Software Enabling for Intel® TDX in Support of TEE-I/O

Revision 1.00

September 2022

Disclaimers

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

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

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

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

No product or component can be absolutely secure.

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

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

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

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

Table of Contents

Overview

The purpose of this white paper is to discuss the software touch points, or new functionalities required in software to enable the Intel TDX technology in support of TEE-I/O, introducing the key features that software developers need to know.

The first generation of virtualization-based Trusted Execution Environment (TEE) VMs (Virtual Machines), such as Intel® Trusted Domain Extensions (TDX) guests do not include devices and accelerators into the trust boundary of TVMs (TEE VMs). Devices are not allowed to read/write TVM confidential memory. Because of the limitations, the current techniques used for TVMs incur significant performance overhead especially for high performance I/O.

The TEE Device Interface Security Protocol (TDISP) defines an architecture for trusted I/O virtualization so that the implementations can address the limitations [PCI-TDISP]. TDISP builds upon the foundation provided by the standard messaging protocols and methodologies, such as CMA (The Component Measurement and Authentication)/SPDM (Security Protocol and Data Model) and IDE (Integrity & Data Encryption); SPDM and IDE are general features for operating systems, and they may already be available there.

To support TEE-I/O, Intel TDX is designed to implement the TDISP architecture with extensions to the current Intel TDX or TDX 1.0 [INTEL-TDX]. In this document, we use Intel TDX to mean Intel TDX that includes such extensions. Software enabling for Intel TDX should take advantage of such a general architecture, defining the common layer and interfaces in support of TDISP support.

We describe a conceptual structure of a VMM (Virtual Machine Monitor) with Intel TDX support as an example, to help clarify the common elements or subsystems required to enable the TDISP architecture and the ones specific to Intel TDX.

The organization of this white paper is as follows:

- Chapter 1 will revisit the existing I/O virtualization technologies and implementations for VMs.
- Chapter 2 will explain the limitations of the current I/O virtualization when used for TVMs.
- Chapter 3 will summarize the TDISP architecture and requirements. And it will introduce the Intel TDX architecture in support of TEE-I/O as Intel's implementation that helps allow direct assignment and establishment of trust between a TD(s) and a TDI(s) hosted by a TDISP-compatible device.
- Chapter 4 will describe high-level software flows for basic operation. And it will show the software touch points, or the new functionality to enable Intel TDX in support of TEE-I/O, describing a conceptual structure of a VMM with Intel TDX support.

1 I/O Virtualization

To implement the capabilities for the direct assignment of devices to VMs, the VMM needs to implement certain functions using hardware features, and we revisit how those are implemented to discuss the requirements for TVMs, which may not include the VMM into in the trust boundary.

I/O virtualization refers to the virtualization and sharing of I/O devices across multiple VMs or container instances. There are multiple existing approaches for I/O virtualization that may be broadly classified as either software-based or hardware-assisted.

I/O virtualization for Existing VMs

With software based I/O virtualization, the VMM exposes a virtual device, such as a Network Interface Controller (NIC), to a VM. A software device model in the VMM or the host OS emulates the behavior of the virtual device. The device model translates from virtual device commands to physical device commands before forwarding to the physical device.

Modern processors provide features to reduce virtualization overhead that may be utilized by VMMs to allow VMs direct access to hardware I/O devices.

Direct access to Hardware I/O Devices

Those features include capabilities for direct memory access (DMA) and interrupt remapping and isolation that can be utilized to minimize the overheads of IO virtualization [INTEL-DIRECTIO].

Specifically, Intel supports the following hardware-assisted I/O virtualization schemes for direct data movement without needing software assistance:

1. Direct Device Assignment – Assignment of entire device to a VM
2. Single Root I/O virtualization (SR-IOV) – Assignment of a device virtualized function.
3. Scalable I/O virtualization (S-IOV) – Assignment of low-level device interfaces composed by the VMM virtualize a device.

Those schemes determine the **device interfaces**, the unit of assignment for an I/O virtualization-capable device.

Device Access via MMIO

When device interfaces are assigned to a VM, the MMIO ranges of the device interface are mapped by the processor-level (memory) virtualization, such as the Extended Page Table (**EPT**) in Intel® Virtualization Technology (VT). When EPT is in use, such MMIO ranges are

treated as guest-physical address (**GPA**), and not used to access the ranges directly. Instead, GPAs are translated by traversing a set of EPT paging structures to produce host-physical addresses (**HPA**s) that are used to access device interfaces. And the EPT paging structures are built by the VMM today for VMs.

Since the VMM itself may not be trusted by TVMs, we also need a trusted entity that utilizes the memory virtualization for direct access to MMIO by TVMs. For example, the VMM would need to use the Intel TDX module API to map such trusted MMIO space to a TVM as private memory in the TVM's Secure-EPT [INTEL-TDX].

DMA Remapping for VMs

To allow a VM to use direct assignment of IO devices, the VMM must also enforce isolation of DMA requests by the VM, and the DMA remapping hardware can be used to restrict DMA from a device interface to the physical memory presently owned by its VM. The address-translation hardware transforms the address in a DMA request issued by a device interface to its corresponding HPA. This functionality is also known as IOMMU (IO Memory Management Unit).

As Chapter 3 in [INTEL-DIRECTIO] describes, the remapping hardware uses the upper portion of the input address (the address in a DMA request issued by a device interface in a VM, for example) to identify a series of paging-structure entries. The last of these entries identifies the physical address of the region to which the input address translates (called the page frame). The format of the paging structures (**VT-d** paging structures, hereafter) is different from the EPT paging structures (above) used by the CPU side, but the format of VT-d paging structures is designed to share the same paging structures with EPT. That allows the VMM to build a single set of paging structures used for consistent address translations between the CPU(s) and I/O device(s) in a VM, and this technique is used by some of the existing VMMs today.

Since the VMM itself may not be trusted by TVMs, we also need a trusted entity that utilizes the DMA remapping hardware. It also needs to isolate and protect the setting of the DMA remapping hardware (IOMMU) when the VMM uses the rest of the DMA remapping hardware for VMs.

2 TVMs and I/O Virtualization

Existing TVMs and I/O Virtualization

This chapter shows the current I/O virtualization for TVMs, and it explains the limitations and the causes of performance overhead.

The first generation of TVMs, such as Intel® TDX guests, do not include (physical) devices and accelerators into the trust computing base of TVMs. Devices are not allowed to read/write TVM confidential memory.

Because of the limitation, today the VMM exposes synthetic device interfaces to TVMs for I/O virtualization. The synthetic device interface is defined to be virtualization-friendly to enable efficient virtualization compared to the overhead associated with I/O emulation.

This, however, requires the TVMs to stage the data that needs to be sent (or "copy-out") or received ("copy-in") from devices in shared memory, which is designed to hold contents accessible to the TVM and the VMM. Further to protect the confidentiality and integrity of such data, the TVM is expected to use cryptographic protections on the I/O data (See Figure 2-1 below).

For some synthetic I/O devices like network and storage, TVM may employ software based cryptographic techniques for data protection. For example, for networking the TVM may use TLS (Transport Layer Security) or such mechanisms to protect the data sent to the NIC (Network Interface Card), and for storage the TVM may employ file system encryption schemes. However, such techniques incur significant performance overhead especially for high performance devices.

