEAMCALL interface function
	23:16	Version Number	Selects the SEAMCALL interface function version
	63:24	Reserved	Must be 0
RCX	IOMMU_ID		The IOMMU ID
RDX	RELEASED_IOMMU_MT_PA		HPA of a 4K shared page, where the TDX Module writes the HPAs of the released pages (acquired during TDH.IOMMU.SETUP). The TDX Module will return (IOMMU_MT_PAGES_COUNT + INVALIDATION_IQ_SIZE$_{\text{from the setup}}$ * 2) HPAs to the VMM.

Table 3.39: TDH.IOMMU.CLEAR Output Operands Definition

Operand	Description
RAX	SEAMCALL instruction return code
Other	Unmodified

Leaf Function Description

Note: The description below is provided at a high level. Actual details, order of checks, returned status codes, etc. may vary.

TDH.IOMMU.CLEAR terminates the TDX Mode on the IOMMU, restores the H/W access to VMM.

The IOMMU and Root Ports hardware teardown can be time consuming, therefore the VMM is required to call this function repeatedly, in case it returns TDX_INTERRUPTED_RESUMABLE or TDX_BUSY statuses.

Intel TDX Connect Application Binary Interface (ABI) Reference

TDH.IOMMU.CLEAR checks the following pre-conditions:

1. IOMMU_ID is valid and the IOMMU is in the Configured state.
2. There are no active SPDM sessions.
3. There are no invalidations in progress.
4. The Trusted DMAR Root table is empty.

If successful, the function performs the I/O stack (IOMMU and Root Ports) teardown and returns TDX_SUCCESS upon the teardown is complete and the IOMMU is in the Init state.

IOMMU and Root Ports hardware teardown can be time consuming. The function checks for IOMMU timeout and may return TDX_INTERRUPTED_RESUMABLE. In this case the VMM must repeat TDH.IOMMU.CLEAR call while TDX_INTERRUPTED_RESUMABLE or TD_BUSY are returned.

Completion Status Codes

Table 3.40: TDH.IOMMU.CLEAR Completion Status Codes (Returned in RAX) Definition

Completion Status Code	Description
TDX_OPERAND_BUSY	
TDX_OPERAND_INVALID	
TDX_SUCCESS	
TDX_IOMMU_INVALID_STATE	
TDX_INTERRUPTED_RESUMABLE	Interrupt is pending, the operation shall be resumed
TDX_IOMMU_ECMD_ERROR	IOMMU TDX Mode programming error

Intel TDX Connect Application Binary Interface (ABI) Reference

3.2.13. TDH.MEM.SHARED.SEPT.WR Leaf

Add mapping from SEPT into shared GPA pages under VMM control,to allow TDX Connect devices DMA into shared GPA space of a TD.

Table 3.41: TDH.MEM.SHARED.SEPT.WR Input Operands Definition

Operand	Description		
RAX	SEAMCALL instruction leaf number and version		
	Bits	**Field**	**Description**
	15:0	Leaf Number	Selects the SEAMCALL interface function
	23:16	Version Number	Selects the SEAMCALL interface function version
	63:24	Reserved	Must be 0
RCX	EPT mapping information:		
	Bits	**Name**	**Description**
	2:0	Level	Level of the Secure EPT entry to write Level must be the level of the Secure EPT page which is indexed the GPA Shared bit (i.e. GPAW + 3).
	11:3	Reserved	Reserved: must be 0
	51:12	GPA	Bits 51:12 of the shared guest physical address for the Secure EPT entry to write. Depending on the level, the following least significant bits must be 0: Level 3 (EPML4E): Bits 38:12 Level 4 (EPML5E): Bits 47:12
	63:52	Reserved	Reserved: must be 0
RDX	Host physical address of the parent TDR page (HKID bits must be 0)		
R8	EPT entry value		

Table 3.42: TDH.MEM.SHARED.SEPT.WR Output Operands Definition

Operand	Description
RAX	SEAMCALL instruction return code
RCX	Extended error information part 1 In case of EPT walk error, Secure EPT entry architectural content where the error was detected – see [ABI Spec] The architectural content represents how the Secure EPT maps a private memory page or a Secure EPT page, and may be different than the actual contents of the Secure EPT entry. Software should consult the Secure EPT information returned in RDX. In other cases, RCX returns 0.
RDX	Extended error information part 2 In case of EPT walk error, Secure EPT entry level and state where the error was detected – see [ABI Spec] In other cases, RDX returns 0.

Operand	Description
Other	Unmodified

Leaf Function Description

Note: The description below is provided at a high level. Actual details, order of checks, returned status codes, etc. may vary.

Add mapping of SEPT shared entries to shared EPT pages for DMA translations of shared GPAs using the Secure-EPT.

The function checks the following conditions:

1. The TDR is valid and the TD is initialized.
2. The GPA shared bit is set, and the specified level is the level indexed by GPA.SHARED bit (i.e., GPAW +3)

If successful, the function walks the Secure EPT based on the GPA operand and find the Secure EPT entry and write the EPT entry value specified by the VMM into it.

Completion Status Codes

Table 3.43: TDH.MEM.SHARED.SEPT.WR Completion Status Codes (Returned in RAX) Definition

Completion Status Code	Description
TDX_EPT_WALK_FAILED	
TDX_OPERAND_ADDR_RANGE_ERROR	
TDX_OPERAND_BUSY	
TDX_OPERAND_INVALID	
TDX_TDCS_NOT_ALLOCATED	
TDX_PAGE_METADATA_INCORRECT	
TDX_SUCCESS	
TDX_SYS_NOT_READY	
TDX_SYS_SHUTDOWN	
TDX_TD_FATAL	
TDX_TD_KEYS_NOT_CONFIGURED	
TDX_TD_NOT_INITIALIZED	

3.2.14. TDH.MMIO.BLOCK Leaf

Block an MMIO page.

Table 3.44: TDH.MMIO.BLOCK Input Operands Definition

Operand	Description		
RAX	SEAMCALL instruction leaf number and version		
	Bits	**Field**	**Description**
	15:0	Leaf Number	Selects the SEAMCALL interface function
	23:16	Version Number	Selects the SEAMCALL interface function version
	63:24	Reserved	Must be 0
RCX	EPT mapping information:		
	Bits	**Name**	**Description**
	2:0	Level	Level of the EPT entry that will map the blocked page: must be 0 (4KB), 1 (2MB) or 2 (1GB)
	11:3	Reserved	Reserved: must be 0
	51:12	GPA	Bits 51:12 of the guest physical address to be mapped for the blocked Secure EPT page
	63:52	Reserved	Reserved: must be 0
RDX	Host physical address of the parent TDR page (HKID bits must be 0)		

Table 3.45: TDH.MMIO.BLOCK Output Operands Definition

Operand	Description
RAX	SEAMCALL instruction return code
RCX	Extended error information part 1 In case of EPT walk error, Secure EPT entry architectural content where the error was detected – see [ABI Spec] The architectural content represents how the Secure EPT maps a private memory page or a Secure EPT page, and may be different than the actual contents of the Secure EPT entry. Software should consult the Secure EPT information returned in RDX. In other cases, RCX returns 0.
RDX	Extended error information part 2 In case of EPT walk error, Secure EPT entry level and state where the error was detected – see [ABI Spec] In other cases, RDX returns 0.
Other	Unmodified

Leaf Function Description

Note: The description below is provided at a high level. Actual details, order of checks, returned status codes, etc. may vary.

TDH.MMIO.BLOCK is used by the VMM to block a MMIO private GPA mapping.

The function checks the following conditions:

1. The TDR is valid, and the TD is initialized.

If successful, the function blocks the MMIO page if it's state is MAPPED or PENDING.

Completion Status Codes

Table 3.46: TDH.MMIO.BLOCK Completion Status Codes (Returned in RAX) Definition

Completion Status Code	Description
TDX_EPT_ENTRY_NOT_FREE	
TDX_EPT_WALK_FAILED	
TDX_OPERAND_ADDR_RANGE_ERROR	
TDX_OPERAND_BUSY	
TDX_OPERAND_INVALID	
TDX_OPERAND_PAGE_METADATA_INCORRECT	
TDX_SUCCESS	
TDX_SYS_NOT_READY	
TDX_SYS_SHUTDOWN	
TDX_TD_FATAL	
TDX_TD_KEYS_NOT_CONFIGURED	
TDX_TD_NOT_FINALIZED	
TDX_TD_NOT_INITIALIZED	

Intel TDX Connect Application Binary Interface (ABI) Reference

3.2.15. TDH.MMIO.MAP Leaf

Map MMIO page to a TD as private GPA.

Table 3.47: TDH.MMIO.MAP Input Operands Definition

Operand	Description		
RAX	SEAMCALL instruction leaf number and version		
	Bits	**Field**	**Description**
	15:0	Leaf Number	Selects the SEAMCALL interface function
	23:16	Version Number	Selects the SEAMCALL interface function version
	63:24	Reserved	Must be 0
RCX	EPT mapping information:		
	Bits	**Name**	**Description**
	2:0	Level	Level of the EPT entry that will map the new page: must be 0 (4KB), 1 (2MB) or 2 (1GB)
	11:3	Reserved	Reserved: must be 0
	51:12	GPA	Bits 51:12 of the guest physical address to be mapped for the new Secure EPT page
	63:52	Reserved	Reserved: must be 0
RDX	Host physical address of the parent TDR page (HKID bits must be 0)		
R8	Host physical address of the target MMIO page to be mapped to the TD (HKID bits must be 0)		

Table 3.48: TDH.MMIO.MAP Output Operands Definition

Operand	Description
RAX	SEAMCALL instruction return code
RCX	Extended error information part 1 In case of EPT walk error, Secure EPT entry architectural content where the error was detected – see [ABI Spec]. The architectural content represents how the Secure EPT maps a private memory page or a Secure EPT page, and may be different than the actual contents of the Secure EPT entry. Software should consult the Secure EPT information returned in RDX. In other cases, RCX returns 0.
RDX	Extended error information part 2 In case of EPT walk error, Secure EPT entry level and state where the error was detected – see [ABI Spec]. In other cases, RDX returns 0.
Other	Unmodified

Leaf Function Description

Note: The description below is provided at a high level. Actual details, order of checks, returned status codes, etc. may vary.

TDH.MMIO.MAP adds a 4KB, 2MB or a 1GB private MMIO page mapping to a TD. The MMIO page is mapped in a pending state and can be accessed only by the guest TD after it accepts it using TDCALL(TDG.MMIO.ACCEPT). TDH.MMIO.MAP does not access the MMIO page.

The function checks the following conditions:

1. The TDR is valid, and the TD is initialized.
2. The target MMIO metadata is correct and the TD is the owner of the MMIO page.

If successful, the function updates the Secure EPT entry with the target page HPA and MMIO_PENDING state.

Completion Status Codes

Table 3.49: TDH.MMIO.MAP Completion Status Codes (Returned in RAX) Definition

Completion Status Code	Description
TDX_EPT_ENTRY_NOT_FREE	
TDX_EPT_WALK_FAILED	
TDX_OPERAND_ADDR_RANGE_ERROR	
TDX_OPERAND_BUSY	Operation encountered a busy operand, indicated by the lower 32 bits of the status. In many cases, this can be resolved by retrying the operation.
TDX_OPERAND_INVALID	
TDX_OPERAND_PAGE_METADATA_INCORRECT	
TDX_SUCCESS	Operation is successful.
TDX_SYS_NOT_READY	
TDX_SYS_SHUTDOWN	
TDX_TD_FATAL	
TDX_TD_KEYS_NOT_CONFIGURED	
TDX_TD_NOT_FINALIZED	
TDX_TD_NOT_INITIALIZED	

Intel TDX Connect Application Binary Interface (ABI) Reference

3.2.16. TDH.MMIO.MT.ADD Leaf

Add MMIO metadata page.

