Intel® TDX Connect Architecture Specification

Notices and Disclaimers:

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

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

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

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

No product or component can be absolutely secure.

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

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

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

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

© Intel Corporation. Intel, the Intel logo, and other Intel marks are trademarks of Intel Corporation or its subsidiaries. Other names and brands might be claimed as the property of others.

354629 003-US

June 2025

CONTENTS

1. INTRODUCTION

1.1. About

This document outlines the system architecture specifications for Intel® TDX Connect, a TDX extension technology for secure and efficient integration with trusted I/O devices.

It is part of the Intel TDX Connect Architecture Specification Set, which also includes the following documents:

Table 1.1: TDX Connect Architecture Specification Set

Document Name	Reference	Description
Intel® TDX Connect Architecture	[TDX Connect Spec]	This document.
Intel® TDX Connect ABI FAS	[TDX Connect ABI Spec]	TDX module Application Binary Interface (ABI) reference specification extensions for TDX Connect.
Intel® TDX Connect Guest-Hypervisor Communication Interface	[TDX Connect GHCI]	Intel TDX specification of the software interface between the Guest TD and the VMM.
Intel® TDX Connect TEE-IO Device Guide	[TDX Connect TEE-IO Device Guide]	An introductory TDISP device overview on how to implement TEE-IO devices which are compliant with PCIe TDISP 1.0 and with Intel® TDX Connect.
Intel® TDX Connect Software Developer Guide	[TDX Connect SW Guide]	Software flow guide and examples for Host VMM, TD OS, and PCIe driver support of TDX Connect.

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. Introduction to TDX Connect

Intel® TDX base architecture [1] defines a Trusted Execution Environment (TEE) for virtual machines, known as Trusted Domains (TDs). TDs are isolated virtual machines with hardware-enforced confidentiality and integrity protections for CPU state, memory, and code execution.

Each TD's memory is divided into two domains:

- Shared Memory: Accessible to host software and devices. Mapped into the TD's shared EPT and accessed using Guest Physical Addresses (GPAs) with the Shared bit set.
- Private Memory: Encrypted on a per-TD basis using dedicated memory encryption keys. Accessible only to the TD itself, while still mapped via the shared EPT with GPA addresses marked as Shared.

While the TDX base architecture provides strong isolation from the hypervisor and untrusted devices, it imposes limitations on direct I/O. By default, all PCIe devices are considered untrusted and are therefore not permitted to access TD private memory directly. Consequently, device I/O operations must occur through shared memory buffers.

In this model, the TD is responsible for copying and encrypting/decrypting data between private and shared memory. This process, commonly known as bounce buffering, introduces overhead and complexity—especially for accelerators that require direct access to plaintext data, such as GPUs or Smart-NICs.

As confidential computing environments evolve, support for efficient trusted I/O device assignment to TDs becomes essential for enabling high-performance workloads that rely on I/O devices requiring access to unencrypted data.

Intel® TDX Connect is designed to address this challenge by enabling direct assignment of trusted PCIe devices, known as TEE Device Interfaces (TDIs), to TDs. TDX Connect builds upon the TDX foundation and leverages industry-standard protocols developed through collaboration within PCI-SIG and DMTF:

- TDISP (TEE Device Interface Security Protocol) [2]: Defines the secure lifecycle, attestation process, and binding of PCIe device functions (TDIs) to TEEs.
- IDE (Integrity and Data Encryption for PCIe) [3]: Provides cryptographic protection for PCIe transactions, supporting confidentiality, integrity, and replay resistance.
- SPDM (Security Protocol and Data Model) [4][5]: A DMTF-defined protocol used to establish authenticated sessions, exchange device certificates and measurements, and manage IDE key provisioning.

Built upon these standards Intel® TDX Connect goal is to enable direct TDISP device assignment to TDs while preserving the isolation and trust guarantees of TEEs.

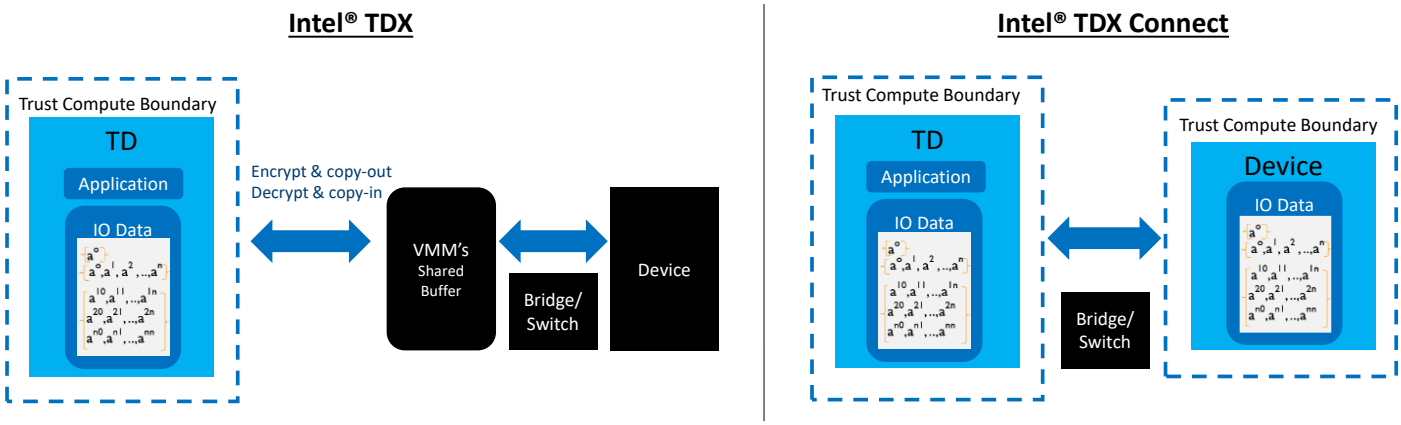


Figure 1.1: IO Virtualization with Intel® TDX vs. Intel® TDX Connect

1.3. Introduction to PCI-SIG TDISP and IDE

Recently, Intel and other hardware vendors, strong advocates of confidential computing, collaborated under the PCI-SIG to define two foundational standards to support secure direct device assignment in Trusted Execution Environments (TEEs). These are the Trusted Execution Environment Device Interface Security Protocol (TDISP), which defines the security lifecycle and assignment protocol for PCIe-based device functions to TEE-based virtual machines, and Integrity and Data Encryption (IDE), which provides a cryptographic transport layer to protect PCIe data-in-flight with confidentiality, integrity, and replay protection.

TDISP enables the binding of PCIe device functions—known as TEE Device Interfaces (TDIs) to isolated virtual machines known as TEE Virtual Machines (TVMs). It introduces a trust model that extends the TEE's protection boundary to include device functions, while maintaining the strong isolation and attestation principles expected of confidential compute platforms.

TDISP introduces four key entities:

- **TEE VM (TVM):** A virtual machine requiring strict isolation and security features. In TDX Connect, this is realized as a Trust Domain (TD).

- **TEE Device Interface (TDI):** TDISP device unit of assignment (e.g., device function) including its associated device resource that can be securely bound to a TVM. This concept remains the same in TDX Connect, referring to the device-side function capable of TDISP interactions.
- **TEE Security Manager (TSM):** The host-site security authority that protects the TVM isolation and extend by TDISP to manage the security of TDI assignment to a TVM. In TDX Connect this is role realized by the TDX module and by the Intel CPU Root Complex extension to support trusted DMA mapping , secure MMIO and PCIe IDE link protection.
- **Device Security Manager (DSM):** A logical component residing on the device that enforces TDISP security policies. It manages device-side operations via TDISP protocol messages, including SPDM session establishment, attestation reporting, PCIe IDE link setup, locking and reporting TDI configurations, TDI state tracking, and maintaining TDI security and isolation.

To protect data in transit between the host and device, TDISP refers to the PCIe Integrity and Data Encryption (IDE). “IDE supports two types of link encryption: Link IDE (port-to-port) and Selective IDE (end-to-end). TDX Connect uses Selective IDE with the goal to exclude switches the trust boundary.

To establish trust, the TSM and DSM initiate a secure SPDM session over the PCIe DOE mailbox transport [6]. TDISP and IDE use protocol messages which are encapsulated within secure SPDM session using AES-256-GCM encryption. This secure session is used to exchange device identity certificates and attestation measurements and to establish a secure channel for provisioning IDE keys and managing TDI state transitions.

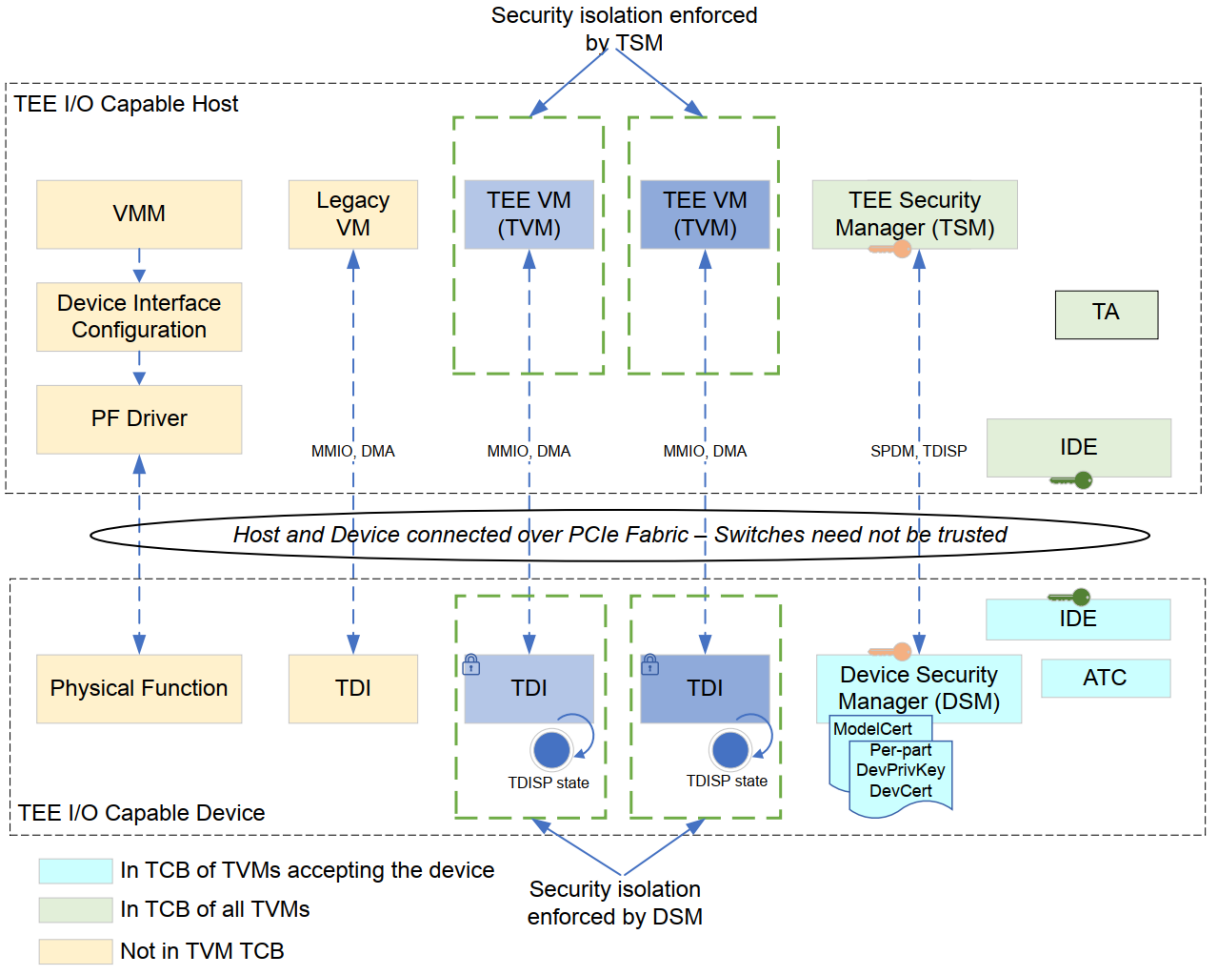


Figure 1.2: PCIe Base Spec TDISP Overview

The TDISP trust model requires the TVM to explicitly and independently inspect and verify the TDISP devices identities and measurements are not automatically part of a TVM’s TCB unless the TVM explicitly accepts and verifies them. This verification process involves:

- Using the TSM, the VMM can manage TVM mapping to TDI’s private MMIO resources and enable trusted DMA mapping to allow TDI access TVM private memory. The trusted MMIO and DMA traffic is transmitted over PCIe IDE Selective Stream.

