Confidential VM Extension I/O (CoVE-IO) for Confidential Computing on RISC-V platforms

Editor - Samuel Ortiz, Jiewen Yao, RISC-V AP-TEE-IO Task Group

Version 0.1, January 15, 2024: This document is in development. Assume everything can change. See http://riscv.org/spec-state for details.

Table of Contents

Preamble . . . 1
Copyright and license information . . . 2
Contributors . . . 3
1. Introduction . . . 4
2. Glossary . . . 6
3. Requirements . . . 8
3.1. Device . . . 8
3.1.1. PCIe or CXL.io . . . 8
TEE Device Interface Protocol (TDISP) . . . 8
Data Object Exchange (DOE) . . . 8
Integrity and Data Exchange (IDE) . . . 9
3.2. Host . . . 9
3.2.1. Ownership . . . 9
3.2.2. Platform Hardware Components . . . 9
PCIe . . . 10
IO Translation Agent . . . 10
Hardware Root-of-Trust . . . 11
CoVE-IO Manifest . . . 11
3.2.3. Software . . . 12
Host . . . 12
TSM . . . 12
3.3. Guest . . . 13
4. Security Model . . . 14
5. Threat Model . . . 16
5.1. Assets . . . 16
5.1.1. Security Objectives . . . 16
5.2. Adversary Model . . . 16
5.3. Threats . . . 16
5.3.1. CoVE-IO-T001 - Trusted MMIO Malicious Access . . . 17
5.3.2. CoVE-IO-T002 - Trusted MMIO Remapping . . . 17
5.3.3. CoVE-IO-T003 - Trusted MMIO PCIe Redirection . . . 17
5.3.4. CoVE-IO-T004 - Trusted MMIO PCIe Pre-Configuration . . . 18
5.3.5. CoVE-IO-T005 - Trusted MMIO Unauthorized Access . . . 18
5.3.6. CoVE-IO-T006 - PCIe Link Man-In-The-Middle . . . 18
5.3.7. CoVE-IO-T007 - PCIe ID Spoofing . . . 18
5.3.8. CoVE-IO-T008 - Confused Deputy DMA Remapping . . . 19
5.3.9. CoVE-IO-T009 - DMA Remapping . . . 19
5.3.10. CoVE-IO-T010 - DMA Remapping . . . 19
5.3.11. CoVE-IO-T011 - TDI Denial of Service . . . 19
5.4. Requirements . . . 20

6. Architectural Overview . . . 21
7. Theory of Operations . . . 23
7.1. Platform Initialization . . . 23
7.1.1. IOMMU Registration and Setup . . . 23
7.1.2. PCIe Root Port Registration . . . 24
Root-of-Trust SPDM Session . . . 24
7.2. SPDM Transport . . . 25
7.2.1. Secure SPDM Session . . . 27
7.3. Device Connection . . . 27
7.3.1. SPDM Session . . . 28
7.3.2. IDE Link . . . 29
7.4. Device Disconnection . . . 32
7.5. Interface Binding . . . 32
7.5.1. Binding Flow . . . 32
7.5.2. TDI Verification and Acceptation . . . 34
7.6. Interface Unbinding . . . 37
7.7. Device and Interface Lifecycle . . . 37
8. Device Attestation . . . 40
8.1. TVM attesting the Device . . . 40
8.1.1. Local Verification . . . 41
8.1.2. Remote Verification . . . 41
8.1.3. Device Runtime Update and Reattestation . . . 41
SPDM Session `TerminationPolicy` . . . 41
TDISP `NO_FW_UPDATE` . . . 42
Post-update reattestation . . . 42
Pre-update attestation . . . 42
8.2. Third party attesting the TVM with the device . . . 43
8.2.1. Local Verification . . . 43
8.2.2. Remote Verification . . . 43
9. Confidential VM Extension (CoVE) IO ABI . . . 44
9.1. CoVE IO Host Extension . . . 44
9.1.1. IOMMU Management . . . 44
Function: CoVE Host Register IOMMU (FID TBD) . . . 44
Function: CoVE Host Notify IOMMU MSI (FID TBD) . . . 44
9.1.2. Root Port Management . . . 44
Function: CoVE Host Register PCIe Root Port (FID TBD) . . . 45
9.1.3. Physical Device Management . . . 45
Function: CoVE Host Connect Device (FID TBD) . . . 45
Function: CoVE Host Disconnect Device (FID TBD) . . . 45
9.1.4. TVM Memory Management . . . 46
Function: CoVE Host Add TVM Interface Region (FID TBD) . . . 46
Function: CoVE Host Reclaim TVM Interface Region (FID TBD) . . . 46
9.1.5. Device Interface Management . . . 47

Function: CoVE Host Bind Interface (FID TBD) . . . 47
Function: CoVE Host Unbind Interface (FID TBD) . . . 47
9.2. CoVE IO Guest Extension . . . 47
9.2.1. Physical Device Query . . . 47
Function: CoVE Guest Get Device Link (FID TBD) . . . 47
Function: CoVE Guest Get Device Certificate (FID TBD) . . . 48
Function: CoVE Guest Get Device Measurements (FID TBD) . . . 48
Function: CoVE Guest Get Device SPDM Attributes (FID TBD) . . . 49
9.2.2. Device Interface Management . . . 49
Function: CoVE Guest Get Interface Report (FID TBD) . . . 50
Function: CoVE Guest Get Interface State (FID TBD) . . . 50
Function: CoVE Guest Map Interface MMIO (FID TBD) . . . 51
Function: CoVE Guest Start Interface (FID TBD) . . . 51
Function: CoVE Guest Stop Interface (FID TBD) . . . 51
References . . . 53

Preamble

This document is in the Development state

Assume everything can change. This draft specification will change before being accepted as standard, so implementations made to this draft specification will likely not conform to the future standard.

Copyright and license information

Contributors

This RISC-V specification has been contributed to directly or indirectly by:

- Samuel Ortiz <sameo@rivosinc.com>
- Jiewen Yao <jiewen.yao@intel.com>
- Ravi Sahita <ravi@rivosinc.com>
- Vedvyas Shanbhogue <ved@rivosinc.com>

Chapter 1. Introduction

The RISC-V Confidential VM Extension [CoVE] specification provides application and virtualized workloads with data confidentiality and integrity, addressing one of the major datacenter security challenges. By building a miminized Trusted Computing Base (TCB), the CoVE interfaces manage to isolate workloads from each others but also from untrusted domain host software components (e.g. the hypervisor). CoVE implementations create a Confidential Computing [CC] framework that allows for mitigating the risks that guest owners are exposed to when running their workloads on shared infrastructures like e.g. public clouds.

The value of such hardware-based Trusted Execution Environments (TEEs) is acknowledged across the industry, and Confidential Computing providers are looking for ways to improve the technology's scalability and performance. With the growing need for securing data processing, expanding Confidential Computing guests TCBs with trusted devices is of particular interest. Data analytics and transformation, artificial intelligence, financial transactions processing are only a few examples of confidential workloads for which a secure and performant I/O architecture is key to their operations. By extending guests' TCBs with trusted accelerators, NICs, or GPUs, such workloads would fully take advantage of their infrastructure provider's capacities while keeping their data protected.

The CoVE interfaces provide TEE Virtual Machines (TVMs) memory with confidential attributes and allow TVM guests to share parts of their address space with an untrusted domain. Although this enables confidential guests to be assigned with devices, directly or not, through para-virtualized I/O implementation, it comes with a significant performance cost. Without additional protection, TVMs have no ways to trust hypervisor-assigned devices and must exclude them from their TCB by not allowing them to directly access confidential memory. Consequently, with the current CoVE specification, data flowing from a device to a TVM must first go through dedicated shared memory regions for the confidential guest to then move it over to its confidential address space. This systematic data copy between shared and confidential memory is called bounce-buffering and can have a major performance impact for confidential workloads.

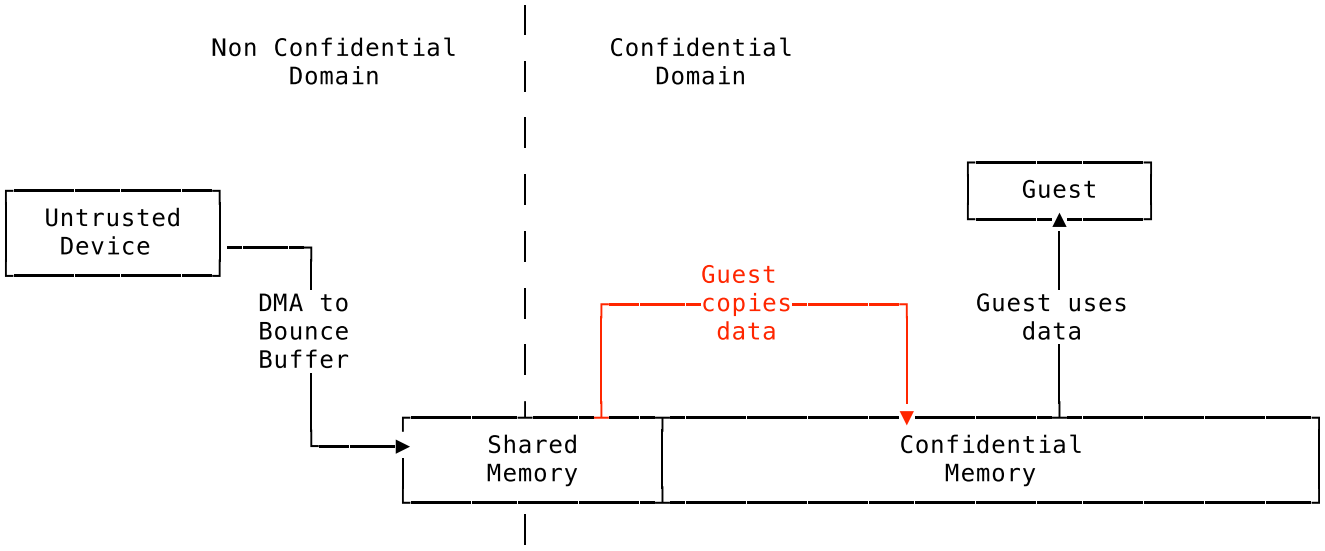

Figure 1. Bounce buffering between an untrusted device and a CoVE TVM

As devices typically expose their programming interfaces through memory mapped registers, using a shared memory buffer requires additional protection of the communication between the device and the TVM (e.g. transport-level data encryption). Such additional security layers can be impractical, intrusive, and may also degrade I/O performance even further.

Ideally, a secure and performant CoVE I/O framework would rely on the ability for hypervisor-assigned devices to directly access TVMs confidential memory, while maintaining the guest confidentality and integrity protection already provided by the CoVE security model. Building such a framework requires enhancing both the host software stack and the assigned devices with new protection mechanisms. In addition to the existing CoVE defined capabilities, the host software must provide ways for TVM guests to establish trust with assigned devices before accepting them into their TCBs and giving them direct access to confidential memory regions. Devices, on the other side of the I/O link, must protect confidential guest workloads and their data from untrusted domain components controlling, accessing, or tampering with them.

This document describes a proposal for extending the CoVE specification with I/O specific flows, interfaces, and intrinsics with the goal of implementing the above-described framework. The CoVE I/O interfaces aim at giving CoVE TVM guests the ability to securily:

- Retrieve a device's identity, configuration, and security state in order for them to establish trust with the device.
- Verify that any untrusted domain component will not be able to intercept, modify, or control data flowing between a guest and its assigned devices.
- Decide to accept devices into their TCB before allowing them to directly access their confidential memory, and before being able to control and configure said devices.

The CoVE I/O framework builds on top of the industry supported and ratified [PCI-SIG] TEE Device Interface Security Protocol [TDISP] specification, which itself relies on the [DMTF] Secure Protocol and Data Model [SPDM] protocol. TDISP compliant devices, also known as TEE-IO devices, implement security protections for isolating guest workloads and confidential data from domains to which a device interface is not assigned to. It also requires TEE-IO devices to provide secure means for confidential guests to attest of any device interface trustworthiness and verify its security configuration.

Chapter 2. Glossary

Term	Acronym	Definition
Application Processor	AP	APs can support commodity operating systems, hypervisors/VMMs, and applications software workloads. The AP subsystem may contain several processing units, on-chip caches, and other controllers for interfacing with memory, accelerators, and other fixed-function logic. Multiple APs may be used within a logical system.
Application Processor- Trusted Execution Environment	AP-TEE	An execution mode that provides HW-isolation for workload assets when in use (user/supervisor code/ data) and provides HW-attestable confidentiality and integrity protection against specific attack vectors per a specified adversary and threat model. The term CoVE, TEE, and hardware-based TEE are also used as synonyms of AP-TEE in this document.
Attestation	N/A	The process by which a relying party can assess the security posture of the confidential workload based on verifying a set of HW-rooted cryptographically-protected evidence.
Confidential Computing	N/A	The protection of data in use by performing computation in a hardware-based TEE.
Confidential VM Extension	CoVE	A set of non-ISA RISC-V extensions that enables confidential computing on RISC-V platforms.
CoVE Guest SBI Extension	COVG	CoVE SBI extensions serviced by the TSM and called by TVMs.
CoVE Host SBI Extension	COVH	CoVE SBI extensions serviced by the TSM and called by host software components.
Device Interface	DI	TDISP term, Device Interface. Same as a TDI. A DI can be a Virtual Function (VF) or a Physical Function (PF).
Device Security Manager	DSM	A DSM is a logical entity on a TEE-IO device that enforces the TDISP security policies, attributes, and states.
Direct Memory Access	DMA	The ability for a device to directly read and write into the host physical memory.
Host platform	N/A	The combination of the host software stack and the host hardware components. The host hardware components include all SoC components (CPU, memory controller, power management IP blocks, etc) and all discrete ones (Root of Trust, IOMMU, physical memory, etc).
Host software	N/A	All software elements including type-1 or type-2 HS-mode VMM and OS; U mode user-space VMM tools; ordinary VMs hosted by the VMM that emulate devices. The hosting platform is typically a multi-tenant platform that hosts multiple mutually distrusting Tenants.
Hypervisor or Virtual Machine Monitor	VMM	HS mode software that manages Virtual Machines by virtualizing hart, guest physical memory and IO resources. This document uses the term VMM and hypervisor interchangeably for this software entity.
Input/Output Memory Management Unit	IOMMU	A system agent that translates device virtual addresses to physical addresses.
Integrity and Data Encryption	IDE	Extended PCIe capability for integrity, confidentiality, and replay protection of PCIe Transport Layer Packets (TLP).
I/O Translation Agent	N/A	Same as an IOMMU.
Platform TCB	TCB	Platform Trusted Computing Base, same as TCB.
Physical Device	N/A	The physical device to which DIs belong to. This is the actual physical PCIe device.
Root of Trust	ROT	Isolated HW/SW subsystem with an immutable ROM firmware and isolated compute and memory elements that form the Trusted Compute Base of a TEE system. The RoT manages cryptographic keys and other security critical functions such as system lifecycle and debug authorization. The RoT provides trusted services to other software on the platform such as verified boot, key provisioning and management, security lifecycle management, sealed storage, device management, cryptography services, attestation, etc. The RoT may be an integrated or discrete element, and may take on the role of a Device Identification Composition Engine (DICE).
Security Protocol and Data Model	SPDM	A DMTF defined specification for exchanging messages between devices over a variety of transports and physical media. In the CoVE-IO context, SPDM is used to exchange TDISP and IDE Key Management messages over PCIe DOE mailboxes.
System TCB	TCB	System Trusted Computing Base, same as TCB.
TEE Device Interface Security Protocol	TDISP	An architecture for trusted I/O virtualization.
TEE I/O Device Interface	TDI	The unit of assignment for a trusted I/O capable device. For example, a TDI can be a Virtual Function (VF) or a Physical Function (PF).
TEE Security Manager	TSM	HS-mode software module that acts as the trusted (in TCB) intermediary between the VMM and the TVM. This module extends the TCB chain on the CoVE platform.
Transaction Layer Packet	TLP	

Term	Acronym	Definition
Trusted Computing Base	TCB	The hardware, software, and firmware elements that are trusted by a relying party to protect the confidentiality and integrity of the relying parties' workload data and execution against a defined adversary model. In a system with separate processing elements within a package on a socket, the TCB boundary is the package. In a multi-socket system the TCB extends across the socket-to-socket interface, and is managed as one system TCB.
Trusted Device Manager	TDM	A CoVE-IO device manager, responsible for verifying, attesting, and accepting CoVE-IO devices into a TVM TCB. This is a TVM guest software stack component.
Trusted Memory Mapped Input Output	Trusted MMIO	A TDI memory mapped I/O region that can only be accessed by a TVM that accepted the TDI in its TCB. TDIs describe their trusted MMIO regions through TDISP. The TVM, with the TSM support, is responsible for verifying that trusted MMIO ranges are correctly mapped into its address space.
TEE VM	TVM	A VM instantiation of an confidential workload.
Virtual Machine	VM	Virtual Machines hosted by a VMM

Chapter 3. Requirements

In order to extend TVM TCBs with external and untrusted devices, both the host platform and the assigned devices must meet a specific set of requirements.