Table 3.50: TDH.MMIO.MT.ADD Input Operands Definition

Operand	Description		
RAX	SEAMCALL instruction leaf number and version		
	Bits	**Field**	**Description**
	15:0	Leaf Number	Selects the SEAMCALL interface function
	23:16	Version Number	Selects the SEAMCALL interface function version
	63:24	Reserved	Must be 0
RCX	MMIOMT_IDX - MMIOMT index (MMIOMT_IDX_T)		
RDX	MMIOMT_PA - The physical address of the new allocated MMIONT page		

Table 3.51: TDH.MMIO.MT.ADD Output Operands Definition

Operand	Description
RAX	SEAMCALL instruction return code
Other	Unmodified

Leaf Function Description

Note: The description below is provided at a high level. Actual details, order of checks, returned status codes, etc. may vary.

TDH.MMIO.MT.ADD used by the VMM to build the MMIOMT tree. The VMM provides the HPA address to allocate the new MMIOMT page and the MMIOMT index (MMIO HPA address and parent entry level).

The function checks the following conditions:

1. MMIOMT_IDX is valid:
   1.1. Reserved fields must be 0.
   1.2. LEVEL must be valid.
   1.3. PA must be a valid 4KB aligned HPA with HKID bits equal to 0.
2. MMIOMT_PA page is free (PT_NDA type in PAMT).

If successful, the function does the following:

1. Walk MMIOMT tree using MMOMT_IDX to get the new MMIOMT page parent entry.
2. Check that the parent entry is not already mapped. Fail otherwise.
3. Map the MMIOMT parent entry to point the new MMIOMT page.
4. Initialize the new MMIOMT page to 0 using MOVDIR64 and TDX reserved HKID.
5. Update PAMT of the new MMIOMT page with type PT_MMIOMT.

Completion Status Codes

Table 3.52: TDH.MMIO.MT.ADD Completion Status Codes (Returned in RAX) Definition

Completion Status Code	Description
TDX_OPERAND_ADDR_RANGE_ERROR	

Completion Status Code	Description
TDX_OPERAND_BUSY	
TDX_OPERAND_INVALID	
TDX_OPERAND_PAGE_METADATA_INCORRECT	
TDX_SUCCESS	

3.2.17. TDH.MMIO.MT.RD Leaf

Read MMIOMT entry.

Table 3.53: TDH.MMIO.MT.RD Input Operands Definition

Operand	Description		
RAX	SEAMCALL instruction leaf number and version		
	Bits	**Field**	**Description**
	15:0	Leaf Number	Selects the SEAMCALL interface function
	23:16	Version Number	Selects the SEAMCALL interface function version
	63:24	Reserved	Must be 0
RCX	MMIOMT_IDX - MMIOMT index (MMIOMT_IDX_T)		

Table 3.54: TDH.MMIO.MT.RD Output Operands Definition

Operand	Description
RAX	SEAMCALL instruction return code
RCX, RDX, R8, R9	MMIOMT DATA or QNODE entry value
Other	Unmodified

Leaf Function Description

Note: The description below is provided at a high level. Actual details, order of checks, returned status codes, etc. may vary.

TDH.MMIO.MT.RD This function is used by the VMM to read MMIOMT entry content (e.g. for debug purposes). The module is not required to be in debug mode since the MMIOMT itself never holds any secrets.

The function checks the following conditions:

1. MMIOMT_IDX is valid:
   1.1. Reserved fields must be 0.
   1.2. LEVEL must be valid.
   1.3. PA must be a valid 4KB aligned HPA with HKID bits equal to 0.

If successful, the function does the following:

1. Walk MMIOMT tree using MMOMT_IDX to get the new MMIOMT page parent entry.

If passed:

2. Return MMIOMT entry 32-byte content in RCX, RDX, R8 and R9.

Completion Status Codes

Table 3.55: TDH.MMIO.MT.RD Completion Status Codes (Returned in RAX) Definition

Completion Status Code	Description
TDX_OPERAND_ADDR_RANGE_ERROR	
TDX_OPERAND_BUSY	
TDX_OPERAND_INVALID	

Completion Status Code	Description
TDX_OPERAND_PAGE_METADATA_INCORRECT	
TDX_SUCCESS	

3.2.18. TDH.MMIO.MT.REMOVE Leaf

Remove an empty MMIOMT page.

Table 3.56: TDH.MMIO.MT.REMOVE Input Operands Definition

Operand	Description		
RAX	SEAMCALL instruction leaf number and version		
	Bits	**Field**	**Description**
	15:0	Leaf Number	Selects the SEAMCALL interface function
	23:16	Version Number	Selects the SEAMCALL interface function version
	63:24	Reserved	Must be 0
RCX	MMIOMT_IDX - MMIOMT index (MMIOMT_IDX_T)		

Table 3.57: TDH.MMIO.MT.REMOVE Output Operands Definition

Operand	Description
RAX	SEAMCALL instruction return code
RCX	Removed MMIOMT page HPA
Other	Unmodified

Leaf Function Description

Note: The description below is provided at a high level. Actual details, order of checks, returned status codes, etc. may vary.

TDH.MMIO.MT.REMOVE reclaims a page from the MMIOMT tree. The VMM provides the MMIOMT index (MMIO HPA address and level) of the MMIOMT page parent entry.

Before removing the MMIOY page, the function verifies that the page is indeed empty (none of the entries is present or is of MMIOMT_DATA type).

The function checks the following conditions:

1. MMIOMT_IDX is valid:
   1.1. Reserved fields must be 0.
   1.2. LEVEL must be valid be valid.
   1.3. PA must be a valid 4KB aligned HPA with HKID bits equal to 0.

If successful, the function does the following:

2. Walk MMIOMT tree using MMOMT_IDX to get the target MMIOMT page parent entry.

If passed:

3. Check that parent MMIOMT entry is valid and MMIOMT page is empty:
   3.1. Parent entry type must be QNODE and with present bit set.
   3.2. Map MMIOMT target page (R permissions using TDX reserved HKID).
   3.3. Verify all entries in MMIOMT are free QNODE entries.

If passed:

4. Unmap the MMIOMT parent entry by clearing its present bit.
5. Update PAMT type of the removed MMIOMT to PT_NDA.

Completion Status Codes

Table 3.58: TDH.MMIO.MT.REMOVE Completion Status Codes (Returned in RAX) Definition

Completion Status Code	Description
TDX_OPERAND_ADDR_RANGE_ERROR	
TDX_OPERAND_BUSY	
TDX_OPERAND_INVALID	
TDX_OPERAND_PAGE_METADATA_INCORRECT	
TDX_SUCCESS	

3.2.19. TDH.MMIO.MT.SET Leaf

Set or release MMIOMT leaf entry ownership.

Table 3.59: TDH.MMIO.MT.SET Input Operands Definition

Operand	Description		
RAX	SEAMCALL instruction leaf number and version		
	Bits	**Field**	**Description**
	15:0	Leaf Number	Selects the SEAMCALL interface function
	23:16	Version Number	Selects the SEAMCALL interface function version
	63:24	Reserved	Must be 0
RCX	MMIOMT_IDX - MMIOMT index (MMIOMT_IDX_T)		
RDX	MMIOMT_INFO – MMIO metadata information		
	Bits	**Field**	**Description**
	0:0	IS_DATA	0 – Release ownership (QNODE), 1 – Set ownership (DATA)
	11:1	RSVD	Reserved – must be 0
	43:12	FUNCTION_ID	FUNCTION_ID of the owner device Only valid when Type is DATA (1)
	63:44	RSVD	Reserved - must be 0

Table 3.60: TDH.MMIO.MT.SET Output Operands Definition

Operand	Description
RAX	SEAMCALL instruction return code
Other	Unmodified

Leaf Function Description

Note: The description below is provided at a high level. Actual details, order of checks, returned status codes, etc. may vary.

TDH.MMIO.MT.SET sets the TDI ownership of the MMIOMT entry or releases the ownership (in case the MMIO page is not mapped) and makes it free again. In the MMIOMT terminology, the free entry state is called QNODE and the owned state is called DATA.

The function checks the following conditions:

1. MMIOMT_IDX is valid:
   1.1. Reserved fields must be 0.
   1.2. LEVEL must be valid.
   1.3. PA must be a valid 4KB aligned HPA with HKID bits equal to 0.
2. MMIOMT_INFO is valid:
   2.1. Reserved fields must be 0.
   2.2. TYPE must be DATA (1) or QNODE (0).
   2.3. If TYPE is Release Ownership (QNODE), OWNER and FUNCTION_ID must be 0.
   2.4. If TYPE is Set Ownership (DATA), FUNCTION_ID fields must be valid.

If successful, the function sets/releases the MMIOMT ownership.

For Set Ownership (DATA), the entry must be free.

For Release Ownership (QNODE), the entry must be present and the MMIO page not mapped.

5

Completion Status Codes

Table 3.61: TDH.MMIO.MT.SET Completion Status Codes (Returned in RAX) Definition

Completion Status Code	Description
TDX_OPERAND_ADDR_RANGE_ERROR	
TDX_OPERAND_BUSY	
TDX_OPERAND_INVALID	
TDX_OPERAND_PAGE_METADATA_INCORRECT	
TDX_TDIMT_NOT_PRESENT	
TDX_SUCCESS:	

Intel TDX Connect Application Binary Interface (ABI) Reference

3.2.20. TDH.MMIO.UNMAP Leaf

Unmap MMIO page from a TD.

Table 3.62: TDH.MMIO.UNMAP Input Operands Definition

Operand	Description		
RAX	SEAMCALL instruction leaf number and version		
	Bits	**Field**	**Description**
	15:0	Leaf Number	Selects the SEAMCALL interface function
	23:16	Version Number	Selects the SEAMCALL interface function version
	63:24	Reserved	Must be 0
RCX	EPT mapping information:		
	Bits	**Name**	**Description**
	2:0	Level	Level of the EPT entry that will map the new page: must be 0 (4KB), 1 (2MB) or 2 (1GB)
	11:3	Reserved	Reserved: must be 0
	51:12	GPA	Bits 51:12 of the guest physical address to be mapped for the new Secure EPT page
	63:52	Reserved	Reserved: must be 0
RDX	Host physical address of the parent TDR page (HKID bits must be 0)		

Table 3.63: TDH.MMIO.UNMAP Output Operands Definition

Operand	Description
RAX	SEAMCALL instruction return code
RCX	Extended error information part 1 In case of EPT walk error, Secure EPT entry architectural content where the error was detected – see [ABI Spec] The architectural content represents how the Secure EPT maps a private memory page or a Secure EPT page, and may be different than the actual contents of the Secure EPT entry. Software should consult the Secure EPT information returned in RDX. In other cases, RCX returns 0.
RDX	Extended error information part 2 In case of EPT walk error, Secure EPT entry level and state where the error was detected – see [ABI Spec] In other cases, RDX returns 0.
R8	Unmapped MMIO HPA
Other	Unmodified

Leaf Function Description

Note: The description below is provided at a high level. Actual details, order of checks, returned status codes, etc. may vary.

TDH.MMIO.UNMAP is used by the VMM to un-map MMIO private GPA from a TD. The MMIO page mapping can only be removed when the device interface TDISP state is UNLOCKED or the IDE stream is blocked.

The function checks the following conditions:

1. The TDR is valid, the TD keys are configured, and the TD has been initialized:
   1.1. The TDR page metadata in PAMT must be correct (PT must be PT_TDR).
   1.2. The TD is not in a FATAL state (TDR.FATAL is FALSE).
   1.3. The TD keys are configured on the hardware (TDR.LIFECYCLE_STATE is TD_KEYS_CONFIGURED).
   1.4. The TD has been initialized locally by TDH.MNG.INIT and no migration session is in progress
2. The corresponding Secure EPT leaf entry for 4KB, 2MB or 1GB page is marked as MMIO_BLOCKED and both IOTLB and TLB tracking are done.
3. The owner TDI's TDISP state is CONFIG_UNLOCKED, or the corresponding IDE stream is blocked.

If successful, the function updates the Secure EPT state as FREE.