- The TSM generates TDISP messages over the secure SPDM session, which are then transported by the VMM to the device's DOE mailbox. The DSM operates the TDI state machine and reports the TDI configuration in response.
- Device Identity Validation – The TVM must independently check the device's identity as reflected in the SPDM certificates and measurements to ensure it is known and trustworthy.
- TDI Configuration Check – The TVM must review the TDI configuration, as reflected in the TDISP report, ensuring that the TDI is properly configured and does not introduce security risks.

The DSM maintains a per TDI state machine which governs its security lifecycle to ensure controlled transitions between states while maintaining confidentiality and integrity for TVMs and bound TDIs.
The state machine consists of the following key states:

- **CONFIG_UNLOCKED (Default State):** The VMM configures the TDI in preparation for assignment to a TVM. Confidential data should not be placed in the TDI while in this state. Memory requests originating from a TVM are rejected.
- **CONFIG_LOCKED (Pre-Assignment Security Lock):** The VMM finalizes the TDI configuration and requests the TSM to lock it. The DSM ensures that the TDI meets security requirements before transitioning. Any detected configuration changes result in a TDISP ERROR state.
- **RUN (Operational State):** The TVM has accepted the TDI into its TCB. The TDI is fully operational and securely interacts with the TVM. Any unauthorized changes or security violations transition the TDI to ERROR.
- **ERROR (Failure State):** Triggered by security violations or misconfigurations. The TDI must not expose confidential TVM data. Memory requests originating from a TVM are rejected.

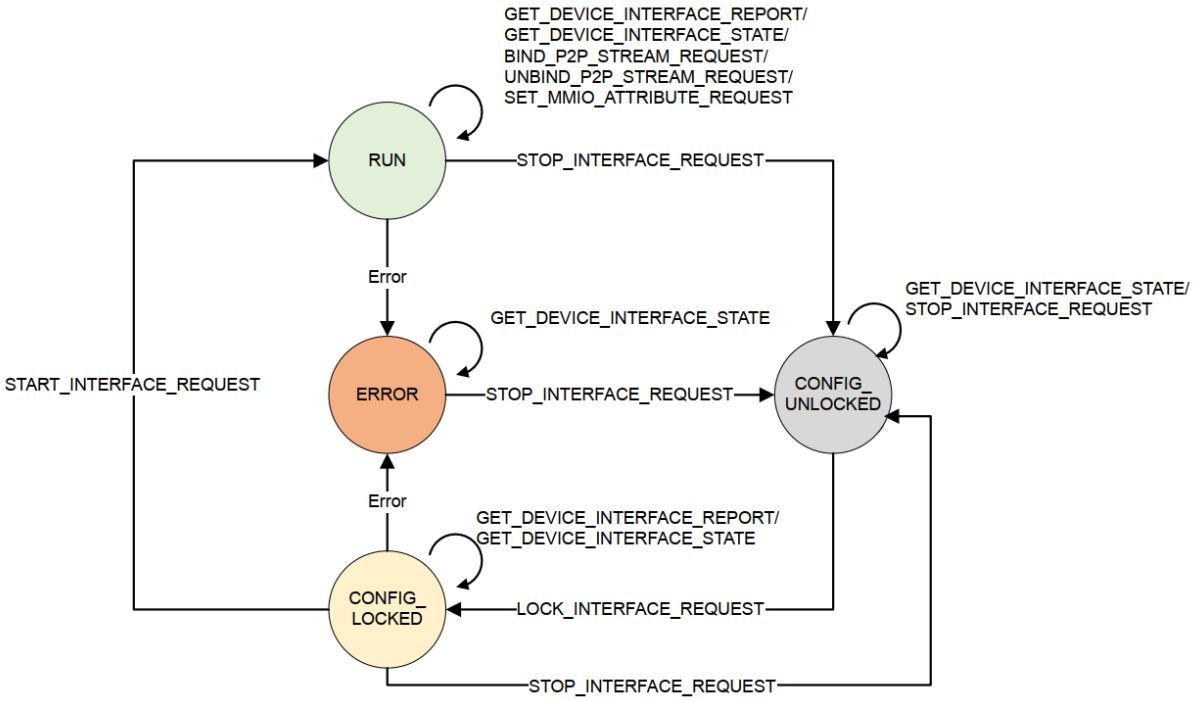


Figure 1.3: PCI-SIG TDI State Machine

These key messages form the TDISP protocol are generated by the TSM to operate the TDI binding state machine and report TDI status and configuration to the TVM:

LOCK_INTERFACE_REQUEST: Transitions the TDI from its default CONFIG_UNLOCKED state to CONFIG_LOCKED, preventing unauthorized configuration changes. The DSM ensures that the TDI remains in a locked configuration state and transitions it to ERROR if the TDI configuration is altered.

GET_DEVICE_INTERFACE_REPORT: Reports the device attestation and TDI locked configuration to the TVM, ensuring the TVM can verify the device identity and TDI configuration before accepting it into its TCB. TDISP relies on SPDM protocol

messages for device attestation and TDISP protocol messages for TDI configuration reporting, both delivered to the TVM by the VMM, with integrity enforced by the TSM.

START_INTERFACE_REQUEST: If the TD accepts the device into its TCB, the TSM enables trusted DMA access from the TDI to the TVM and trusted MMIO access from the TVM to the TDI. The TSM then issues the TDISP START_INTERFACE_REQUEST message, instructing the DSM to transition the TDI into its operational state (RUN).

STOP_INTERFACE_REQUEST: Used during the TDI unbind process, ensuring that the VMM reclaims TDI resources (e.g., MMIO pages) and removes DMA access to the TVM. Before detaching the TDI, the TSM generates this message to securely instruct the DSM to quiesce the TDI and drain outstanding TLPs, ensuring that all traffic related to the TDI binding is completed or flushed.

2. ARCHITECTURE OVERVIEW

2.1. TDX Connect Framework

The TDX Connect framework is composed of the following core components:

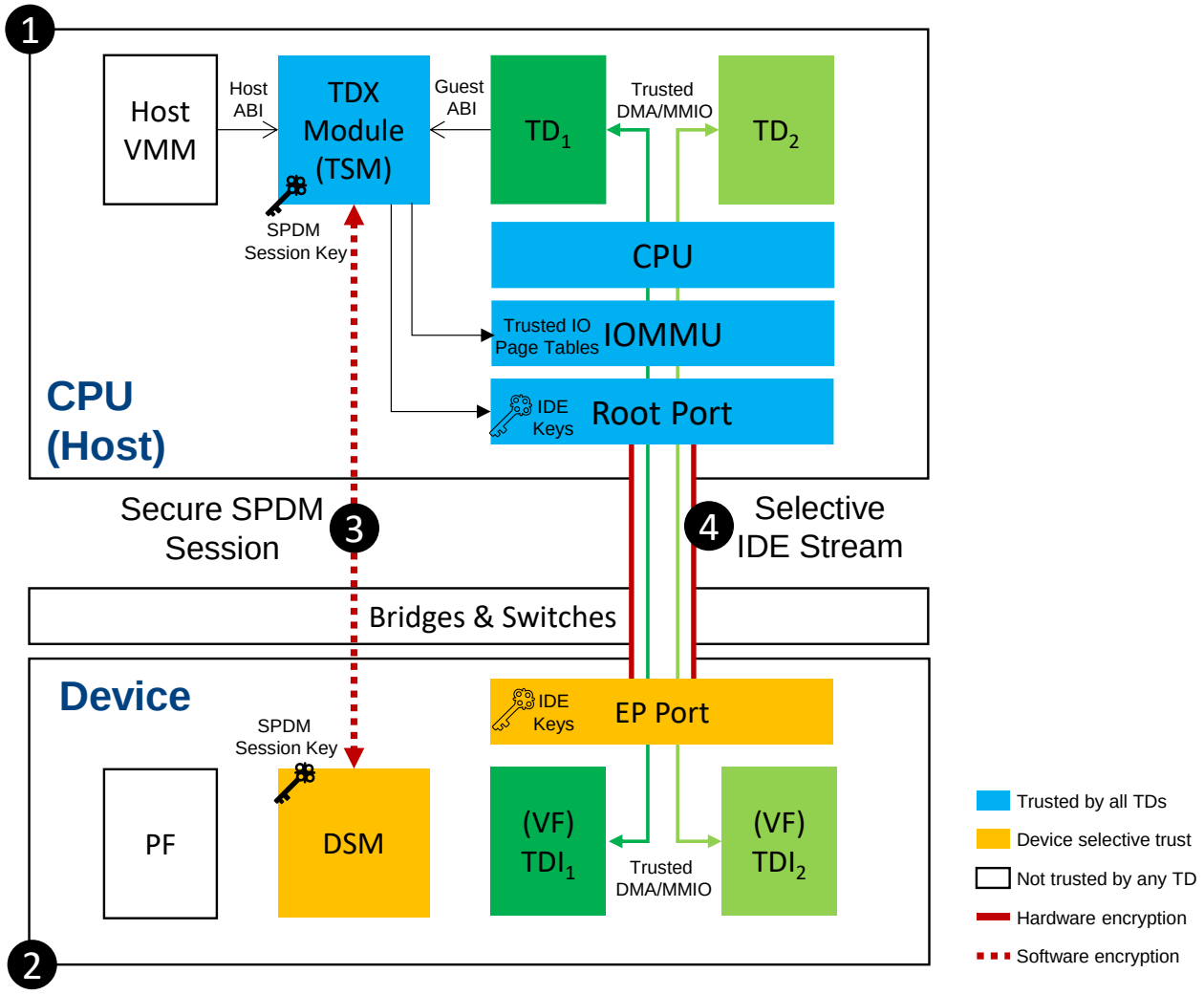


Figure 2.1: Intel TDX Connect Framework

1. **TDX Connect CPU and Module Extensions (TSM):**

   The **TDX module (TSM)** provides critical ABI extensions to securely manage the lifecycle of TDISP device connections and the binding of TDIs to TDs. This functionality is achieved through several enhancements at both the hardware and software layers:

   - **CPU Root Complex Extensions:** The CPU Root Complex has been extended to support **PCIe Integrity and Data Encryption (IDE)** and trusted IOMMU mechanisms. These enhancements isolate between non-confidential I/O page tables and invalidation interfaces allowing orthogonal and TDX module controlled secure IOMMU management. The VMM is responsible for opting into TDX Connect per IOMMU (and corresponding Root Complex), once this is enabled, the TDX module is the only entity allowed to write into specific VTd registers or modify the security related PCIe configuration registers of the Root Port IDE and related settings.

   - **TDISP Device Connection and Secure Configuration:** The TDX module exposes a secure ABI that facilitates the entire TDISP device connection process. This includes the establishment of a secure **SPDM (Security Protocol and Data Model)** session and the management of IDE encryption keys. Once the secure connection between the

CPU and the TDISP device is established, the VMM can utilize the TDX module's TDISP ABIs to lock down the TDI configuration, request configuration reports, and proceed with the TDI-TD binding process.

- **TDI Binding to TD:** After successfully binding the TDI to a TD, the TDX module ensures the mapping of **private MMIO (Memory-Mapped I/O)** and DMA operations specific to that TDI. The TDX module also provides the TD with a set of ABIs to verify the TDI attestation report and confirm its integrity before transitioning the TDI into its operational state.

- **TDI Unbinding and Disconnection:** In addition to binding, the TDX module includes ABIs for the VMM to unbind a TDI from a TD and to safely disconnect the TDISP device. Furthermore, the TDX module offers specific ABIs for TDs, ensuring that when a TDI is unbound, all resources, configurations, and cryptographic keys are securely cleaned up, maintaining the integrity and security of the system.

In essence, the TDX module acts as the trusted intermediary between the VMM, TD, and TDISP devices, enforcing secure device management, binding, and ensuring the protection of the TEE during the lifecycle of I/O operations.

2. **TDISP Compatible Device (DSM):**

   The Device Security Manager (DSM) is embedded in the trusted device. It operates in coordination with TDX module (TSM) responding to request to setup SPDM session, manage IDE keys and the secure binding of TDIs to TDs using TDISP protocol messages.

3. **Attestation and Secure Message Transport (DMTF SPDM Protocol and Secure Session):**

   The Security Protocol and Data Model (SPDM) helps ensure secure communication between the TDX module (TSM) on the CPU and DSM on the device. It enables attestation of the device's identities and measurements required to establish TD trust and provides a secure messaging framework that protects data exchanged between the TSM and the DSM for the IDE Key management and TDISP protocols.

4. **PCIe Link Encryption (Selective IDE Stream):**

   PCIe Integrity and Data Encryption (IDE) ensures that data exchanged over the PCIe link between the CPU and the device is encrypted, protecting it from eavesdropping and tampering. IDE selectively encrypts streams to secure data paths for trusted devices without incurring unnecessary overhead on non-critical data paths.

Together, these ingredients form the TDX Connect framework, which securely enables the direct assignment of TEE I/O Device Interfaces (TDIs) to TDs, while maintaining uniform TD and TDI TEE confidentiality and integrity.