3.1. Device

CoVE-IO compliant implementations support extending TVMs TCBs with devices if and only if they meet the following requirements:

3.1.1. PCIe or CXL.io

A CoVE-IO supported device must be a PCIe or CXL.io compliant device with optional SR-IOV capabilities. When supporting SR-IOV, the device can be split into multiple Virtual Functions (VFs). A TVM-assigned interface can be the whole device, also known as the Physical Device (PF), or any of its VFs. In both cases, this unit of assignment is referred to as a TEE I/O Device Interface (TDI).

A host enumerated TDI must provide the following PCIe capabilities:

TEE Device Interface Protocol (TDISP)

In order to interoperate with a RISC-V CoVE-IO implementation, PCIe devices must support the [TDISP] protocol version 1.0 or above, as defined in the latest TDISP ECN. CoVE-IO devices must support all required TDISP requests and may support the optional ones, as defined by the TDISP request code table.

Moreover, CoVE-IO compatible device firmwares must run a Device Security Manager (DSM) to enforce the TDISP-defined security attributes and policies. The DSM must support both full PCIe devices (PFs) and Virtual Functions (VFs) assignment to TVMs.

Enumeration

TDIs PCIe Device Capabilities Registers must have the "TEE-IO Supported" bit in "Device Capabilities Register" set in order for the host to discover their TDISP capability.

Data Object Exchange (DOE)

The TDI implements the optional PCIe Data Object Exchange mechanism. The minimal supported version is 1.0 defined in [PCIE].

A CoVE-IO supported device must support the following DOE object types:

Vendor ID	Object Type	Description
0001	00	DOE Discovery
0001	01	CMA/SPDM
0001	02	Secure CMA/SPDM

Secure Protocol and Data Model (SPDM)

RISC-V CoVE-IO compliant hosts use the Security Protocol and Data Model ([SPDM]) protocol to exchange TDISP and IDE Key Management (IDE_KM) messages with physical devices, over DOE mailboxes. CoVE-IO also relies on SPDM for gathering devices' certificates and measurements.

As a consequence, a CoVE-IO supported DSM must support the SPDM specification version 1.2 or above. It must also support sending and receiving SPDM Secured Messages as defined in the [SecuredSPDM] specification version 1.1 or above.

CoVE-IO compatible DSMs must support the following SPDM responder capabilities: `CERT_CAP`, `MEAS_CAP` (10b), `ENCRYPT_CAP`, `MAC_CAP`, and `KEY_EX_CAP`.

Moreover, CoVE-IO compatible DSMs must support below algorithms: 1. For `BaseAsymAlgo`, one or more of the following ones: `TPM_ALG_RSASSA_3072`, `TPM_ALG_ECDSA_ECC_NIST_P256`, `TPM_ALG_ECDSA_ECC_NIST_P384`. 2. For `BaseHashAlgo`, one or more of the following ones: `TPM_ALG_SHA_256`, `TPM_ALG_SHA_384`. 3. For `MeasurementHashAlgo`, one or more of the following ones: `TPM_ALG_SHA_256`, `TPM_ALG_SHA_384`. 4. For `DHE Group`, one or more of the following ones: `secp256r1`, `secp384r1`. 5. For `AEAD` Cipher Suite, `AES-256-GCM` with 16 byte MAC.

Integrity and Data Exchange (IDE)

The data that flows between a TDI and a TVM must be confidential and both integrity and replay attack protected. PCIe IDE provides this protection for all Transport Layer Packets (TLPs) moving between those two endpoints. As a consequence, CoVE-IO compatible devices must implement IDE and expose this PCIe capability defined in [PCIE].

Selective Streams

As all PCIe switches are excluded from a TVM's TCB, CoVE-IO compliant hosts' TSMs exclusively establish selective IDE streams between the host PCIe Root Port and TEE-IO devices.

Key Management

To fully support IDE, CoVE-IO compatible devices must implement the IDE Key Management (IDE_KM) protocol. IDE_KM messages are initiated by the TSM and sent over SPDM to the DSM.

CoVE-IO compatible devices must keep track of the IDE key invocation field, which is in bit 63:0 of the IDE sub-stream specific AES-GCM initialization vector (IV). If the IDE key invocation field overflows, the IDE stream must enter Insecure state. Before overflowing, the device may notify the host to let the host software perform the IDE_KM defined key refresh process.

3.2. Host

In order to support TEE-IO devices with the above described capabilities, a CoVE implementation must meet software and hardware requirements, as described in the next sections.

3.2.1. Ownership

In a CoVE-IO context, the non-confidential host software stack is the sole platform resource's owner. In particular, the physical devices to which TDIs belong are owned and managed by the host, typically through the device PF. Host software stack components, e.g. the VMM, assign, unassign, and expose TDIs resp. to, from, and to TVMs.

3.2.2. Platform Hardware Components

PCIe

All CoVE-IO-compliant devices in a CoVE platform must be connected to the CoVE platform through a PCIe Root Port (RP) that is part of a PCIe Root Complex (RC).

A CoVE-IO PCIe link can be direct or go via any topology or PCIe switches and bridges. Since those are excluded from a TVM's TCB, only selective IDE streams must be used between a CoVE-IO device and its corresponding PCIe Root Port.

The TSM establishes one single selective IDE stream for each physical device from which TDIs may be attached to TVMs. All TDIs within a CoVE-IO device share the same IDE stream.

For a given selective IDE stream, the TSM generates, owns, and distributes the IDE keys to both the CoVE-IO device and its upstream PCIe Root Port.

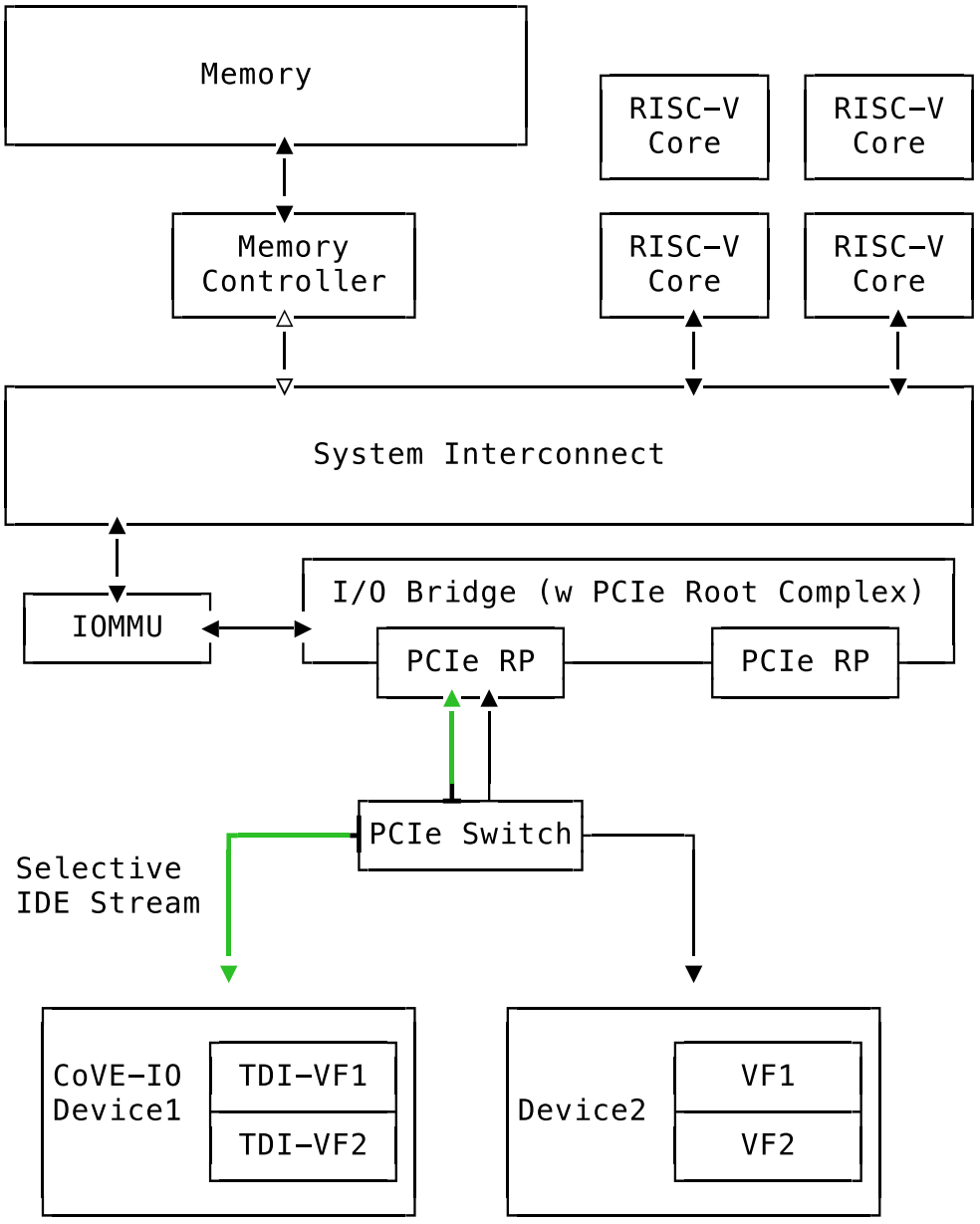

Figure 2. CoVE-IO PCIe Topology

IO Translation Agent

CoVE-IO-compliant platforms must have at least one IO translation agent, also known as an IOMMU.

Platform IOMMUs must follow the RISC-V [IOMMU] architecture specification v1.0.

In order to protect direct access to confidential memory, a CoVE-IO device must be attached to a PCIe Root Port that is bound to a platform IOMMU. All inbound traffic from a TDI must then be translated by the upstream IOMMU.

As the confidential Domain Security Manager (DoSM), the TSM is responsible for setting TDI-specific DMA mapping, MSI page tables, and also for configuring platform IOMMUs own MSIs. As a consequence, all IOMMUs on a CoVE-IO platform must provide a domain isolated Register Programming Interface (RPI) that is exclusively accessible to the TSM.

Hardware Root-of-Trust

As described in Section 3.2.2.1, the TSM generates and sets the IDE keys into both the CoVE-IO PCIe endpoint and its upstream Root Port, for all maintained selective IDE streams.

When setting IDE keys into a CoVE-IO device, the TSM relies on the DSM IDE Key Management (`IDE_KM`) support, and its ability to receive IDE_KM messages over a Secured SPDM session. However, there are no architecturally-defined PCIe protocol for managing Root Port IDE keys.

Instead of adding multiple vendor-specific `IDE_KM` implementations to the TSM, the TSM relies on the platform hardware Root-of-Trust (HROT) to implement the `IDE_KM` protocol and abstract the platform specific PCIe RP implementation away from the TSM. The TSM establishes a Secured SPDM session with the HROT over a host accessible DOE mailbox, and then sets platform RP IDE keys over that session.

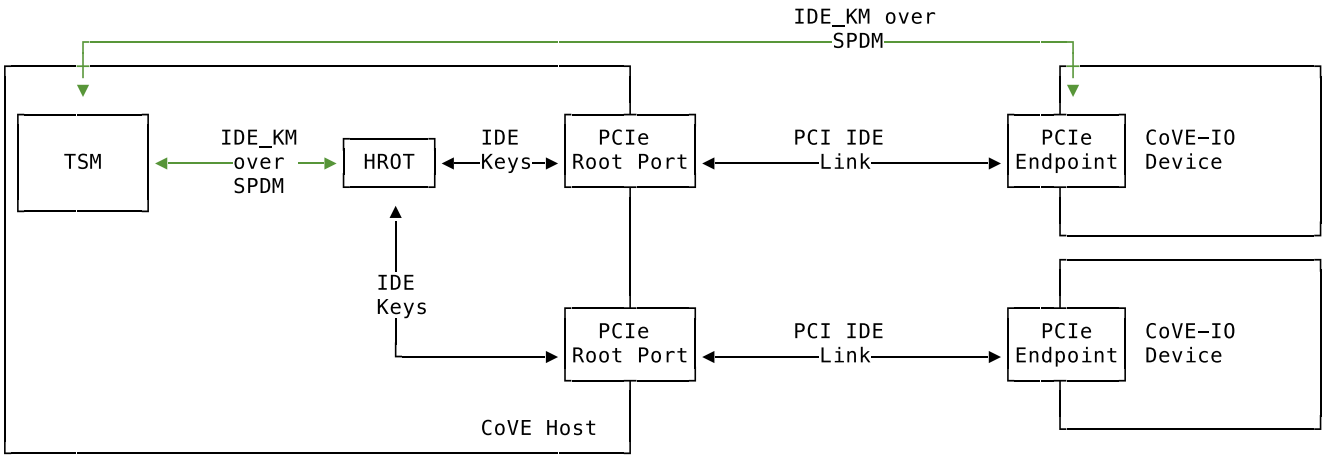

Figure 3. PCIe Root Port IDE Key Management through Hardware Root-of-Trust

As a consequence, a CoVE-IO-compliant platform must have at least one PCIe accessible HROT, with the following requirements:

1. The HROT must support the DOE mechanism
2. The HROT must support Secured SPDM sessions
3. The HROT must support the IDE Key Management protocol

CoVE-IO Manifest

The TSM must be provisioned with a trusted piece of data describing the required CoVE-IO platform components. The hardware Root-of-Trust provides the TSM with a CoVE-IO manifest containing the following pieces of information:

Trust anchor

A list of root certificates that the TSM uses to verify DSM certificates received through SPDM.

IOMMUs

For each IOMMU present in the platform:

- The IOMMU RPI MMIO base address. This is used as the IOMMU identifier.

PCIe Root Ports

For each PCIe Root Ports present in the platform:

- A PCIe Segment:Bus:Device:Function identifier.
- The IOMMU identifier the RP is bound to.
- The list of all MMIO ranges routed throught that RP.
- The RP ECAM base address.
- All downstream PCIe Endpoints linked to that RP, identified by their PCIe RID (i.e. the device PCIe Bus:Device:Function triplet).

TODO: More precise CoVE-IO manifest format.

3.2.3. Software

Host

To support extending TVMs with CoVE-IO devices, the untrusted domain software stack must:

- Implement the [CoVE] Host Extension (`COVH`).
- Support the RISC-V [IOMMU] programming interface with an IOMMU driver.
- Implement the CoVE-IO host ABI, as described in Chapter 8 of this document.

TSM

The trusted Domain Security Manager, i.e. the TSM, is the trusted intermediary between the untrusted domain and the TVM. To allow for securely assigning TDIs into TVMs, it must:

- Support the [CoVE] Host Extension (`COVH`).
- Implement the [CoVE] Guest Extension (`COVH` and `COVG`).
- Support the RISC-V [IOMMU] programming interface with an IOMMU driver.
- Support the CoVE-IO host ABI, as described in Chapter 8 of this document:
  - Implement the SPDM requester protocol and flows.
  - Implement the TDISP requester protocol and flows.
  - Implement the PCIe IDE Key Management protocol.
- Implement the CoVE-IO guest ABI, as described in Chapter 8 of this document.

3.3. Guest

A TVM guest must verify and explictly accept any TDI into their TCBs. The TSM prevents both TDIs from directly accessing the TVM confidential memory and the TVM from doing memory mapped I/O with TDIs, unless the TVM guest accepts the TDI.

By implementing the CoVE-IO guest ABI, the TSM allows for a TVM guest to verify the trustworthiness of an assigned TDI. The TVM also uses the same ABI to notify the TSM about its TDI acceptance decision.

The TDI verification process from the TVM guest not only requires support from the TSM through the CoVE-IO guest ABI but may also include running local or remote attestation of the physical device the assigned TDI belongs to. In order to minimize the TVM guest software stack changes needed to support the CoVE-IO TDI verification, attestation, and acceptance flows, the CoVE-IO guest must run a Trusted Device Manager (TDM) as a separate TVM guest process. Although the TDM can be architectured in a TEE-agnostic fashion, it must support the CoVE-IO guest ABI.