Completion Status Codes

Table 3.64: TDH.MMIO.UNMAP Completion Status Codes (Returned in RAX) Definition

Completion Status Code	Description
TDX_EPT_ENTRY_NOT_FREE	
TDX_EPT_WALK_FAILED	
TDX_OPERAND_ADDR_RANGE_ERROR	
TDX_OPERAND_BUSY	
TDX_OPERAND_INVALID	
TDX_OPERAND_PAGE_METADATA_INCORRECT	
TDX_SUCCESS	
TDX_SYS_NOT_READY	
TDX_SYS_SHUTDOWN	
TDX_TD_FATAL	
TDX_TD_KEYS_NOT_CONFIGURED	
TDX_TD_NOT_FINALIZED	
TDX_TD_NOT_INITIALIZED	

3.2.21. TDH.SPDM.CONNECT Leaf

Establish a new SPDM session with the device.

Table 3.65: TDH.SPDM.CONNECT Input Operands Definition

Operand	Name	Description		
RAX	Leaf and Number	SEAMCALL instruction leaf number and version		
		Bits	**Field**	**Description**
		15:0	Leaf Number	Selects the SEAMCALL interface function
		23:16	Version Number	Selects the SEAMCALL interface function version
		63:24	Reserved	Must be 0
RCX	SPDM_ID	SPDM Session Id, see SPDM_ID_T		
RDX	SPDM_CONFIG_PA	HPA of a page containing the SPDM configuration info, see SPDM_CONFIG_INFO_T		
R8	SPDM_RSP_HPA_ARR	Page buffer of type HPA_ARRAY_T, with SPDM_DOE_PAGES_COUNT free page(s) for the SPDM Response message (Input)		
R9	SPDM_REQ_HPA_ARR	Page buffer of type HPA_ARRAY_T, with SPDM_DOE_PAGES_COUNT free page(s) for the SPDM Request message (Output)		
R10	SPDM_OUT_HPA_ARR	Page buffer of type HPA_ARRAY_T, with SPDM_MAX_DEV_INFO_PAGES shared 4KB page HPAs containing the device's configuration info, see SPDM_DEVICE_ATTESTATION_INFO_T		

Table 3.66: TDH.SPDM.CONNECT Output Operands Definition

Operand	Description
RAX	SEAMCALL instruction return code
RCX	If RAX returns TDX_SPDM_REQUEST – Generated SPDM output message length in bytes. If RAX returns TDX_SUCCESS – SPDM_DEVICE_ATTESTATION_INFO_T length in bytes. If RAX returns TDX_SPDM_MESSAGE_ERROR / TDX_TDISP_MESSAGE_ERROR, then the length of the SPDM extended error information, returned in SPDM_REQ_HPA_ARR buffer. For other RAX values – 0.
RDX	0. Reserved for future use.
Other	Unmodified

Leaf Function Description

Note: The description below is provided at a high level. Actual details, order of checks, returned status codes, etc. may vary.

TDH.SPDM.CONNECT establishes a new SPDM session with the device.

The function checks the following conditions:

1. SPDM_ID must be valid session id, returned by TDH.SPDM.CREATE.
2. Input parameters SPDM_RSP_HPA_ARR, SPDM_REQ_HPA_ARR, SPDM_OUT_HPA_ARR are valid.
3. SPDM session state is DISCONNECTED.

If successful, the function prepares the SPDM session connect request and returns TDX_SPDM_REQUEST. The VMM will send the request to the device and recall the function, see [the VMM Common SPDM Interface]. When the function returns TDX_SUCCESS, the SPDM session has been established.

Completion Status Codes

Table 3.67: TDH.SPDM.CONNECT Completion Status Codes (Returned in RAX) Definition

Completion Status Code	Description
TDX_IOMMU_INVALID_STATE	IOMMU is not in IOMMU_CONFIGURED state
TDX_OPERAND_BUSY	Operation encountered a busy operand, indicated by the lower 32 bits of the status. In many cases, this can be resolved by retrying the operation.
TDX_OPERAND_INVALID	
TDX_SPDM_ENTRY_NOT_PRESENT	
TDX_SPDM_INVALID_MESSAGE	Invalid SPDM message or decryption/integrity check failure
TDX_SPDM_REQUEST	TDX Module requires VMM to send the request in OUT_PA with length
TDX_SPDM_INVALID_STATE	The associated SPDM session is in CONNECTED state
TDX_SUCCESS	Operation is successful
TDX_INTERRUPTED_RESUMABLE	Interrupt is pending, the operation can be resumed
TDX_SPDM_SESSION_ERROR	The SPDM operation failed
TDX_SPDM_MESSAGE_ERROR	The SPDM session establishment failed

3.2.22. TDH.SPDM.CREATE Leaf

Create a new SPDM session metadata.

Table 3.68: TDH.SPDM.CREATE Input Operands Definition

Operand	Name	Description		
RAX	Leaf and Number	SEAMCALL instruction leaf number and version		
		Bits	**Field**	**Description**
		15:0	Leaf Number	Selects the SEAMCALL interface function
		23:16	Version Number	Selects the SEAMCALL interface function version
		63:24	Reserved	Must be 0
RCX	FUNCTION_ID	**Bits**	**Field**	**Description**
		31:0	FUNCTION_ID	TDISP FUNCTION_ID of the device PF (the segment and bus define the IO stack)
		63:32	Reserved	Must be 0
RDX	SPDM_MT_HPA_ARR	Page buffer of type HPA_ARRAY_T, with SPDM_MT_PAGES_COUNT free page(s) for hosting the TDX Module SPDM Metadata and internal secure buffer(s). See: TDX Connect System Enumerations.		

Table 3.69: TDH.SPDM.CREATE Output Operands Definition

Operand	Description
RAX	SEAMCALL instruction return code
RCX	SPDM_ID handle
RDX	Bits 7:0 TDI bind policy see TDX_DEVICE_POLICY_T Bits 63:8 Reserved, 0.
Other	Unmodified

Leaf Function Description

Note: The description below is provided at a high level. Actual details, order of checks, returned status codes, etc. may vary.

TDH.SPDM.CREATE allocates a metadata page to hold the device SPDM session information.

The function checks the following conditions:

1. FUNCTION_ID is a valid FUNCTION_ID of a device on a TDX Connect-configured IOMMU.
2. SPDM_MT_HPA_ARR is valid, and SPDM_MT_PAGES_COUNT shared pages are available.
3. The number of created SPDM sessions is less than MAX_SPDM_SESSIONS.

If successful, the function creates and initializes the SPDM context.

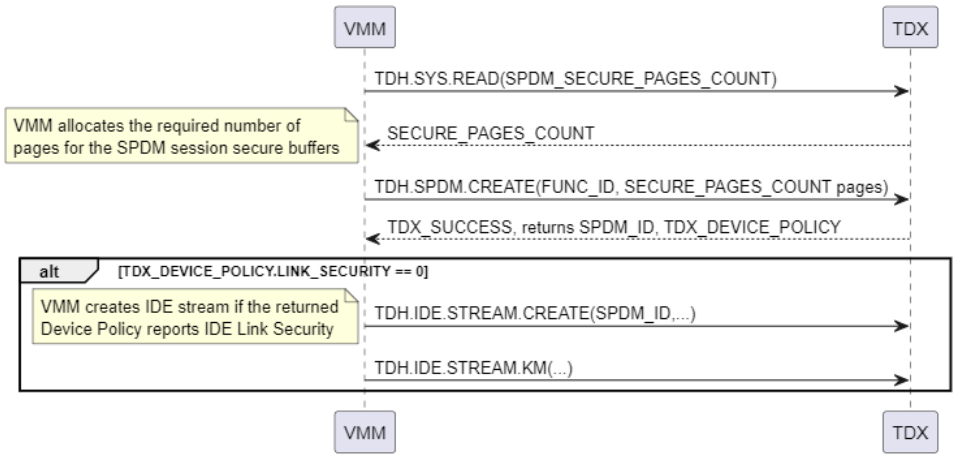
The output TDX_DEVICE_POLICY_T tells the VMM the setup required for TDI assignment.

Table 3.70: TDX_DEVICE_POLICY_T

Bit	Name	Description
0:0	LINK_SECURITY	Link Security Enumeration: **0** – IDE: VMM must create and setup a Selective IDE stream for the SPDM session before initiating a TDI assignment (TDH.TDI.BIND). **1** – IDE Not required. VMM shall bypass IDE stream creation and proceed to the TDI assignment (see TDH.TDI.BIND) over the connected SPDM session.
7:1	Reserved	**0**

Below is a sample VMM flow for SPDM session creation and checking the TDX_DEVICE_POLICY_T if the IDE Stream needs to be created:

Table 3.71: Sample TDH.SPDM.CREATE and Device Policy checking flow



Completion Status Codes

Table 3.72: TDH.SPDM.CREATE Completion Status Codes (Returned in RAX) Definition

Completion Status Code	Description
TDX_IOMMU_INVALID_STATE	IOMMU is not in IOMMU_CONFIGURED state
TDX_OPERAND_BUSY	Operation encountered a busy operand, indicated by the lower 32 bits of the status. In many cases, this can be resolved by retrying the operation.
TDX_NO_FREE_SPDM_SESSIONS	Max number of SPDM sessions reached, no free SPDMDIR entry for a new session
TDX_OPERAND_INVALID	
TDX_SUCCESS	Operation is successful

3.2.23. TDH.SPDM.DELETE Leaf

Delete SPDM session metadata structure.

Table 3.73: TDH.SPDM.DELETE Input Operands Definition

Operand	Name	Description		
RAX	Leaf and Number	SEAMCALL instruction leaf number and version		
		Bits	**Field**	**Description**
		15:0	Leaf Number	Selects the SEAMCALL interface function
		23:16	Version Number	Selects the SEAMCALL interface function version
		63:24	Reserved	Must be 0
RCX	SPDM_ID	SPDM Session Id, see SPDM_ID_T		
RDX	AUX_PAGE_PA	HPA of a shared 4K auxiliary page: • If **SPDM_MT_PAGES_COUNT == 1:** TDX Module will not use the auxiliary page, although will check its validity for forward compatibility. The HPA of the released IDE Metadata page will be in the output register. • If **SPDM_MT_PAGES_COUNT > 1:** TDX Module uses the auxiliary page for HPA_ARRAY_T object with the released SPDM Metadata pages (acquired in TDH.SPDM.CREATE). The object is returned in the output register, see: HPA_ARRAY_T.		

Table 3.74: TDH.SPDM.DELETE Output Operands Definition

Operand	Description
RAX	SEAMCALL instruction return code
RCX	HPA_ARRAY_T type containing the released SPDM Metadata pages: • If SPDM_MT_PAGES_COUNT == 1: the value is the HPA of the released page. • If SPDM_MT_PAGES_COUNT > 1: the value is *HPA_ARRAY_T(HPA == AUX_PAGE_PA, LAST_IDX == SPDM_MT_PAGES_COUNT)*. The VMM needs to map the AUX_PAGE_PA page to read the HPAs of the released pages.
Other	Unmodified

Leaf Function Description

Note: The description below is provided at a high level. Actual details, order of checks, returned status codes, etc. may vary.

TDH.SPDM.DELETE deletes an SPDM session metadata page after checking there are no IDE Streams attached to it.

The function checks the following conditions:

1. SPDM_ID must be valid session id, returned by TDH.SPDM.CREATE.
2. The active IDE streams count for the session must be 0.

Commented [AA66]: Bin: Simplify and change to HPA_ARRAY_T

Commented [RI67R66]: Isn't that an older comment?, I explained it already in the email thread with examples. If Metadata is 1 page, there is no need in AUX page (map it, fill with the HPA, etc.).

If all checks passed, the function releases the SPDM session metadata page(s). Note, the VMM can delete the SPDM session without a graceful session disconnect by TDH.SPDM.DISCONNECT.