2.2. Trusted DMA

According to the TDISP specification, when a TDI is bound and in the RUN state, it must send DMA requests to the host via the default IDE stream marked as a TEE request (IDE T = 1). Once the host Root Port (RP) decrypts and authenticates the IDE request, it forwards it to the IOMMU for access control, preserving the T = 1 designation. The IOMMU identifies these as TEE-IO requests.

TDX Connect introduces IOMMU support for TEE-IO requests, allowing access to TD private memory and includes Trusted VT-d interfaces to let the TDX module enforce binding permissions for TD private memory. Only TEE-IO DMA requests can access TD private memory, and only the TDX module can configure the access control permissions for TEE-IO.

Specifically, IOMMU and VT-d are enhanced with:

1. **Trusted DMA translation root table** – A second root table address register, restricted to SEAM SAI, which the TDX module programs with the root of the TDX-managed DMA translation tables for TD-assigned devices. This allows the TDX module to enforce the integrity of DMA translations.

2. **Trusted invalidation queue** – A second invalidation queue address register, along with corresponding head and tail registers, also restricted to SEAM SAI. The TDX module manages these, allowing it to control IOMMU and device TLB invalidations securely.

3. **I/O TLB and DID Isolation** – When IOMMU is enabled by the VMM to support TDX Connect, the IOMMU restricts the VMM's DID setting, reserving the MSB bit for the TDX module. The TDX module always sets this reserved bit

on the trusted DMA table. IOMMU tags IOTLB, PASID cache, and context entries to indicate whether they were created from TEE-IO transactions, ensuring isolation between TEE and non-TEE requests in translation caches.

The TDX module provides ABIs for the VMM to manage trusted (TEE-IO) DMA page tables, while the VMM directly manages shared (non-TEE-IO) DMA page tables. Trusted DMA mappings require acceptance from the TD owning the TDI, proving to the TDX module that the TD accepts the TDI into its trust boundary. Once accepted, the TDX module connects the second-level page table (EPT) to the TD secure EPT, ensuring consistency between the I/O and core view of the GPA space, preventing effective remapping attacks. Trusted and shared page tables are isolated with separate root table address registers.

Additionally, the TDX module provides an IOTLB invalidation ABI to invalidate cached I/O translations of TEE-IO requests as necessary to protect the TD. The TDX module ensures that the VMM completes IOTLB invalidations before freeing a TD private page or unbinding a TDI and its trusted DMA mappings.

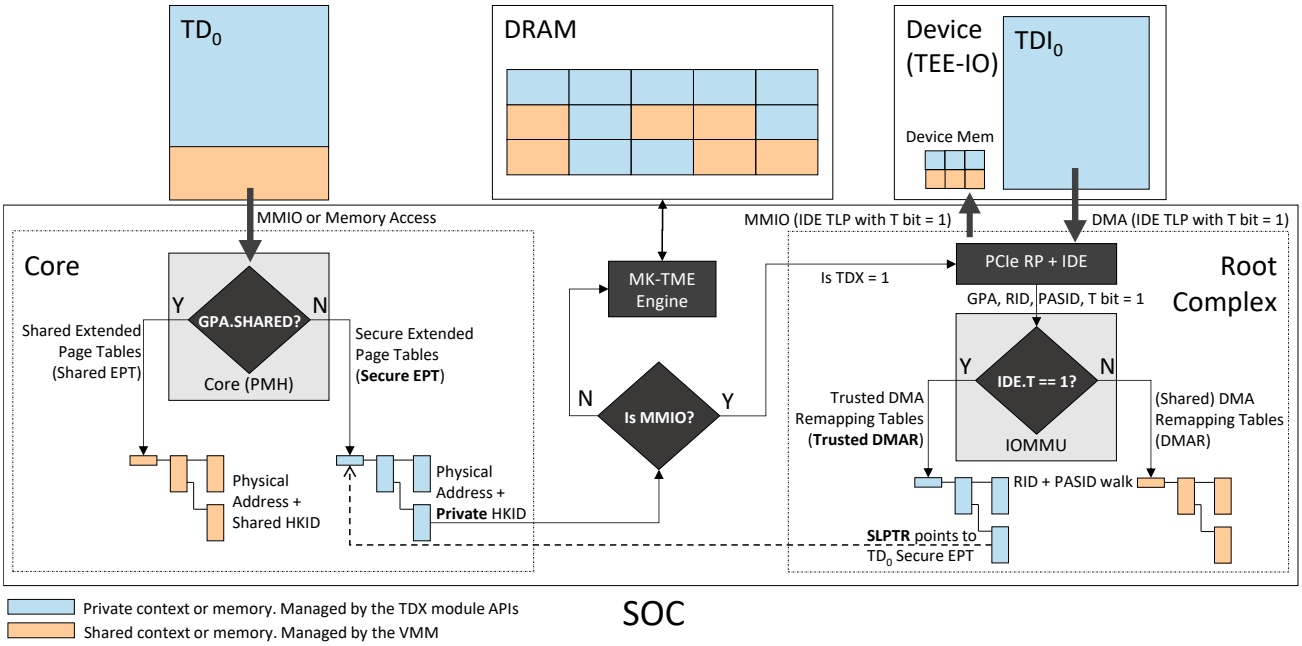


Figure 2.2: Trusted DMA and Secure-EPT

2.3. TDX Connect Enumeration

TDX Connect enumeration discovery is done similarly to other TDX module features, using `TDH.SYS.RD` ABI. The VMM first reads the TDX_FEATURES[TDX_CONNECT] (bit 6) to check if TDX Connect is supported by this TDX module on this platform.

When TDX Connect capability is supported, TDX_CONNECT_FEATURES bitmask provides the features supported by TDX module and platform (e.g., ATS, PRS, etc.). For more information refer to the TDX Connect ABI FAS specification.

2.4. Enabling TDX Connect

Intel host CPUs may support multiple PCIe Root Complex IO hubs, each forming a full x16 PCIe hierarchy with a corresponding IOMMU and Root Ports. To maintain integrity of PCIe transaction routing between the hierarchy, CPU memory, and cores, TDX Connect restricts the host VMM's control over the IOMMU VTd BAR, the IDE Key Configuration BAR, and the PCIe Root Port IDE, ACS and PCI header registers to enforce the integrity of private MMIO and DMA over PCIe IDE transactions.

Initially, TDX Connect is disabled on all IO Hubs, allowing VMM to configure the IOMMU and enumerate the PCIe hierarchy.

Once setup is complete, the VMM requests the TDX module to enable TDX Connect per IO Hub using the `TDH.IOMMU.SETUP` command. The VMM can disable TDX Connect via `TDH.IOMMU.CLEAR`, for example if PCIe links need to be retrained or re-enumerated, but first, the VMM must unbind all TDIs from TDs and disconnect all the TDISP devices from TDX module.

2.5. TDX Connect Life Cycle Overview

The diagram below illustrates the conceptual TDISP Lifecycle for TDX Connect. The process starts with the VMM Host establishing an SPDM (Security Protocol and Data Model) session and programming the IDE Selective Stream keys during the device connection phase. This phase outcome is that the SPDM session and IDE link with the device are established and the VMM gets the device SPDM attestation structures it can cache for later TD to TDI binding attestation purposes.

This lifecycle supports both SR-IOV (Single Root I/O Virtualization), where device FVs and/or PF can be assignable by a device as TEE-IO Device Interfaces (**TDI**). All the TDIs hosted by a physical device, shared the same SPDM session and default IDE Stream which collectively represent TDISP device to TDX host connection.

A TDI may be assigned as **Shared TDI** and used as non-confidential device interface by a TD. During TD runtime, the TD guest may enumerate Shared TDI as TDISP capable and request the host VMM to configure and bind it as a **Private TDI**.

The VMM configures the devices and issues the TDH.TDI.BIND ABI command. The TDI bind command outputs the TDISP report which the host VMM returns to the guest along with the SPDM attestation reports obtained during the device connect.

As part of the device attestation, TD must verify the integrity of the device attestation info and TDISP report using TDG.TDI.VALIDATE. If the TD decides to accept the TDI into its TCB, the guest uses TDG.MMIO.ACCEPT and TDH.DMAR.ACCEPT to enable the TDI DMA and MMIO mappings which allow private memory access between the guest and TDI. Finally, the guest notifies the device to transition the TDI into RUN state using the TDG.TDI.START and requesting the VMM to orchestrate and deliver the message. The VMM involves the TDH.TDI.START ABI command which generates the SPDM secure TDISP message delivered by the VMM to the device DOE mailbox.

In case the TDI encounters an error or if the guest wishes to disconnect the TDI from its TCB, it issues a VMM GHCI TDI unbind request. The host VMM uses the TDH.TDI.UNBIND ABI to generate the TDISP request and follow the subsequent MMIO and DMA mapping process and return to the guest indicating TDI unbind is complete. The TD guest can confirm TDI is unbound reading the TDI state using `TDG.TDI.RD`.

In any case, the host VMM is in full control and can unbind any TDI from a TD and disconnect TDISP device (stopping SPDM session and disable IDE). The TDX module also allows a secure process of **aborting** SPDM device connection only from the host side in case the device hangs or cannot respond to SPDM messages (i.e. DOE mailbox timeout).

The diagram and steps below describe a typical end-to-end life cycle stating with device connection followed by TDI binding, unbinding and device disconnect.

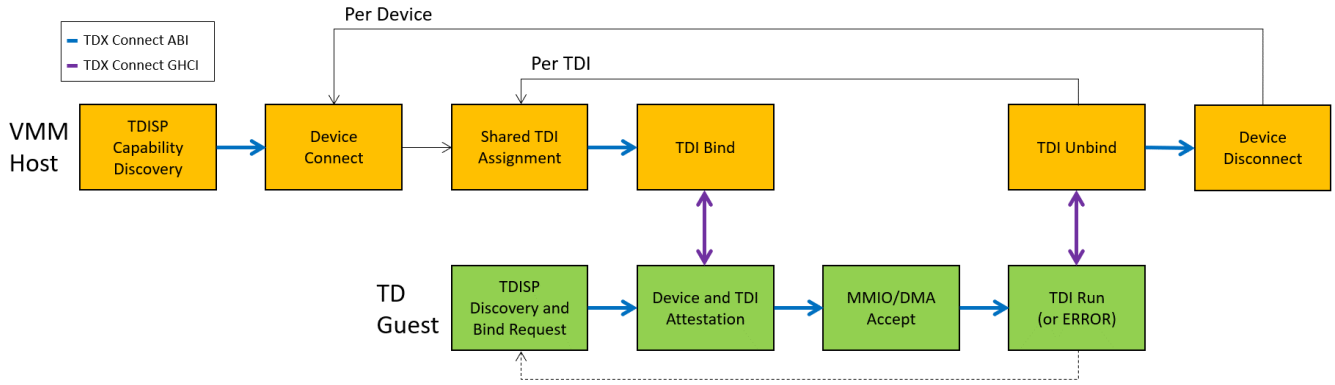


Figure 2.3: TDX Connect Conceptual Life Cycle

1. **TDISP Capability Discovery**: The VMM Host begins by discovering the TDISP capabilities of the device using the device configuration space.

2. **Device Connect**: The host establishes a connection with the TDISP device using TDX Connect ABI. It first uses `TDH.SPDM.CONNECT` to establish the SPDM session, followed by `TDH.IDE.STREAM.KM` to set up the IDE stream and program its keys on both the host and the device.

3. **Shared TDI Assignment**: The host VMM assigns the TDI to the guest as a shared (non-confidential) in CONFIG_UNLOCKED state. The host VMM may not know if the guest is willing or capable of supporting TDISP. Assigning the TDI as shared, allows the guest to enumerate TDISP (using virtualized PCI config space or para-virtualized guest-host interface).

4. **TDI Bind**: At runtime, after the TD guest discovers the device's TDISP capability and decides to use it as a private TDI, the guest requests the host VMM to bind the TDI to itself. The host VMM uses the `TDH.TDI.BIND` command to bind the TDI in the CONFIG_LOCKED state to the TD. As part of the binding process, the host VMM maps the TDI MMIO using `TDH.MMIO.MAP` and trusted DMA using `TDH.DMAR.ADD`. Both the MMIO and DMA mappings are initially mapped as “pending”, allowing the TD guest to independently accept the TDI into its trust boundary before the mappings become active.

5. **Device and TDI Attestation**: During the SPDM session establishment, the device generates an attestation composed of its certificate chain and measurements, both signed by the device's private key. Likewise, during the TDI bind, the TDI generates its TDISP report. The TDX module calculates and stores a SHA-384 digest of the attestation and report. The host VMM retrieves this data, passing it to the TD. The TD verifies the attestation and report, ensuring the authenticity of the device and the TDI. If the TDI is accepted, the TD guest invokes `TDG.TDI.ACCEPT`, providing the digests of the attestation and report to the TDX module. The module compares the digests to verify the integrity of the TDI information passed by the host VMM. Once verified, the TDX module marks the TDI as valid for subsequent TD operations.