Chapter 4. Security Model

The CoVE-IO securiy model is built on the following assumptions:

- The host platform physical devices are owned by untrusted domain software components (e.g. the host VMM or the hypervisor) that are not part of any TVM TCB and are thus untrusted by TVMs.
- Only the TVM owner can assess of a TDI trustworthiness. Based on that assessment, it explicitly accepts or rejects a TDI into its TCB.
- A TDI may access a TVM confidential memory through DMA only when all the following conditions are met:
  - The TVM owner has explicitly allowed the TDI to access its confidential memory by accepting it.
  - The TDI is exclusively assigned to the TVM, i.e. it must not be shared other TVMs or any host software component.
- A TVM may access a TDI trusted MMIO space only when all the following conditions are met:
  - The TVM owner has explicitly accepted the TDI.
  - The TDI is exclusively assigned to the TVM, i.e. it must not be shared other TVMs or any host software component.
- Until a TDI is accepted by the TVM:
  - The TDI is not allowed to DMA into the TVM confidential memory.
  - Trusted MMIO access to the TDI is blocked by TSM.
  - Untrusted MMIO access to the TDI may still be allowed by the VMM for ordinary VMs (if and only if the TDI is in the TDISP unlocked state).

By means of the CoVE-IO guest ABI, TVMs are required to explicitly accept TDIs into their TCBs. Accepting or rejecting a TDI is a TVM specific decision, based on TVM specific set of verification policies and criteria. A TVM accepting a TDI does not imply that other TVMs on the same host platform would accept it as well.

Shareable CoVE-IO-compliant devices may expose multiple TDIs, assigned to different TVMs. It is the DSM responsibility to guarantee isolation between all assigned TDIs, on a per-TVM basis.

Each TDI must be exclusively assigned to no more than one TVM. However, a single TVM can simultaneously be assigned several TDIs, irrespective of whether they originate from the same physical device or not.

As a TVM TCB does not include PCIe switches and bridges, a selective PCIe IDE stream must be setup to guarantee end-to-end confidentiality and integrity protection between a TVM and a TDI. The host VMM is responsible for reserving a single selective IDE stream per physical device from which one or more TDIs are assigned to TVMs. That single PCIe IDE stream is then shared by all TDIs originating from the same corresponding physical device. Since the host VMM is not in the TCB, its role in the stream setup is limited to selecting an available stream ID and configuring the device IDE capability. The selective IDE stream keys, for both link endpoints, are managed and configured by TCB elements (the TSM, DSM, and platform RoT).

The main security objective of the CoVE-IO security model is to protect a TVM's confidential data integrity and confidentiality while TDIs are assigned to it. At the same time, availability of those

assigned TDIs is out of this model scope as e.g. the host VMM could remove them from the TVM at any time it sees fit. Either the DSM or the TDI itself must clear and wipe all TVM confidential data in the TDI before the host software stack can fully reclaim an assigned TDI.

Chapter 5. Threat Model

5.1. Assets

The CoVE-IO security model aims at protecting the following assets:

1. The TVM confidential data, which includes its confidential main memory, code and execution state. A TVM execution state includes both its CPUs micro-architectural states and its assigned TDIs states.
2. An assigned TDI trusted I/O space that is mapped into the TVM address space, also known as a TDI trusted MMIO.

5.1.1. Security Objectives

The CoVE-IO security model objectives is to protect the above-described TVM assets **confidentiality** and **integrity** from components outside of the TCB.

Availability of these assets is out of the CoVE-IO security model scope.

5.2. Adversary Model

The CoVE-IO security model aims at protecting the above-described TVM assets from the following adversaries:

- *Privileged host software adversary*: This includes host software components executing in S and M mode like the host firmware, kernel, hypervisor, VMM, etc. As the system resource owner but also the TVMs lifecycle manager, those components can access and control all devices on the system.
- *Unprivileged host software adversary*: This includes host software components executing in U mode like e.g. the userspace parts of the host VMM.
- *Privileged guest software adversary*: This includes guest software components executing in VS mode like e.g. the guest kernel.
- *Unprivileged guest software adversary*: This includes guest software components executing in VU mode like e.g. the guest application workload.
- *Device firmware adversary*: This includes any firmware driving a device within the system's PCIe topology. As PCIe transaction generators those device firmware components can gain direct access to a TVM confidential memory.
- *Simple hardware adversary*: This includes adversaries that are able to probe visible buses on the motherboard, use JTAG based debuggers, power cycle the system, subject the system to thermal radiation.
- *Advanced hardware adversary*: This includes adversaries that, in addition to the simple hardware adversary capabilities, can also probe high speed buses, place interposers on visible buses, glitch clocks and voltage rails.

5.3. Threats

5.3.1. CoVE-IO-T001 - Trusted MMIO Malicious Access

Asset	Threat	Adversary	Scope	Result
TVM confidential data	Tamper and Disclosure	Privileged host software	In scope	Host component reads TVM confidential data
Description				
A privileged host software component programs a TDI that is assigned to a TVM. By accessing the device MMIO space, the host component can program direct memory access destination addresses to either its own address space or unintended parts of the TVM address space. Device generated data intended to be copied to the TVM confidential memory is respectively accessed by the host component instead or redirected to unintended parts of the TVM address space.				
Mitigations				
The `CoVE-IO-T001` threat can be addressed by preventing untrusted domain software components from accessing an assigned TDI, as follows: • With TEE-I/O, a PCIe root port generates TLPs with the T-bit set only if the MMIO access originates from the trusted domain. Untrusted domain MMIO accesses must have the T-bit cleared. • A TDI is assigned to a TVM when the TVM accepts it into its TCB, by notifying the TSM about it. • The TEE-I/O DSM enforces that: ◦ Before it is assigned to a TVM, a TDI must not directly access the TVM confidential memory. ◦ Once assigned to a TVM, a TDI is in either the `LOCKED` or `RUN` TDISP state. ◦ In both the `LOCKED` and `RUN` TDISP state, a TDI trusted MMIO space can only be accessed by a trusted domain generated TLP (T-bit set), through the TDI bound PCIe selective IDE stream.				

Table 1. CoVE-IO-T001

5.3.2. CoVE-IO-T002 - Trusted MMIO Remapping

Asset	Threat	Adversary	Scope	Result
Device trusted MMIO	Tamper	Privileged host software	In scope	TVM programs a TDI that is unassigned to it
Description				
A privileged host software component remaps a TVM assigned TDI MMIO guest physical address to an unassigned TDI MMIO host physical address. The TVM programs a different TDI than the one that is assigned to it.				
Mitigations				
The `CoVE-IO-T002` threat can be addressed as follows: • The TSM maintains second stage page tables (from trusted domain physical addresses to untrusted host domain physical addresses) in confidential memory. • The untrusted domain software component must not set the second stage mappings for the TDI trusted MMIO. It can requests the TSM to do so on its behalf, through the CoVE-IO host ABI. • The TSM must not enable Trusted MMIO mappings for an assigned TDI until the TVM accepts it. • The TVM receives the TDI device interface report through TDISP, via the the TSM CoVE-IO guest ABI. This report is trusted by the TVM and contains the trusted MMIO ranges and order in which they must be mapped to the TVM address space. • The TVM must explicitly accept the reported MMIO ranges, and the TSM must not enable them until they are accepted by the TVM.				

Table 2. CoVE-IO-T002

5.3.3. CoVE-IO-T003 - Trusted MMIO PCIe Redirection

Asset	Threat	Adversary	Scope	Result
Device Trusted MMIO	Tamper	Privileged host software	In scope	TVM accesses an unassigned TDI trusted MMIO space
Description				
A privileged host software component configures PCIe switches to redirect (or drop) MMIO accesses from the TVM to one of its assigned TDIs. The host software component can trick the TVM into tampering with an untrusted device or an unassigned TDI MMIO.				
Mitigations				

Asset	Threat	Adversary	Scope	Result
The `CoVE-IO-T003` threat can be addressed as follows: • PCIe switches must not be included in the TVM trust boundary. This is achieved by only allowing PCIe selective IDE streams to be established between a physical device and the untrusted host domain. • Although the VMM can tamper with the device IDE extended capabilities, the PCIe root port IDE settings must only be available to a TVM TCB component, either the TSM or a hardware root-of-trust.				

Table 3. CoVE-IO-T003

5.3.4. CoVE-IO-T004 - Trusted MMIO PCIe Pre-Configuration

Asset	Threat	Adversary	Scope	Result
TVM confidential data	Tamper and Disclosure	Privileged or unprivileged host software	In scope	Guest software reads and writes resp. from and to another TVM confidential memory
Description				
The VMM maliciously pre-configures a TDI trusted MMIO and assigns it to a TVM. If either the TVM accepts the TDI as-is into its TCB, or the TDI is made operational before the TVM accepts it, the TDI can now access or tamper with the TVM confidential data on behalf of the host software component.				
Mitigations				
TBD				

Table 4. CoVE-IO-T004

5.3.5. CoVE-IO-T005 - Trusted MMIO Unauthorized Access

Asset	Threat	Adversary	Scope	Result
Device trusted MMIO	Tamper	Privileged host software	In scope	TVM accesses an unassigned TDI trusted MMIO space
Description				
A privileged host software component maps a TDI trusted MMIO space into TVM1 as part of the TDI assignement. Then it unassigns the TDI from TVM1 and assigns it to TVM2, withouth unmapping the TDI trusted MMIO space from TVM1. TVM1 can tamper with a TDI trusted MMIO while it is not assigned to it.				
Mitigations				
TBD				

Table 5. CoVE-IO-T005

5.3.6. CoVE-IO-T006 - PCIe Link Man-In-The-Middle

Asset	Threat	Adversary	Scope	Result
TVM confidential data	Tamper and Disclosure	Advanced hardware	In scope	A hardware adversary probes or places an interposer on the PCIe physical link between a TVM and its assigned TDI
Description				
A skilled hardware adversary with system physical access probes or places an interposer in the PCIe physical link. It can then eavesdrop, replay or event tamper with a TVM confidential data.				
Mitigations				
TBD				

Table 6. CoVE-IO-T006

5.3.7. CoVE-IO-T007 - PCIe ID Spoofing

Asset	Threat	Adversary	Scope	Result
TVM confidential data	Tamper and Disclosure	Device firmware	In scope	Host software reads and writes from and to a TVM confidential memory
Description				

Asset	Threat	Adversary	Scope	Result
A device firmware spoofs a PCIe Requester ID (RID) to generate PCIe packets with an existing, assigned TDI RID and get direct memory access to the corresponding TVM confidential memory.				
Mitigations				
TBD				

Table 7. CoVE-IO-T007

5.3.8. CoVE-IO-T008 - Confused Deputy DMA Remapping

Asset	Threat	Adversary	Scope	Result
TVM confidential data	Tamper and Disclosure	Privileged guest software	In scope	Guest software reads and writes resp. from and to another TVM confidential memory
Description				
TVM1 and TVM2 are assigned resp. TDI1 and TDI2. TDI1 and TDI2 belong to the same physical device. TVM1 programs TDI1 with TVM2's address space. TVM2 confidential memory is accessed by an unassigned TDI.				
Mitigations				
TBD				

Table 8. CoVE-IO-T008

5.3.9. CoVE-IO-T009 - DMA Remapping

Asset	Threat	Adversary	Scope	Result
TVM confidential data	Tamper and Disclosure	Privileged host software	In scope	Host software reads and writes from and to a TVM confidential memory
Description				
The privileged host software component manipulates an assigned TDI guest physical address (GPA) to host physical address (HPA) mappings. The TDI direct memory access to and from the TVM confidential data is then redirected to the host software component address space, allowing it to eavesdrop or tamper with the TVM confidential data.				
Mitigations				
TBD				

Table 9. CoVE-IO-T009

5.3.10. CoVE-IO-T010 - DMA Remapping

Asset	Threat	Adversary	Scope	Result
TVM confidential data	Tamper	Privileged host software	In scope	TDI writes into unintended portions of a TVM confidential memory
Description				
The privileged host software component manipulates the guest physical address (GPA) to host physical address (HPA) mappings to create inconsistencies between the TVM and its assigned TDI mappings for the same GPA ranges. The TDI writes physical adresses that are different than the ones the TVM programmed it with, and tampers the TVM confidential memory. Moreover, the TVM memory reads from the intended GPA return results that are inconsistent with the actual device operation.				
Mitigations				
TBD				

Table 10. CoVE-IO-T010

5.3.11. CoVE-IO-T011 - TDI Denial of Service

Asset	Threat	Adversary	Scope	Result
TVM confidential data	Denial of service	Privileged host software	Not in scope	TVM can not access a TDI that is assigned to it
Description				

Asset	Threat	Adversary	Scope	Result
A privileged host software component resets or powers down an assigned TDI or its physical device, while the TDI is assigned to a TVM. The TVM is no longer able to directly access its assigned TDI.				
Mitigations				
TBD				

Table 11. CoVE-IO-T011

5.4. Requirements

List CoVE-IO security requirements to address the threat model.

Chapter 6. Architectural Overview

CoVE-IO is made of multiple hardware and software components.

Figure 1 below illustrates the overall CoVE-IO architectural components and how they interact together:

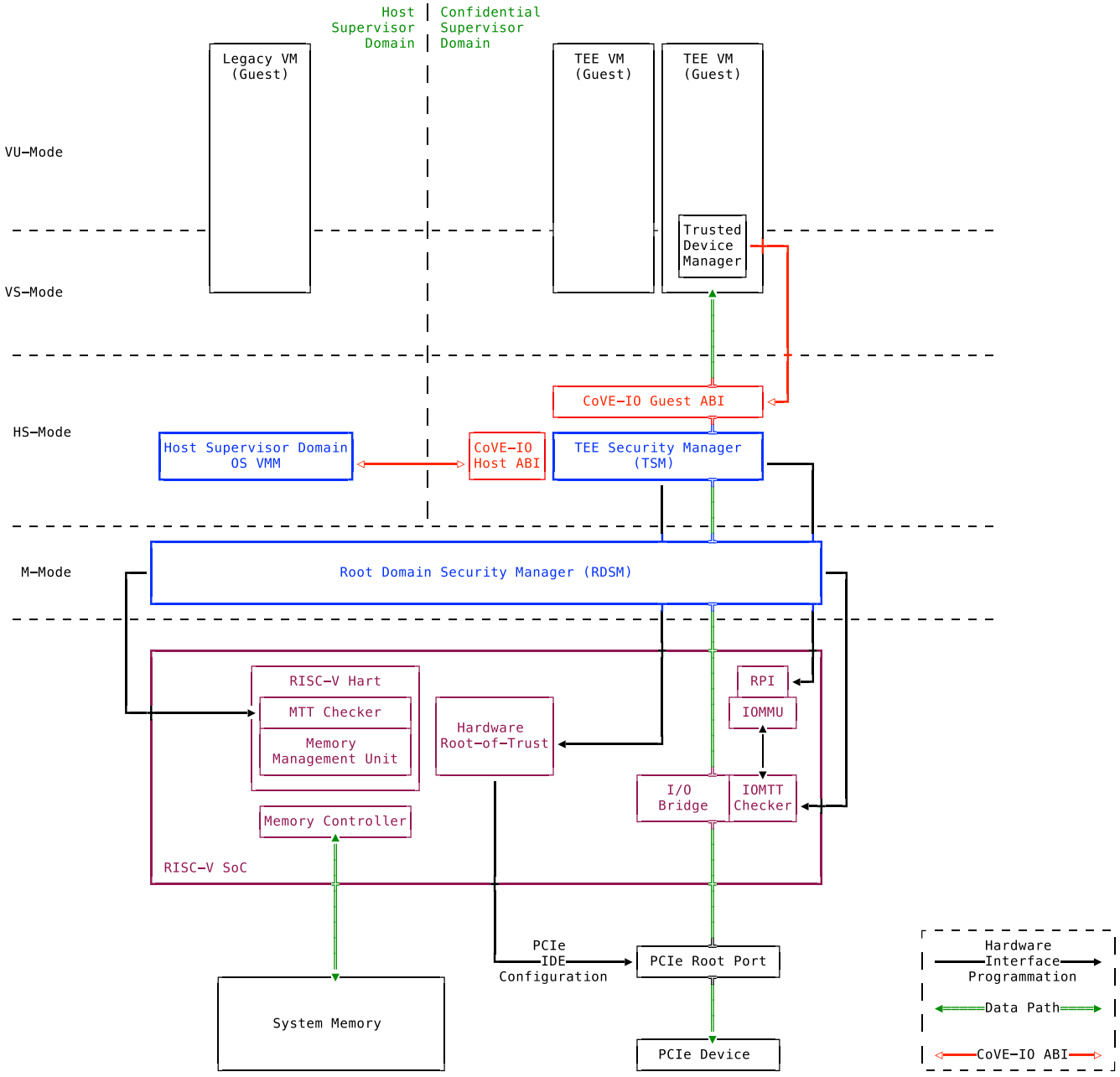

Figure 4. CoVE-IO High Level Architecture

The TEE Security Manager (TSM) orchestrates and manages TEE Virtual Machines (TVM), as defined by the [CoVE] specification. The TSM operates in HS-Mode and is the confidential supervisor domain manager. It uses the [Smmtt] specified Memory Tracking Table (MTT) to enforce TVM memory isolation from the host supervisor domain.