Completion Status Codes

Table 3.75: TDH.SPDM.DELETE Completion Status Codes (Returned in RAX) Definition

Completion Status Code	Description
TDX_IOMMU_INVALID_STATE	IOMMU is not in IOMMU_CONFIGURED state
TDX_OPERAND_BUSY	Operation encountered a busy operand, indicated by the lower 32 bits of the status. In many cases, this can be resolved by retrying the operation.
TDX_OPERAND_INVALID	
TDX_SUCCESS	Operation is successful

3.2.24. TDH.SPDM.DISCONNECT Leaf

Disconnect (end) the SPDM session on the device.

Table 3.76: TDH.SPDM.DISCONNECT Input Operands Definition

Operand	Name	Description		
RAX	Leaf and Number	SEAMCALL instruction leaf number and version		
		Bits	**Field**	**Description**
		15:0	Leaf Number	Selects the SEAMCALL interface function
		23:16	Version Number	Selects the SEAMCALL interface function version
		63:24	Reserved	Must be 0
RCX	SPDM_ID	SPDM Session Id, see SPDM_ID_T		
RDX	SPDM_RSP_HPA_ARR	Page buffer of type HPA_ARRAY_T, with SPDM_DOE_PAGES_COUNT free page(s) for the SPDM Response message (Input)		
R8	SPDM_REQ_HPA_ARR	Page buffer of type HPA_ARRAY_T, with SPDM_DOE_PAGES_COUNT free page(s) for the SPDM Request message (Output)		

Table 3.77: TDH.SPDM.DISCONNECT Output Operands Definition

Operand	Description
RAX	SEAMCALL instruction return code
RCX	If RAX returns TDX_SPDM_REQUEST – Generated SPDM output message length in bytes. If RAX returns TDX_SPDM_MESSAGE_ERROR / TDX_TDISP_MESSAGE_ERROR, then the length of the SPDM extended error information, returned in SPDM_REQ_HPA_ARR buffer. The SPDM extended error information is defined in [SPDM] For other return codes – 0.
Other	Unmodified

Leaf Function Description

Note: The description below is provided at a high level. Actual details, order of checks, returned status codes, etc. may vary.

The function gracefully ends the SPDM session on the device.

The function checks the following conditions:

1. SPDM_ID must be valid session id, returned by TDH.SPDM.CREATE.
2. SPDM state is Connected.
3. The active IDE streams count for the session must be 0.
4. No other SPDM-related operations are in progress (see: VMM Common SPDM Interface).

If successful, the function prepares the SPDM END_SESSION request and returns TDX_SPDM_REQUEST. The VMM will send the request to the device and recall the function, see [the VMM Common SPDM Interface]. When the function returns TDX_SUCCESS, the SPDM session has been disconnected.

Completion Status Codes

Table 3.78: TDH.SPDM.DISCONNECT Completion Status Codes (Returned in RAX) Definition

Completion Status Code	Description
TDX_IOMMU_INVALID_STATE	IOMMU is not in IOMMU_CONFIGURED state
TDX_OPERAND_BUSY	Operation encountered a busy operand, indicated by the lower 32 bits of the status. In many cases, this can be resolved by retrying the operation.
TDX_OPERAND_INVALID	
TDX_SPDM_ENTRY_NOT_PRESENT	
TDX_SPDM_INVALID_MESSAGE	Invalid SPDM message or decryption/integrity check failure
TDX_SPDM_REQUEST	TDX Module requires VMM to send the request in OUT_PA with length
TDX_SPDM_INVALID_STATE	The associated SPDM session is not in CONNECTED state
TDX_SUCCESS	Operation is successful
TDX_INTERRUPTED_RESUMABLE	Interrupt is pending, the operation can be resumed
TDX_SPDM_SESSION_ERROR	The SPDM operation failed
TDX_SPDM_MESSAGE_ERROR	The SPDM session establishment failed

Intel TDX Connect Application Binary Interface (ABI) Reference

3.2.25. TDH.SPDM.MNG Leaf

Manages the SPDM session.

Table 3.79: TDH.SPDM.MNG Input Operands Definition

Operand	Name	Description		
RAX	Leaf and Number	SEAMCALL instruction leaf number and version		
		Bits	**Field**	**Description**
		15:0	Leaf Number	Selects the SEAMCALL interface function
		23:16	Version Number	Selects the SEAMCALL interface function version
		63:24	Reserved	Must be 0
RCX	SPDM_ID	SPDM Session Id, see SPDM_ID_T		
RDX	SPDM_OPERATION	SPDM operation type: HEARTBEAT – 0 KEY_UPDATE – 1 DEV_INFO_RECOLLECTION – 2 ABORT - 3		
R8	MNG_PARAM_PA	In case of DEV_INFO_RECOLLECTION, HPA of a page with 32B nonce, used in case of DEV_INFO_RECOLLECTION, see MNG_PARAM_T. Otherwise, NULL PA.		
R9	SPDM_RSP_HPA_ARR	Page buffer of type HPA_ARRAY_T, with SPDM_DOE_PAGES_COUNT free page(s) for the SPDM Response message (Input)		
R10	SPDM_REQ_HPA_ARR	Page buffer of type HPA_ARRAY_T, with SPDM_DOE_PAGES_COUNT free page(s) for the SPDM Request message (Output)		
R11	SPDM_OUT_HPA_ARR	In case of DEV_INFO_RECOLLECTION, page buffer of type HPA_ARRAY_T, with SPDM_MAX_DEV_INFO_PAGES shared 4KB page HPAs containing the device's configuration info, see SPDM_DEVICE_ATTESTATION_INFO_T. Otherwise, NULL PA.		

Table 3.80: TDH.SPDM.MNG Output Operands Definition

Operand	Description
RAX	SEAMCALL instruction return code
RCX	If RAX returns TDX_SPDM_REQUEST – Generated SPDM output message length in bytes. If RAX returns TDX_SUCCESS – DEVICE_SPDM_INFO length in bytes. If RAX returns TDX_SPDM_MESSAGE_ERROR / TDX_TDISP_MESSAGE_ERROR, then the length of the SPDM extended error information, returned in SPDM_REQ_HPA_ARR buffer. The SPDM extended error information is defined in [TPA Spec]. Zero means no SPDM extended error information returned. Zero otherwise.
Other	Unmodified

Leaf Function Description

Note: The description below is provided at a high level. Actual details, order of checks, returned status codes, etc. may vary.

TDH.SPDM.MNG performs the management operations on the given SPDM session in the Connected state.

The operations supported:

HEARTBEAT – Issue SPDM heartbeat.

KEY_UPDATE – Issue SPDM Key Update.

DEV_INFO_RECOLLECTION – Issue the Device Attestation Info recollection.

ABORT – Abort the SPDM Session.

To understand the table and text below, please refer to the [TDX Module Spec] chapter discussing general aspects of the Intel TDX Module API.

Table 3.81: TDH.SPDM.MNG Operands Information Definition

Explicit/ Implicit	Reg.	Ref. Type	Resource	Resource Type	Access	Access Semantics	Align. Check	Concurrency Restrictions		
								Operand	Contain. 2MB	Contain. 1GB
Explicit	RCX	Index	SPDM Info Entry	IOMMU metadata	RW	Hidden	N/A	Exclusive	N/A	N/A
Explicit	R8	HPA	VMM page	N/A	RW	Opaque	4KB	N/A	N/A	N/A
Explicit	R9	HPA	VMM page	N/A	RW	Opaque	4KB	N/A	N/A	N/A
Explicit	R10	HPA	VMM page	N/A	R	Opaque	4KB	N/A	N/A	N/A
Explicit	R11	HPA_ARRAY	VMM page	N/A	RW	Opaque	4KB	N/A	N/A	N/A

TDH.SPDM.MNG checks the memory operands per the table above when applicable during its flow. The text below does not explicitly mention those checks, except when necessary.

The function checks the following conditions:

1. SPDM_ID must be valid session id, returned by TDH.SPDM.CREATE.
2. SPDM state is Connected.
3. No other SPDM-related operations are in progress (see: VMM Common SPDM Interface).

If successful, the function does the following:

For HEARTBEAT, KEY_UPDATE and DEV_INFO_RECOLLECTION: prepares the SPDM management request according to the operation requested and returns TDX_SPDM_REQUEST. The VMM will send the request to the device and recall the function, see [the VMM Common SPDM Interface]. When the function returns TDX_SUCCESS, the management operation has been completed.

For ABORT: SPDM Session is aborted. No SPDM operations will be allowed for the session. The VMM may only release the resources of the TDIs that are in not UNLOCKED state.

Completion Status Codes

Table 3.82: TDH.SPDM.MNG Completion Status Codes (Returned in RAX) Definition

Completion Status Code	Description
TDX_IOMMU_INVALID_STATE	IOMMU is not in IOMMU_CONFIGURED state
TDX_OPERAND_BUSY	Operation encountered a busy operand, indicated by the lower 32 bits of the status. In many cases, this can be resolved by retrying the operation.
TDX_OPERAND_INVALID	
TDX_SUCCESS	Operation is successful
TDX_SPDM_REQUEST	TDX Module requires VMM to send the request in OUT_PA with length
TDX_INTERRUPTED_RESUMABLE	Interrupt is pending, the operation can be resumed
TDX_SPDM_SESSION_ERROR	The SPDM operation failed
TDX_SPDM_MESSAGE_ERROR	The SPDM session establishment failed
TDX_SPDM_SESSION_KEY_REQUIRE_REFRESH	The SPDM session requires a key refresh

Intel TDX Connect Application Binary Interface (ABI) Reference

3.2.26. TDH.TDI.BIND Leaf (Change in Progress)

Start TDI binding 1st step: lock the Interface configuration and obtain the Interface Report.

Note: This ABI is ongoing changes to add VMM input for guest virtual function ID as well as add TDISP report checks to verify that TEE MMIO ranges are not assigned to any TDI and DMA entry for FUNCTION_ID (bdf) is not already mapped.

Table 3.83: TDH.TDI.BIND Input Operands Definition

Operand	Description				
RAX	SEAMCALL instruction leaf number and version				
	Bits	**Field**	**Description**		
	15:0	Leaf Number	Selects the SEAMCALL interface function		
	23:16	Version Number	Selects the SEAMCALL interface function version		
	63:24	Reserved	Must be 0		
RCX	FUNCTION_ID		**Bits**	**Field**	**Description**
			31:0	Function ID	TDI Function ID
			63:32	Reserved	Must be 0
RDX	TDISP_PARAM		TDISP-related parameters for TDI binding		
			Bits	**Field**	**Description**
			15:0	LOCK_FLAGS	Interface Lock flags, only bits reported by the global field **LOCK_INTERFACE_FLAGS_SUPPORTED** can be set to 1.
			63:16	Reserved	Must be 0
R8	**SPDM_RSP_HPA_ARR**– Page buffer of type HPA_ARRAY_T, with SPDM_DOE_PAGES_COUNT free page(s) for the SPDM Response message (Input)				
R9	**SPDM_REQ_HPA_ARR** - Page buffer of type HPA_ARRAY_T, with SPDM_DOE_PAGES_COUNT free page(s) for the SPDM Request message (Output)				
R10	**SPDM_OUT_HPA_ARR** - Page buffer of type HPA_ARRAY_T, with TDISP_MAX_TDI_REPORT_PAGES free page(s) for the TDI Interface report output.				

Table 3.84: TDH.TDI.BIND Output Operands Definition

Operand	Description
RAX	SEAMCALL instruction return code
RCX	If RAX returns **TDX_SUCCESS** – TDISP Interface Report length in SPDM_OUT_HPA_ARR in bytes. SPDM_REQ_HPA_ARR – Unused. If RAX returns **TDX_SPDM_REQUEST** – Generated in SPDM_REQ_HPA_ARR TDISP output message (DOE+Secure SPDM enveloped) length in bytes. If RAX returns **TDX_TDISP_ERROR** – Returns the TDISP ERROR_CODE (bytes 0-3), TDISP ERROR_DATA (bytes 0-3), see [TDISP] (bytes 4-8), see [TDISP]. For other return codes – unmodified.