6. **MMIO/DMA Accept**: After a successful attestation and the acceptance of the TDI by the TD guest, the TD uses the information provided in the TDISP report, which includes the device function ID and its MMIO ranges. The TD then explicitly accepts both the DMA and MMIO mappings of the TDI. This process makes the mappings present, allowing both the TD and TDI to securely access each other’s private (TEE) memory.

7. **TDI Run or Error**: To complete the binding process, the TD guest uses `TDG.TDI.START`, signaling to the TDX module that the TD is ready for the TDI to transition into the RUN state. The TD guest then requests the host VMM to start the TDI. In turn, the VMM calls TDH.TDI.START, requesting the TDX module to transition the TDI to the RUN state. At this point, the TDI is bound to the TD and becomes operational.

   Note: As indicated by TDISP, the device may experience a security-related failure, leading to its voluntary transition into the ERROR state. The discovery of the TDI ERROR state is vendor-specific and outside the scope of this specification. If an ERROR is detected, the TD guest can request the host VMM to re-bind the TDI, which is handled as a sequence of TDI unbind and re-bind operations.

8. **TDI Unbind**: In the event of a TD teardown, migration, device maintenance, or error handling, the host VMM can unbind the TDI from the TD by using the `TDH.TDI.UNBIND` command. Unbinding can also be initiated by the TD guest, either because the TD no longer requires the device, for handling device errors, or for device maintenance. If the TD requests a TDI unbind, it should not rely on the VMM to complete the operation. The TD must independently verify the TDI binding status reading its INTERFACE_STATE using the `TDG.TDI.RD` command. Failing to do so could result in the TD releasing the TDI resources (e.g., DMA), while the TDI remains connected to the TD, potentially allowing access to the released DMA buffer.

9. **Device Disconnect**: Once no TDI hosted by a TDISP device is bound to a TD, the host VMM can disconnect the device by disabling its IDE stream using the `TDH.IDE.STREAM.BLOCK` command and tearing down its SPDM session using the `TDH.SPDM.DISCONNECT` command. In some cases, disconnecting a device is necessary to update the device firmware and demonstrate its new security posture to a new tenant through attestation. In other cases, device disconnection is performed as a recovery step following PCIe link, IDE, or device errors, prompting the host VMM to reprogram the IDE stream keys or reset the SPDM session.

2.6. SPDM Management

TDX Connect uses SPDM 1.2 for TDISP device attestation and sets up a Secure SPDM over (unprotected) DOE message transport channel to protect communication between the device's DSM and the TDX module (TSM). This secure channel is essential for ensuring the integrity and confidentiality of device attestation, while also securing IDE key management and TDISP protocol messages.

The SPDM session management in TDX Connect involves the secure initialization, handling, and termination of communication between the TDX module and a TDISP device, with the VMM acting as the orchestrator. The VMM is responsible for issuing the necessary commands to establish and manage the SPDM session and transporting (copying) the messages between the SPDM command buffers and the device's DOE mailbox, while the TDX module handles the actual secure communication with the TDISP device.

2.6.1. Static Device Attestation

During device life cycle management, the VMM issues the `TDH.SPDM.CONNECT` command requesting the TDX module to establish the SPDM session with the device. TDX module collects the SPDM attestation info and returns the VMM SPDM_DEVICE_ATTESTATION_INFO structure. The VMM can then cache this object and deliver it to any TD as part of TDI binding and attestation flow. In order to prove the integrity of VMM delivered SPDM_DEVICE_ATTESTATION_INFO, the TD must calculate the SHA-384 digest of this structure call TDG.TDI.VALIDATE which verifies the TD calculated digest matches with the one stored by the TDX module during the SPDM session setup with the device. [For Arie to consider: While the TDX Module plays a role in enabling static device attestation, attestation functionality remains the responsibility of the tenant [or is it VMM?]]

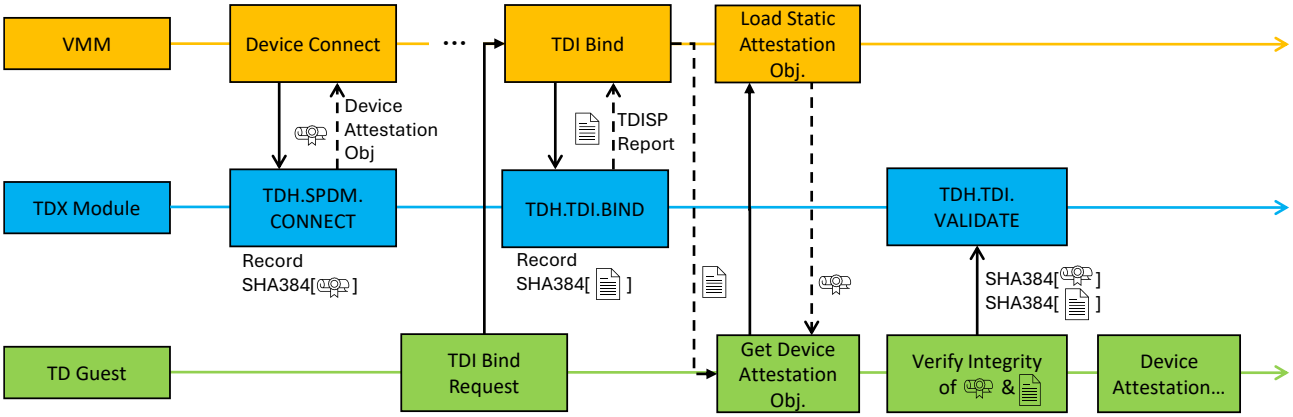


Figure 2.4: Static Device Attestation

2.6.2. Dynamic Device Attestation (Measurement Recollection)

The dynamic attestation model extends the static approach by supporting real-time measurement re-collection. In this model, the TD can request the VMM to provide a fresh SPDM_DEVICE_ATTESTATION_INFO. To prove freshness the TD may generate a nonce and send it to the VMM along with the measurement recollection request. The VMM uses the `TDH.SPDM.MNG(GET_MEASUREMENT)` command to request the device measurement re-collection providing the TD generated none as input. When the device attestation is re-collected with the TD provided nonce, the device DSM signs updated certificates and measurements along with the nonce.

The dynamic attestation could be useful when for example the TD remote verifier needs to ensure TCB recovery or requires to verify specific firmware or components are updated even accepting the TDI into the TD TCB. [For Arie to consider: While the TDX Module plays a role in enabling dynamic device attestation, attestation functionality remains the responsibility of the tenant [or is it VMM?]]

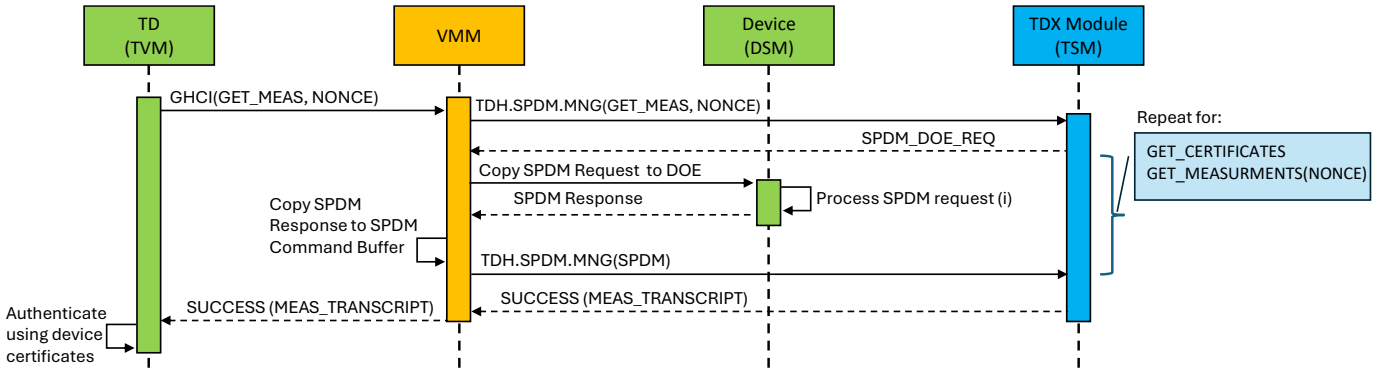


Figure 2.5: Dynamic Attestation Model

2.6.3. Device Attestation Objects

The static device attestation objects, generated by `TDH.SPDM.CONNECT` is called SPDM_DEVICE_ATTESTATION_INFO_T and it contains:

- **DEVICE_INFORMATION:** Includes key device details such as the SPDM session index and interface ID, essential for communication between the TDX module and the device.
- **SPDM_POLICY:** Outlines SPDM session rules, including supported SPDM versions and security capabilities.
- **TDISP_POLICY:** Specifies details related to the TDISP version and capabilities.
- **DEVICE_SPDM_INFORMATION:** Contains session-specific data, such as cryptographic algorithms (e.g., AES-GCM-256), session ID, and heartbeat interval.
- **DEVICE_SPDM_CERTIFICATE:** Stores the SPDM certificate chain for device authentication.
- **DEVICE_SPDM_MEAS_TRANSCRIPT:** Records of the full measurement transcript with the digital signature.

Note that delivering a signed measurement transcript (instead of just the measurement record from the SPDM GET_MEASUREMENTS response) provide several functionality and security advantages:

- **Enhanced Protection:** The measurement record is safeguarded by a digital signature, eliminating the need for other protective mechanisms (e.g., TLS) during transmission between the host and a remote verifier.
- **Device-Specific Data:** The SPDM MEASUREMENTS response includes opaque data that can store extra device-specific information, useful for device-specific verification processes.
- **SPDM Connection Parameter Attestation:** Verifiers can attest to negotiated SPDM connection parameters such as version, capabilities, and algorithms.
- **Resistance to Internal Device Attacks:** A digital signature secures the measurement record against internal device attacks. For instance, a device with a Root of Trust (RoT) may generate measurements, hold private keys, and sign the data. However, if the SPDM session is managed by a higher-level component, that component could forge the measurement record. Digital signature protection limits the Trusted Computing Base (TCB) to the device's RoT, whereas session-based protection extends the TCB to include the higher-level SPDM component.

2.6.4. SPDM Create

For each TDISP device, the host VMM must establish an SPDM session by first calling the `TDH.SPDM.CREATE` command. This command receives a VMM input of an ordered HPA list (of 4KB pages) which the VMM allocates as the TDX module SPDM object context. These pages are used for tracking the SPDM session's AES-GCM-256 cryptographic context and serve as a scratchpad buffer for the TDX module to securely generate, process, and manage SPDM, IDE_KM, and TDISP protocol messages.

When the SPDM-associated IDE stream is disabled and no device TDI is bound to a TD, the host VMM may reclaim the SPDM object pages by using the `TDH.SPDM.DELETE` command.

2.6.5. SPDM Connect and Disconnect

To initiate the SPDM connection with the device, the host VMM calls the `TDH.SPDM.CONNECT` ABI with the `START_SESSION` command. This command generates SPDM requests into the SPDM command buffer and causes `multiple SPDM_DOE_REQ` exits from the TDX module. Upon each such exit, the host VMM retrieves the SPDM request from the command buffer output and copies it into the device's DOE mailbox.

When the device generates a response, the host VMM retrieves the response from the DOE mailbox, copies it into the SPDM command buffer input, and re-invokes the `TDH.SPDM.CONNECT` ABI with the SPDM command to continue the process.