As the confidential security domain manager, the TSM provides the **COVH** ABI for the host OS/VMM to create and destroy TVMs, and donate or reclaim confidential memory from them. TVMs can interact with the TSM through the **COVG** ABI.

CoVE-IO extends both the COVH and COVG ABIs for bringing TEE-IO capable devices into the trust boundary of TVMs. These extensions allow for configuring and registering IOMMUs and PCIe root ports, and then binding physical devices interfaces (TDI) and TVMs together.

With the CoVE-IO ABIs and flows, TDIs can access TVM confidential memory directly. CoVE-IO uses the Smmtt I/O MTT extension and the platform IOMMUs security domain specific Register Programming Interfaces (RPI) to grant TDIs with direct access to their bound TVM confidential memory and isolate it from DMA originating from any unbound TDI.

With the CoVE-IO model, the TVM ultimately holds the decision to accept or reject a bound TDI into or from its TCB. Access to a TVM confidential memory from a TDI is disabled until a TVM notifies the TSM that it trusts the TDI and its configuration. The TVM communicates this decision after verifying the TDI, which generally includes attesting to the physical device the TDI belongs to. The device attestation process is specific to each TVM workload configuration, but it may be handled by the Trusted Device Manager (TDM).

In order to protect the confidentiality and integrity of TVM data between the PCIe root port and a bound TDI, CoVE-IO relies on the TSM to configure both the PCIe physical device and the root port it's attached to in order to establish IDE streams between the two. The PCIe root port IDE configuration from the TSM is abstracted through the platform Root-of-Trust (RoT).

Chapter 7. Theory of Operations

The CoVE-IO specification extends the CoVE host and guest SBI extensions to allow TVMs to establish trust with TEE-IO devices, and then use and interact with those devices. Untrusted supervisor domain components are responsible for assigning TEE-IO devices to TVMs, and also for supporting the TVM acceptance or rejection of the assigned devices. The TSM, on the other hand, establishes and maintains the secure physical and logical links between TVMs and their assigned devices.

The following sections describe the functionality of the TSM-provided CoVE-IO extensions to support trusted I/O on CoVE-enabled platforms.

7.1. Platform Initialization

7.1.1. IOMMU Registration and Setup

The TSM relies on the availability of at least one IOMMU instance exclusvely associated with the TSM supervisor domain. Those IOMMUs allow the TSM to enforce the integrity of address translations and protection from DMA into confidential memory, as well as interrupts originating from assigned TDIs. The host supervisor domain may assign one or more IOMMU instances to the TSM supervisor domain, after which, only the TSM can access and program the assigned IOMMU instances.

IOMMUs assigned to the TSM security domain may generate MSIs in order to signal the TSM about command completions, transaction faults or device page requests. Those MSIs target system physical memory, which is owned by the host security domain manager, e.g. the host VMM. As a consequence, it is the host security domain's responsibility to reserve the MSI addresses and then rely on the TSM to program the IOMMUs with those reserved addresses. This IOMMU registration process is driven by the untrusted domain manager for all IOMMUs that participate in TEE-IO and operates as described in the following steps:

1. The TSM is loaded into a supervisor domain and provisioned with a CoVE-IO manifest. It is recommended that the TSM is measured by the root-of-trust for measurement (RTM) for subsequent attestation.
2. The host supervisor domain manager (e.g. the host VMM) enumerates all platform IOMMUs as PCIe devices.
3. The host IOMMU driver is loaded and probed.
4. The host IOMMU driver allocates MSI vectors for the trusted domain IOMMUs. Those vectors must point to the untrusted IOMMU driver as the TSM can not handle external interrupts.
5. The host supervisor domain manager registers the IOMMUs with the TSM by calling the `sbi_covh_register_iommu() COVH` function. The TSM gets the allocated MSI vectors and configures the trusted IOMMU `s_msi_cfg_table` register accordingly.
6. When a trusted IOMMU sends an MSI, the untrusted IOMMU driver handles it and notifies the TSM about a pending trusted IOMMU MSI by calling the `sbi_covh_notify_iommu_msi() COVH` function.
7. The TSM verifies that there is a pending MSI by reading the IOMMU `s_ipsr` register, and handles the interrupts as needed.

The host IOMMU driver may be malicious and attempt to trick the TSM by either invoking the `sbi_covh_notify_iommu_msi() COVH` function while there are no pending MSI from the trusted

IOMMU or not invoking it when there actually is one such pending interrupt. In the former case, the TSM can verify that there really is a pending MSI by checking the trusted IOMMU status registers. The latter case could cause a denial of service, which is not in scope of the confidential computing threat model. If the TSM must ensure that a given command is completed without having to rely on untrusted IOMMU driver MSI notifications, it can queue an `IOFENCE.C` command after the desired command and check for the `cqh` advancing past the `IOFENCE.C` command index.

7.1.2. PCIe Root Port Registration

When the data link between the TDI and the TVM must be secured, trusted I/O relies on the PCIe IDE protocol. IDE provides confidentiality and integrity protection for TLPs received and transmitted between the physical device the TDI belongs to and its assigned PCIe root port (RP). Both endpoints (The RP and the physical device) must be configured with the same encryption keys through the IDE key management protocols and have their IDE PCIe extended capability configured as well.

As described in the threat model and requirement sections, the following trusted I/O rules apply:

- As PCIe switches are outside of the TVM TCB, only selective IDE streams are used to protect the PCIe link between the TVM and the TDI.
- A single selective IDE stream is established between the physical device and its Root Port. All TDIs from the physical device share this single IDE stream.
- For each physical device from which a TDI is attached to a TVM, the TSM generates, owns and distributes the IDE stream keys to both the physical device and the RP.
- The TSM configures the RP PCIe IDE extended capability.

As the TSM is responsible for setting both the RP IDE keys and PCI IDE capabilities, it must be the IDE operations owner for any downstream device for a given RP. As the overall platform resources owner, the host supervisor domain software stack must register a RP for TEE I/O and IDE ownership with the TSM, by calling the `sbi_covh_register_rp() COVH` function. This function associates a RP id to its MMIO space (for the IDE capability configurations) and all the MMIO ranges that are routed through it. The TSM must compare these 2 arguments with the information it received from the platform ROT through the TEE I/O manifest. If both match, it can proceed into establishing an SPDM session with the RP.

Root-of-Trust SPDM Session

For each TDI that gets attached to a TVM, its corresponding PCIe root port must be configured with per-device IDE keys. This configuration is done through the PCIe IDE key management protocol (IDE_KM) that runs on top of an SPDM session. It is thus necessary to establish an SPDM session with all trusted I/O registered root ports. This is akin to considering the RP as just another trusted device for which the DSM is the platform ROT. The SPDM establishment thus happens between the TSM and the platform ROT. By having the ROT playing the RP DSM role, the RP vendor-specific IDE key management interface is abstracted by the ROT.

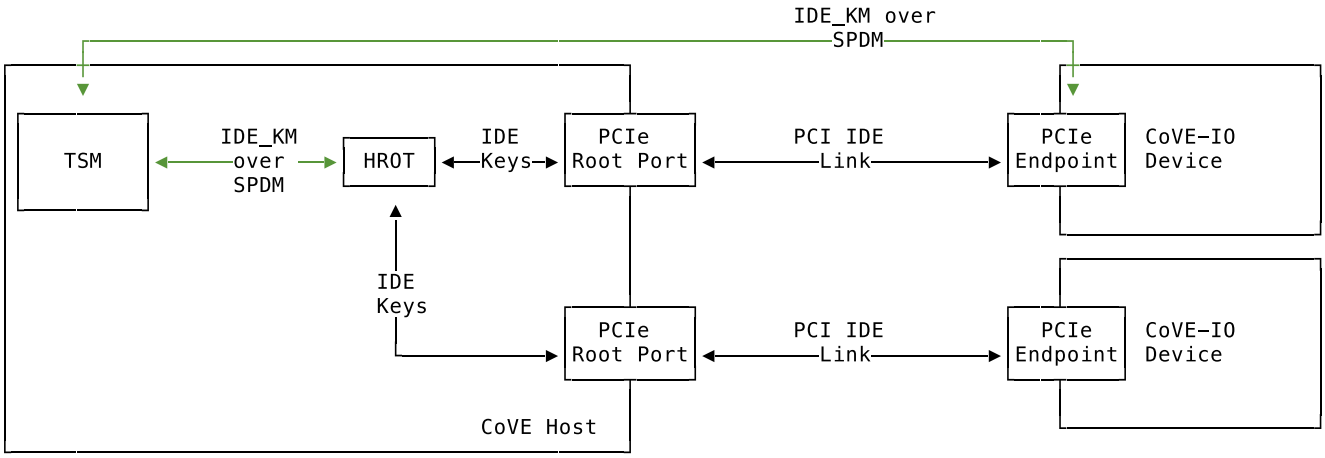

Figure 5. PCIe Root Port IDE Key Management through Hardware Root-of-Trust

7.2. SPDM Transport

SPDM is the main transport protocol for securely setting both the RP and the physical device IDE keys and also for driving the TDI through the TDISP state machine.

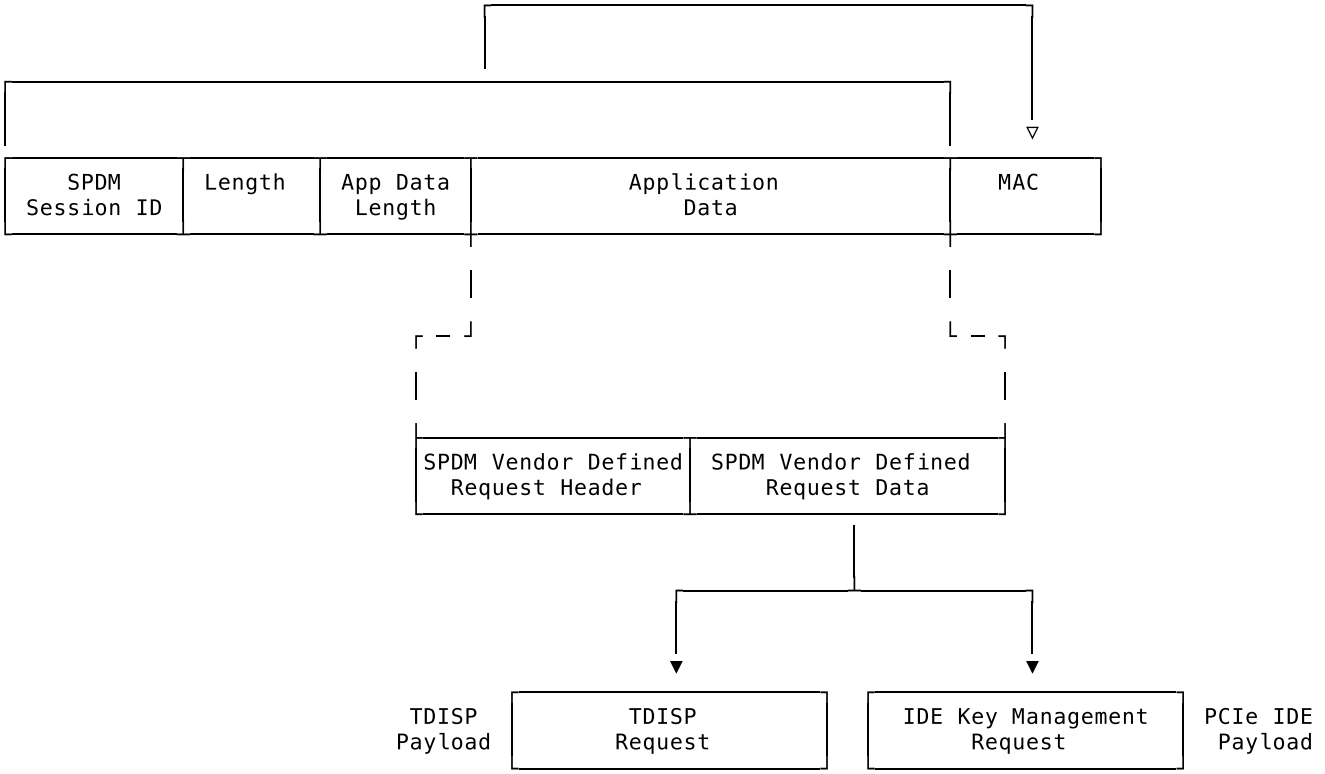

Figure 6. SPDM Message Layout

Whether the SPDM session is established with the ROT or the physical device, the TSM is the requester and it must reach and communicate with DOE mailboxes that are owned by the host supervisor domain manager. As a consequence, after requesting the TSM to establish an SPDM session with a device (through the `sbi_covh_connect_device()` `COVH` call), it acts as an untrusted SPDM messages proxy between the TSM and the device DSM.

After a successful call to `sbi_covh_connect_device()`, host supervisor domain initiated CoVE-IO `COVH` calls that require SPDM requests to be sent to the device DOE mailbox follows the flow described below:

Listing 1. SPDM Flow With CoVE

```
%%{init: {'theme': 'neutral', 'themeVariables': {'darkMode': true}, "flowchart" : { "curve" : "basis" } }
}%%
sequenceDiagram

autonumber

participant Responder as Device or ROT (SPDM Responder)
participant VMM as Host Supervisor Domain Manager (VMM)
participant TSM

VMM ->> TSM: [COVH] - sbi_covh_tee_io_action()
TSM ->> TSM: Generate SPDM request REQ_1
TSM ->> VMM: [COVH] - SBI_SUCCESS(SPDM_PENDING_REQUEST REQ_1)
VMM ->> Responder: [DOE] - SPDM_REQ_1
Responder ->> VMM: [DOE] - SPDM_RESP_1
VMM ->> TSM: [COVH] - sbi_covh_tee_io_action(RESP_1)
TSM ->> TSM: Generate SPDM request REQ_2
TSM ->> VMM: [COVH] - SBI_SUCCESS(SPDM_PENDING_REQUEST REQ_2)
VMM ->> Responder: [DOE] - SPDM_REQ_2
Responder ->> VMM: [DOE] - SPDM_RESP_2
VMM ->> TSM: [COVH] - sbi_covh_tee_io_action(RESP_2)
TSM ->> VMM: [COVH] - SBI_ERROR_CODE(SPDM_REQUEST_COMPLETED)
```

The TSM generates the SPDM request to support the initial CoVE-IO `COVH` call and copies the request into the per-vcpu shared, non-confidential memory region that the host supervisor domain and the TSM share as per the CoVE specification. The TSM replies to the host supervisor domain manager request with the `SBI_SUCCESS` error code and the `SPDM_PENDING_REQUEST (0x1)` value through the `sbiret` structure. The host supervisor domain manager then sends the pending SPDM request to the device DOE mailbox. It forwards the device SPDM response to the TSM, by copying it to the same SPDM buffer it fetches the SPDM request from and by calling again the same CoVE-IO `COVH` call. This process continues until the initial CoVE-IO call is completed. The TSM then replies to the last `COVH` call with the appropriate error code and the `SPDM_REQUEST_COMPLETED (0x0)` value through the `sbiret` structure.

The TSM only supports one pending SPDM transaction per device, and the CoVE NACL shared memory holds one pending SPDM action buffer per device the TSM is connected to. Each pending SPDM action buffer is structured as described in the following layout:

Offset (bytes)	Field	Length (bytes)	Description
0h	FUNCTION_ID	4	The SBI Function ID this pending SPDM transaction applies to.
4h	DEVICE_ID	4	The PCIe device ID this pending SPDM transaction applies to.
8h	SPDM_PAYLOAD_LENGTH	4	SPDM payload length.
12h	SPDM_PAYLOAD	See SPDM_PAYLOAD_LENGTH	SPDM payload data (e.g. TDISP, IDE_KM).