RDX	If RAX returns **TDX_TDISP_ERROR** – Returns the TDISP request message type (see [TDISP] that caused the error (byte 0), bytes 1-7 are 0. For other return codes – unmodified.
Other	Unmodified

Leaf Function Description

Note: The description below is provided at a high level. Actual details, order of checks, returned status codes, etc. may vary.

TDH.TDI.BIND initiates TDI assignment (see LOCK_INTERFACE and GET_DEVICE_INTERFACE_REPORT in [TDISP]) for a previously unassigned (UNLOCKED) interface.

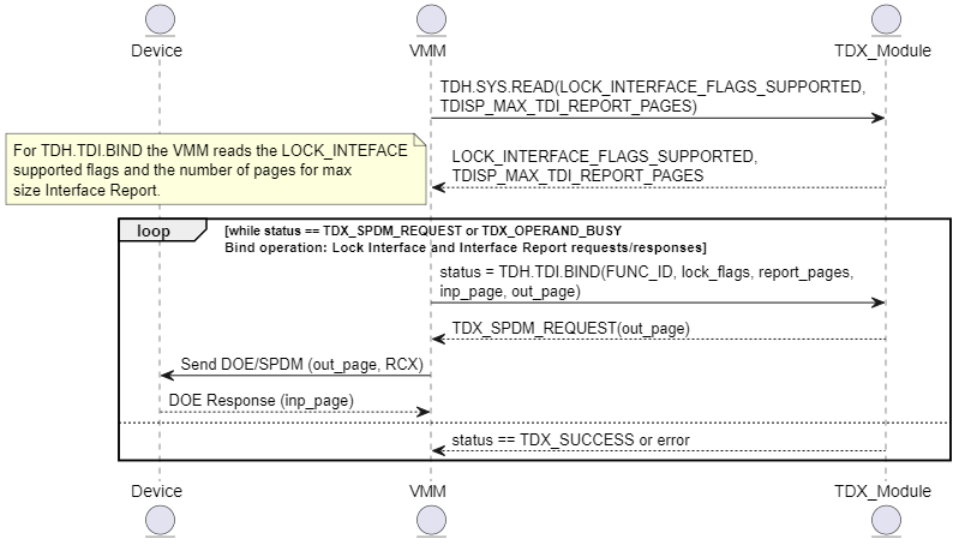
The function checks the following conditions:

1. FUNCTION_ID is valid.
2. TDISP_PARAM.LOCK_FLAGS are in the allowed global field LOCK_INTERFACE_FLAGS_SUPPORTED bitmask. Reserved field is 0.
3. No other SPDM operation is in progress, see VMM Common SPDM Interface.
4. TDI is in UNLOCKED state.
5. The secure session with physical device is connected and active: SPDM session is Connected, IDE Stream is configured and not blocked.

If successful, the function prepares a serious the binding requests, returning TDX_SPDM_REQUEST so the VMM will send the request to the device and recall the function, see [the VMM Common SPDM Interface] and the Figure below. When the function returns TDX_SUCCESS, the TDI is bound, the device Interface Report is written into SPDM_OUT_HPA_ARR pages and RCX has the Interface Report length in bytes.

Important note: if TDH.TDI.BIND returns a failure status, the TDI state is unknown (can be either LOCKED or UNLOCKED). In case a failure, the VMM shall sync with the actual interface by calling either TDH.TDI.GET.STATE or TDH.TDI.STOP.

Figure 3.85: TDH.TDI.BIND Sample flow sequence



Completion Status Codes

Table 3.86: TDH.TDI.BIND Completion Status Codes (Returned in RAX) Definition

Completion Status Code	Description
TDX_OPERAND_BUSY	
TDX_OPERAND_INVALID	
TDX_TDI_NOT_PRESENT	
TDX_TDI_INVALID_METADATA	Invalid TDIMT_IDX path, or the path for the entry index already exists
TDX_SPDM_ENTRY_NOT_PRESENT	
TDX_SPDM_SESSION_KEY_REQUIRE_REFRESH	The SPDM session requires a key refresh
TDX_SPDM_INVALID_MESSAGE	Invalid SPDM message or decryption/integrity check failure
TDX_SPDM_REQUEST	TDX Module requires VMM to send the request in OUT_PA with length
TDX_SPDM_INVALID_STATE	The associated SPDM session is not in CONNECTED state
TDX_IDE_STREAM_INVALID_STATE	The IDE stream is not configured or blocked
TDX_TDISP_INVALID_MESSAGE	Unexpected/invalid TDISP response
TDX_TDISP_ERROR	TDSP Error Response returned
TDX_TDI_INVALID_TDI_CONFIGURATION	Interface Report INTERFACE_INFO capability(ies) is not supported by TDX Module and doesn't allow to make the Interface secure
TDX_SUCCESS	

3.2.27. TDH.TDI.CREATE Leaf

Create a TDI interface control structure and metadata.

Table 3.87: TDH.TDI.CREATE Input Operands Definition

<table>
<tr><th>Operand</th><th>Name</th><th colspan="3">Description</th></tr>
<tr><td rowspan="4">RAX</td><td rowspan="4">Leaf and Version</td><td colspan="3">SEAMCALL instruction leaf number and version</td></tr>
<tr><td>Bits</td><td>Field</td><td>Description</td></tr>
<tr><td>15:0</td><td>Leaf Number</td><td>Selects the SEAMCALL interface function</td></tr>
<tr><td>23:16</td><td>Version Number</td><td>Selects the SEAMCALL interface function version</td></tr>
<tr><td></td><td></td><td>63:24</td><td>Reserved</td><td>Must be 0</td></tr>
<tr><td>RCX</td><td>STREAM_ID</td><td colspan="3">IDE-Stream identifier</td></tr>
<tr><td>RDX</td><td>TDI_INFO</td><td colspan="3">Device interface ID
<table><tr><th>Name</th><th>Bits</th><th>Description</th></tr><tr><td>TDISP_FUNC_ID</td><td>31:0</td><td>TDI FUNCTION_ID See: [TDISP]</td></tr><tr><td>Type</td><td>39:32</td><td>Device Interface Type (see: TDI_TYPE_T)</td></tr><tr><td>Reserved</td><td>63:40</td><td>Reserved. Must be zero.</td></tr></table></td></tr>
<tr><td>R8</td><td>TDR</td><td colspan="3">Host physical address of the parent TDR page (HKID bits must be 0)</td></tr>
<tr><td>R9</td><td>TDICS_MT_HPA_ARR</td><td colspan="3">Page buffer of type HPA_ARRAY_T, with TDICS_MT_PAGES_COUNT free page(s) for hosting a new TDI Metadata. See: TDX Connect System Enumerations.</td></tr>
</table>

Table 3.88: TDH.TDI.CREATE Output Operands Definition

Operand	Description
RAX	SEAMCALL instruction return code
Other	Unmodified

Leaf Function Description

Note: The description below is provided at a high level. Actual details, order of checks, returned status codes, etc. may vary.

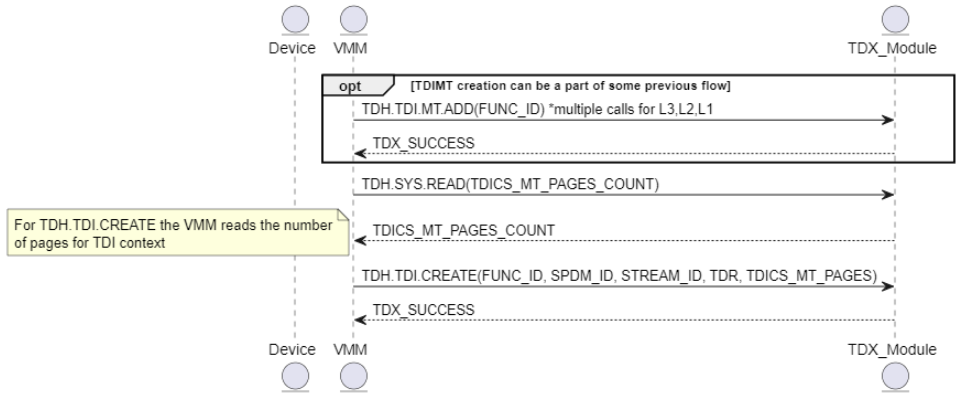
TDH.TDI.CREATE creates and initializes the TDI metadata. The TDI is identified by the FUNCTION_ID and TDI type). The TDI will be assigned to the TD (identified by TDR) with the STREAM_ID being their communication channel.

The function checks the following conditions:

1. TDI_INFO.type == TDI_TYPE::PFVF
2. TDI_INFO. FUNCTION_ID is valid.
3. The TDR is valid and the TD is in operational state.
4. The secure session with physical device is connected and active: SPDM session is Connected, IDE Stream is configured and not blocked.
5. The FUNCTION_ID's (Requestor ID) is inside the Stream's Extended Info IDE RID Association range.

If successful, the function initializes the TDI metadata and marks the TDI assigned to the TD.

Figure 3.89: TDH.TDI.CREATE Sample flow



Completion Status Codes

Table 3.90: TDH.TDI.CREATE Completion Status Codes (Returned in RAX) Definition

Completion Status Code	Description
TDX_OPERAND_INVALID	
TDX_OPERAND_BUSY	Operation encountered a busy operand, indicated by the lower 32 bits of the status. In many cases, this can be resolved by retrying the operation.
TDX_TD_FATAL	
TDX_TD_KEYS_NOT_CONFIGURED	
TDX_TDCS_NOT_ALLOCATED	
TDX_OP_STATE_INCORRECT	
TDX_IOMMU_INVALID_STATE	IOMMU is not in IOMMU_CONFIGURED state
TDX_SPDM_INVALID_STATE	SPDM session must be valid and connected
TDX_IDE_STREAM_INVALID_STATE	IDE-Stream metadata in TDX module is not configured or it is blocked
TDX_SUCCESS	Operation is successful

3.2.28. TDH.TDI.GET.STATE Leaf

Queries the TDI TDISP Interface State.

Table 3.91: TDH.TDI.GET.STATE Input Operands Definition

Operand	Description				
RAX	SEAMCALL instruction leaf number and version				
	Bits	**Field**	**Description**		
	15:0	Leaf Number	Selects the SEAMCALL interface function		
	23:16	Version Number	Selects the SEAMCALL interface function version		
	63:24	Reserved	Must be 0		
RCX	FUNCTION_ID	**Bits**	**Field**	**Description**	
		31:0	Function ID	TDI Function ID	
		63:32	Reserved	Must be 0	
RDX	**SPDM_RSP_PA**– HPA of a page with SPDM response message input.				
R8	**SPDM_REQ_HPA_ARR** - HPA of a page for the TDISP request message output.				

Table 3.92: TDH.TDI.GET.STATE Output Operands Definition

Operand	Description
RAX	SEAMCALL instruction return code
RCX	If RAX returns **TDX_SUCCESS** – TDISP Interface State (see [TDISP]). If RAX returns **TDX_SPDM_REQUEST** – Generated TDISP output message (DOE+Secure SPDM enveloped) length in bytes. If RAX returns **TDX_TDISP_ERROR** – Returns the TDISP ERROR_CODE (bytes 0-3) and ERROR_DATA (bytes 4-7), see [TDISP]. For other return codes – 0.
Other	Unmodified

Leaf Function Description

Note: The description below is provided at a high level. Actual details, order of checks, returned status codes, etc. may vary.

TDH.TDI.GET.STATE issues an Interface State request (see GET_INTERFACE_STATE in [TDISP]) and processes the response to get the TDISP state of TDI.