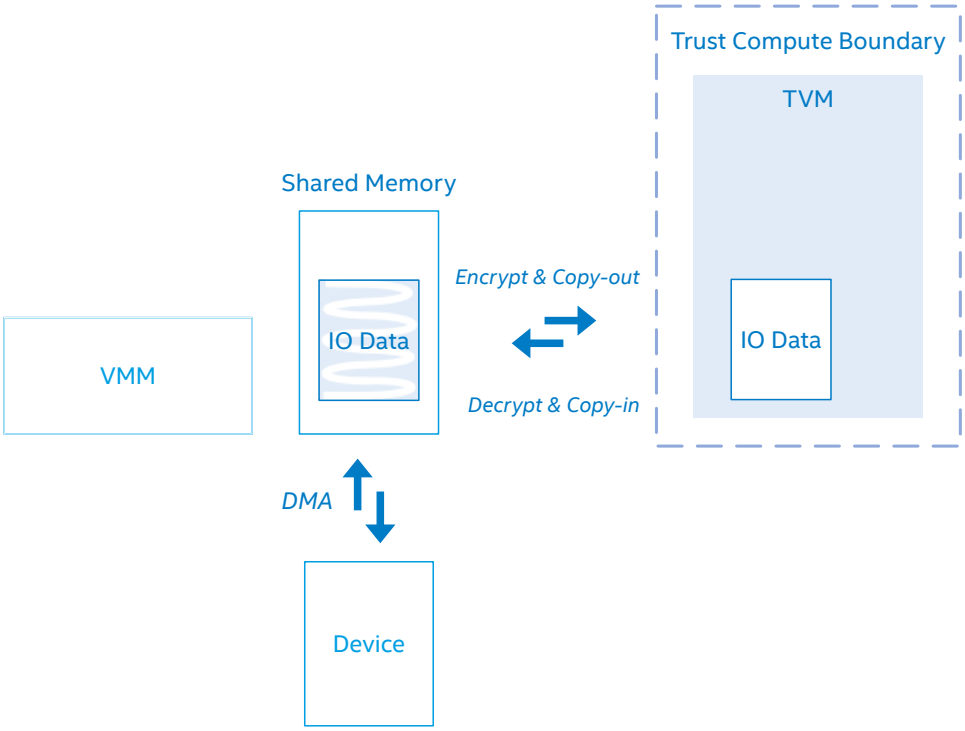

Figure 2-1: Current I/O Handling in TVM

Even if there is a particular device such as a GPU or an FPGA accelerator that the TVM includes into its TCB using its own security policy, the cryptographic data protection does not allow the TVM to offload computation directly to such GPUs or FPGA accelerators.

MKTME and DMA Remapping

For legacy Intel VMX-based VMs where the VMM needs to be included in their TCB, the multi-key, total-memory-encryption (MKTME) technology can be used to enable memory encryption and enhance it by adding support for cryptographic-integrity protection. For directed I/O use cases mentioned in Chapter 1, the VMM needs to set a KeyID[1] as part of the physical addresses in the VT-d paging structures corresponding to the KeyID as part of the physical addresses in EPT employed for the CPU side. This will allow DMAs to be able to access memory without requiring changes to I/O devices and/or I/O drivers in the legacy VM.

For TVMs that may not include the VMM into TCB, the above implementation is not available because the KeyIDs are visible to the VMM, and because they need to be programmed by the VMM.

For Intel TDX the set of KeyID(s) supported by MKTME can be partitioned into two groups of KeyID(s) – a set of private KeyID(s) and a set of shared KeyID(s). Intel TDX uses a set of private

[1] A KeyID identifies a key used by MKTME.

KeyIDs, and private KeyIDs can only be programmed with keys when called from Intel TDX module. This implies that Intel TDX module needs to be extended to help provide secure DMA remapping or IOMMU management functions to the VMM to add or remove mappings from the VT-d paging structures.

In addition, if the Data Links between the TVM and the Trusted Device (Figure 2-2 below) are not in TCB of the TVM because of the intermediate switches that are out of the trust compute boundary, for example, the TVM still needs to use the shared memory combined with cryptographic data protection. Also, such GPUs or FPGA accelerators need to make DMA operations with encryption/decryption sharing the key with the CPU side.

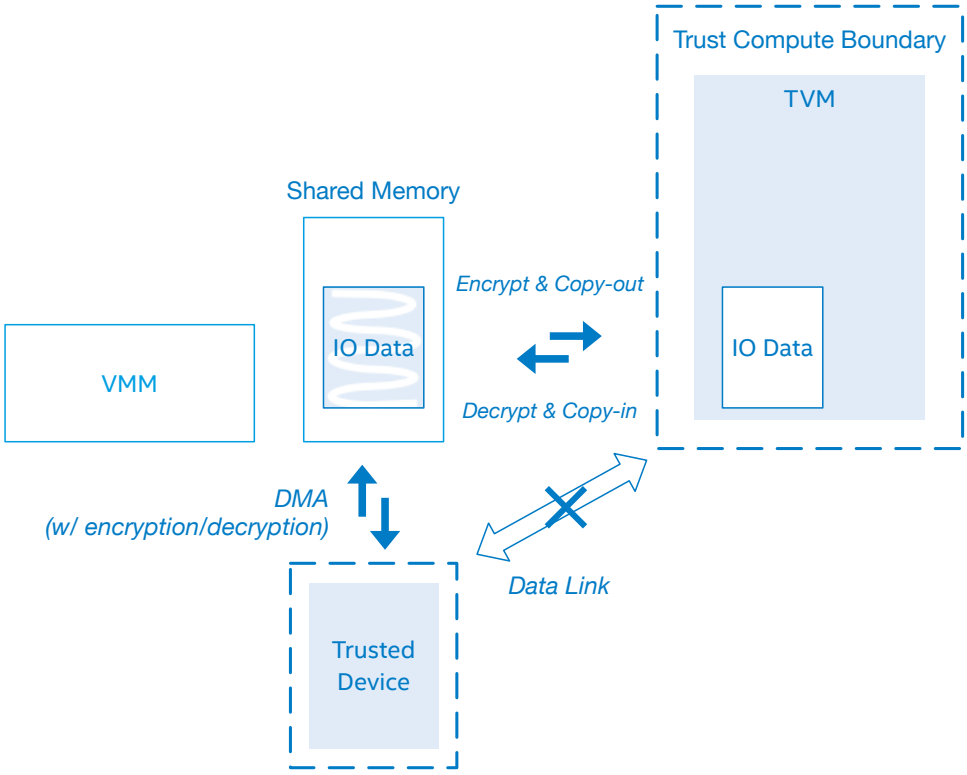

Figure 2-2: GPU or FPGA accelerator with TVM

As we discussed above, we need to extend various areas, including hardware, to enable trusted I/O virtualization. The TEE Device Interface Security Protocol (TDISP) defines an architecture for trusted I/O virtualization, and we will look at an overview of TDISP in the next chapter, introducing the Intel's implementation of the functionality defined by TDSIP. See [TDISP] for details of TDISP.