Table 12. Pending SPDM Transaction Buffer Layout

In this document, for readability reasons, flow definitions that involve SPDM based exchanges are simplified and do not include the above described flow between the TSM, the host supervisor domain manager and the SPDM responder. In particular, the steps that cover the SPDM request generation from the TSM, the transmission to the host VMM, the transmission to the SPDM responder and finally the path back to the TSM are reduced into 2 steps:

1. TSM sends SPDM request #1

2. TSM receives SPDM response #1

For example the above example would be described through those simplified steps:

1. VMM calls `sbi_covh_tee_io_action()`
2. TSM sends SPDM request #1
3. TSM receives SPDM response #1
4. TSM sends SPDM request #2
5. TSM receives SPDM response #2
6. TSM returns `sbi_covh_tee_io_action()`

7.2.1. Secure SPDM Session

With trusted I/O, the TSM establishes SPDM sessions with both the ROT and the physical devices DSMs. As those sessions are used to exchange IDE keys through the `IDE_KM` protocol while going through the host supervisor domain manager, they need to be confidentiality and integrity protected. Establishing a secured SPDM session for IDE key management or TDISP operations between the TSM and either PCIe root ports or devices is a mandatory requirement for TVMs to accept TDIs into their TCB.

Any trusted I/O SPDM session is established through the SPDM responder DOE mailbox, which could be either the ROT acting as a DSM for the PCIe root ports, or the physical device DSM. In either case the mailboxes are resources owned by the host security domain manager which thus initiates the SPDM session establishment. It acts as an untrusted proxy between the TSM and the DSM by requesting the TSM to generate SPDM requests through the CoVIO `COVH`TH ABI, sending those requests to the DOE mailbox and forwarding the SPDM responses back to the TSM, as described in the SPDM flow section.

7.3. Device Connection

After the IOMMU is registered with the TSM, the host supervisor domain manager must establish a logical connection with any device from which a TDI could be bound to a TVM. To do so, it must cooperate with the TSM to properly initialize all such physical devices.

The device initialization process aims at establishing secured, integrity-protected control and data planes between the TSM and the DSM running in either the platform ROT or a physical PCIe device. The secured control plane is based on the SPDM protocol and is an encrypted, integrity-protected software session that is used for passing TDISP and IDE_KM messages between the TSM and the DSM. The data place is a hardware session based on the PCIe Integrity and Data Encryption (IDE) specification and is used to secure the PCIe TLPs.

When the host supervisor domain detects a new TEE-IO capable device, it must go through two device initialization steps:

1. Establish a secured SPDM session between the TSM (The SPDM Requester) and the device DSM (The SPDM responder).
2. Set the PCIe IDE stream up for encrypting the PCIe link.

The CoVE-IO `COVH` extension supports those two initialization steps through one single function: `sbi_covh_connect_device()`.

When the host supervisor domain manager calls `sbi_covh_connect_device()`, it requests the TSM to establish an SPDM session with a device, and to set an PCIe IDE link between the device and the TSM.

7.3.1. SPDM Session

The first step for initializing a TEE-IO capable device is to establish a secured SPDM session between the TSM an the device. The secured SPDM sessions will then be used to carry `TDISP` and `IDE_KM` messages, in order to respectively secure the physical link between the device and its PCIe root port, and for binding or unbinding TDIs to or from a TVM.

The host supervisor domain manager requires the TSM to establish a secured SPDM session with the physical device by calling the `sbi_covh_connect_device()` COVH function. Before proceeding into actually establishing the session, the TSM must check that:

1. The physical device is a downstream endpoint of a root port that the host supervisor domain manager has previously registered with the TSM.
2. A secured SPDM session between the TSM and the upstream root port is established.

The TSM establishes a secured SPDM session with the physical device DSM by going through the steps described in the Secured SPDM Session section.

Listing 2. Device Connection - Secured SPDM Session

```
%%{init: {'theme': 'neutral', 'themeVariables': {'darkMode': true}, "flowchart" : { "curve" : "basis" } }
}%%
sequenceDiagram

autonumber

participant DSM as Device DSM or ROT
participant VMM as Host Supervisor Domain Manager (VMM)
participant TSM

VMM ->> TSM: [COVH] - sbi_covh_connect_device(device_id, stream_id)

Note over TSM,DSM: TSM: SPDM Connection Setup

TSM ->> TSM: Generate SPDM request: GET_VERSION
TSM ->> VMM: [COVH] - spdm_req(GET_VERSION)
VMM ->> DSM: [DOE] - SPDM_GET_VERSION
DSM ->> VMM: [DOE] - SPDM_VERSION
VMM ->> TSM: [COVH] - spdm_resp(VERSION)

TSM ->> TSM: Generate SPDM request: GET_CAPABILITIES
TSM ->> VMM: [COVH] - spdm_req(GET_CAPABILITIES)
VMM ->> DSM: [DOE] - SPDM_GET_CAPABILITIES
DSM ->> VMM: [DOE] - CAPABILITIES
VMM ->> TSM: [COVH] - spdm_resp(CAPABILITIES)

TSM ->> TSM: Generate SPDM request: NEGOTIATE_ALGORITHMS
TSM ->> VMM: [COVH] - spdm_req(NEGOTIATE_ALGORITHMS)
VMM ->> DSM: [DOE] - SPDM_NEGOTIATE_ALGORITHMS
DSM ->> VMM: [DOE] - ALGORITHMS
VMM ->> TSM: [COVH] - spdm_resp(ALGORITHMS)

TSM ->> TSM: Generate SPDM request: GET_CERTIFICATE
TSM ->> VMM: [COVH] - spdm_req(GET_CERTIFICATE)
VMM ->> DSM: [DOE] - GET_CERTIFICATE
DSM ->> VMM: [DOE] - CERTIFICATE
VMM ->> TSM: [COVH] - spdm_resp(CERTIFICATE)
TSM ->> TSM: Verify and store device certificate chain
```

```
Note over TSM,DSM: TSM: SPDM Key Exchange

TSM ->> TSM: Generate ephemeral SPDM session DHE key pair - DheKey0
TSM ->> TSM: Generate SPDM request: KEY_EXCHANGE(DheKey0Pub)
TSM ->> VMM: [COVH] - spdm_req(KEY_EXCHANGE)
VMM ->> DSM: [DOE] - SPDM_KEY_EXCHANGE
DSM ->> DSM: Generate ephemeral SPDM session DHE key pair - DheKey1
DSM ->> VMM: [DOE] - KEY_EXCHANGE_RSP(DheKey1Pub)
VMM ->> TSM: [COVH] - spdm_resp(KEY_EXCHANGE_RSP)
TSM ->> TSM: Derive DHE secret
TSM ->> TSM: Derive SPDM session handshake secrets
TSM ->> TSM: Generate SPDM request: FINISH
TSM ->> VMM: [COVH] - spdm_req(FINISH)
VMM ->> DSM: [DOE] - SPDM_FINISH
DSM ->> VMM: [DOE] - FINISH_RSP
VMM ->> TSM: [COVH] - spdm_resp(FINISH_RSP)
TSM ->> TSM: Derive SPDM session application secrets

Note over TSM,DSM: TSM: SPDM Session Created
```

7.3.2. IDE Link

The SPDM session is a software link between the TSM and the DSM, secured after both entities go through a DHE key exchange over the untrusted host supervisor domain manager proxy. SPDM is used as a control link for configuring the rest of the device and then running the TDI binding flows.

The last part of the device connection process is about securing the data link between the TSM and the device, and that must be done through the IDE Key Management protocol. Here again, the host supervisor domain manager implictly initiates the PCI IDE link setup by calling the `sbi_covh_connect_device() COVH` function, and relies on the TSM to generate and send IDE KM messages over SPDM.

The TSM is responsible for:

1. Configuring the PCIe root port IDE Extended Capability.
2. Generating the IDE keys for all sub-streams for a given stream ID.
3. Setting the PCIe root port IDE keys for a given stream ID, through IDE KM requests.
4. Refreshing the PCIe root port IDE keys for a given stream ID.
5. Generating all IDE KM requests and encapsulating them into SPDM messages.
6. Setting the PCIe device IDE keys through IDE KM requests.

The host supervisor domain manager is responsible for:

1. Generating and managing system wide PCIe stream IDs.
2. Setting the device PCIe device IDE PCI Extended Capability.
3. Programing the PCIe switch between the device and PCIe root port, such as IDE Control Register - Flow-Through IDE Stream Enabled.
4. Initiating the IDE link setup.

The IDE link initial setup must go through the following steps:

1. The host supervisor domain manager finds an available stream ID and configures the device IDE

Extended Capability accordingly.

2. The host supervisor domain manager programs the device IDE extended capability: All RIDs and all memory is allowed, IDE is disabled for the selected stream.
3. The host supervisor domain manager implicitly initiates the IDE link setup by calling the `sbi_covh_connect_device()` `COVH` function, and passing the selected stream ID as an argument to it. The TSM must first establish a secured SPDM session, as described in the previous section.
4. After it establishes a secured SPDM session with the device, the TSM starts setting the IDE link up and programs the Root Port IDE Extended Capability with the proper RID range and the stream ID selected by the host supervisor domain manager in step 1. IDE is disabled for that stream ID. The capability is only accessible to the TSM, i.e. any writes to it with the C-bit set to 0 are dropped and reads return all 1s.
5. The TSM generates an IDE key for each sub-stream for the stream ID.
6. The TSM, for each Rx and Tx sub-stream (6 of them), programs the generated keys into the physical device:
    1. Generates and sends an `IDE_KM KEY_PROG` message to the DSM. The message is encapsulated in a vendor-defined SPDM request.
    2. Receives an `IDE_KM_KEY KP_ACK` from the DSM.
7. The TSM, for each Rx and Tx sub-stream (6 of them), programs the generated keys into the physical device's PCIe root port:
    1. Generates and sends an `IDE_KM KEY_PROG` message to the ROT. The message is encapsulated into a vendor-defined SPDM request.
    2. Receives an `IDE_KM_KEY KP_ACK` from the ROT.
8. The TSM, for each Rx sub-stream (3 of them), triggers IDE in the physical device:
    1. Generates and sends an `IDE_KM SET_GO(Rx)` message to the DSM. The message is encapsulated into a vendor-defined SPDM request.
    2. Receives an `IDE_KM_KEY K_GOSTOP_ACK` from the DSM.
9. The TSM, for each Rx sub-stream (3 of them), triggers IDE in the physical device's PCIe root port:
    1. Generates and sends an `IDE_KM SET_GO(Rx)` message to the ROT. The message is encapsulated into a vendor-defined SPDM request.
    2. Receives an `IDE_KM_KEY K_GOSTOP_ACK` from the ROT.
10. The TSM, for each Tx sub-stream (3 of them), triggers IDE in the physical device:
    1. Generates and sends an `IDE_KM SET_GO(Tx)` message to the DSM. The message is encapsulated into a vendor-defined SPDM request.
    2. Receives an `IDE_KM_KEY K_GOSTOP_ACK` from the DSM.
11. The TSM, for each Tx sub-stream (3 of them), triggers IDE in the physical device's PCIe root port:
    1. Generates and sends an `IDE_KM SET_GO(Tx)` message to the ROT. The message is encapsulated into a vendor-defined SPDM request.
    2. Receives an `IDE_KM_KEY K_GOSTOP_ACK` from the DSM.

Listing 3. Device Connection - IDE Link Setup

```
%%{init: {'theme': 'neutral', 'themeVariables': {'darkMode': true}, "flowchart" : { "curve" : "basis" } }
}%%
```

```
sequenceDiagram

autonumber

participant DSM as Device DSM
participant Device as PCIe Device
participant RootPort as PCIe Root Port
participant RoT as RoT (RP DSM)
participant VMM as Host Supervisor Domain Manager (VMM)
participant TSM

VMM ->> VMM: Find an available IDE stream ID
VMM ->> Device: Program IDE extended capability (Allow all RIDs and memory, disable IDE).

VMM ->> TSM: [COVH] - sbi_covh_connect_device(device_id, stream_id)

Note over TSM,DSM: TSM: SPDM Connection Initialization

Note over TSM,DSM: TSM: SPDM Connection Created

TSM ->> RoT: Program RP IDE extended capability (Allow RID and memory for the device, IDE is disabled).

TSM ->> TSM: Generate 6 IDE keys [(Rx, Tx) * sub-streams(P, NP, C)]

Loop 6 times - TSM programs IDE keys into Device (For the given stream ID)
     TSM ->> TSM: Generate SPDM request IDE_KM_KEY_PROG
     TSM ->> VMM: [COVH] - spdm_req(IDE_KM_KEY_PROG)
     VMM ->> DSM: [DOE] - SPDM_IDE_KM_KEY_PROG
     DSM ->> Device: Program  IDE Key
     DSM ->> VMM: [DOE] - SPDM_IDE_KM_KP_ACK
     VMM ->> TSM: [COVH] - spdm_resp(IDE_KM_KP_ACK)
end

Loop 6 times - TSM programs IDE keys into Root Port through RoT (For the given stream ID)
     TSM ->> TSM: Generate SPDM request IDE_KM_KEY_PROG
     TSM ->> VMM: [COVH] - spdm_req(IDE_KM_KEY_PROG)
     VMM ->> RoT: [DOE] - SPDM_IDE_KM_KEY_PROG
     RoT ->> RootPort: Program  IDE Key
     RoT ->> VMM: [DOE] - SPDM_IDE_KM_KP_ACK
     VMM ->> TSM: [COVH] - spdm_resp(IDE_KM_KP_ACK)
end

Loop 3 times (For each sub-stream)
     TSM ->> TSM: Generate SPDM request IDE_KM_K_SET_GO
     TSM ->> VMM: [COVH] - spdm_req(IDE_KM_K_SET_GO)
     VMM ->> DSM: [DOE] - SPDM_IDE_KM_K_SET_GO
     DSM ->> Device: Trigger Tx IDE
     DSM ->> VMM: [DOE] - SPDM_IDE_KM_K_GOSTOP_ACK
     VMM ->> TSM: [COVH] - spdm_resp(IDE_KM_K_GOSTOP_ACK)
end

Loop 3 times (For each sub-stream)
     TSM ->> TSM: Generate SPDM request IDE_KM_K_SET_GO
     TSM ->> VMM: [COVH] - spdm_req(IDE_KM_K_SET_GO)
     VMM ->> RoT: [DOE] - SPDM_IDE_KM_K_SET_GO
     RoT ->> Device: Trigger Tx IDE
     RoT ->> VMM: [DOE] - SPDM_IDE_KM_K_GOSTOP_ACK
     VMM ->> TSM: [COVH] - spdm_resp(IDE_KM_K_GOSTOP_ACK)
end

Loop 3 times (For each sub-stream)
     TSM ->> TSM: Generate SPDM request IDE_KM_K_SET_GO
     TSM ->> VMM: [COVH] - spdm_req(IDE_KM_K_SET_GO)
     VMM ->> DSM: [DOE] - SPDM_IDE_KM_K_SET_GO
     DSM ->> Device: Trigger Rx IDE
     DSM ->> VMM: [DOE] - SPDM_IDE_KM_K_GOSTOP_ACK
     VMM ->> TSM: [COVH] - spdm_resp(IDE_KM_K_GOSTOP_ACK)
```

```
end

Loop 3 times (For each sub-stream)
     TSM ->> TSM: Generate SPDM request IDE_KM_K_SET_GO
     TSM ->> VMM: [COVH] - spdm_req(IDE_KM_K_SET_GO)
     VMM ->> RoT: [DOE] - SPDM_IDE_KM_K_SET_GO
     RoT ->> Device: Trigger Rx IDE
     RoT ->> VMM: [DOE] - SPDM_IDE_KM_K_GOSTOP_ACK
     VMM ->> TSM: [COVH] - spdm_resp(IDE_KM_K_GOSTOP_ACK)
end

TSM ->> VMM: [COVH] - spdm_covh_connect_device()

VMM ->> Device: Enable IDE for the selected stream
VMM ->> RootPort: Enable IDE for the selected stream
```

7.4. Device Disconnection

7.5. Interface Binding

Once both the SPDM session and the IDE link are secured and established, the host supervisor domain manager may bind a TDI and a TVM together, through the `COVH` interface. This is a four steps process:

1. The host supervisor domain manager initiates the interface binding flow by having the TSM move the TDI into the TDISP `CONFIG_LOCKED` state. This is achieved through the `sbi_covh_bind_interface()` `COVH` ABI.
2. The TVM verifies and accepts the locked TDI into its TCB.
3. The TVM asks the TSM to move the TDI to the TDISP `RUN` state, by calling the `sbi_covg_start_interface()` `COVG` ABI.
4. The TVM verifies that the TDI is in the TDISP `RUN` state and starts using it. This verification is provided by the `sbi_covg_get_interface_state()` `COVG` ABI.