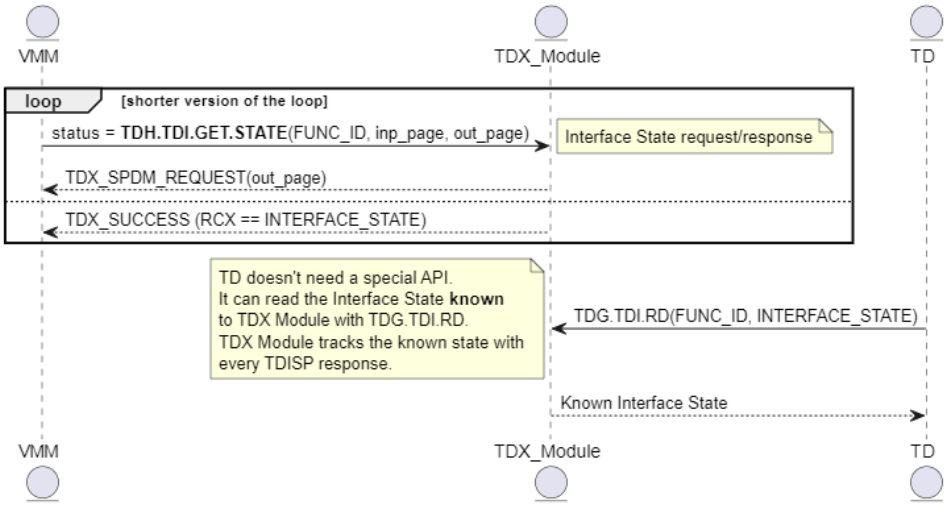
The function checks the following conditions:

1. FUNCTION_ID is valid.
2. No other SPDM operation is in progress, see VMM Common SPDM Interface.
3. The secure session with physical device is connected and active: SPDM session is Connected, IDE Stream is configured and not blocked.

If successful, the function prepares the GET_STATE request and returns TDX_SPDM_REQUEST. The VMM will send the request to the device and recall the function, see [the VMM Common SPDM Interface]. When the function returns TDX_SUCCESS, the TDI Interface state is returned in RCX.

The following sequence diagram demonstrates the flow with TDH.TDI.GET.STATE.

Figure 3.93: TDH.TDI.GET.STATE in a Sample GET_INTERFACE_STATE flow



Completion Status Codes

Table 3.94: TDH.TDI.GET.STATE Completion Status Codes (Returned in RAX) Definition

Completion Status Code	Description
TDX_OPERAND_BUSY	
TDX_OPERAND_INVALID	

Completion Status Code	Description
TDX_TDI_NOT_PRESENT	
TDX_TDI_INVALID_METADATA	Invalid TDIMT_IDX path, or the path for the entry index already exists
TDX_SPDM_ENTRY_NOT_PRESENT	
TDX_SPDM_SESSION_KEY_REQUIRE_REFRESH	The SPDM session requires a key refresh
TDX_SPDM_INVALID_MESSAGE	Invalid SPDM message or decryption/integrity check failure
TDX_SPDM_REQUEST	TDX Module requires VMM to send the request in OUT_PA with length
TDX_SPDM_INVALID_STATE	The associated SPDM session is not in CONNECTED state
TDX_IDE_STREAM_INVALID_STATE	The IDE stream is not configured or blocked
TDX_TDISP_ERROR	TDSP Error Response returned
TDX_TDISP_INVALID_MESSAGE	Unexpected/invalid TDISP response
TDX_SUCCESS	

3.2.29. TDH.TDI.MT.ADD Leaf

Add a page for TDI Metadata.

Table 3.95: TDH.TDI.MT.ADD Input Operands Definition

Operand	Description		
RAX	SEAMCALL instruction leaf number and version		
	Bits	**Field**	**Description**
	15:0	Leaf Number	Selects the SEAMCALL interface function
	23:16	Version Number	Selects the SEAMCALL interface function version
	63:24	Reserved	Must be 0
RCX	TDIMT_IDX – Parent entry index (See: TDIMT_IDX_T)		
RDX	TDIMT_PA - The physical address of the new allocated TDIMT page		

Table 3.96: TDH.TDI.MT.ADD Output Operands Definition

Operand	Description
RAX	SEAMCALL instruction return code
Other	Unmodified

Leaf Function Description

Note: The description below is provided at a high level. Actual details, order of checks, returned status codes, etc. may vary.

TDH.TDI.MT.ADD used by the VMM to build the TDIMT tree. The VMM provides the HPA of a free page to be used as the TDIMT page and the TDIMT parent entry index.

The function checks the following conditions:

1. TDIMT_IDX is valid:
   1.1. Reserved fields must be 0.
   1.2. LEVEL must be valid.
2. TDIMT_PA must be a free 4KB aligned HPA with HKID bits equal to 0.
3. Parent entry is not already mapped.

If successful, the function adds the new page to TDIMT page structure.

Completion Status Codes

Table 3.97: TDH.TDI.MT.ADD Completion Status Codes (Returned in RAX) Definition

Completion Status Code	Description
TDX_OPERAND_ADDR_RANGE_ERROR	
TDX_OPERAND_BUSY	
TDX_OPERAND_INVALID	
TDX_OPERAND_PAGE_METADATA_INCORRECT	
TDX_TDI_INVALID_METADATA	Invalid TDIMT_IDX path, or the path for the entry index already exists
TDX_SUCCESS	

3.2.30. TDH.TDI.MT.REMOVE Leaf

Remove empty TDI Metadata page.

Table 3.98: TDH.TDI.MT.REMOVE Input Operands Definition

Operand	Description		
RAX	SEAMCALL instruction leaf number and version		
	Bits	**Field**	**Description**
	15:0	Leaf Number	Selects the SEAMCALL interface function
	23:16	Version Number	Selects the SEAMCALL interface function version
	63:24	Reserved	Must be 0
RCX	TDIMT_IDX - Parent entry index (See: TDIMT_IDX_T)		

Table 3.99: TDH.TDI.MT.REMOVE Output Operands Definition

Operand	Description
RAX	SEAMCALL instruction return code
RCX	Removed TDIMT page HPA
Other	Unmodified

Leaf Function Description

Note: The description below is provided at a high level. Actual details, order of checks, returned status codes, etc. may vary.

TDH.TDI.MT.REMOVE reclaims an unused (containing no entries) page from the TDIMT tree.

The function checks the following conditions:

1. TDIMT_IDX is valid level and reserved fields are 0.
2. Parent entry is valid and TDIMT page is empty.

If successful, the function removes the page from the TDIMT structure and returns its HPA to the VMM.

Completion Status Codes

Table 3.100: TDH.TDI.MT.REMOVE Completion Status Codes (Returned in RAX) Definition

Completion Status Code	Description
TDX_OPERAND_ADDR_RANGE_ERROR	
TDX_OPERAND_BUSY	
TDX_OPERAND_INVALID	
TDX_OPERAND_PAGE_METADATA_INCORRECT	
TDX_TDI_INVALID_METADATA	Invalid TDIMT_IDX path, or the path to the index doesn't exist
TDX_SUCCESS	

3.2.31. TDH.TDI.MT.RD Leaf

Read TDI Metadata entry.

Table 3.101: TDH.TDI.MT.RD Input Operands Definition

Operand	Description		
RAX	SEAMCALL instruction leaf number and version		
	Bits	**Field**	**Description**
	15:0	Leaf Number	Selects the SEAMCALL interface function
	23:16	Version Number	Selects the SEAMCALL interface function version
	63:24	Reserved	Must be 0
RCX	TDIMT_IDX – Entry index (See: TDIMT_IDX_T)		

Table 3.102: TDH.TDI.MT.RD Output Operands Definition

Operand	Description
RAX	SEAMCALL instruction return code
RCX	TDIMT entry data
Other	Unmodified

Leaf Function Description

Note: The description below is provided at a high level. Actual details, order of checks, returned status codes, etc. may vary.

TDH.TDI.MT.RD used by the VMM to read the TDIMT entry (e.g. for debugging purposes).

The function checks the following conditions:

1. TDIMT_IDX is valid:
   1.1. Reserved fields must be 0.
   1.2. LEVEL must be valid.

If successful, the function does the following:

2. Walk TDIMT tree using TDIMT_IDX to get the new TDIMT page parent entry.

If passed:

3. Return TDIMT entry 64-bit value in RCX.

Completion Status Codes

Table 3.103: TDH.TDI.MT.RD Completion Status Codes (Returned in RAX) Definition

Completion Status Code	Description
TDX_OPERAND_ADDR_RANGE_ERROR	
TDX_OPERAND_BUSY	
TDX_OPERAND_INVALID	
TDX_OPERAND_PAGE_METADATA_INCORRECT	
TDX_SUCCESS	
TDX_TDIMT_NOT_PRESENT	

3.2.32. TDH.TDI.REMOVE Leaf (Change in Progress)

Removes the TDI control structure and releases the metadata page(s).

Note: This ABI is ongoing changes to remove the MMIO and DMA removal pre-condition for TDI removal. These pre-conditions will move to TDH.TDI.BIND time instead.

Table 3.104: TDH.TDI.REMOVE Input Operands Definition

Operand	Name	Description
RAX	Leaf and Version	SEAMCALL instruction leaf number and version **Bits** / **Field** / **Description** 15:0 / Leaf Number / Selects the SEAMCALL interface function 23:16 / Version Number / Selects the SEAMCALL interface function version 63:24 / Reserved / Must be 0
RCX	FUNCTION_ID	FUNCTION_ID **Bits** / **Field** / **Description** 31:0 / Function ID / TDI Function ID 63:32 / Reserved / Must be 0
RDX	AUX_PAGE_PA	HPA of a shared 4K auxiliary page: • If **TDICS_MT_PAGES_COUNT == 1:** TDX Module will not use the auxiliary page, although will check its validity for forward compatilibity. The HPA of the released IDE Metadata page will be in the output register. • If **TDICS_MT_PAGES_COUNT > 1:** TDX Module uses the auxiliary page for HPA_ARRAY_T object with the released TDI Metadata pages (acquired in TDH.TDI.CREATE). The object is returned in the output register, see: HPA_ARRAY_T.

Table 3.105: TDH.TDI.REMOVE Output Operands Definition

Operand	Description
RAX	SEAMCALL instruction return code
RCX	HPA_ARRAY_T type containing the released IDE Metadata pages: • If TDICS_MT_PAGES_COUNT == 1: the value is the HPA of the released page. • If TDICS_MT_PAGES_COUNT > 1: the value is HPA_ARRAY_T(HPA == AUX_PAGE_PA, LAST_IDX == TDICS_MT_PAGES_COUNT). The VMM needs to map the AUX_PAGE_PA page to read the HPAs of the released pages.
Other	Unmodified

Leaf Function Description

Note: The description below is provided at a high level. Actual details, order of checks, returned status codes, etc. may vary.

TDH.TDI.REMOVE reclaims the TDI control and metadata page(s) after all TDI resources were released.

The function the following pre-conditions:

1. The TDI is in TDISP UNLOCKED state (see TDH.TDI.STOP). In case the device had to be forcefully removed by the VMM, its associated IDE stream must be blocked.
2. All MMIO pages were unmapped, unless the TD is in the FATAL state and the VMM needs forcefully remove the TDI.
3. PASIDTE DMAR entry was removed.

If successful, the function removes the TDI control structure and releases its metadata pages to the VMM.

Completion Status Codes

Table 3.106: TDH.TDI.REMOVE Completion Status Codes (Returned in RAX) Definition

Completion Status Code	Description
TDX_OPERAND_INVALID	
TDX_OPERAND_BUSY	Operation encountered a busy operand, indicated by the lower 32 bits of the status. In many cases, this can be resolved by retrying the operation.
TDX_OPERAND_ADDR_RANGE_ERROR	
TDX_OPERAND_PAGE_METADATA_INCORRECT	
TDX_TDI_INVALID_STATE	TDI_CS TDISP state is not CONFIG_UNLOCKED and its associated IDE-Stream is not blocked
TDX_TDI_HAS_BOUND_PASIDS	TDISP bound PASIDs must be removed
TDX_TDI_HAS_MAPPED_MMIO	Device MMIO mappings must be removed
TDX_SUCCESS	Operation is successful

3.2.33. TDH.TDI.START Leaf

Start the TDI and complete the TDI assignment.