At the end of this sequence, the command returns the `SPDM_DEVICE_ATTESTATION_INFO_T` structure to the host VMM, while the TDX module securely stores its SHA-384 digest in the SPDM object to ensure the integrity of the attestation data which is verified later by the TD calling `TDG.TDI.ACCEPT`

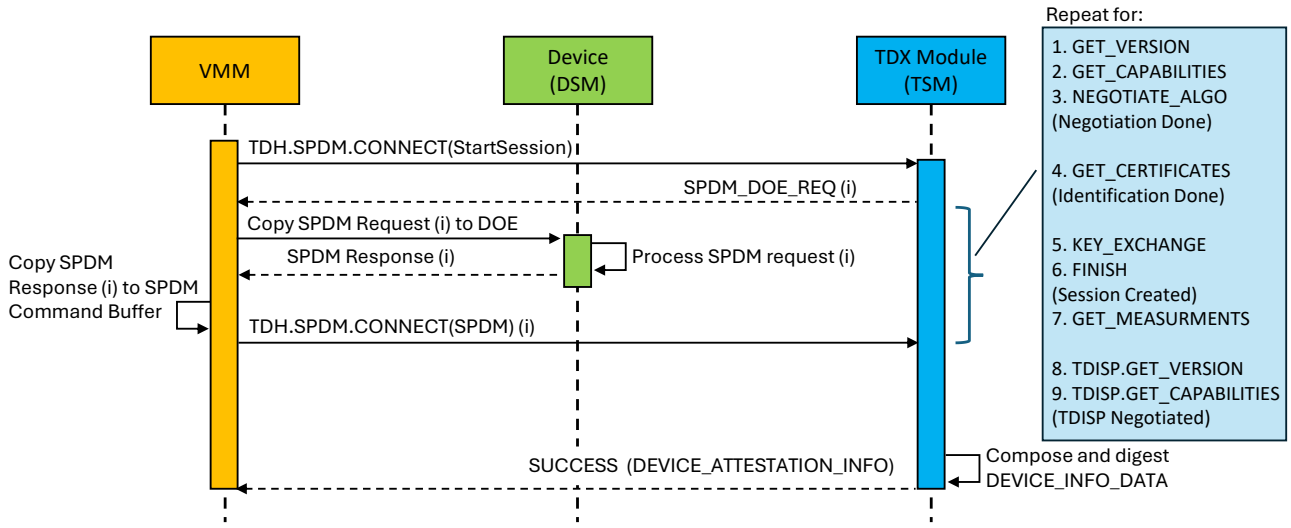


Figure 2.6: TDH.SPDM.CONNECT Sequence Diagram

Once a device is no longer required to maintain the SPDM session—when none of its TDIs are assigned to a TD and its associated IDE streams are disabled—the host VMM may disconnect the SPDM session using the `TDH.SPDM.DISCONNECT` command and reclaim the SPDM object memory resources via `TDH.SPDM.DELETE`.

In the event of a broken SPDM session, the VMM can forcefully remove the SPDM resources without first calling the SPDM disconnect command. There is no security risk in doing so, as the TDX module ensures that the IDE streams are disabled and all associated TDIs are unbound from their respective TDs.

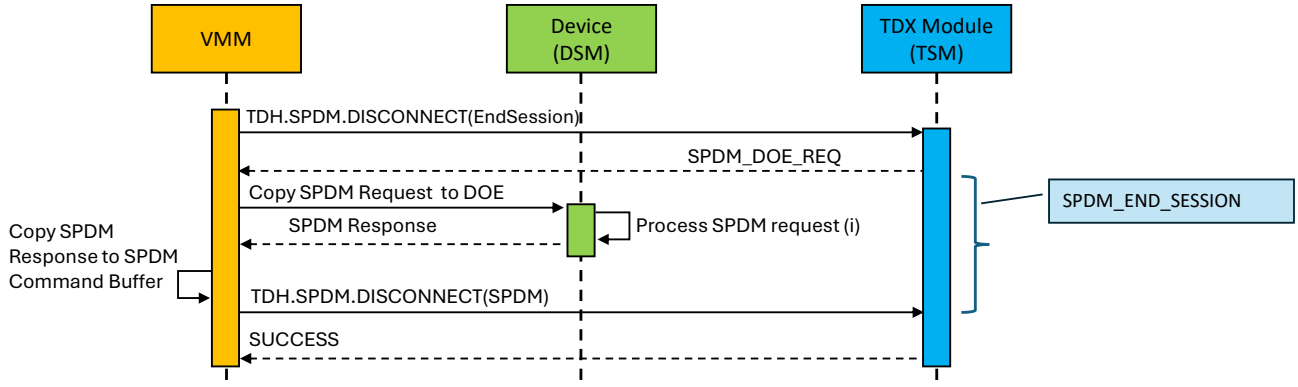


Figure 2.7: TDH.SPDM.DISCONNECT Sequence Diagram

2.6.6. Device Capabilities and TDX Connect Security Considerations

Some device capabilities enumerated via the PCIe configuration space may not be supported or deemed secure by the TDX Connect module, which acts as the TSM. For instance, while the TDX module may allow ATS (Address Translation Services) to be enabled for Integrated TDISP devices, it might restrict such enabling for external devices unless an appropriate ATS access control mechanism is in place.

This per-device capability enabling policy enumeration is communicated to the host VMM as part of the SPDM session establishment. The details are conveyed via the `TDH.SPDM.CONNECT` command. For further information, refer to the TDX Connect ABI FAS.

2.6.7. Other SPDM Operations

During an SPDM session lifetime, TDX Connect supports other management operations through the `TDH.SPDM.MNG` ABI:

- **KEY_REFRESH:** The TDX module monitors the SPDM session's cryptographic algorithms and security policies, and if the SPDM session key shows signs of wear, it will need to be refreshed. When this happens, the TDX module will fail to generate a new SPDM request message and will exit with an `SPDM_KEY_REFRESH_REQ` code. Upon receiving this exit code, the host VMM must call `TDH.SPDM.MNG(KEY_REFRESH)` to begin the SPDM key refresh process.

- **HEART_BEAT:** When the SPDM session is set up with the `TDH.SPDM.CONNECT` command, the TSM and DSM can negotiate the frequency of optional HEART_BEAT messages that must be exchanged to keep the session active. If the HEART_BEAT feature is enabled, the VMM must manage these messages by regularly calling the `TDH.SPDM.MNG(HEART_BEAT)` command, ensuring the SPDM session remains alive.

- **DEV_INFO_RECOLLECT:** At the start of the SPDM session, the `TDH.SPDM.CONNECT` command collects initial device measurements, which are used to create the SPDM_DEVICE_ATTESTATION_INFO_T structure. However, the device might receive updates or new firmware during the session, and the initial measurements may no longer be valid indicators of the device's current security posture. In such cases, the TDX module allows for the recollection of updated measurements. The TD generates its own nonce and send it to the VMM to requests a new TDISP device measurement via the `TDH.SPDM.MNG(DEV_INFO_RECOLLECT)` command. The device signs this new set of measurements along with the TD's nonce to ensure freshness and integrity. The TDX module returns the updated `DEVICE_SPDM_MEAS_TRANSCRIPT` structure to the host VMM, which in turn passes it to the TD guest. The guest can then verify the authenticity of the new measurements by checking the signature against the device's public key, which is stored in its certificate.

2.7. IDE Stream Management

The TDX Connect architecture secures communication with TDISP PCIe devices using the PCI-SIG IDE protocol, which defines two types of streams:

- **Link IDE streams**: Secure communication between two directly connected PCIe ports (e.g., device and switch).
- **Selective IDE streams**: End-to-end secure communication between ports connected through PCIe switches, where the switches are excluded from the trust boundary.

TDX Connect exclusively utilizes Selective IDE streams to secure communication between PCIe device endpoints, ensuring that switches, routers, or any PCIe connectivity fabric devices remain outside the TD trust boundary.

A single TDISP device may host multiple devices, and TDISP defines a model that uses a shared IDE stream to protect all TDI traffic between the device and the CPU. This shared IDE stream, known as the device's default IDE stream, ensures that all traffic with the device's upstream port is secured using that single stream, providing end-to-end protection for the communication.

2.7.1. IDE Stream Create, Block and Delete

The TDH.IDE.STREAM.CREATE command creates the IDE stream object and pre-configures the root port's IDE-ECAP registers using inputs provided by the host VMM. This command initializes the necessary metadata for the IDE stream, which the TDX module uses to track the stream's lifecycle.

The VMM can use this command to create either a Selective or Link IDE stream. While TDX Connect exclusively uses Selective IDE streams, this command also supports the creation of Link IDE streams, giving the VMM the flexibility to manage and configure IDE streams for its own purposes, even if they are not used by TDX Connect.

The command assigns a unique STREAM_ID to the stream and configures it accordingly as either a Link or Selective IDE stream. It also pre-configures the root port's IDE capability registers, including the RID (Requester ID) and address association ranges, based on the inputs provided by the host VMM.

To disable the IDE stream and remove it, the VMM first calls the TDH.IDE.STREAM.BLOCK and then TDH.IDE.STREAM.DELETE command. On IDE Stream deletion, TDX module verifies that no TDI (Trusted Device Interface) associated with the IDE stream is bound to any TD (Trusted Domain) before proceeding.

2.7.2. IDE Key Management

The TDH.IDE.STREAM.KM ABI is used by the VMM to manage key programming for an IDE stream, involving three main commands: SETUP, REFRESH, and STOP. The commands operate using the PCIe IDE_KM protocol over the Secure SPDM session previously established with the device.

The ABI input consists of an SPDM command buffer with host VMM, and each command generates IDE_KM request messages based on the operation type. After each request is generated, the ABI exits with SPDM_DOE_REQ code, prompting the VMM to send the IDE_KM request to the device via its DOE mailbox. The VMM resumes the command with the device's response, and once the entire sequence completes successfully, the operation returns with a success status.

The commands TDH.IDE.STREAM.KM are:

- **SETUP:** Programs the initial Tx/Rx keys for all three sub-streams (completion, non-posted, posted) of the IDE stream. The command generates 256-bit random keys using HMAC & RDSEED, then programs these keys into the relevant root
- **REFRESH**: Run time key refresh by programming the alternate key set Tx/Rx keys with new ones for all three sub-streams along with generation of the IDE_KM requests and authentication of the device responses. Once the new key set is and programmed and all the device IDE_KM response messages are authenticated, the refresh is complete and TDX module triggers the root port to switch into the new key set.
- **STOP:** Deletes the Tx/Rx keys from of a specific IDE stream and key set while generating the corresponding IDE_KM key stop messages to the device and confirming the device response indicating the device is following the protocol.

2.8. TDI Life Cycle Management

TDX Connect Supports TDISP Secure TDI Binding and Unbinding. On a typical TDI device assignment life cycle, the VMM assigns the TDI to the TD as non-confidential (i.e., in the CONFIG_UNLOCKED state), allowing the TD to discover the TDI as TDISP capable.

TDX Connect extends the GHCI with a TD VMCALL, enabling the TD to request a VMM service to bind the TDI as private (i.e., converting the TDI from shared to private).

To bind the TDI to the TD, the VMM must follow these steps:

1. **TDI Assignment:** The VMM creates the TDI context structure using TDH.TDI.MT.ADD and TDH.TDI.CREATE to allocate TDI metadata pages for the TDI's FUNCTION_ID. The VMM also allocates MMIO metadata pages

using TDH.MMIO.MT.ADD to support the TDI's MMIO bars and associates the MMIO pages with the TDI using TDH.MMIO.MT.SET.

2. **TDI Bind:** The VMM requests the TDX module to bind the TDI using the TDH.TDI.BIND command, which generates TDISP protocol messages transitioning the TDI to the CONFIG_LOCKED state and produces the TDI TDISP Report.
3. **Trusted DMA Mapping:** The VMM uses the TDH.DMAR.ADD command to build trusted DMA page tables and map the TDI as "pending" to the TD. The TD must accept the mapping using TDG.DMAR.ACCEPT.
4. **Private MMIO Mapping:** The VMM maps the TDI's private (TEE) MMIO pages in the TD Secure-EPT using TDH.MMIO.MAP. These MMIO pages are mapped as "pending" and become present only when explicitly accepted by the TD using TDG.MMIO.ACCEPT.
5. **TDI Start:** Once the VMM completed with the TDI binding, MMIO and DMA mapping, it resumes the guest with the TDI attestation objects (Device Info Data and TDISP Report).The TD invokes the TDG.TDI.VALIDATE command, providing the calculated SHA-384 digests of the Device Info Data and TDI Report to verify their integrity and then, it performs the device attestation. Following the attestation, the TD accepts both the MMIO and DMA private mappings and finally invokes the TDG.TDI.START command and request the VMM host to transition the TDI into the RUN state. The VMM then invokes the TDH.TDI.START command which generates the TDISP START)INTERFACE_REQUEST message which transition the TDI to RUN state.