3 TDISP and Intel TDX

TEE Device Interface Security Protocol (TDISP)

TDISP defines an architecture for trusted I/O virtualization [PCI-TDISP] which includes:

A. Establishing a trust relationship between a TVM and a TDISP-compliant or TEE-I/O device. A TEE-I/O device interface (TDI) is the unit of assignment for an IO-virtualization capable device. For example, a TDI may be an entire (physical) device, a non-IOV Function, or a virtual function (VF).
B. Securing the data-path PCIe interconnect between the host and device. Physical access may be used to tamper with the data transferred between the host and the device. Transfers must be cryptographically protected to provide confidentiality, integrity, and replay protection to TVM data. Such schemes must also guard against violations of producer-consumer ordering.
C. Support TDISP assignment and removal life cycle in a trusted manner.

TDISP builds upon the foundation provided by:

- CMA/SPDM. The SPDM (Security Protocol and Data Model) specification [DMTF-0274] defines standard messages, data objects, and sequences for performing message exchanges between devices. The message exchanges include authentication and provisioning of hardware identities, measurement for firmware identities, session key exchange protocols to enable confidentiality with integrity protected data communication and other related capabilities. The Component Measurement and Authentication (CMA) [CMA] defines optional security features based on the adaptation of the data objects and underlying protocol defined in SPDM. <Examples will be shown later>
- IDE (Integrity & Data Encryption). IDE provides confidentiality, integrity, and replay protection for Transaction Layer Packets (TLPs) [PCIe 6.0]. This provides the implementation for B above.

Figure 3-1shows the TDISP Host/Device Reference Architecture. 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. In other words, the TSM needs to be implemented by a particular technology. When implementing the TSM in the Intel TDX architecture, the Intel TDX module is extended to function as the TSM.

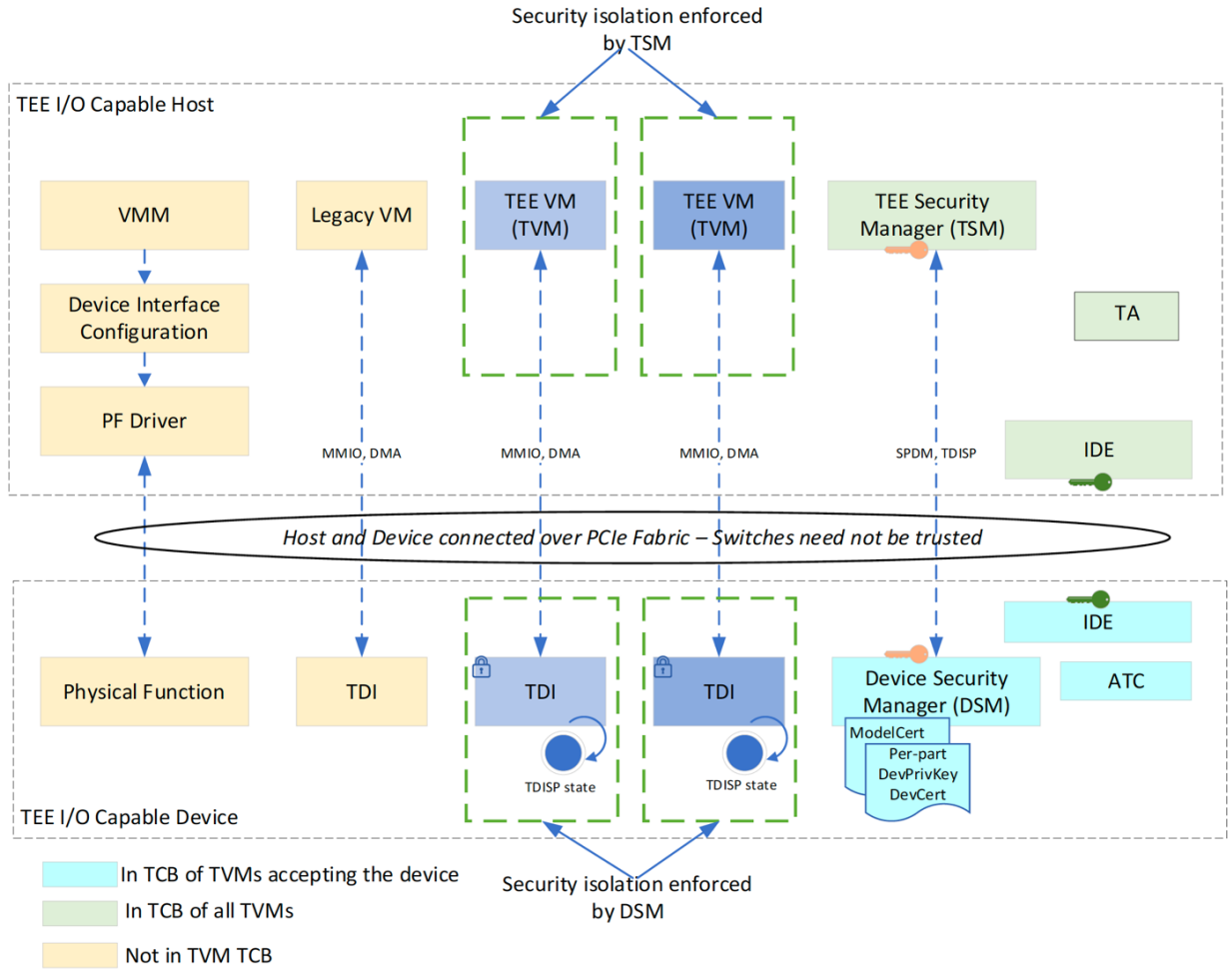

Figure 3-1: TDISP Host/Device Reference Architecture (from [PCI-TDISP])

As we show below, we also use TPA (TDX Provision Agent) to implement part of the TSM. TPA is an architectural TD that extends the Intel TDX module.

The TSM tracks the TDISP state of the TDIs in the figure above, and Figure 3-2 shows the state machine. Some of the transitions are triggered by the TVM that the TDI is assigned to. For example, the TVM requests the TSM to move the TDI to the RUN state by sending a TDISP *START_INTERFACE_REQUEST* **indirectly**; as we explain in Chapter 4, the VMM communicates with the device (DSM) and sends/receives actual TDISP messages, regardless of the originator of the message, getting the secured payload from the Intel TDX module.