The next two sections respectively give a detailed description of the overall process for binding an interface to a TVM, and one if its most critical steps: the TVM decision of accepting or rejecting the TDI into its TCB.

7.5.1. Binding Flow

Binding an interface and a TVM together goes through the following steps:

1. For the binding process to succeed, the host supervisor domain manager must first pass each of the bound TDI MMIO ranges to the TVM. It does so by calling the `sbi_covh_add_tvm_interface_region()` `COVH` ABI. The TSM services those requests by creating uncacheable G-stage mappings for each range, so that the TVM outbound transaction will not trigger MMIO page faults. The TSM must not enable the above mappings until the TVM accepts the TDI in its TCB, by moving it to the TDISP `RUN` state.
2. The host supervisor domain manager calls the `sbi_covh_bind_interface()` `COVH` function to start binding the TDI and the TVM together. The TSM rejects that request if a secured SPDM session is not established with the DSM or if a stream IDE link is not set up.
3. The TSM discovers and queries the TDI's TDISP version and capabilities by respectively generating and sending the TDISP `GET_TDISP_VERSION` and `GET_TDISP_CAPABILITIES` messages to

the DSM. The respective `TDISP_VERSION` and `TDISP_CAPABILITIES` responses let the TSM select a common TDISP version and set of capabilities to be used in all future TDISP communication with the DSM.

4. The TSM locks the bound TDI by generating and sending a TDISP `LOCK_INTERFACE_REQUEST` message to the DSM, which is encapsulated into a vendor-defined SPDM request.
5. The TSM receives the TDISP `LOCK_INTERFACE_RESPONSE` message, which contains a device-generated nonce. The TSM stores the nonce. The TDI is now in the TDISP `CONFIG_LOCKED` state.
6. The TSM programs the DMA mappings into the trusted IOMMU space, in order for the TDI inbound transactions to map into the TVM address space. The IOMMU mappings are programmed but not validated. They will be validated only once the TDI moves to the TDISP `RUN` state, i.e. when it accepts the TDI into its TCB through the `sbi_covg_start_interface()` `COVG` ABI.
7. Through regular discovery mechanisms (ACPI, PCI bus scanning), the TVM detects the assigned TDI. It is important to note that the TDI configuration space, including its BARs, is emulated by the host security domain manager. The TDI MMIO ranges are mapped into the TVM address space by the host security domain manager, through the `sbi_covh_add_tvm_interface_region()` `COVH` ABI.
8. Before using the TDI, the TVM must accept it into its TCB. Moreover, the TVM must not use the TDI until it can verify from the TSM that it's been put in the TDISP `RUN` state (step 12 below). To notify the TSM about its decision to accept or reject the bound TDI, the TVM respectively call into the `sbi_covg_start_interface()` or `sbi_covg_stop_interface()` `COVG` ABI.
9. Upon acceptation of the TDI by the TVM, the TSM generates and sends a TDISP `START_INTERFACE_REQUEST` message that includes the `LOCK_INTERFACE_RESPONSE` nonce received on step 5. The message is encapsulated into a vendor-defined SPDM request.
10. The TSM receives the TDISP `START_INTERFACE_RESPONSE`. The TDI is now in the TDISP `RUN` state.
11. The TSM enables the IOMMU and the TVM G-stage mappings configured on steps 6 and 1, for respectively enabling the DMA and MMIO operations with the bound TDI.
12. The TVM verifies that the TDI is in the TDISP `RUN` state by calling the `sbi_covg_get_interface_state()` `COVG` ABI.
13. The TVM can start using the device.

Listing 4. Device Interface Binding - Bind Interface

```
%%{init: {'theme': 'neutral', 'themeVariables': {'darkMode': true}, "flowchart" : { "curve" : "basis" } }
}%%
sequenceDiagram

autonumber

participant TDI as Device Interface
participant DSM as Device DSM
participant VMM as Host Supervisor Domain Manager (VMM)
participant TSM
participant IOMMU

note over TDI: CONFIG_UNLOCKED

loop For all the TDI MMIO regions exposed to the TVM
     VMM ->> TSM: [COVH] sbi_covh_add_tvm_interface_region()
     TSM ->> TSM: Prepare G-stage mappings and mark them invalid
     TSM ->> TSM: Store the interface MMIO gpa -> hpa mapping
     TSM ->> VMM: [COVH] sbi_covh_add_tvm_interface_region()
end
```

```
VMM ->> TSM: [COVH] sbi_covh_bind_interface()

TSM ->> TSM: Generate TDISP GET_TDISP_VERSION
TSM ->> VMM: [COVH] spdm_req(GET_TDISP_VERSION)
VMM ->> DSM: [DOE] SPDM_GET_TDISP_VERSION
DSM ->> VMM: [DOE] SPDM_TDISP_VERSION
VMM ->> TSM: [COVH] spdm_resp(TDISP_VERSION)
TSM ->> TSM: Decrypt TDISP_VERSION

TSM ->> TSM: Generate TDISP GET_TDISP_CAPABILITIES
TSM ->> VMM: [COVH] spdm_req(GET_TDISP_CAPABILITIES)
VMM ->> DSM: [DOE] SPDM_GET_TDISP_CAPABILITIES
DSM ->> VMM: [DOE] SPDM_TDISP_CAPABILITIES
VMM ->> TSM: [COVH] spdm_resp(TDISP_CAPABILITIES)
TSM ->> TSM: Decrypt TDISP_CAPABILITIES

TSM ->> TSM: Generate TDISP LOCK_INTERFACE_REQUEST
TSM ->> VMM: [COVH] spdm_req(LOCK_INTERFACE_REQUEST)
VMM ->> DSM: [DOE] SPDM_LOCK_INTERFACE_REQUEST
DSM ->> TDI: LOCK
note over TDI: CONFIG_LOCKED
DSM ->> VMM: [DOE] SPDM_LOCK_INTERFACE_RESPONSE
VMM ->> TSM: [COVH] spdm_resp(LOCK_INTERFACE_RESPONSE)
TSM ->> TSM: Decrypt LOCK_INTERFACE_RESPONSE
TSM ->> TSM: Store LOCK_Nonce
TSM ->> IOMMU: Disable IOMMU translation for devif_id
TSM ->> IOMMU: Configure C-IOMMU mappings to G-stage
TSM ->> VMM: [COVH] sbi_covh_bind_interface()

note over IOMMU,TDI: TVM verifies the device interface
note over IOMMU,TDI: TVM accepts and uses the device interface
```

7.5.2. TDI Verification and Acceptation

It is the TVM responsibility to accept or reject the assigned TDI into its TCB, and to explicitly notify both the TSM and the host supervisor domain manager about its decision. The TVM should verify the following security attributes before being able to decide whether or not it can safely accept a TDI into its TCB:

1. **SPDM session establishment**: A secured SPDM session must be established between the TDI's DSM and the TSM. TVM verifies that attribute from the TSM, through the `sbi_covg_get_device_link()` `COVG` ABI.
2. **IDE link**: The PCIe physical link between the Root Port and the physical device must be confidentiality and integrity protected through IDE. As for the SPDM session, the TVM calls into the `sbi_covg_get_device_link()` `COVG` ABI to verify that attribute from the TSM.
3. **TDISP and SPDM configuration**: The TVM must verify that the TDI TDISP configuration and the SPDM session attributes comply with its security policy. For example, the TVM could check for the allowed device firmware update policy by combining the TDI TDISP report `NO_FW_UPDATE` setting with the SPDM session measurement freshness capabilities (`MEAS_FRESH_CAP`). It is then the TVM choice to accept or reject a TDI depending on the inferred physical device firmware update policy. The TDI interface report and the SPDM session attributes are provided by respectively the `sbi_covg_get_interface_report()` and the `sbi_covg_get_device_spdm_attrs()` `COVG` ABI.
4. **TDI state**: Before accepting a TDI into its TCB, a TVM must verify that its configuration is immutable, and in particular that the host can not modify it without having all in-flight transactions being discarded. TEE-IO capable physical devices follow the TDISP specification and can guarantee that immutability state once the TDI has been moved to the TDISP `CONFIG_LOCKED`

state. The transition from TDISP `CONFIG_UNLOCKED` to `CONFIG_LOCKED` is triggered by the host supervisor domain manager through the `COVH` ABI. As such, the TVM can query the TSM for the TDI state through the `sbi_covg_get_interface_state()` `COVG` ABI. A TVM must not accept a TDI if it's in any other TDISP state than `CONFIG_LOCKED`.

5. **Device trustworthiness**: Verifying that the TDI is in an immutable state across a secured SPDM and physical link is mandatory but not sufficient. The TVM must also attest to the physical device trustworthiness in order to decide if it can accept one of its TDIs into its TCB. A TVM can trust a PCIe device by first authentictating it. Once authenticated, the TVM challenges the device and then verifies its measurements:
    1. First the TVM must first verify the authenticity of the device by getting its certificate chain from the TSM, through the `sbi_covg_get_device_certificate()` `COVG` ABI. The TVM should then verify the chain against a provisioned and measured trust anchor list.
    2. Once the device certificate authenticity is verified, the TVM must then challenge it by having it sign a piece of data, making sure that the device actually owns the private key bound to its certificate. This is achieved by getting the TDI measurements from the TSM through the `sbi_covg_get_device_measurement()` `COVG` ABI. This set of device-signed measurements, also knonw as the device attestation evidence, must be verified against the TDI certificate acquired in the previous step.
    3. Finally, the TVM should attest to the device configuration trustworthiness (code, SVN, state, etc) by verifying the previously fetched device attestation evidence. This is typically done through a remote or local attestation procedure.
6. **TDI IO ranges**: The TVM will likely interact with and program the TDI through a set of memory mapped IO ranges (e.g. a PCI BAR defined memory range). However, when discovering the TDI in its address space, the TVM only sees guest physical addresses (GPA) for those ranges, as exposed by the host supervisor domain manager PCI emulation. When communicating with the TDI, the TVM will use those GPAs and must rely on their corresponding translations to host physical addresses (HPA) to be properly set. In particular, it must rely on the fact that the TDI MMIO ranges GPAs do not map to non-confidential memory that could be otherwise accessed by a host domain component. To verify that security attribute, the TVM must retrieve the TDISP report for the TDI, through the `sbi_covg_get_interface_report()`. The TDISP report, among other things, contains the list of MMIO ranges for the TDI sorted by BAR indexes. First, the TVM must verify that the host VMM exposed BARs have the same sizes as the TDISP reported ones. To further validate those ranges, the TVM must check from the TSM that they're correctly mapped to host physical ranges. Prior to the TVM being able to accept a TDI, the host VMM must have requested the TSM to map all the TDI MMIO ranges to TVM GPA ranges, through `sbi_covh_add_tvm_interface_region()` `COVH` calls. The TVM then verifies from the TSM that a GPA exposed TDI MMIO range will be mapped to the TDISP reported range through the TSM managed G-stage page tables, by calling into the `sbi_covg_map_interface_mmio()` `COVG` ABI. The TVM can accept a TDI only if the TSM confirms the validity of all MMIO range mappings, in the TDISP reported order (i.e. BAR #N in the TVM address space will be mapped to the TDISP reported MMIO range #N).

Once the TVM has verified the above security attributes, it lets the TSM know that it is ready to use the TDI, by calling into the `sbi_covg_start_interface()` `COVG` ABI.

Listing 5. Device Interface Verification

```
%%{init: {'theme': 'neutral', 'themeVariables': {'darkMode': true}, "flowchart" : { "curve" : "basis" } }
}%%
sequenceDiagram

autonumber
```

```
participant DSM as Device DSM
participant VMM as Host Supervisor Domain Manager (VMM)
participant TSM
participant TVM

VMM ->> TSM: [COVH] sbi_covh_run_vcpu()

note over TVM,TSM: Verify Device Link (Secured SPDM and IDE keys)
    TVM ->> TSM: [COVG] sbi_covg_get_device_link()
    TSM ->> TVM: [COVG] sbi_covg_get_device_link()
    TVM ->> TVM: Verify that Secured SPDM and IDE are established

note over TVM,TSM: Verify TDISP and SPDM Configuration
    TVM ->> TSM: [COVG] sbi_covg_get_device_spdm_attrs()
    TSM ->> TVM: [COVG] sbi_covg_get_device_spdm_attrs()
    TVM ->> TSM: [COVG] sbi_covg_get_interface_report()
    TSM ->> TSM: Generate TDISP request GET_DEVICE_INTERFACE_REPORT
    TSM ->> VMM: [COVH] spdm_req(GET_DEVICE_INTERFACE_REPORT)
    VMM ->> DSM: [DOE] SPDM_GET_DEVICE_INTERFACE_REPORT
    DSM ->> VMM: [DOE] SPDM_DEVICE_INTERFACE_REPORT
    VMM ->> TSM: [COVH] spdm_resp(DEVICE_INTERFACE_REPORT)
    TSM ->> TSM: Decrypt DEVICE_INTERFACE_REPORT
    TSM ->> TVM: [COVG] sbi_covg_get_interface_report()
    TVM ->> TVM: Verify that the TDISP and SPDM configuration comply with the TVM policy

note over TVM,TSM: Verify Device Interface State (TDISP CONFIG_LOCKED)
    TVM ->> TSM: [COVG] sbi_covg_get_interface_state()
    TSM ->> TVM: [COVG] sbi_covg_get_interface_state()
    TVM ->> TVM: Check that the interface state is CONFIG_LOCKED

note over TVM,TSM: Verify Device Identity
    TVM ->> TSM: [COVG] sbi_covg_get_device_certificate()
    TSM ->> TVM: [COVG] sbi_covg_get_device_certificate()
    TVM ->> TVM: Verify the returned device certificate chain

note over TVM,DSM: Challenge Device
    TVM ->> TSM: [COVG] sbi_covg_get_device_measurement()
    TSM ->> TSM: Generate SPDM GET_MEASUREMENTS request
    TSM ->> VMM: [COVH] spdm_req(GET_MEASUREMENTS(Nonce))
    VMM ->> DSM: [COVH] SPDM_GET_MEASUREMENTS(Nonce)
    DSM ->> VMM: [DOE] SPDM_MEASUREMENTS(Nonce)
    VMM ->> TSM: [COVH] spdm_resp(MEASUREMENTS)
    TSM ->> TSM: Decrypt SPDM MEASUREMENTS
    TSM ->> TVM: [COVG] sbi_covg_get_device_measurement()
    TVM ->> TVM: Verify the device measurements with cert chain
    TVM ->> TVM: Device attestation (Local or remote)

note over TVM,TSM: Check Device Interface MMIO mappings
    TVM ->> TVM: Re-use previously fetched TDISP interface report
    loop For all TDISP reported MMIO ranges
         TVM ->> TSM: [COVG] sbi_covg_map_interface_mmio(dev_id, gpa, offset_hpa, size)
         TSM ->> TSM: Compare with the VMM donated MMIO regions (sbi_covh_add_tvm_interface_region)
         TSM ->> TSM: Compare with the RP IDE ranges
         TSM ->> TVM: [COVG] sbi_covg_map_interface_mmio()
    end
```

Listing 6. Device Interface Acceptation

```
%%{init: {'theme': 'neutral', 'themeVariables': {'darkMode': true}, "flowchart" : { "curve" : "basis" } }
}%%
sequenceDiagram

autonumber

participant TDI as Device Interface
```

```
participant DSM as Device DSM
participant VMM as Host Supervisor Domain Manager (VMM)
participant TSM
participant TVM

note over TDI: CONFIG_LOCKED
note over TVM: Device interface verified

VMM ->> TSM: [COVH] sbi_covh_run_vcpu()

TVM ->> TSM: [COVG] sbi_covg_start_interface()
TSM ->> TSM: Generate TDISP START_INTERFACE_REQ(LOCK_Nonce)
TSM ->> VMM: [COVH] spdm_req(START_INTERFACE_REQ)
VMM ->> DSM: [DOE] SPDM_START_INTERFACE_REQ
DSM ->> TDI: START
note over TDI: RUN
DSM ->> VMM: [DOE] SPDM_START_INTERFACE_RESP
VMM ->> TSM: [COVH] spdm_resp(START_INTERFACE_RESP)
TSM ->> TSM: Decrypt START_INTERFACE_RESP
TSM ->> TSM: Store device state
TSM ->> TSM: Enable IOMMU translation for devif_id
TSM ->> TSM: Enable G-stage MMIO mappings
TSM ->> TVM: [COVG] sbi_covg_start_interface()
TVM ->> TSM: [COVG] sbi_covg_get_interface_state()
TSM ->> TVM: [COVG] sbi_covg_get_interface_state()
TVM ->> TVM: Check that the interface state is RUN
TVM ->> TVM: Use device interface
```