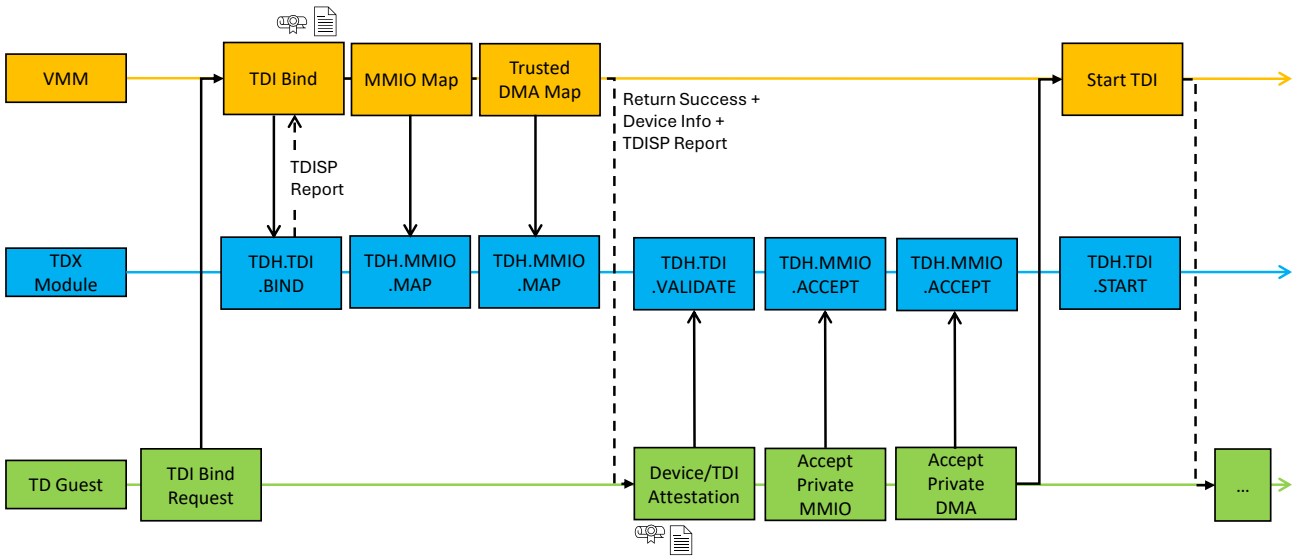


Figure 2.8: TDI Binding Sequence

To unbind the TDI from the TD, the VMM must follow these steps:

1. **TDI Unbind:** The VMM requests the TDX module to unbind the TDI using the TDH.TDI.UNBIND command, which generates TDISP protocol messages transitioning the TDI back to the CONFIG_UNLOCKED state.
2. **MMIO Unmap:** The VMM removes the TDI private MMIO mappings using TDH.MMIO.BLOCK, invalidating all TLBs. It then removes the mappings and frees the MMIO pages using the TDH.MMIO.UNMAP command. Optionally, the VMM may also remove the MMIO metadata tables using TDH.MMIO.MT.REMOVE.
3. **Trusted DMA Unmap:** The VMM blocks the DMA mapping using TDH.DMAR.BLOCK, invalidates the IOTLB, and removes the TDI trusted DMA mapping using TDH.DMAR.REMOVE.
4. **Reclaim the TDI:** Finally, the VMM removes the TDI context object using TDH.TDI.REMOVE. Optionally, the VMM may reclaim the TDI metadata tables using TDH.TDI.MT.REMOVE.

For TD-Initiated unbinding, TDX Connect extends the GHCI with a TD VMCALL, enabling the TD to request the VMM to unbind the TDI from its TCB. The TD can confirm that the TDI was removed from its TCB by reading the TDI binding state using the TDG.TDI.RD(INTERFACE_STATE) command. If the TDI is not bound, this command will return the TDX_TDI_NOT_BOUND error. Otherwise, it will return SUCCESS along with the TDISP TDI state.

2.8.1. TDI Control Structure (TDICS) and Metadata Table (TDIMT)

The TDISP standard defines a unique ID for a TDI called `INTERFACE_ID`. This ID is composed of a `FUNCTION_ID` field, which identifies the function of the device hosting the TDI by its RID (Bus, Device, Function) and, when supported, Segment.

The TDX module enforces the following TDI binding and unbinding properties:

- **Single Assignment:** Ensures that each TDI can be uniquely assigned to a single TD at a time.
- **Trusted DMA Provisioning:** Only the TD owner of the TDI can enable trusted DMA access to its private memory.
- **Trusted MMIO Provisioning:** TDs ensure that TDI TEE MMIO resources are correctly mapped to their Secure-EPT.
- **Secure TDI Unbinding:** The TDX module ensures proper TDI unbinding through the TDISP protocol and properly unmap TEE MMIO and trusted DMA from its TD owner before allowing the TDI to be reassigned to the same or another TD.

To facilitate tracking of these TDI properties, the TDX module manages a per-TDI Control Structure (TDICS), which the VMM creates using the `TDH.TDI.CREATE` command and removes using the `TDH.TDI.REMOVE` command. TDX module enumerates how many pages are needed to allocate per TDICS.

To enforce unique TDI assignment and tracking per `FUNCTION_ID`, the TDX module uses a TDI Mapping Table (TDIMT) to index and track TDICS objects. TDIMT is a fully dynamic structure, consisting of:

- `Non-leaf entries`, which map to a child TDIMT (4KB) page.
- `Leaf entries`, which map to a TDICS page.

The TDIMT hierarchy is defined as follows:

- **Segment Level:** A TDX Connect platform may be configured to support segments. A single Segment Page is needed to support maximum number of 256 segments. When the platform has no Segment support, the first Segment page entry (0) is used to point the Bus page.
- **BUS Level:** Each Segment page entry points to a single 4KB BUS page hosting up to 256 BUS entries, each representing a unique PCIe BUS, and potentially pointing to a single Device Function page.
- **Device-Function Level:** Each Device Function 4KB page has up to 256 TDIMT `leaf entries`, each potentially pointing to a single TDICS structure (of one or more pages).

Before adding a TDICS, the host VMM must ensure that its TDIMT pages for its `FUNCTION_ID` exist or create them using the `TDH.TDI.MT.ADD` command. A TDIMT page can be reclaimed using the `TDH.TDIMT.REMOVE` command when all its entries are free.

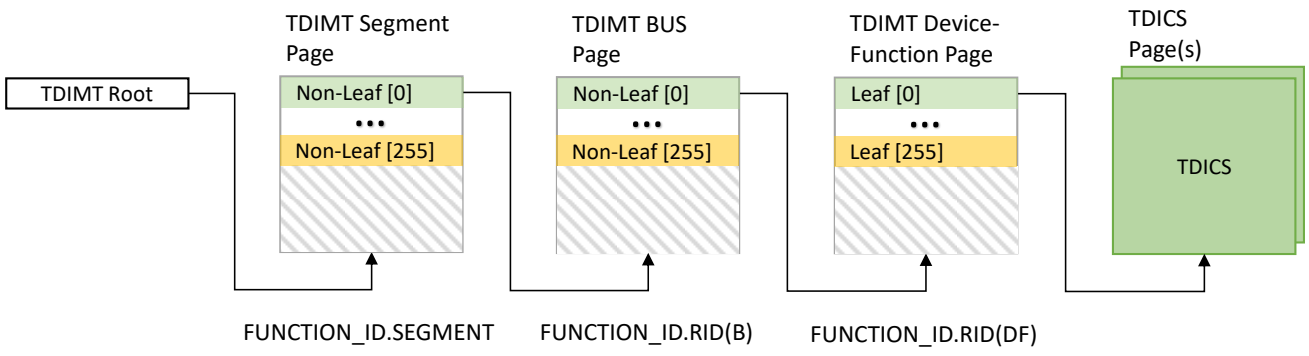


Figure 2.9: TDIMT and TDICS

2.8.2. MMIO Metadata Table (MMIOMT)

To enforce MMIO to TDI assignment and mapping state to TDs, the TDX module uses an MMIO Metadata Table (MMIOMT). The MMIOMT is a fully dynamic structure, consisting of:

- `Non-leaf entries`, which map to a set of 4KB MMIOMT child pages.
- `Leaf entries`, which hold the MMIO metadata with an enumerated entry size.

The MMIOMT hierarchy is defined as follows:

- MMIOMT_L0: A 4KB page set indexed by HPA bits 20:12, holding up to 512 `4KB leaf entries`.
- MMIOMT_L1: A 4KB page set indexed by HPA bits 29:21, holding a combination of up to 512 `2MB leaf entries` or `non-leaf entries` with pointers to an `MMIOMT_L0` page set covering a 2MB range.
- MMIOMT_L2: A 4KB page set indexed by HPA bits 38:30, holding a combination of up to 512 `1GB leaf entries` or `non-leaf entries` with pointers to an MMIOMT_L1 page set covering a 1GB range.
- MMIOMT_L3: A 4KB page set indexed by HPA bits 47:39, holding up to 512 `non-leaf entries` with pointers to an MMIOMT_L2 page set.
- MMIOMT_L4 (Optional): Let `TDX_MAX_PA` be defined as platform MAX_PA minus the number of HKID bits. This level is only required if `TDX_MAX_PA` > 48. It holds up to 512 `non-leaf` entries with pointers to an MMIOMT_L3 page set.
- MMIOMT_ROOT: The TDX module maintains a global MMIOMT root entry, pointing to MMIOMT_L4 if applicable or MMIOMT_L3 otherwise.

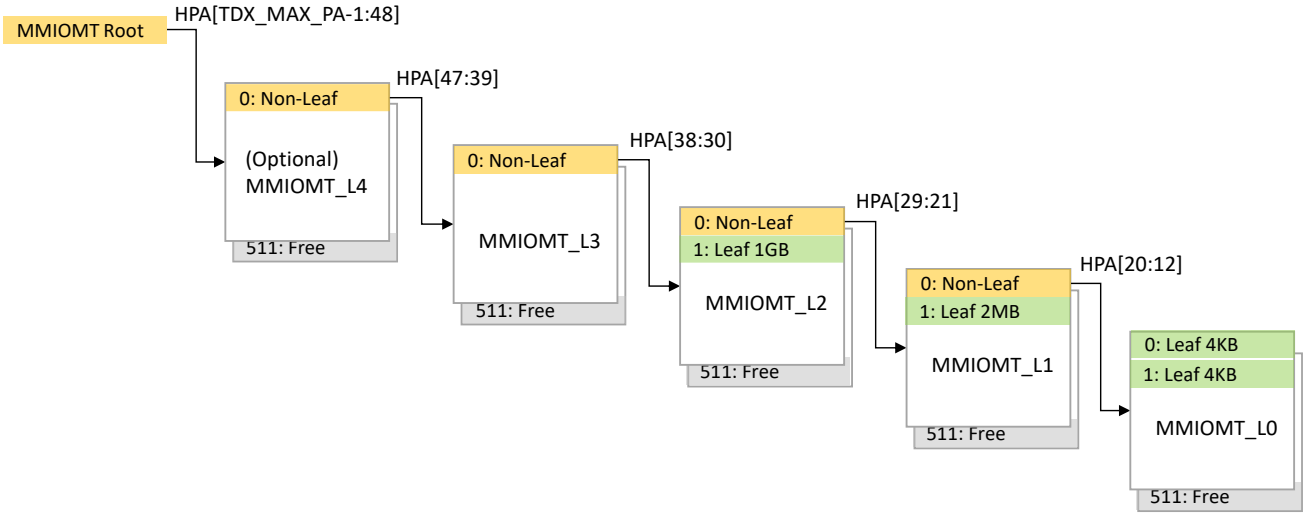


Figure 2.10: MMIOMT

The MMIOMT entries are opaque to the VMM, but the VMM needs to allocate the MMIOMT pages and associate each MMIOMT page to a TDI before MMIO pages can be mapped to a TD.

Typically, the host VMM binds a TDI after its MMIO BARs have been enumerated. At this point, the VMM can allocate the MMIOMT `TDH.MMIO.MT.ADD` command or ensure the MMIOMT pages to cover the TDI BARs already exist.

TDX module enumerates the MMIOMT entry sizes. Depending on the MMIOMT entry size, each MMIOMT level will require one or more pages to host up to 512 `MMIOMT` entries. For example, if the entry size is 16 Bytes, 512 entries require the parent `MMIOMT` entry to point to a set 2×4KB pages. The VMM enumerates the MMIOMT entry size to know how many pages it needs to allocate when building the MMIOMT.

Before mapping the MMIO pages to the TD, the VMM must assign each MMIO page owner to the TDI and determine its size using the `TDH.MMIO.MT.SET` command. The VMM may default to 4KB mapping of the MMIO pages or, to optimize and allocate pages at 2MB granularity, the VMM may inspect the TDISP report and decide based on the MMIO ranges if larger page sizes are possible (Note, TDX Connect does not support MMIO page promote and demote, and this feature may be added later).

The MMIOMT entries are used by the TDX module to enforce secure MMIO management through the following commands:

- `TDH.MMIO.MAP`, `TDH.MMIO.BLOCK`, and `TDH.MMIO.UNMAP`: Track MMIO GPA to HPA mapping. Handle blocking, TLB invalidation, and removal, similar to how PAMT entries are used for tracking TD memory (DRAM) pages.

- TDG.MMIO.ACCEPT: Used by the TD to enforce the secure TDI binding process. The TD uses this command to verify that MMIO pages have been correctly mapped and assigned by the VMM to the TD owner of the TDI, ensuring alignment with the TDISP report.
- TDH.TDI.UNBIND: MMIOMT is used by the TDX module to enforce TDI unbinding is completed and the TDI can be freed once all its MMIO pages have been unmapped, and their MMIOMT ownership has been cleared.

A MMIOMT page set can only be reclaimed using the TDH.MMIOMT.REMOVE command when all its entries are free.

2.8.3. TD Memory Pinning

In a typical non-confidential VM pass-through device assignment, the host VMM pins the guest VM memory mappings to prevent device DMA from being blocked.