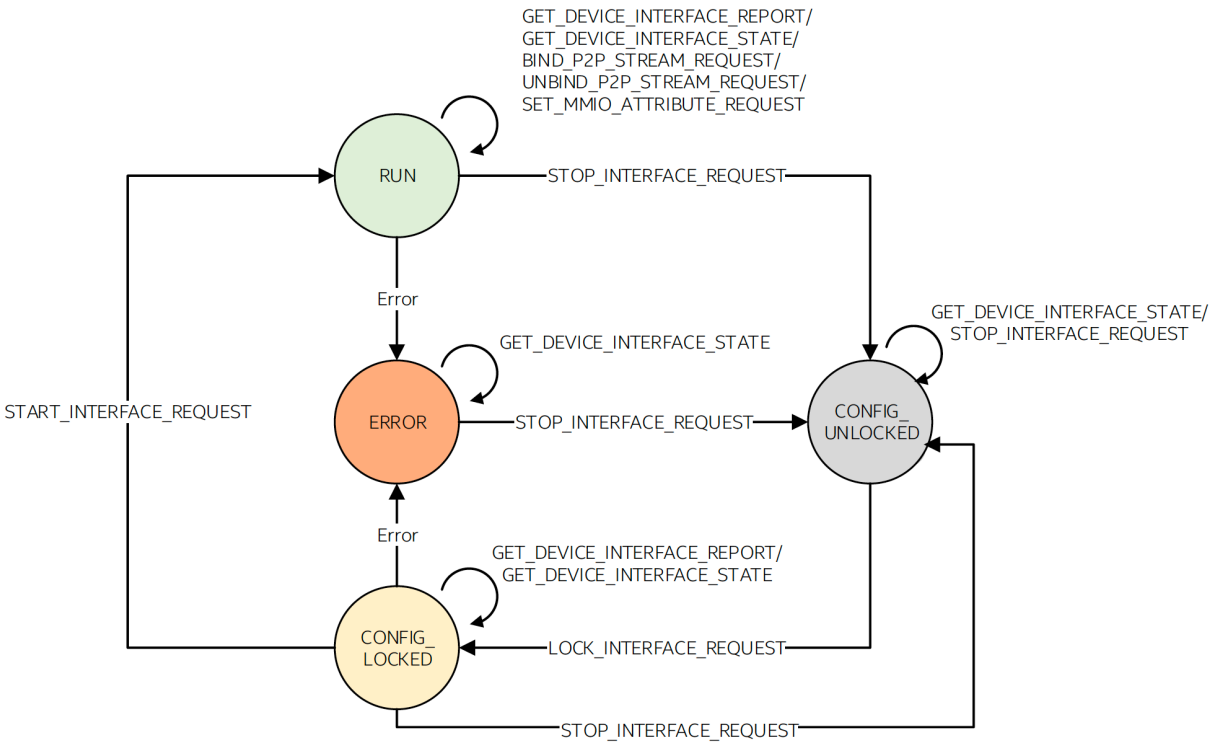

Figure 3-2: TDISP State Machine from [PCI-TDISP]

Since a TVM is implemented as TD (Trusted Domain) virtual machine on Intel TDX, we use "TD" below to indicate as such.

Intel TDX in Support of TEE I/O

Intel TDX is designed to implement the functionality defined by TDISP with extensions to the first generation of Intel TDX or Intel TDX 1.0 [INTEL-TDX], addressing the restrictions discussed in Chapter 2 to include devices and accelerators into TDs. Such devices admitted into the trust boundary of a TD can thus access TD confidential memory and can compute on the data provided by the TDs.

Specifically, the framework of Intel TDX defines the following elements, and the circled numbers in Figure 3-3 correspond to the ones below:

1. **TDISP-compliant (TEE-I/O) device**. As an example, the figure shows that a (physical) TEE-I/O device has two TDIs (TDI1 and TDI2) and Device Security Manager (DSM), which is explained below. It also has a legacy VF (Virtual Function) and physical function device. TDI1 is assigned to TVM1, and TDI2 to TVM2. Multiple TEE-I/O devices may exist in the system, and their configuration can be different.

2. **Integrity and Data Encryption (IDE) on PCIe/CXL Links**. IDE is defined by [PCI-IDE], and is designed to provide confidentiality, integrity, and replay protection for Transaction Layer Packets (TLPs).
3. **Trusted MMIO and DMA access controls**.
4. **Intel TDX module (and TPA) extensions**. The extensions provide the functionality of TEE Security Manager (TSM) defined by TDISP, which is a logical entity in a host that is in the TCB for a TVM and enforces security policies on the host [TDISP]. **TPA** (TDX Provision Agent) is an architectural TD that extends the Intel TDX module which is responsible for the SPDM session establishment between the Intel TDX module and the device.

Note that the TEE-IO Device is moved to the right-hand side from the bottom in Figure 3-3 compared with Figure 3-1, but the Intel TDX architecture is consistent with the TDISP reference architecture.

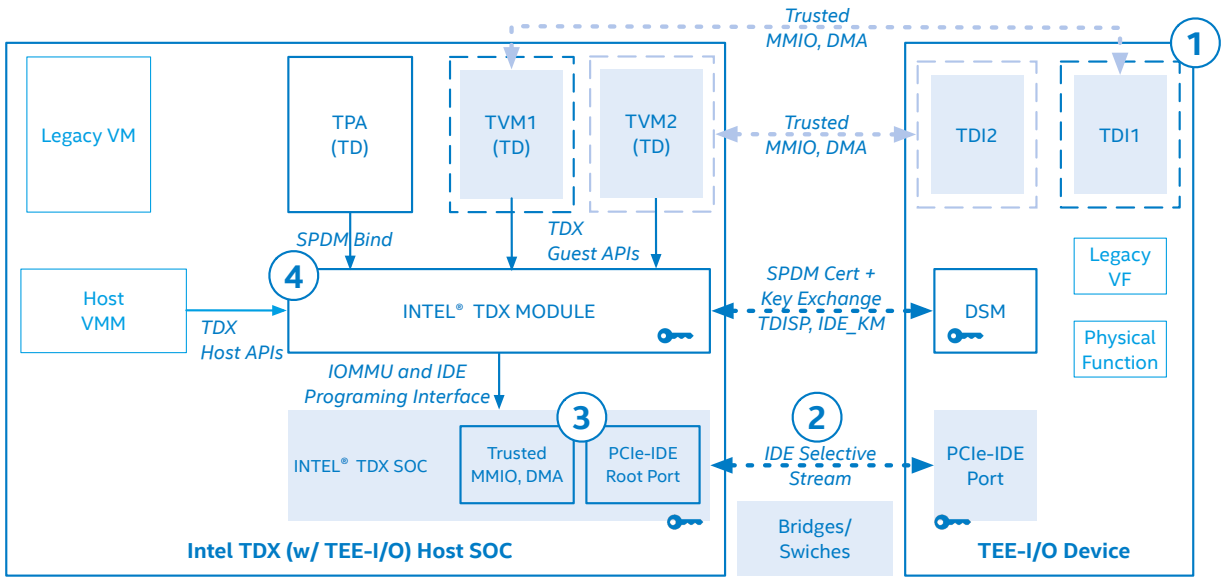

Figure 3-3: Intel TDX (w/ TEE-I/O)

TDISP-Compliant Devices

The Device Security Manager (DSM) is a logical entity, and it provides the following functions:

- Authentication of device identities and measurement reporting.
- Configuring the IDE encryption keys in the device. The Intel TDX module provides the keys for the initial configuration and subsequent key refreshes to the DSM using the IDE key management protocol.

- Locking the TDI configuration, reporting TDI configurations, attaching, and detaching TDIs to TDs.
- Implementing access control and security mechanisms to isolate TD data from entities not in the TCB of the TD.

A TDISP-compliant device must satisfy the TDISP specification for TEE-I/O device security requirements and have a DSM implementation with the following capabilities:

1. Device function 0 must implement a DOE (Data Object Exchange) [PCI-OED] mailbox and support the CMA/SPDM 1.2 protocols for device authentication.
2. At least one Selective IDE Stream and implement IDE key management (IDE_KM) protocol as a Responder.
3. Per TDI implementation of the TDISP state machine management with support for locking the TDI configuration, reporting the TDI configurations and moving the TDI in and out of RUN state.
4. Implementing access control and security mechanisms to isolate TVM provided data from entities not in the TCB of the TD.

The host CPU and the device must use the established Secure SPDM session of 1. As secure transport for passing host requests and device response messages for the protocols implemented for 2 and 3.