7.6. Interface Unbinding

7.7. Device and Interface Lifecycle

When combined together, the flows and ABIs described in the previous sections are used to build the lifecycle of a TDISP capable device on a CoVE-IO compatible platform, as illustrated in the follwing figure:

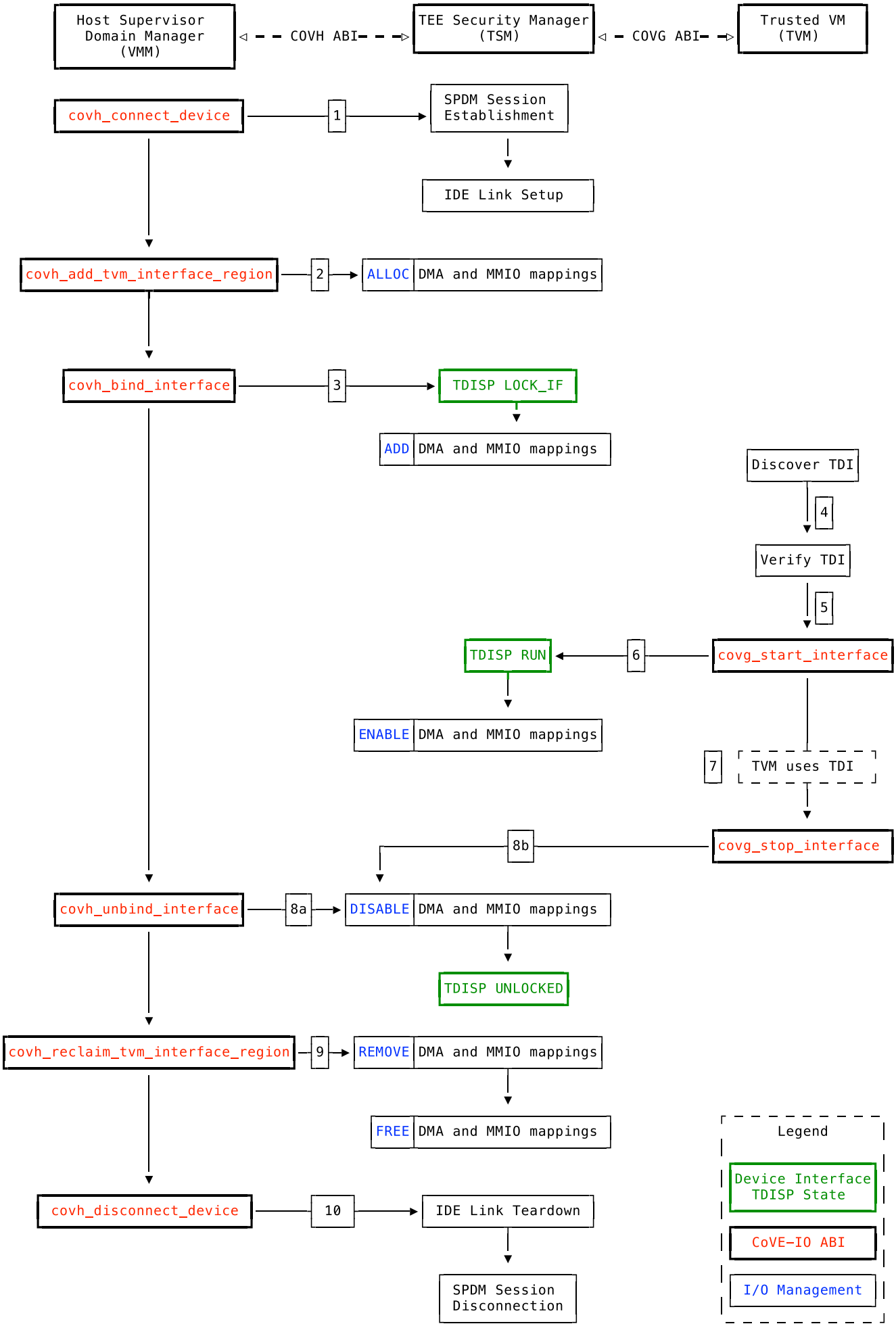

Figure 7. Device and Interface Lifecycle

The host supervisor domain manager owns the physical device, and manages its lifecycle. The TSM enforces that this lifecycle management is done without compromising any TVM confidential assets.

As the platform resources owner, the host supervisor domain manager can assign a TDI to a TVM by binding them together (step 3). At any point in time, it can reclaim that physical resource by unbinding (step 8a) it from its TVM.

Before binding a TDI and a TVM together, the host supervisor domain manager must first require the TSM to connect (step 1) to the physical device through secured SPDM. As part of servicing that request, the TSM also protects the physical link with PCIe IDE.

The host supervisor domain manager is also required to explictly add the TDI MMIO regions to the TDI (step 2). The TSM can prepare and allocate the TVM second stage page tables and map those I/O regions into the guest physical address space. The TSM does not enable those tables until the TVM starts the interface (step 6)

Only once the TSM is securily connected to the physical device, The host supervisor domain manager can proceed with binding a device interface and a TVM together. Once bound to a TVM, the device interface is locked but the MMIO and DMA paths between the two are not enabled yet.

The TVM to which a TDI is bound to is the I/O gatekeeper. After detecting and verifying (steps 4 and 5) the bound interface, it may accept it into its TCB and enable all I/O paths between the two parties. The TVM may only use a bound TDI (step 7) after accepting it, by requesting the TSM to start the device interface (step 6).

Both the host supervisor domain manager and the TVM can disable I/O between the TDI and the TVM, by respectively unbinding (step 8a) or stopping the device interface (step 8b). The host supervisor domain manager may first remove the TDI MMIO regions from the TVM address space (step 9).

Finally, the host supervisor domain manager can fully reclaim the physical device by requesting the TSM to disconnect (step 10) from it.

Chapter 8. Device Attestation

There are two attestation stages involved in the CoVE-IO architecture.

- TVM attests the device
- A third party attests the TVM with the device

The IETF RATS Remote Attestation Architecture [RATS] can be applied to both.

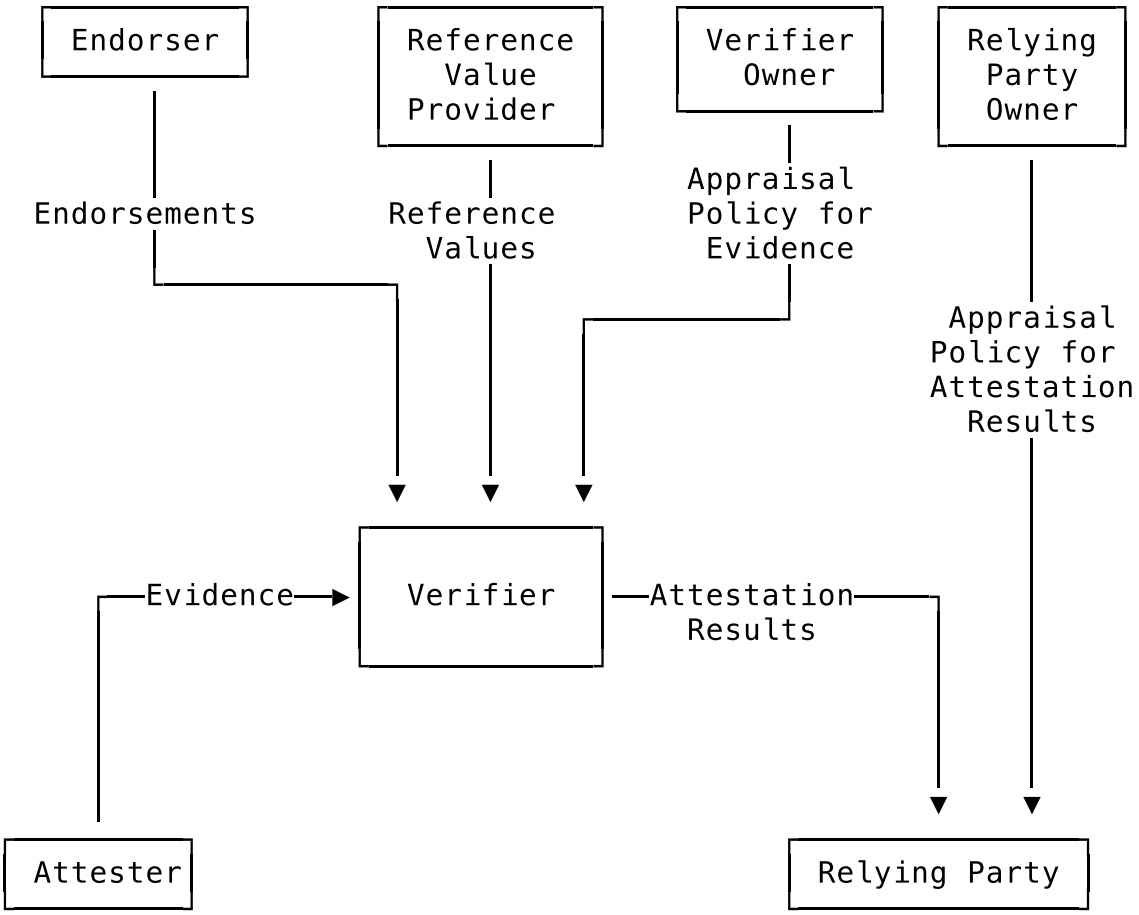

Figure 8. Remote Attestation Framework (IETF RATS)

8.1. TVM attesting the Device

The TVM must verify and trust the device before offloading the workload to the TDI. According to the [TDISP] architecture, the high level steps for that stage are as follows:

1. The TSM collects the device certificate and measurement from the DSM as the evidence, via SPDM protocol.
2. The TSM presents the evidence to the TVM.
3. A TVM verifier appraises the evidence and produce the attestation result to the relying party in the TVM, such as device driver.
4. The local relying part running in the TVM accepts (or rejects) the device into (from) its TCB.

The above process follows the background-check model, as defined by [RATS]. The [RATS] passport model is not supported by the CoVE-IO attestation architecture, because it adds complexity to the

TSM as the TSM would need to perform the verification and pass the attestation result to TVM. Since the TSM is in the TVM's TCB, its attack surface should be kept minimal.

In step 3, the verifier needs to perform the verification for the device. There are two possible use cases:

8.1.1. Local Verification

With local verification, the endorsement and reference values are provisioned into the TVM.

Static provisioning means the data is built into the TVM image at build time, while dynamic provisioning means the data is provisioned into the TVM at runtime.

The endorsement may include the device root CA certificate and its associated certificate revocation list (CRL). The reference value includes the reference integrity manifest (RIM), also known as golden measurement.

When running locally, the verifier in the TVM validates the device certificate according to the device root CA certificate and CRL, and then compares the device measurement with the measurement from its locally provisioned RIM.

The TVM may have a local policy update mechanism to update the endorsement and reference value. How to trigger the update is out of scope of this specification. It could be a pull model, such as TVM querying for updates periodically. Or it could be a push mode, such as orchestrator posting the update when it notices the device vendor update. To maintain the integrity of the policy update, the update package should have protection mechanism, such as digital signature.

8.1.2. Remote Verification

When the device is verified remotely, the endorsement and reference values are not provisioned into the TVM, but are accessible from an externally available service, also known as a relying party for the device.

The verifier itself can either run locally or be hosted on a remote service as well, typically available through an HTTPS-based protocol. It always collects the up-to-date endorsement and RIM from the device relying party.

Use Case	Local Verification	Remote Verification
Endorsement (Root CA Cert, CRL)	Provisioned in TVM (Static or Dynamic)	Remove Service
Reference Value (RIM)	Provisioned in TVM (Static or Dynamic)	Remove Service
Policy Update (Endorsement, Reference)	No update or scheduled update (Pull or Push)	Always up-to-date
Verifier Location	Inside of TVM	Inside of TVM or Remove Service

Table 13. Device Attestation Comparison

8.1.3. Device Runtime Update and Reattestation

A TEE-IO device may support runtime update. Currently, there are two policy settings controlling device runtime update.

SPDM Session `TerminationPolicy`

This SPDM session attribute controls if runtime update will cause SPDM session termination or device may keep the session.

> **Note** The session termination means the IDE stream will change to `insecure` state and TDI in `ERROR` state.

The SPDM `TerminationPolicy` is set in SPDM `KEY_EXCHANGE` message by TSM. The TSM should return such policy information to the TVM via `sbi_covg_get_device_spdm_attrs()` `COVG` ABI.

TDISP `NO_FW_UPDATE`

This TDISP interface attribute controls if runtime update is allowed after TDI in either `CONFIG_LOCKED` or `RUN` state.

> **Note** Runtime update is always allowed if TDI is in `CONFIG_UNLOCKED` or `ERROR`.

The TDISP `NO_FW_UPDATE` is set in TDISP `LOCK_INTERFACE_REPORT` message by TSM, and is returned in BITO of `INTERFACE_INFO` in `TDI Report Structure` by the device. The TVM can get the full `TDI Report Structure` via the `sbi_covg_get_interface_report()` `COVG` ABI.

The TVM should consult both setting to decide if it can accept the device runtime update.

If the TVM allows the device runtime update after lock (`NO_FW_UPDATE`=0) and update without session termination (`TerminationPolicy`=1), then the TVM may want to do attestation again to verify the device.

There are 2 possible models.

Post-update reattestation

After the device performs the update, the TVM will collect both the certificate nd the measurement and do verification again to determine if it still wants to accept the device. If the TVM decides to no longer accept the device any more, the TVM will use `sbi_covg_stop_interface()` ABI to request the VMM to use `sbi_covh_unbind_interface()` to unbind the TDI, then the TVM will verify with TSM if the TDI is really unbound by the VMM.

How the device performs the update is out of scope of this specification. It could be initiated by current TVM, other TVMs, other legacy VMs, VMM, or even out of band (OOB) mechanism, such as Baseboard Management Controller (BMC).

When the TVM performs the reattestation is also out of scope of this specification. It could be a pull mode with policy inside of TVM, such as reattestation every week. Or it could be an interrupt mode such as request triggered by the orchestrator.

Pre-update attestation

With post-update reattestation, there is a gap between update time and attestation time. Confidential data access from the TDI may happen before the updated device is re-attested and re-accepted by the TVM.

SPDM 1.3 added an optional event mechanism and `MeasurementPreUpdate` event notification. If the device, TSM, and TVM all support SPDM 1.3, the TVM may get the `MeasurementPreUpdate` event notification and verify the measurement before the update takes effect. If the TVM cannot accept the new measurement, then the TVM can unbind the TDI before runtime update.

The difference between this pre-update attestation and post-update reattestation is that the update takes effect when the verification happens. Pre-update attestation gives a chance to unbind TDI before runtime update, while post-update reattestation can only unbind TDI after runtime update. That can remove the gap between update time and attestation time.

8.2. Third party attesting the TVM with the device

The third party must follow the CoVE defined attestation mechanism to perform local or remote attestation and verify the TVM.

The CoVE-IO architecture adds the additional device related information to the TVM attestation evidence, in order to allow the third party to verify the attached device TDI.

Depending on the verification process being local or remote, the TVM needs to add different pieces of device-related information to the TVM report:

8.2.1. Local Verification

The TVM report should include the device verifier code, the device policy, including the provisioned device root CA certificate, CRL, and RIM.

If the TVM supports device policy update, then the secure signed update mechanism should be implemented. In that case, the signer of the update data shall be included in the TVM report as the trust anchor.

The device measurement and certificate are not required in the TVM report, however the TVM should provide a mechanism to return the device measurement and certificate to the verifier for further verification.

8.2.2. Remote Verification

The TVM report should include measurements of either the device verifier or the proxy connecting to the verifier service. It should also include the remote service URL and the public certificate as the trust anchor.

The detailed, up-to-date endorsement or reference value is **not** required to be part of the TVM report.

Although the device measurement and certificate are not required to be included in the TVM report, the TVM should provide a mechanism to return the device measurement and certificate for the verifier to perform further verification.