Table 3.107: TDH.TDI.START Input Operands Definition

Operand	Description				
RAX	SEAMCALL instruction leaf number and version				
	Bits	**Field**	**Description**		
	15:0	Leaf Number	Selects the SEAMCALL interface function		
	23:16	Version Number	Selects the SEAMCALL interface function version		
	63:24	Reserved	Must be 0		
RCX	FUNCTION_ID	**Bits**	**Field**	**Description**	
		31:0	Function ID	TDI Function ID	
		63:32	Reserved	Must be 0	
RDX	**SPDM_RSP_HPA_ARR**– Page buffer of type HPA_ARRAY_T, with SPDM_DOE_PAGES_COUNT free page(s) for the SPDM Response message (Input)				
R8	**SPDM_REQ_HPA_ARR** - Page buffer of type HPA_ARRAY_T, with SPDM_DOE_PAGES_COUNT free page(s) for the SPDM Request message (Output)				

Table 3.108: TDH.TDI.START Output Operands Definition

Operand	Description
RAX	SEAMCALL instruction return code
RCX	If RAX returns **TDX_SPDM_REQUEST** – Generated TDISP output message (DOE+Secure SPDM enveloped) length in bytes. If RAX returns **TDX_TDISP_ERROR** – Returns the TDISP ERROR_CODE (bytes 0-3) and ERROR_DATA (bytes 4-7), see [TDISP]. For other return codes – 0.
Other	Unmodified

Leaf Function Description

Note: The description below is provided at a high level. Actual details, order of checks, returned status codes, etc. may vary.

TDH.TDI.START issues the START request (see START_INTERFACE in [TDISP]) and processes the response, as the final step of the TDI assignment.

The function checks the following conditions:

1. FUNCTION_ID is valid.
2. No other SPDM operation is in progress, see VMM Common SPDM Interface.
3. The secure session with physical device is connected and active: SPDM session is Connected, IDE Stream is configured and not blocked.
4. TD has approved the Interface Start (see TDG.TDI.START).

If successful, the function prepares START_INTERFACE request and returns TDX_SPDM_REQUEST. The VMM will send the request to the device and recall the function, see [the VMM Common SPDM Interface]. When the function returns TDX_SUCCESS, the TDI Interface is in the RUNNING state.

Completion Status Codes

Table 3.109: TDH.TDI.START Completion Status Codes (Returned in RAX) Definition

Completion Status Code	Description
TDX_OPERAND_BUSY	
TDX_OPERAND_INVALID	

Completion Status Code	Description
TDX_TDI_NOT_PRESENT	
TDX_TDI_INVALID_METADATA	Invalid TDIMT_IDX path, or the path for the entry index already exists
TDX_SPDM_ENTRY_NOT_PRESENT	
TDX_SPDM_SESSION_KEY_REQUIRE_REFRESH	The SPDM session requires a key refresh
TDX_SPDM_INVALID_MESSAGE	Invalid SPDM message or decryption/integrity check failure
TDX_SPDM_REQUEST	TDX Module requires VMM to send the request in OUT_PA with length
TDX_SPDM_INVALID_STATE	The associated SPDM session is not in CONNECTED state
TDX_IDE_STREAM_INVALID_STATE	The IDE stream is not configured or blocked
TDX_TDISP_ERROR	TDSP Error Response returned
TDX_TDISP_INVALID_MESSAGE	Unexpected/invalid TDISP response
TDX_TDI_INVALID_STATE	The interface is in invalid TDISP state or TD hasn't requested Start Interface
TDX_SUCCESS	

3.2.34. TDH.TDI.STOP Leaf

Stop TDI Interface.

Table 3.110: TDH.TDI.STOP Input Operands Definition

Operand	Description				
RAX	SEAMCALL instruction leaf number and version				
	Bits	**Field**	**Description**		
	15:0	Leaf Number	Selects the SEAMCALL interface function		
	23:16	Version Number	Selects the SEAMCALL interface function version		
	63:24	Reserved	Must be 0		
RCX	FUNCTION_ID	**Bits**	**Field**	**Description**	
		31:0	Function ID	TDI Function ID	
		63:32	Reserved	Must be 0	
RDX	**SPDM_RSP_HPA_ARR**– Page buffer of type HPA_ARRAY_T, with SPDM_DOE_PAGES_COUNT free page(s) for the SPDM Response message (Input)				
R8	**SPDM_REQ_HPA_ARR** - Page buffer of type HPA_ARRAY_T, with SPDM_DOE_PAGES_COUNT free page(s) for the SPDM Request message (Output)				

Table 3.111: TDH.TDI.STOP Output Operands Definition

Operand	Description
RAX	SEAMCALL instruction return code
RCX	If RAX returns **TDX_SPDM_REQUEST** – Generated TDISP output message (DOE+Secure SPDM enveloped) length in bytes. If RAX returns **TDX_TDISP_ERROR** – Returns the TDISP ERROR_CODE (bytes 0-3) and ERROR_DATA (bytes 4-7), see [TDISP]. For other return codes – 0.
Other	Unmodified

Leaf Function Description

Note: The description below is provided at a high level. Actual details, order of checks, returned status codes, etc. may vary.

TDH.TDI.STOP issues the STOP_INTERFACE request (see [TDISP]) and processes the response, as the first step of the TDI tear down.

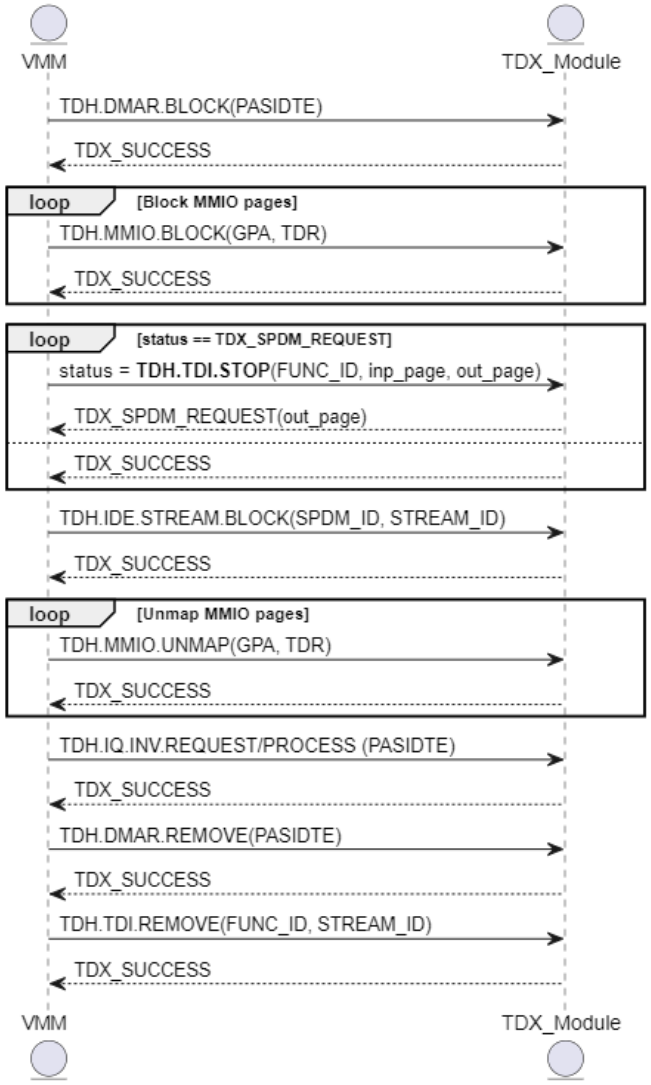
The function checks the following conditions:

1. FUNCTION_ID is valid.
2. No other SPDM operation is in progress, see VMM Common SPDM Interface.
3. The secure session with physical device is connected and active: SPDM session is Connected, IDE Stream is configured and not blocked.

If successful, the function prepares the STOP_INTERFACE request and returns TDX_SPDM_REQUEST. The VMM will send the request to the device and recall the function, see [the VMM Common SPDM Interface]. When the function returns TDX_SUCCESS, the TDI Interface state is UNLOCKED.

Important note: if TDH.TDI.STOP returns a failure, the TDI state is unknown (can be either LOCKED or UNLOCKED). In case a failure, the VMM shall sync with the actual interface by calling either TDH.TDI.GET.STATE or retrying TDH.TDI.STOP.

Figure 3.112: TDH.TDI.STOP in a Sample TDI Disconnect flow



Notes. If the known TDI state is INTERFACE_UNLOCKED, the TDX Module will return TDX_SUCCESS immediately.

Completion Status Codes

Table 3.113: TDH.TDI.STOP Completion Status Codes (Returned in RAX) Definition

Completion Status Code	Description
TDX_OPERAND_BUSY	
TDX_OPERAND_INVALID	
TDX_TDI_NOT_PRESENT	
TDX_TDI_INVALID_METADATA	Invalid TDIMT_IDX path, or the path for the entry index already exists
TDX_SPDM_ENTRY_NOT_PRESENT	
TDX_SPDM_SESSION_KEY_REQUIRE_REFRESH	The SPDM session requires a key refresh
TDX_SPDM_INVALID_MESSAGE	Invalid SPDM message or decryption/integrity check failure
TDX_SPDM_REQUEST	TDX Module requires VMM to send the request in OUT_PA with length
TDX_SPDM_INVALID_STATE	The associated SPDM session is not in a CONNECTED state
TDX_IDE_STREAM_INVALID_STATE	The IDE stream is not configured or blocked
TDX_TDISP_ERROR	TDSP Error Response returned
TDX_TDISP_INVALID_MESSAGE	Unexpected/invalid TDISP response
TDX_SUCCESS	

3.3. TDX Connect Guest-Side (TDCALL) Interface Functions (Change in Progress)

Note: The TD guest ABI commands in this section are changing to use function ID input as guest virtual function ID that the VMM allocates during TDI bind.

3.3.1. TDG.DMAR.ACCEPT Leaf

Accept a PASID table entry.

Table 3.114: TDG.DMAR.ACCEPT Input Operands Definition

Operand	Name	Description		
RAX	Leaf and Version	SEAMCALL instruction leaf number and version		
		Bits	**Field**	**Description**
		15:0	Leaf Number	Selects the SEAMCALL interface function
		23:16	Version Number	Selects the SEAMCALL interface function version
		63:24	Reserved	Must be 0
RCX	FUNCTION_ID	**Bits**	**Field**	**Description**
		31:0	Function ID	TDI Function ID
		63:32	Reserved	Must be 0
RDX	TARGET: Bits 0-7: **VM_IDX**, **0** – for non-partitioned TD or L1, **1-3 -** L2 VM 1-3 Bits 8-27: Reserved for Gen2. Must be 0. Bits 28-63: Reserved for Gen2. Must be 0			
R8-R15	RSVD – Must be 0			

Table 3.115: TDG.DMAR.ACCEPT Output Operands Definition

Operand	Description
RAX	SEAMCALL instruction return code
Other	Unmodified

Leaf Function Description

Note: The description below is provided at a high level. Actual details, order of checks, returned status codes, etc. may vary.

TDG.DMAR.ACCEPT is used by the TD to enable DMA access from the trusted device.

Before enabling the DMA access by setting the present bit of the DMAR PASID table entry, the TDX Module checks that the device interface is assigned to and validated by the TD accepting it (see TDG.TDI.VALIDATE).

For the partitioned TD only: the L1 may choose the TARGET: an L2 VM to assign the DMA to. Value 0 assigns the DMA to the L1, values 1-3 assign the DMA to VMs 1-3 respectively.

Work in progress!

Before reassigning the DMA to another VM, the L1 must ensure the PASID was invalidated, see TDG.DMAR.RELEASE.

The function checks the following conditions:

1. FUNCTION_ID is valid and the TDI is owned by the TD.
2. TDI has been validated, see TDG.TDI.VALIDATE.
3. DMAR PASIDTE table entry is DMAR_PENDING and the invalidation is done.

If successful, the function marks the DMAR present (setting the PASID table entry Present), allowing DMA access to the TD private memory via its secure EPT.