IDE on PCIe/CXL Links

As defined by TDISP, Intel TDX requires establishment of a secure connection with a device such that:

- Data that flows between the TD and a trusted device must remain confidentiality, integrity, and replay protected, using the keys derived in above step.
- Metadata associated with transfers (e.g. logical addresses of DMA transfers in the TD's address space) must be integrity and replay protected.

A port-to-port connection established using the mechanisms defined by IDE to secure TLP traffic between the two ports. The connection may be in the form of a **selective IDE stream**, in which case it is possible for IDE TLPs to flow through switches without affecting their security, or in the form of a **link IDE stream**, in which case the two ports must be connected without intervening switches. When there are no switches between the ports, then it is possible to secure all, or only selected, TLP traffic on the Link.

The Intel TDX security architecture excludes switches from the TCB of TD and devices participating in the Intel TDX framework. To that effect, the Intel TDX architecture exclusively uses the **selective IDE streams** to protect the TLPs flowing between the SOC root complex and the Intel TDX capable endpoint devices.

Trusted MMIO and DMA Access Controls

Trusted MMIO access controls

As we discussed in Chapter 2, the VMM needs to use Intel-TDX module APIs to map a TDI trusted MMIO space to a TD as private memory in the TD's Secure-EPT. When the TD accesses the MMIO page using private guest-physical address (GPA), the translated host-physical address (HPA) gets the TD private (Host KeyID) HKID. The base Intel TDX already enforces that no other context (e.g., VMM or other TD) may use the TD HKID.

MMIO pages can only be mapped to TDs if the TD accepted GPA as MMIO page and it is matching the device owner and offset from MMIO base address as indicated by the TD.

According to TDISP spec, TDI enforces that:

1. When a TDI is not in RUN state, trusted MMIO accesses (IDE with TLP T bit set) must be rejected
2. When TDI is in CONFIG_LOCKED, the TDI must capture the IDE stream ID from the *INTERFACE_LOCK_REQUEST* message and use it later to ensure trusted IO transactions only use this IDE stream.
3. When DI is in RUN state, non-trusted MMIO accesses (IDE with TLP T bit clear) or with the wrong IDE stream ID, must be rejected and when TDI is in TDISP RUN state.

Trusted DMA access controls

Intel TDX enforces Secure DMA access controls to help ensure that a TDI may only access TD private memory after:

1. The TD has accepted the device into its TCB
2. The TDI is in TDISP CONFIG_LOCKED or RUN state (TDI is responsible not to access or accept TD memory accesses when it's not in TDISP RUN state)
3. The TD has verified the TDI report and explicitly accepted the TDI MMIO pages and trusted DMA mappings between the TDI and the TD private memory.

Secure IOTLB Invalidation

IOMMU has caches that are used to save DMA remapping table translation overhead for recurrent DMA accesses. On legacy VT-d, for example, the VMM is responsible for the blocking and the invalidation of the relevant IOTLB (I/O Translation Lookaside Buffer) caches before changing or removing the mapping structures that may have been cached.

The Intel TDX module is responsible for invalidating the IOTLB caches to help 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. This is enforced by extending the current Intel TDX TLB tracking with IOTLB tracking, which is done by the Intel TDX module following the TLB tracking sequence: BLOCK, TRACK, INVALIATE, and REMOVE [INTEL-TDX].

4 Software Enabling for Intel TDX

In this chapter, we assume that 1) the VMM already supports the Intel TDX 1.0 specified by [INTEL TDX], and 2) the guest OS also has changes for Intel TDX 1.0 (also known as “TDX enlightenments”) to run as a TD.

First, we describe the high-level software flows for the basic operations to identify the components and to show the interactions among them based on the TDISP architecture with the Intel TDX functionalities. Second, we discuss the software touch points to implement Intel TDX support in the VMM and TDs, taking advantage of the software subsystems required to implement the TDISP framework.

High-Level Software Flows

The following (the step A – D) shows high-level software flows for the basic operations with Intel TDX when a TDI is assigned to a TD. The TDI is owned by a TEE-I/O device below. The steps correspond to the circled A-C in Figure 4-1 and the D in Figure 4-2.

A. **Initialize the host machine for Intel TDX**.

   1. The Intel TDX module extensions for TEE-I/O require a reliable mechanism to enumerate the underlying Intel TDX hardware. The emulation is given to the Intel TDX module as a global “IO System Information Table” which describes the (MCHECK verified) IO configuration information (per socket and IOMMU). See [INTEL-TDX] for MCHECK.
   2. The VMM loads the Intel TDX module (if not already done) and completes initialization of the Intel TDX module, checking for the presence of the TEE-I/O support capability (A1).
   3. It configures the Trusted IOMMU and the PCI express root ports (A2).
   4. It launches the TPA TD (A3).

   This step is global for the system.

B. **Start the SPDM session with the device for secure communication**.

   1. The VMM checks the device capability by reading the device PCI DOE capability and IDE capability (B1). As described in Chapter 3, as one of the requirements for a TDISP-compliant device, the device function 0 must implement a DOE mailbox.
   2. It creates SPDM metadata for the Intel TDX module (B2).
   3. It invokes TPA to start the SPDM session with a physical device, injecting an interrupt event (B3).
   4. TPA uses SPDM to collect the device certificate and measurement and return to the VMM for device attestation later (B4).

5. TPA passes the SPDM session keys to the Intel TDX module. The parameter also includes a hash of the device information, which is used later to ensure that the TD get the unmodified device information from the VMM. The Intel TDX module provides functions that can be invoked only by a TPA TD to bind keys, capability, and device identity into the SPDM session metadata (B5).
6. TPA returns device certificates and measurements to the VMM.

This step is done per device.

C. **Set up the IDE stream for link encryption**.

1. The VMM creates IDE stream by calling the Intel TDX module.
2. It queries and configure the device IDE extended capabilities.
3. It programs keys for all IDE sub streams with both receive and transmit direction. The Intel TDX module provides functions for the VMM to generate IDE_KM messages (*KEY_PROG*) and to process the response. Upon a *KEY_PROG* request, the Intel TDX module generates an ephemeral key and program it to the root-port key programming register.
4. It starts all IDE sub streams with both receive and transmit direction. The Intel TDX module provides functions for the VMM to generate IDE messages (*K_SET_GO*) and to process the response. If the Intel TDX module receives *K_GOSTOP_ACK* response for the *K_SET_GO* request, the Intel TDX-module will trigger IDE stream for the Root Port.

After this step, the device and host SOC have two communication secure sessions. The SPDM session is a software session, which is used for software configuration such as IDE_KM and TDISP. The IDE session is a hardware session, which is used to secure the PCI express TLP.

This step is also per device.

As Figure 3-2 shows the TDISP state machine [PCI-TDISP], and each TDI is associated with that. The goal of Step D is to move the TDI into the TDISP *RUN* state.

In general, the VMM initiates the steps with the TD, and the TD is responsible for checking the information provided or the operations made by the VMM by using the trusted information from the Intel TDX module (including TPA) and the DSM. The TD is also responsible for accepting the configuration upon successful verification to move on to the next step.