Use Case	Local Verification	Remote Verification
Device Verifier Code in TVM report	TVM Verifier	TVM Verifier or Stub Function to Verifier Service
Device Policy Data in TVM report	Endorsement and Reference Value	Remote Service URL and Public Cert (trust anchor)
Device Policy Update in TVM report	Signer of Update (as trust anchor)	N/A
Device identity NOT in TVM report	Device Measurement and Certifcate	Device Measurement and Certifcate

Table 14. TVM Attestation Comparison

Chapter 9. Confidential VM Extension (CoVE) IO ABI

9.1. CoVE IO Host Extension

9.1.1. IOMMU Management

Function: CoVE Host Register IOMMU (FID TBD)

```
struct sbiret sbi_covh_register_iommu(unsigned long iommu_id,
                                      unsigned long msi_cfg[3]);
```

Registers an IOMMU (`iommu_id`) with a TSM. The `msi_cfg` MSI vectors are allocated by the host security domain manager (e.g. the host VMM).

The TSM configures the IOMMU TSM security domain programming interface `s_msi_cfg_table` with the provided MSI vectors.

The possible error codes returned in `sbiret.error` are shown below.

Error code	Description
SBI_SUCCESS	The operation completed successfully.
SBI_ERR_INVALID_PARAMS	The `iommu_id` index is invalid.
SBI_ERR_FAILED	The operation failed for unknown reasons.

Table 15. CoVE Host Register IOMMU

Function: CoVE Host Notify IOMMU MSI (FID TBD)

```
struct sbiret sbi_covh_notify_iommu_msi(unsigned long iommu_id);
```

Notifies the TSM about a pending MSI for the TSM security domain.

When servicing this request, the TSM must first verify that there are pending MSIs, by reading the IOMMU `s_ipsr` register from the IOMMU TSM security domain programming interface. If there are some, it handles the interrupts as needed.

The possible error codes returned in `sbiret.error` are shown below.

Error code	Description
SBI_SUCCESS	The operation completed successfully.
SBI_ERR_INVALID_PARAMS	The `iommu_id` index is invalid.
SBI_ERR_FAILED	The operation failed for unknown reasons.

Table 16. CoVE Host Notify IOMMU MSI

9.1.2. Root Port Management

Function: CoVE Host Register PCIe Root Port (FID TBD)

```
struct sbiret sbi_covh_register_rp(unsigned long rp_id,
                                   unsigned long mmio_ranges);
```

Registers a PCIe Root Port (RP) for TEE I/O and IDE ownership with the TSM.

This function associates a RP id to its MMIO space (for configuring the RP IDE capability) and all the MMIO ranges that are routed through it.

The TSM must compare these 2 arguments with the information it received from the platform ROT through the TEE I/O manifest. If both match, it can proceed into establishing an SPDM session with the RP.

The possible error codes returned in `sbiret.error` are shown below.

Error code	Description
SBI_SUCCESS	The operation completed successfully.
SBI_ERR_INVALID_PARAMS	The `rp_id` index is invalid.
SBI_ERR_FAILED	The operation failed for unknown reasons.

Table 17. CoVE Host Register PCIe Root Port

9.1.3. Physical Device Management

Function: CoVE Host Connect Device (FID TBD)

```
struct sbiret sbi_covh_connect_device()(unsigned long device_id,
                                        unsigned long stream_id);
```

Establishes a secure SPDM session between the TSM and a device identified by `device_id`, and configure the PCIe IDE link between that device and its upstream Root Port. The configured IDE link must use the selective IDE stream identified by `stream_id`.

The possible error codes returned in `sbiret.error` are shown below.

Error code	Description
SBI_SUCCESS	The operation completed successfully.
SBI_ERR_INVALID_PARAMS	The `device_id` index is invalid.
SBI_ERR_FAILED	The operation failed for unknown reasons.

Table 18. CoVE Host Connect Device

Function: CoVE Host Disconnect Device (FID TBD)

```
struct sbiret sbi_covh_disconnect_device()(unsigned long device_id);
```

Disconnects the TSM from the device identified by `device_id`.

The TSM must first stop all TVM-bound interfaces belonging to `device_id`, clear the IDE link and keys for `device_id` and then terminate the secure SPDM session established with `device_id`.

The possible error codes returned in `sbiret.error` are shown below.

Error code	Description
SBI_SUCCESS	The operation completed successfully.
SBI_ERR_INVALID_PARAMS	The `device_id` index is invalid.
SBI_ERR_FAILED	The operation failed for unknown reasons.

Table 19. CoVE Host Disconnect Device

9.1.4. TVM Memory Management

Function: CoVE Host Add TVM Interface Region (FID TBD)

```
struct sbiret sbi_covh_add_tvm_interface_region(unsigned long tvm_id,
                                                unsigned long device_if_id,
                                                unsigned long base_addr,
                                                unsigned long num_pages);
```

Adds a device interface MMIO region into a TVM address space.

The host supervisor domain manager must call this function for all `device_if_id` MMIO regions before binding it to `tvm_id`. This function returns an error if `device_if_id` is already bound to `tvm_id`.

The TSM creates uncacheable G-stage mappings for the added region, so that the TVM outbound transaction will not trigger MMIO page faults. The TSM must not enable the created mappings until the TVM accepts the TDI in its TCB.

The possible error codes returned in `sbiret.error` are shown below.

Error code	Description
SBI_SUCCESS	The operation completed successfully.
SBI_ERR_FAILED	The operation failed for unknown reasons.

Table 20. CoVE Host Add TVM Interface Region

Function: CoVE Host Reclaim TVM Interface Region (FID TBD)

```
struct sbiret sbi_covh_reclaim_tvm_interface_region(unsigned long tvm_id,
                                                    unsigned long device_if_id,
                                                    unsigned long base_addr,
                                                    unsigned long num_pages);
```

Reclaims a device interface MMIO region previously added to a TVM address space.

The host supervisor domain manager may call this function when the `tvm_id` TVM stops the `device_if_id` device interface, in order for the TSM to unmap the device interface MMIO ranges from the TVM address space.

If `device_if_id` is still bound to `tvm_id` when this call is made, the TSM unbinds the device interface from the TVM first.

The possible error codes returned in `sbiret.error` are shown below.

Error code	Description
SBI_SUCCESS	The operation completed successfully.
SBI_ERR_FAILED	The operation failed for unknown reasons.

Table 21. CoVE Host Reclaim TVM Interface Region

9.1.5. Device Interface Management

Function: CoVE Host Bind Interface (FID TBD)

```
struct sbiret sbi_covh_bind_interface()(unsigned long tvm_id,
                                        unsigned long device_if_id);
```

Binds a TVM and a device interface together.

The TSM returns an error if a secured SPDM session is not established with the DSM or if a stream IDE link is not set up.

After this calls completes successfully, the `device_id_interface` is in the TDISP `CONFIG_LOCKED` state.

The possible error codes returned in `sbiret.error` are shown below.

Error code	Description
SBI_SUCCESS	The operation completed successfully.
SBI_ERR_FAILED	The operation failed for unknown reasons.

Table 22. CoVE Host Bind Interface

Function: CoVE Host Unbind Interface (FID TBD)

```
struct sbiret sbi_covh_unbind_interface()(unsigned long tvm_id,
                                          unsigned long device_if_id);
```

Unbinds a device interface from a TVM.

After this calls completes successfully, the `device_id_interface` is moved back to the TDISP `CONFIG_UNLOCKED` state, from one of the `CONFIG_LOCKED`, `CONFIG_UNLOCKED` or `CONFIG_ERROR` states.

The possible error codes returned in `sbiret.error` are shown below.

Error code	Description
SBI_SUCCESS	The operation completed successfully.
SBI_ERR_FAILED	The operation failed for unknown reasons.

Table 23. CoVE Host Unbind Interface

9.2. CoVE IO Guest Extension

9.2.1. Physical Device Query

Function: CoVE Guest Get Device Link (FID TBD)

```
struct sbiret sbi_covg_get_device_link(unsigned long device_if_id);
```

Gets the status of the link between the physical device hosting `device_if_id` and the TVM. This covers both the SPDM and IDE links.

Returns the a link status bitmap through `sbiret.value`.

```
// A secure SPDM Session is established.
#define SECURED_SPDM (1 << 0)

// A PCIe IDE link is established.
#define PCIE_IDE (1 << 1)
```

The possible error codes returned in `sbiret.error` are shown below.

Error code	Description
SBI_SUCCESS	The operation completed successfully.
SBI_ERR_FAILED	The operation failed for unknown reasons.

Table 24. CoVE Guest Get Device Link

Function: CoVE Guest Get Device Certificate (FID TBD)

```
struct sbiret sbi_covg_get_device_certificate(unsigned long device_if_id,
                                              unsigned long slot_id,
                                              unsigned long cert_addr_out,
                                              unsigned long cert_size);
```

Get the certificate chain for the physical device hosting `device_if_id` based on `slot_id`.

The TSM returns the certificate chain in the form of the [SPDM] defined `certificate chain format`.

The TVM calls this function in order to verify the authenticity of the physical device. A TVM must not accept a device interface without doing that verification first.

`slot_id` must be between 0 and 7 inclusive.

`cert_addr_out` must be page aligned.

The possible error codes returned in `sbiret.error` are shown below.

Error code	Description
SBI_SUCCESS	The operation completed successfully.
SBI_ERR_FAILED	The operation failed for unknown reasons.

Table 25. CoVE Guest Get Device Certificate

Function: CoVE Guest Get Device Measurements (FID TBD)

```
struct sbiret sbi_covg_get_device_measurements(unsigned long device_if_id,
                                               unsigned long nonce_addr,
                                               unsigned long msmt_req_attr,
                                               unsigned long msmt_addr_out,
                                               unsigned long msmt_size);
```

Gets the measurements of the physical device hosting `device_if_id`.

The TSM returns the measurements in the form of the complete, signed [SPDM] measurement transcript, including `VCA` and all `{GET_MEASUREMENTS, MEASUREMENTS}` pairs that are exchanged between the SPDM measurement requester and the responder. Only the last `MEASUREMENTS` shall include the digital signature of the measurement transcript.

The `nonce_addr` parameter points at an optional 32 bytes long buffer holding a cryptographic nonce.

For any non zero value, the nonce is used as the SPDM `GET_MEASUREMENTS` request `Nonce` field. When set to `0x0`, the TSM ignores this argument and generates a nonce on behalf of the TVM.

`msmt_req_attr` is used as the measurement request attributes in SPDM `GET_MEASUREMENT` request `param1` field. Only `RawBitStreamRequested` bit is valid and the rest bits are igored. The last `GET_MEASUREMENT` request must set `SignatureRequested` bit to request the digital signaure of the measurement transcript.

Both `msmt_addr_out` and `nonce_addr` must be page aligned.

The possible error codes returned in `sbiret.error` are shown below.

Error code	Description
SBI_SUCCESS	The operation completed successfully.
SBI_ERR_FAILED	The operation failed for unknown reasons.

Table 26. CoVE Guest Get Device Measurements

Function: CoVE Guest Get Device SPDM Attributes (FID TBD)

```
struct sbiret sbi_covg_get_device_spdm_attrs(unsigned long device_if_id,
                                             unsigned long spdm_attrs_addr_out,
                                             unsigned long spdm_attrs_size);
```

Gets the attributes for the Secure SPDM session between the physical device hosting `device_if_id` and the TSM.

`spdm_attrs_addr_out` must be page aligned.

```
struct SPDMAttributes {
       bool measurement_freshness;
}
```

The possible error codes returned in `sbiret.error` are shown below.

Error code	Description
SBI_SUCCESS	The operation completed successfully.
SBI_ERR_FAILED	The operation failed for unknown reasons.

Table 27. CoVE Guest Get Device SPDM Attributes

9.2.2. Device Interface Management

Function: CoVE Guest Get Interface Report (FID TBD)

```
struct sbiret sbi_covg_get_interface_report(unsigned long device_if_id
                                            unsigned long if_report_out,
                                            unsigned long if_report_size);
```

Gets the TDISP interface report for the device interface.

The TSM returns the interface report, as defined by the [TDISP] TDI Report Structure, at the `if_report_out` address.

`if_report_out` must be page aligned.

The possible error codes returned in `sbiret.error` are shown below.

Error code	Description
SBI_SUCCESS	The operation completed successfully.
SBI_ERR_FAILED	The operation failed for unknown reasons.

Table 28. CoVE Guest Get Interface Report

Function: CoVE Guest Get Interface State (FID TBD)

```
struct sbiret sbi_covg_get_interface_state(unsigned long device_if_id);
```

Gets the TDISP state for the device interface.

The TVM calls this function to verify that a bound device interface is in the TDISP `RUN` state.

Returns the interface state through `sbiret.value`.

```
enum Interface State {
     /* TDISP CONFIG_UNLOCKED */
     Unlocked,

     /* TDISP CONFIG_LOCKED */
     Locked,

     /* TDISP RUN */
     Running,

     /* TDISP ERROR */
     Error,
};
```

The possible error codes returned in `sbiret.error` are shown below.

Error code	Description
SBI_SUCCESS	The operation completed successfully.
SBI_ERR_FAILED	The operation failed for unknown reasons.

Table 29. CoVE Guest Get Interface State

Function: CoVE Guest Map Interface MMIO (FID TBD)

```
struct sbiret sbi_covg_map_interface_mmio(unsigned long device_if_id
                                          unsigned long gpa_addr,
                                          unsigned long hpa_addr,
                                          unsigned long size);
```

Maps a TVM MMIO region (from `gpa_address`, `size` bytes long) to a `TDISP`-reported physical region (`hpa_addr`).

The TVM uses that function to verify from the TSM that all the device interface MMIO regions exposed by the host security domain manager are correctly mapped to the trusted `TDISP`-reported MMIO regions. The TSM will enable those mappings if the TVM calls the starts the device interface through the `sbi_covg_start_interface` function.

All of `gpa_addr`, `hpa_addr` and `size` values must be page aligned.

The possible error codes returned in `sbiret.error` are shown below.

Error code	Description
SBI_SUCCESS	The operation completed successfully.
SBI_ERR_FAILED	The operation failed for unknown reasons.

Table 30. CoVE Guest Map Interface MMIO

Function: CoVE Guest Start Interface (FID TBD)

```
struct sbiret sbi_covg_start_interface(unsigned long device_if_id);
```

Starts a bound device interface.

The TVM calls this function in order to accept a bound device interface into its TCB. While servicing this request, the TSM moves the device interface TDISP state from `CONFIG_LOCKED` to `RUN`.

After this calls completes successfully, the device interface I/O is ready and available for the bound TVM.

The possible error codes returned in `sbiret.error` are shown below.

Error code	Description
SBI_SUCCESS	The operation completed successfully.
SBI_ERR_ALREADY_STARTED	The device interface is already started.
SBI_ERR_FAILED	The operation failed for unknown reasons.

Table 31. CoVE Guest Start Interface

Function: CoVE Guest Stop Interface (FID TBD)

```
struct sbiret sbi_covg_stop_interface(unsigned long device_if_id);
```

Stops a bound device interface.

The TVM calls this function for either removing a bound device interface from its TCB or initially

rejecting it.

After this calls completes successfully, the device interface and the TVM are no longer bound together.

The possible error codes returned in `sbiret.error` are shown below.

Error code	Description
SBI_SUCCESS	The operation completed successfully.
SBI_ERR_ALREADY_STOPPED	The device interface is already stopped.
SBI_ERR_FAILED	The operation failed for unknown reasons.

Table 32. CoVE Guest Stop Interface

References

- [CoVE] RISC-V Confidential VM Extension, github.com/riscv-non-isa/riscv-ap-tee
- [CC] Confidential Computing, confidentialcomputing.io/about/
- [PCI-SIG] PCI SIG, pcisig.com/
- [DMTF] Distributed Management Task Force, www.dmtf.org/
- [SPDM] Secure Protocol and Data Model Specification, www.dmtf.org/dsp/DSP0274
- [SecuredSPDM] Secured Messages using SPDM Specification, www.dmtf.org/dsp/DSP0277
- [PCIE] PCI Express Base Specification, members.pcisig.com/wg/PCI-SIG/document/18363
- [TDISP] TEE Device Interface Protocol ECN, members.pcisig.com/wg/PCI-SIG/document/18268?uploaded=1
- [IOMMU] RISC-V IOMMU, github.com/riscv-non-isa/riscv-iommu/blob/main/riscv-iommu.pdf
- [RATS] IETF RFC 9334 Remote ATtestation procedureS (RATS) Architecture, datatracker.ietf.org/doc/rfc9334/
- [Smmtt] RISC-V Supervisor Domain Access Protection, github.com/riscv/riscv-smmtt