Completion Status Codes

Table 3.116: TDG.DMAR.ACCEPT Completion Status Codes (Returned in RAX) Definition

Completion Status Code	Description
TDX_OPERAND_INVALID	
TDX_OPERAND_BUSY	
TDX_SUCCESS	Operation is successful

3.3.2. TDG.IQ.INV.REQUEST

Request L1 GPA Range IOTLB invalidations. Work in progress!

Table 3.117: TDG.IQ.INV.REQUEST Input Operands Definition

Operand	Description		
RAX	TDCALL instruction leaf number and version		
	Bits	**Field**	**Description**
	15:0	Leaf Number	Selects the TDCALL interface function
	23:16	Version Number	Selects the TDCALL interface function version
	63:24	Reserved	Must be 0
RCX	**NUM_INV_DESCS** - Number of invalidation descriptors provided in REQ_PA		
RDX	**REQ_PA** - GPA of the 4K page containing invalidation descriptors		

Table 3.118: TDG.IQ.INV.REQUEST Output Operands Definition

Operand	Description
RAX	TDCALL instruction return code
Other	Unmodified

Leaf Function Description

Note: The description below is provided at a high level. Actual details, order of checks, returned status codes, etc. may vary.

TDG.IQ.INV.REQUEST is called by a TD to request IOTLB invalidations.

The function checks the following conditions:

1. There is no outstanding invalidation request.
2. There are TDIs attached to the TD, or TDX_GUEST_INV_NOT_REQUIRED is returned.
3. The DMAR PASIDTE is Present (mapped).

If successful, the function invokes TDEXIT to the VMM with TDX_IOTLB_INV_REQUEST status and number of descriptors requested, so the VMM can request and process the invalidations. The VMM resumes the TD at some point.

Completion Status Codes

Table 3.119: TDG.IQ.INV.REQUEST Completion Status Codes (Returned in RAX) Definition

Completion Status Code	Description
TDX_OPERAND_INVALID	
TDX_OPERAND_BUSY	
TDX_GUEST_INV_INVALID_DESC	Invalid invalidation descriptor
TDX_GUEST_INV_NOT_REQUIRED	IOTLB invalidation is not required
TDX_GUEST_INV_IN_PROGRESS	TD already requested the invalidation request and it's in progress
TDX_SUCCESS	

3.3.3. TDG.MMIO.ACCEPT Leaf (Change in Progress)

Accept a pending private MMIO page.

Note: This ABI is changing now to accept TDI guest function ID and operate on MMIO ranges from the TDISP report instead of MMIO page GPA and HPA addresses.

Table 3.120: TDG.MMIO.ACCEPT Input Operands Definition

Operand	Description		
RAX	TDCALL instruction leaf number and version		
	Bits	**Field**	**Description**
	15:0	Leaf Number	Selects the TDCALL interface function
	23:16	Version Number	Selects the TDCALL interface function version
	63:24	Reserved	Must be 0
RCX	EPT mapping information:		
	Bits	**Name**	**Description**
	2:0	Level	Level of the Secure EPT entry that maps the private page to be accepted: either 0 (4KB), 1 (2MB) or 2 (1GB).
	11:3	Reserved	Reserved: must be 0
	51:12	GPA	Bits 51:12 of the guest physical address of the private MMIO page to be accepted
	63:52	Reserved	Reserved: must be 0
RDX	MMIO page offset (see: [TDISP] spec)		

Table 3.121: TDG.MMIO.ACCEPT Output Operands Definition

Operand	Description
RAX	TDCALL instruction return code
Other	Unmodified

Leaf Function Description

Note: The description below is provided at a high level. Actual details, order of checks, returned status codes, etc. may vary.

TDG.MMIO.ACCEPT is used by the guest TD to verify and accept MMIO private GPA mapping.

The function checks the following conditions:

1. The Secure EPT walk based on the GPA operand is the requested level is succeeded and the page in the MMIO_PENDING state.

If successful, the function sets the MMIO page as mapped (MMIO_MAPPED).

Completion Status Codes

Table 3.122: TDG.MMIO.ACCEPT Completion Status Codes (Returned in RAX) Definition

Completion Status Code	Description
TDX_OPERAND_INVALID	
TDX_OPERAND_BUSY	

Completion Status Code	Description
TDX_MMIO_INVALID_HPA_OFFSET	
TDX_PAGE_ALREADY_ACCEPTED	
TDX_PAGE_SIZE_MISMATCH	
TDX_MMIOMT_RANGE_MISMATCH	
TDX_SUCCESS	TDG.MMIO.ACCEPT is successful.

3.3.4. TDG.TDI.RD Leaf

Read device interface related fields.

Table 3.123: TDG.TDI.RD Input Operands Definition

Operand	Description				
RAX	TDCALL instruction leaf number and version				
	Bits	**Field**	**Description**		
	15:0	Leaf Number	Selects the TDCALL interface function		
	23:16	Version Number	Selects the TDCALL interface function version		
	63:24	Reserved	Must be 0		
RCX	FUNCTION_ID		**Bits**	**Field**	**Description**
			31:0	FUNCTION_ID	TDI Function ID
			63:32	Reserved	Must be 0
RDX	Field code:				
	Bits	**Field**	**Description**		
	7:0	Field code	**Code**		**Description**
			0x1 (TDISP version)		TDISP version negotiated by TSM and DSM
			0x2 (INTERFACE_STATE)		TDI TDISP Known Interface State
	63:8	Reserved	Must be 0		

Table 3.124: TDG.TDI.RD Output Operands Definition

Operand	Description
RAX	TDCALL instruction return code
RCX	Field value
Other	Unmodified

Leaf Function Description

Note: The description below is provided at a high level. Actual details, order of checks, returned status codes, etc. may vary.

TDG.TDI.RD returns device interface related information based on input field code and parameter.

The function checks the following conditions:

1. FUNCTION_ID is valid and TDI is assigned to the TDI.
2. Field code and code parameter combination is valid, reserved fields are zero.

If the above conditions are satisfied, the function returns the requested TDI_CS field in RCX.

INTERFACE_STATE field can be used by the guest to peek the TDI's Known TDISP state. The TDX Module updates the known state on each TDISP response, therefore it would represent the **actual** TDISP state most of the times. Since TDI can change the state any time, TD can't fully rely on the Known RUN state in any scenario. However, TD can rely on the Known RUN state after TDG.TDI.START as a confirmation of the Interface Start.

From the Guest perspective, querying the INTERFACE_STATE is a correct way for TD to query the TDI status. If the call status is TDX_SUCCESS then the TDI is BOUND. Other statuses indicate that the TDI is UNBOUND:

TDX_TDIMT_NOT_PRESENT or TDX_TDI_INVALID_METADATA mean the TDI is UNBOUND and the TDI control structure has been removed or reassigned.

TDX_TDI_INVALID_STATE means TDI is UNBOUND or in the ERROR state (TDI in TDISP ERROR state is considered UNBOUND from TD's perspective).

Completion Status Codes

Table 3.125: TDG.TDI.RD Completion Status Codes (Returned in RAX) Definition

Completion Status Code	Description
TDX_TDI_INVALID_STATE	TDI is UNBOUND
TDX_OPERAND_BUSY	Operation encountered a busy operand, indicated by the lower 32 bits of the status. In many cases, this can be resolved by retrying the operation.
TDX_TDI_INVALID_METADATA	TDI is UNBOUND
TDX_TDIMT_NOT_PRESENT	TDI is UNBOUND
TDX_OPERAND_INVALID	
TDX_SUCCESS	TDI is in BOUND state

3.3.5. TDG.TDI.START Leaf

TD requests (authorizes) to Start the Interface.

Table 3.126: TDG.TDI.START Input Operands Definition

Operand	Description				
RAX	SEAMCALL instruction leaf number and version				
	Bits	**Field**	**Description**		
	15:0	Leaf Number	Selects the TDCALL interface function		
	23:16	Version Number	Selects the TDCALL interface function version		
	63:24	Reserved	Must be 0		
RCX	FUNCTION_ID		**Bits**	**Field**	**Description**
			31:0	Function ID	TDI Function ID associated with TDI_CS
			63:32	Reserved	Must be 0

Table 3.127: TDG.TDI.START Output Operands Definition

Operand	Description
RAX	SEAMCALL instruction return code
Other	Unmodified

Leaf Function Description

Note: The description below is provided at a high level. Actual details, order of checks, returned status codes, etc. may vary.

TDG.TDI.START indicates the TD's readiness to start the TDI interface. TD must call the function after it accepted DMA and MMIO (see TDG.DMAR.ACCEPT, TDG.MMIO.ACCEPT).

The function checks the following pre-conditions:

1. Function ID is valid and assigned to the TD.
2. TDI is validated, see TDG.TDI.VALIDATE

If successful, the TDX Module allows the VMM to start the TDI, see TDH.TDI.START.

Completion Status Codes

Table 3.128: TDG.TDI.START Completion Status Codes (Returned in RAX) Definition

Completion Status Code	Description
TDX_OPERAND_BUSY	
TDX_OPERAND_INVALID	

Completion Status Code	Description
TDX_TDI_NOT_PRESENT	
TDX_TDI_INVALID_METADATA	Invalid TDIMT_IDX path, or the path for the entry index already exists
TDX_TDI_INVALID_STATE	The interface is in invalid TDISP state or TD hasn't requested Start Interface
TDX_SUCCESS	

3.3.6. TDG.TDI.VALIDATE Leaf

Validates the integrity of the TDI's Device Attestation info and Interface Report that the VMM exchanges directly with TD.

Table 3.129: TDG.TDI.VALIDATE Input Operands Definition

Operand	Description			
RAX	TDCALL instruction leaf number and version			
	Bits	**Field**	**Description**	
	15:0	Leaf Number	Selects the TDCALL interface function	
	23:16	Version Number	Selects the TDCALL interface function version	
	63:24	Reserved	Must be 0	
RCX	FUNCTION_ID	**Bits**	**Field**	**Description**
		31:0	Function ID	TDI Function ID associated with TDI_CS
		63:32	Reserved	Must be 0
RDX	GPA of the TD private page containing the device interface hashes:			
	Name	**Bytes**	**Description**	
	TDI_PKH	47:0	Device interface certificate public key hash (SHA384)	
	TDI_IRH	95:48	Actual device Interface Report hash (SHA384)	
	Reserved	4095:96	Reserved, must be 0.	

Table 3.130: TDG.TDI.VALIDATE Output Operands Definition

Operand	Description
RAX	TDCALL instruction return code
Other	Unmodified

Leaf Function Description

Note: The description below is provided at a high level. Actual details, order of checks, returned status codes, etc. may vary.

This function verifies that the TDI is assigned and bound to the TD (see TDH.TDI.BOUND) and that the Device Attestation and Interface Report hashes (the VMM provided to the TD as the result of the SPDM session establishment, see TDH.SPDM.CONNECT and TDI Bind) were not replaced or tampered with.

The function checks the following pre-conditions:

1. Function ID is valid and assigned to the TD.
2. TDI is bound.
3. TDI_PKH matches the SPDM session's Device Attestation Information.
4. TDI_IRH matches the Device Interface Report.

If successful, the function returns TDX_SUCCESS, indicating that the TD can accept the TDI's MMIO and DMAR, see TDG.DMAR.ACCEPT and TDG.MMIO.ACCEPT.

Completion Status Codes

Table 3.131: TDG.TDI.VALIDATE Completion Status Codes (Returned in RAX) Definition

Completion Status Code	Description
TDX_TDI_ALREADY_VALID	
TDX_TDI_SPDM_PKH_MISMATCH	Interface Public Key hash mismatch
TDX_TDI_SPDM_IRH_MISMATCH	Interface Report hash mismatch, or there is no actual Interface Report for the interface (TDH.TDI.BIND wasn't executed successfully).
TDX_OPERAND_BUSY	Operation encountered a busy operand, indicated by the lower 32 bits of the status. In many cases, this can be resolved by retrying the operation.
TDX_OPERAND_INVALID	
TDX_SUCCESS	

Intel TDX Connect Application Binary Interface (ABI) Reference