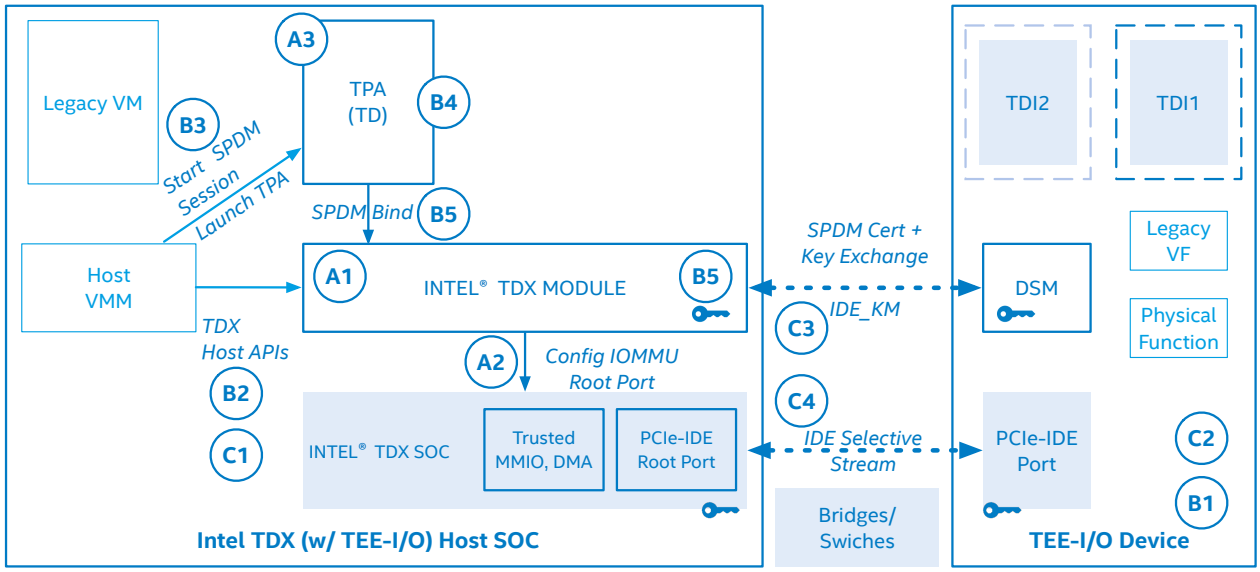

Figure 4-1: Step A-C

TDISP messages are integrity-protected, and the VMM communicates with the device (DSM) and sends/receives actual TDISP messages, regardless of the originator of the message (the VMM or TD). But the VMM needs to call the Intel TDX module to prepare (encrypt and protect) a required TDISP message, and the Intel TDX module returns the secured payload that the VMM can send to the device.

If the TD initiates the steps, the interaction with the VMM needs to use a structure like the following:

1. The TD informs the Intel TDX module of the intent to send a TDISP message in advance. This step is required so that the Intel TDX module can later check that the VMM didn't tamper with the request (step 3 below).
2. The TD requests the VMM to send the TDISP message to, a device (DSM), for example, using a hypercall (TDG.VP.VMCALL).
3. As described above, the VMM needs to call the Intel TDX module because the TDISP message needs a secured payload, which be prepared only by the Intel TDX module. The Intel TDX module checks that the message is what the device intended to send (step 1 above) and returns the secure payload to the VMM.
4. The VMM sends the TDISP message to the device.

For example, the above applies to the Step D-6 below.

D. **Start the TDI**.
   1. The VMM creates a data structure for the device interface and configures the TDI (D1). The state of the TDI is TDISP *CONFIG_UNLOCKED* state. It also prepares the MMIO metadata structure and associates the TDI MMIO pages with the data structure.

2. The VMM calls the Intel TDX module to move it to TDISP *CONFIG_LOCKED* state (D2), by generating a TDISP message – *LOCK_INTERFACE_REQUEST*.
3. The VMM creates a TD and assigns the TDI to the TD, by advertising the TDI in PCIe enumeration.
4. The TD gets the device certificate and measurement from the VMM, then the TD verifies the device based on a TD specific policy. For example, the device certificate must have a trusted root CA (Certification Authorities) and the device measurement must match the latest reference manifest published by the device vendor.
5. If the verification passes, the TD can accept the TDI to the Intel TDX module.
6. The TD uses the TDISP protocol to send TDISP message – *GET_DEVICE_INTERFACE_REPORT* from the DSM. Note that the Intel-TDX IO provides functions that the TD can use to send TDISP messages (D6).
7. The TD accepts the MMIO and DMA mappings in the report by the functions provided by the Intel TDX module.
8. The TD requests the Intel TDX module and VMM to move the TDI into the TDISP *RUN* state (D8). It receives a TDISP message – *START_INTERFACE_REQUEST* from the VMM.
9. The TD pins/unpins private GPA pages as part as DMA allocate/free. Now TD can start the TDI and use trusted DMA to communicate directly with the TDI (D9).

This step is per device interface.

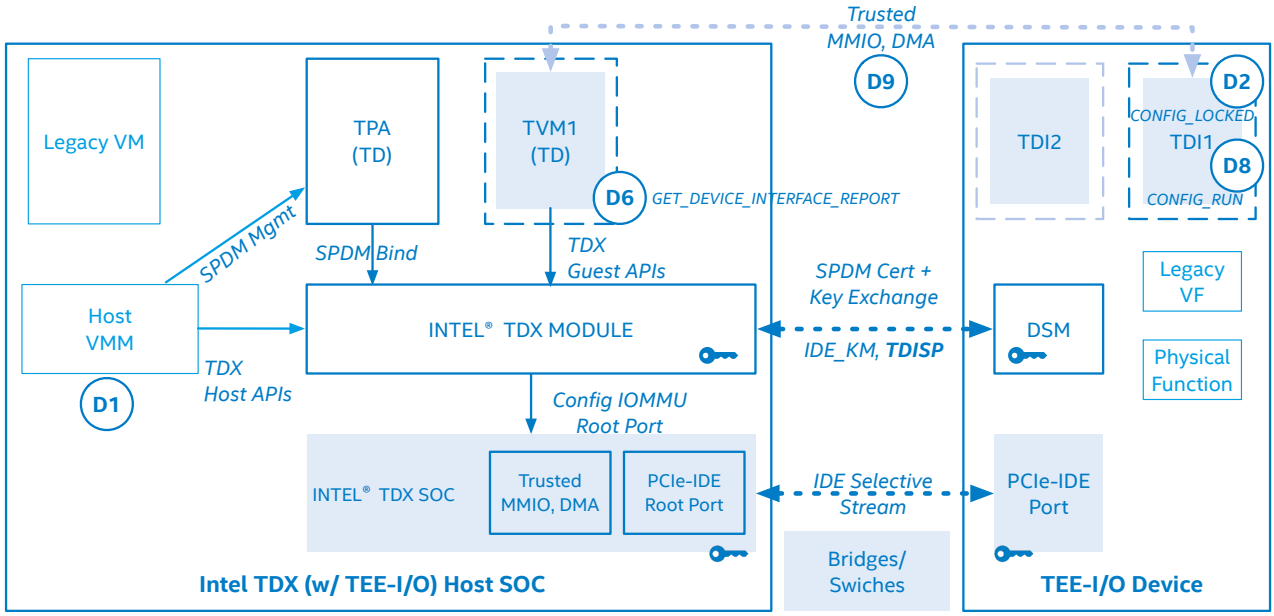

Figure 4-2: Step D

The VMM can repeat same above process to assign another TDI to another TD. Note that one TD may accept multiple TDIs but one TDI must not be assigned to more than one TD.