The TDX module does not dictate any software pinning solution or design. However, the TDX module enforces that the VMM cannot block DMA while TDIs are assigned to a TD and does not allow binding a TDI to a TD while TD pages that can be accessed by the TD are blocked.

This mechanism is referred to as implicit Secure-EPT pinning and is implicitly enforced by the TDX module when the per-TD TDX_CONNECT_EN flag is set during TD build time.

If TDX_CONNECT_EN is set:

- Attaching a TDI to a TD is allowed only when no TD pages are blocked.
- When any TDI is bound to the TD, TD-mapped pages cannot be blocked (implicitly pinned).

To handle page promotion and demotion, the TDX module introduces a new version of TD page promotion and demotion that does not require blocking (allowing TLB invalidation only for reclaiming Secure-EPT pages during page promotion).

To support page conversion from private to shared, the TDX module introduces a new TDX Connect enlightened guest ABI, TDG.MEM.PAGE.RELEASE, which allows the TD to explicitly release (un-pin) a private page.

2.8.4. I/O TLB Invalidation

IOMMU has context-cache, PASID-cache, IOTLB, and paging structure caches storing IO translations page table entries save DMA remapping table translation overhead for recurrent DMA accesses.

For non-confidential VMs, the VMM software is responsible for the blocking and the invalidation of the relevant IOTLB caches before changing the mapping structures that may have been cached. The invalidation operation is done using the VTd invalidation queue (IQ) interface. The IQ is a software (producer) and hardware (consumer) FIFO. To request IO invalidation, software pushes invalidation requests (descriptors) and updates the IQ tail, and hardware consumes requests from the IQ head.

With TDX Connect, the TDX module is responsible for invalidating the IOTLB caches to ensure that stale entries cannot be used to perform DMA attacks and access trusted TD memory assets once a TDI binding has been removed from the TD or when TD private page permissions are downgraded (e.g., on TD page removal).

This tracking is done by extending the TDX module memory management TLB invalidation and follows the BLOCK, TRACK, INVALIDATE, REMOVE paradigm.

In addition, the removal of a trusted DMA entries and pages also require IOTLB invalidation of their cached information in the IOMMU and to fence potential hardware races of entry removal during an ongoing trusted IOMMU translation (DMA walk).

The TDX module provides two ABI functions for the VMM to handle IO TLB invalidations:

- **TDH.IQ.INV.REQUEST:** Enqueue invalidation request in the IOMMU's invalidation queue.
- **TDH.IQ.INV.PROCESS:** Process completed IOMMU invalidations and update their TDX tracking state.

The VMM calls TDH.IQ.INV.REQUEST indicating the invalidation type (e.g., IOTLB, DMAR entry), IOMMU_ID, RID, PASID, and the "invalidation wait descriptor" allowing the VMM to define if and how to get notified about the invalidation completion.

The TDX module does not have a synchronous mechanism to poll or get notified about invalidation processing by the hardware. The VMM needs to call `TDH.IQ.INV.PROCESS` to request the TDX module to poll the trusted IQ head and update corresponding IO TLB invalidation completion status.

Before the VMM can remove a TD private page (regular or MMIO page) mapping from a TD's Secure-EPT, the TDX module enforces that in addition to core TLB invalidations, also the relevant IOTLB invalidations are completed. This fundamental security requirement ensures that TD private memory cannot be accessed due to stale Core or IOMMU TLB and mapping entry caches.

The TDX Connect extends TLB tracking steps to handle IOTLB tracking as follows:

1. **Block:** When the VMM blocks a page in Secure-EPT, the TDX module logs the current TD-EPOCH in the BEPOCH field of regular pages in the PAMT and for MMIO pages in the MMIOMT.
2. **Track:** `TDH.MEM.TRACK` is extended to update new TDCS records of current and previous EPOCH IOTLB reference counters, which snapshot the total attached IOMMUs to a TD (connected to the TD assigned device interfaces).
3. **Invalidate:** The VMM is required to invalidate the relevant IOTLBs using the `TDH.IQ.INV.*` functions. When the relevant IOTLB is invalidated, the corresponding TD IOTLB reference counter is decremented.
4. **Verify:** TDX APIs that require IOTLB invalidation (e.g., `TDH.MEM.PAGE.REMOVE`) are extended to ensure both core TLB and IOTLB tracking is complete for the target page BEPOCH.

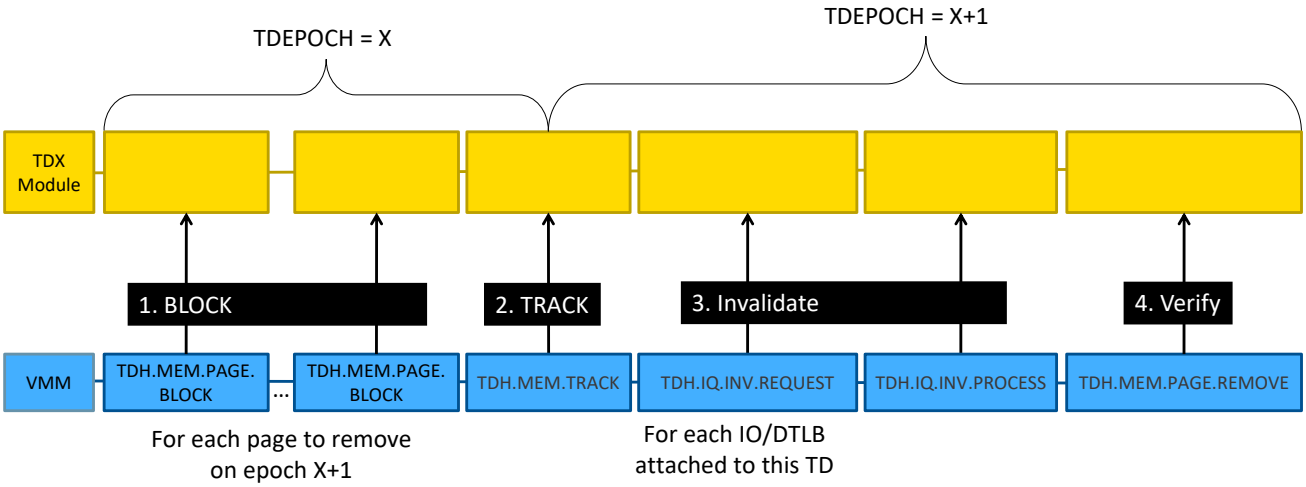


Figure 2.11: TD Memory Management with IOTLB Tracking

Before a PASID (PASIDTE) or a context table entry (CTE) can be removed from the trusted DMA remapping tables, the TDX module verifies that the relevant IOMMU caches were invalidated.

To remove trusted DMA entries and invalidate the corresponding IOMMU caches, the VMM should follow these steps:

1. Block the PASIDTE/CTE using `TDH.DMAR.BLOCK`
2. Request TDX to enqueue the invalidation using `TDH.IQ.INV.REQUEST`
3. When the invalidation is done, invoke `TDH.IQ.INV.PROCESS`
4. Remove the trusted DMA entry mapping using `TDH.DMAR.REMOVE`

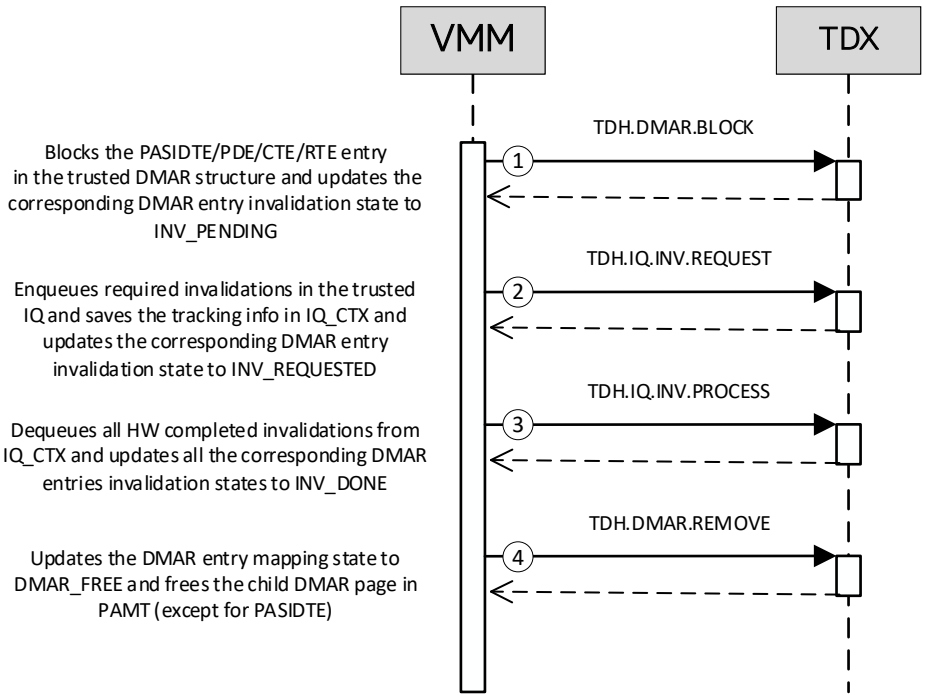


Figure 2.12: Trusted DMA Remove and IOTLB Invalidation

2.9. Interaction with TD Partitioning

To support TD Partitioning, TDX Connect is enhanced to enable the L1 VM context to manage TDX Connect enlightened commands and effectively control the TDI binding within the TD. From the host VMM's perspective, the TDI is bound to the entire TD.

This translates to the following key aspects from the TD guest perspective:

- **Exclusive TD L1 VM Context Control:** The TD L1 VM context is the only context capable of accepting the TDI for itself or for any of its L2 VMs.
- **Private GPA Space Permissions Management:** The TD L1 VM context, responsible for managing private GPA space permissions for any L2 VM context, is extended with TDX Connect interfaces to request and track IOTLB invalidations, ensuring that DMA access permissions align with the L2 VMs' core perspective access permissions.
- **Dynamic TDI Reassignment:** Unlike the host VMM, the L1 VM can remove and reassign a TDI to an L2 VM even while the TDI remains in the RUN state.

The VMM binds the TDI to a TD, mapping the private MMIO and DMA in a pending state. Recall that each VM index has its own Secure-EPT root, but each TDI has only one PASID table entry that can be mapped to a single Secure-EPT at a time.

To enable the L1 VM context to assign the TDI to itself or any L2 VM context (without requiring L2 VM enlightenment), the L1 VM can invoke `TDG.DMAR.ACCEPT`, providing a VM index. A default value of 0 means the TDI is self-assigned to the L1, while any other VM index value corresponds to an L2 VM index. The TDI is then mapped by aliasing the Secure-EPT root of the corresponding L2 VM.

For MMIO mappings, the L2 VM does not need to accept MMIO pages itself. The L1 VM accepts the MMIO pages and then assigns permissions to each L2 VM GPA using the existing TD partitioning `TDG.MEM.ATTR.WR` ABI command.

Note that the L1 VM is responsible for tracking IOTLB invalidations for L2 VMs upon downgrading their private GPA permissions or unbinding a TDI from an L2 VM. Since the L1 VM cannot trust the host VMM to properly invalidate the IOTLB, the TDX module provides a para-virtualized IOTLB invalidation interface for the L1 VM.

To perform this, the L1 VM must follow this sequence:

1. The L1 VM uses `TDG.DMAR.RELEASE` to release the TDI binding from an L2 VM or `TDG.MEM.ATTR.WR` to remove page access permissions.
2. The L1 VM requests IOTLB invalidation using the `TDG.IQ.INV.REQUEST` command, which allows up to 256 virtual invalidation requests, including virtual wait descriptor indicating the TD GPA address for the IQ invalidation completion status write address and value. The TDX module identifies the IOMMUs associated with the TD at the time of the request and exits to the host VMM with a new exit status and an IOMMU bitmap, indicating the TD request and hinting at which IOMMU requires invalidation.
3. The VMM processes the request for each IOMMU using `TDH.IQ.INV.REQUEST`, with special parameters indicating the TD-requested IOTLB invalidation.
4. Once the hardware completes the invalidations, the VMM invokes `TDH.IQ.INV.PROCESS`. The TDX module marks the request as complete only after ensuring IOTLB invalidation is finalized on all IOMMUs identified at request time.
5. Finally, the L1 VM reads which IOTLB invalidation status to check verify its completion (the status write is virtualized by the TDX module on the previous step).

2.9.1. L1 TDI Accept

The VMM binds the TDI to a TD, mapping the private MMIO and DMA in a pending state. Each TD VM context has its own Secure-EPT root, but each TDI has only one PASID table entry that can be mapped to a single VM context (Secure-EPT) at a time.

To assign the TDI to itself or any L2 VM context (without requiring L2 VM enlightenment), the L1 VM invokes `TDG.DMAR.ACCEPT`, providing a VM index. A default value of 0 means the TDI is self-assigned to the L1 VM, while any other VM index value corresponds to an L2 VM index. The TDI is then mapped by aliasing the Secure-EPT root of the corresponding L2 VM.