If a TDI is no longer needed, then the TD or VMM can stop the TDI. If all TDIs in a device are removed, the VMM can remove the IDE stream and terminate the SPDM session for the device. If all devices managed by an IOMMU are disconnected, the VMM can reinitialize the IOMMU.

Conceptual Structure of a VMM with Intel TDX in Support of TEE-I/O

In the following sections, we discuss the software touch points or new functionality to enable Intel TDX in support of TEE-I/O.

As we showed above, software enabling for Intel TDX can take advantage of the standard and general security operating-system (OS) features, which are not necessarily related with virtualization:

- General OS features – PCI-SIG DOE, SPDM, IDE subsystem
- Virtualization features – PCI I/O virtualization subsystem and IOMMU.
- TEE-I/O support – TDISP support (the host and guest side)
- Hardware-based TEE support, such as Intel TDX support

Some of the above may already be present or under development because of the generic features.

The following Figure 4-3 conceptually shows the touch points in the VMM subsystems with a TD, or the code required to be modified/added, and we categorize them into the following:

- Required for TDISP support – general OS functionality. The code may already be present or under development.
- TDISP support – general functionality to support TDISP TEE-I/O. Since Intel TDX is an implementation of the TDISP functionalities, it would depend on the common code.
- Intel TDX support – functionalities that specifically utilize the Intel TDX. They are typically placed under the common or abstracted layer.
- I/O virtualization support – general functionality required for direct assignment of I/O devices to (legacy) VMs. This includes CPU, memory, interrupts virtualization (including Intel TDX support), and the PCI virtual I/O subsystem. Only the related areas are shown in the figure.

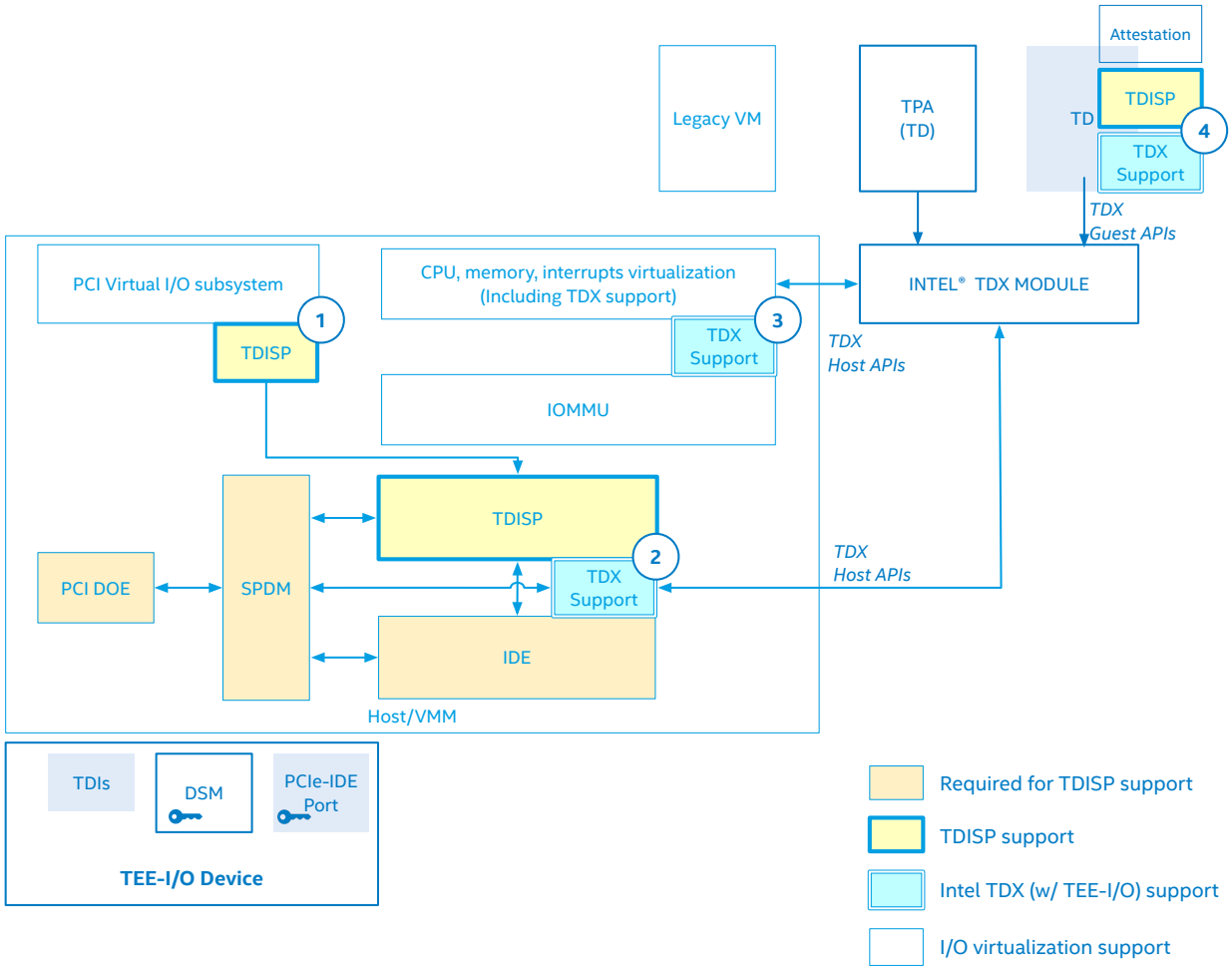

Figure 4-3: Software Touch Points for Enabling Intel TDX

Next, we describe more details of the subsystems and the touch points with the circled numbers in the figure. The changes to TDs (④) will be described later in details.

Host/VMM Changes

As we showed, the VMM is not required to be TCB of TDs, but it still performs resource allocations as it does for the current Intel TDX support. In this paper, we focus on PCIe devices, and TEE-I/O devices are enumerated as PCIe devices.

PCI Virtual I/O Subsystem

Since TEE-I/O devices will have “TEE-I/O Supported” in the Device Capabilities Register, the PCI subsystem should be able to identify the TEE-I/O devices. For such TEE-I/O devices, additional initialization is required by the VMM so that they can be assigned to a TD(s). If a device(s) identified as “TEE-I/O capable”, then the TDSP subsystem should manage the device (①).

TDISP Subsystem

The TDISP subsystem supports TEE-I/O use cases, and implements the following functionalities:

- Initialize the device(s) based on the TDISP requirements, managing the TDISP state machine of TDIs: *CONFIG_UNLOCKED, CONFIG_LOCKED, CONFIG_RUN,* and *CONFIG_ERROR*.
- Request SPDM session. As we see in the TDISP architecture, the SPDM session is required for the secure communication between the TSM (TEE Security Manager, Intel TDX module and TPA in Intel TDX) and DSM (Device Security Manager).
- Request IDE selective streams.
- Bind a TDI(s) to the target TVM.

As described, the requests from the VMM or from the TD(s) are sent to the Intel TDX module first (②) to prepare the secured payload, and then they are handled by SPDM. Such requests include:

- Lock (*TDISP CONFIG_LOCKED*) or stop (*TDISP CONFIG_UNLOCKED*) TDIs
- Get Device Interface Report (TDISP *GET_DEVICE_INTERFACE_REPORT*)

IDE Subsystem

As we showed at the Step C above, the IDE subsystem implements the following functionalities:

- Set up IDE selective streams between two ports.
- Configure the IDE extended capability for the target device by the PCIe driver.

The following are related with Intel TDX support (②):

- Configure the IDE extended capability for Root Port by requesting the Intel TDX module. IDE needs to use the functions provided by the Intel TDX module.
- As we showed above, IDE Key Programming for the Root Port is done by the TDX module (Step C-3), and IDE needs to use the functions provided by the Intel TDX module.
- IDE Key Programming for the target device by Intel TDX module (*IDE_KM_KEY_PROG*). It requests the Intel TDX module to generate IDE_KM requests and sends them via SPDM session/DOE mailbox.

IOMMU Subsystem

This subsystem needs to exist for the existing I/O virtualization, especially to support direct assignment of I/O devices, as we discussed in Chapter 1. Thus, we focus on the additional functionality required for Intel TDX (③). The IOMMU subsystem implements the following functions:

- Configure Trusted IOMMU to Intel TDX mode during initialization (Step A).
- Use the functions provided by the Intel TDX module to access a trusted IOMMU register.

As we discussed in Chapter 1, the format of VT-d (IOMMU) paging structures is designed to share the same paging structures with EPT. This also applies to Intel TDX, that is, the IOMMU paging structures can be shared with Secure-EPT (S-EPT), and this is the implementation used by the Intel TDX module for Intel TDX to add trusted IOMMU functionality effectively and efficiently.

Since the CPU-side already manages S-EPT, one of the options would be:

- The IOMMU subsystem requests the S-EPT management component to build the S-EPT in advance. This is because most I/O devices cannot tolerate a page fault, unlike the CPU-side where the faulted instruction is restarted after handling of S-EPT violation.

MMIO Subsystem

As discussed in Chapter 1, the VMM uses the EPT to map the MMIO ranges for the device interfaces assigned to VMs. Unlike for the mapping for the usual memory, however, the EPT memory type is typically set to UC (uncacheable) for the device interfaces. This also applies to Intel TDX, and the current Intel TDX module doesn't allow the VMM to set UC for the S-EPT memory type. For Intel TDX, the Intel TDX module provides functions for the VMM to map the TDI MMIO pages, and the VMM needs to use the functions (③).

Interrupts Handling

Non-NMI interrupt injection into TDs by the VMM or the IOMMU can be done through the posted-interrupt mechanism [INTEL-TDX].

TVM (TD) Changes

In general, the TVM (TD) is responsible for checking the devices, the SPDM sessions, IDE keys, and TDIs using the trusted information before accepting or using the TDIs.

Enumeration of TDISP Devices

First-level enumeration of TDSP devices can be done by checking "TEE-I/O Supported" in the Device Capabilities Register as the host does. And the PCI subsystem should be able to identify the TEE-I/O devices.

Checking prior to accepting an TDI

A TD must ask the following questions before it accepts an TDI into its TCB:

1. Is the identity of the device and the measurements reported by the device hosting the TDI acceptable?

2. Is there a SPDM secure session established between TSM and the DSM and is the identity authenticated by the Intel TDX-module to set up the SPDM secure session match the identity reported to the TD?
3. Were all IDE keys for the IDE stream used by the TDI established by the Intel TDX-module?
4. Has the VMM configured the TDI and mapped the TDI into the TD address space as expected?

The TD gets the device certificate and measurements from the VMM to determine the answer to question 1. The information should have been reported by the TPA to the VMM (Step B-4). The TD calculates a hash of the data structure, and it requests the Intel TDX-module to verify that the hash matches the one calculated by the TPA (Step B-5).

The TD queries the Intel TDX-module to determine the answers to question 2 and 3. The TD uses the report of the TDI configuration (TDISP specified – *GET_DEVICE_INTERFACE_REPORT*) from the DSM. The TD may then use the report and communicate with the Intel TDX-module to determine the answer to the question 4 (④). If the answer to each of these questions is a yes, then the TD may accept the TDI into its TCB.

DMA Subsystem

Today, DMA buffers are allocated from shared memory in TDs because of the limitation described in Chapter 2, and DMA buffers for TDIs need to be allocated from private memory.

Also, the TD needs to explicitly accept or remove the mapping done by the VMM using the function provided by the Intel TDX module. TDI assignment to TDs requires the VMM to set up the trusted translations in the IOMMU for DMA remapping. And the mapping becomes valid once the TD accepts the mapping.

MMIO Subsystem

Likewise, the TD needs to accept all the device MMIO pages described in all the MMIO_RANGEs in the report from the TDISP message – *GET_DEVICE_INTERFACE_REPORT*.

Attestation

Attestation implementations are out of scope of this white paper.

For device attestation, the white paper “Device Attestation in Confidential Computing Environment” convers the details required for TEE-I/O devices [DEV-ATTEST].

For a TVM (TD), it is also required to verify the device driver for the TDI before using the device interface unless the device driver is already covered by the attestation for the OS (kernel, etc.). The actual change would depend on the current attestation implementation of the TVM.

References

[DMTF-2058] DMTF, DSP2058: Security Protocol and Data Model (SPDM) Architecture White Paper, https://www.dmtf.org/dsp/DSP2058

[DMTF-0274] DMTF, DSP0274: Security Protocol and Data Model (SPDM) Specification, https://www.dmtf.org/dsp/DSP0274

[PCI-PCIE] PCI, PCI Express Base 6.0 Specification, https://members.pcisig.com/wg/PCI-SIG/document/16609

[PCI-IDE] PCI, Integrity & Data Encryption (IDE) – Revision A, https://members.pcisig.com/wg/PCI-SIG/document/16599

[PCI-TDISP] PCI, Trusted Device Interface Security Protocol (TDISP) ECN, https://members.pcisig.com/wg/PCI-SIG/document/16849

[CXL-CXL] CXL, CXL 2.0 Specification, https://www.computeexpresslink.org/

[OCP-SIOV] OCP, Scalable I/O virtualization, https://www.opencompute.org/documents/ocp-scalable-io-virtualization-technical-specification-revision-1-v1-2-pdf

[AMD-SEV] AMD, AMD Secure Encrypted Virtualization (SEV), https://developer.amd.com/sev/

[ARM-RME] ARM, ARM Realm Management Extension (RME), https://developer.arm.com/documentation/den0129

[INTEL-TDX] Intel, Intel Trust Domain Extensions (TDX), https://software.intel.com/content/www/us/en/develop/articles/intel-trust-domain-extensions.html

[RISCV-APTEE] RISCV, RISCV AP TEE, https://github.com/riscv-non-isa/riscv-ap-tee/blob/main/specification/attestation.adoc

[INTEL-DIRECTIO] Intel, Intel Virtualization Technology for Directed I/O Architecture Specification, https://www.intel.com/content/www/us/en/developer/tools/overview.htm

[INTEL-MKTME] Intel, Intel® Architecture Memory Encryption Technologies Specification, https://software.intel.com/sites/default/files/managed/a5/16/Multi-Key-Total-Memory-Encryption-Spec.pdf

[DEV-ATTEST] Device Attestation Model in Confidential Computing Environment, https://software.intel.com/content/www/us/en/develop/articles/intel-trust-domain-extensions.html