For MMIO mappings, the L2 VM does not need to accept MMIO pages itself. The L1 VM accepts the MMIO pages and assigns permissions to each L2 VM GPA using the existing TD partitioning `TDG.MEM.ATTR.WR` ABI command.

2.9.2. L1 IOTLB Invalidation

The L1 VM is responsible for tracking IOTLB invalidations for L2 VMs upon downgrading their private GPA permissions or unbinding a TDI from an L2 VM. Since the L1 VM cannot trust the host VMM to properly invalidate the IOTLB, the TDX module provides a para-virtualized IOTLB invalidation interface for the L1 VM. This interface supports Domain-Selective or Page-Selective-Within-Domain descriptors.

The process involves the following steps:

1. **Remove TDI Binding or Adjust GPA Permissions:** The L1 VM uses `TDG.DMAR.RELEASE` to release the TDI DMA mapping converting it back to `PENDING` state enabling its re-assignment to L1 or L2 VM. Alternatively, the L1 VM may change L2 VM page access permissions using `TDG.MEM.ATTR.WR`. In both cases, the L1 VM needs to first block the DMA mapping and then request IOLB invalidation.

2. **Request IOTLB Invalidation:** The L1 VM uses the `TDG.IQ.INV.REQUEST` command to request IOTLB invalidation. This command supports up to 127 invalidation requests per 4KB page, including a Wait Descriptor. The Wait Descriptor allows the L1 VM to specify a Status Address (GPA) and Status Value that will be written upon completion of the invalidation.

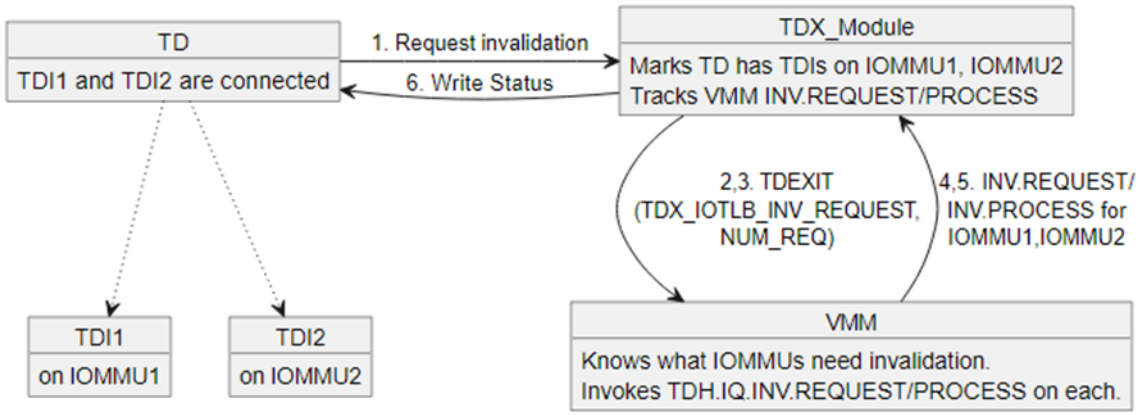
3. **TDX Module virtual IOTLB Invalidation:** The TDX module initializes an internal TD Invalidation Tracker with the IOMMUs associated with the TD based on the TDIs bound at the time of the request. The TDX module exits to the host VMM with `TDX_IOTLB_INV_REQUEST` code, indicating the TD request and providing the relevant IOMMU bitmap and total invalidation request count.

4. **VMM Processes the Invalidation:** For each IOMMU in the bitmap, the VMM processes the invalidation request using `TDH.IQ.INV.REQUEST`, with TD_INV_REQ type and TDR input. The VMM determines schedule and invalidation request capacity to enqueue on each call per each IOMMU.

5. **Invalidation Completion:** The VMM invokes `TDH.IQ.INV.PROCESS` once the hardware completes the invalidations. The TDX module ensures that invalidation is finalized on all IOMMUs before marking the operation complete.

6. **Completion Confirmation:** Upon finalizing all descriptors, the TDX module writes the Wait Descriptor's Status Value to the provided GPA Status Address, signaling to the L1 VM that the invalidation is complete.

7. **L1 VM Verification:** The L1 VM polls the Wait Descriptor's Status Address to confirm the completion of the invalidation request before reassigning the TDI.



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

Figure 2.13: Trusted DMA Remove and IOTLB Invalidation

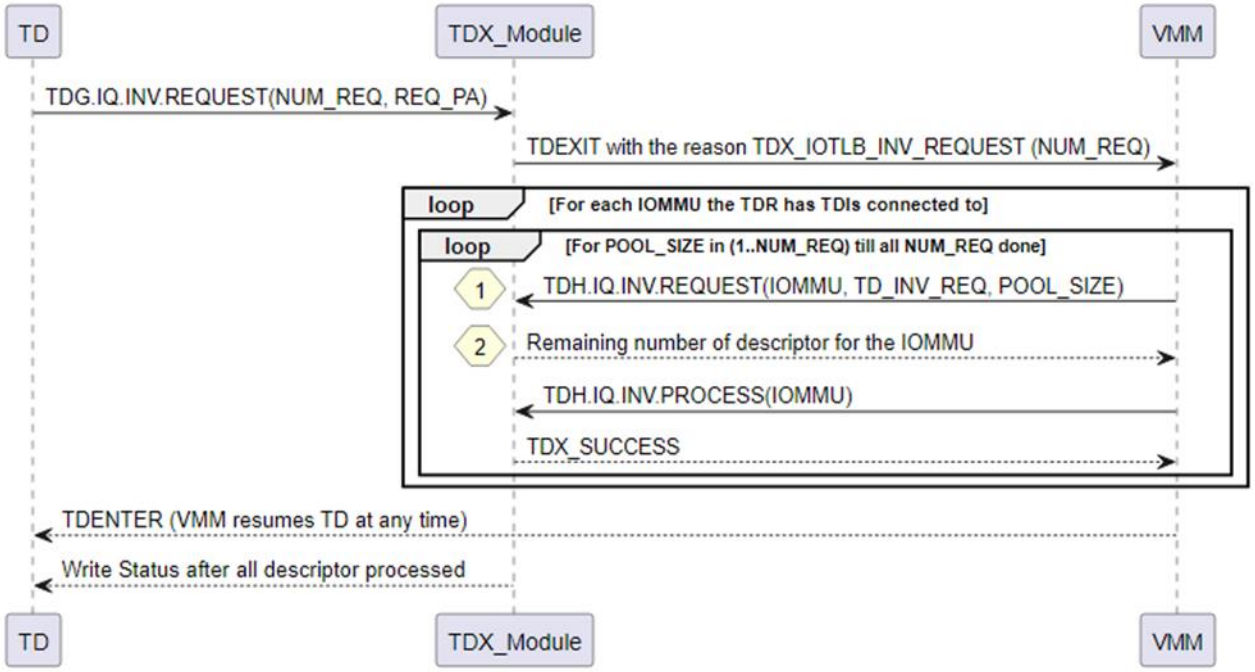


Figure 2.14: Trusted DMA Remove and IOTLB Invalidation

2.10. Interaction with TDX Seamless Updates

The TDX seamless update mechanism is microarchitecturally extended to preserve TDX Connect context and metadata without requiring explicit changes to host VMM software.

The existing requirement for the host VMM software to exit and pause all TDs before and during a TDX module update remains unaltered.

While TDs are paused, devices attached to the TDs may remain in the RUN state, performing private DMA read/write operations to TD private memory. The TD memory remains intact during the update process.

After the TDX module update, when TDs are resumed, the TDs can continue interacting with attached TDIs since the DMA buffers retain confidentiality and integrity, ensuring secure and seamless operation.

3. APPENDIX

3.1. Glossary

Acronym	Full Name	Description
ACS	**Access Control Service**	PCIe ACS defines a set of control points within a PCI Express topology to determine whether a TLP is to be routed normally, blocked, or redirected. ACS is applicable to RCs, Switches, and Multi-function Devices (See [PCIe Base spec])
ATS	**Address Translation Service**	ATS is a PCIe extended capability which uses a request-completion protocol between a Device and a root complex (RC) to provide translation services. In addition, a new AT field is defined within the memory read and memory write TLP to enable the RC to determine whether a given request has been translated or not via the ATS protocol (See [PCIe Base spec])
CPL	**Completion**	Completion PCIe TLP. All read, non-posted write, DMWR, and AtomicOp requests require Completion. Completions include a Completion header that, for some types of Completions, will be followed by some number of DWs of data (See [PCIe Base spec])
DOE	**Data Object Exchange**	Data Object Exchange (DOE) is a PCIe optional mechanism for system firmware/software to perform data object exchanges with a function or RCRB. Software discovers DOE support via the Data Object Exchange (DOE) extended capability structure (See [PCIe Base spec])
DSM	**Device Security Manager**	Device Security Manager (DSM) is a logical entity in the device that may be admitted into the TCB for a TVM by the TSM and enforces security policies on the device (See [TDISP spec])
FLR	**Function Level Reset**	FLR is a PCIe optional mechanism which enables software to quiesce and reset endpoint hardware with function-level granularity (See [PCIe Base spec])
IDE	**Integrity & Data Encryption**	Extended PCIe capability for Integrity & Data Encryption (IDE) for confidentiality, integrity, and replay protection of PCIe transport layer packets (See [PCIe IDE spec])
IDE_KM	**IDE Key Management**	IDE Key Management (IDE_KM) protocol builds upon [SPDM] and [Secured-SPDM], and can be used over multiple transports (See [PCIe IDE spec])
IOMMU	**Input–Output Memory Management Unit**	CPU-RC Input-Output Memory Management Unit (IOMMU) that translates device virtual addresses to physical addresses
KCBAR	**Key Config Bar**	Intel CPU implementation of PCIe IDE key configuration interface (See [Intel RC IDE Programing Guide])
RC	**Root Complex**	PCIe root of an I/O hierarchy that connects the CPU/memory subsystem to the I/O. RC may support one or more PCI Express Ports. Each interface defines a separate hierarchy domain (See [PCIe IDE spec])

Acronym	Full Name	Description
RP	**Root Port**	A PCIe port on a root complex that maps a portion of a hierarchy through an associated virtual PCI-PCI bridge (See [PCIe Base spec])
EP	**End Point**	Function that can be the Requester or Completer of a PCI Express. Endpoints are classified as either legacy, PCI Express, or root complex Integrated Endpoints RCiEPs (See [PCIe Base spec])
SPDM	**Secure Protocol and Data Model**	The Security Protocol and Data Model (SPDM) Specification which defines messages, data objects, and sequences for performing message exchanges between devices over a variety of transport and physical media. (See [SPDM spec])
TCB	**Trusted Computing Base**	A trusted computing base (TCB) is everything in a computing system that provides a secure environment for operations. This includes its hardware, firmware, software, operating system, physical locations, built-in security controls, and prescribed security and safety procedures.
TDI	**TEE Device Interface**	The unit of assignment for an IO-virtualization capable device. For example, a TDI may be an entire device, a non-IOV Function, or a VF (See [TDISP spec])
TEE	**Trusted Execution Environment**	A secure area of a main processor. It helps ensure code and data loaded inside to be protected with respect to confidentiality and integrity.
TEE-IO	**Trusted Execution Environment with IO Devices**	A conceptual framework for establishing and managing Trusted Execution Environments (TEEs) that include a composition of resources from one or more devices (See [TDISP spec])
TSM	**TEE Security Manager**	The TEE security manager (TSM) is a logical entity in a host that is in the TCB for a TVM and enforces security policies on the host (See [TDISP spec])
TVM	**TEE Virtual Machine**	A Trusted Execution Environment Virtual Machine as defined in the TEE Device Interface Security Protocol (TDISP) reference architecture

3.2. External References

Table 3.1: Specification References

#	Document Name	Reference	Version, Date
1	**Intel® TDX Module Base Architecture Specification**	[Intel TDX Base]	April 2025
2	**PCI-SIG TEE Device Interface Security Protocol (TDISP)**	[PCIe TDISP]	5.x, Aug 2022
3	**PCI-SIG Integrity and Data Encryption (IDE) ECN**	[PCIe IDE]	A, Jan 2022
4	**DMTF Security Protocol and Data Model (SPDM) Specification**	[SPDM]	1.4, May 2025
5	**DMTF Secured Messages using SPDM Specification**	[Secure SPDM]	1.2, Jun 2024
6	**PCI-SIG Data Object Exchange (DOE) ECN**	[PCIe DOE]	5.x, March 2020
7	**Intel® Virtualization Technology for Directed I/O Architecture Specification**	[Intel VTd]	4.1, March 2023

#	Document Name	Reference	Version, Date
8	**Intel® Root Complex IDE Key Configuration Unit - Software Programming Guide**	[Intel RC-IDE Guide]	1.03, Feb 2025