Intel® TDX Module Architecture Specification: TD Migration

DRAFT

348550-002

January 2023

Notices and Disclaimers

Table of Contents

Section 1: Introduction and Overview

SECTION 1: TD MIGRATION INTRODUCTION AND OVERVIEW

1. About this Document

1.1. Scope of this Document

This document describes the architecture and the external Application Binary Interface (ABI) of the Intel® Trust Domain Extensions (Intel® TDX) module's Live Migration feature, implemented using the Intel TDX Instruction Set Architecture (ISA) extensions, for cold or live migration of Trust Domains in an untrusted hosted cloud environment.

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

Table 1.1: TDX Module Architecture Specification Set

Document Name	Reference	Description
TDX Module **Base Architecture Specification**	[TDX Module Base Spec]	Base TDX module architecture overview and specification, covering key management, TD lifecycle management, memory management, virtualization, measurement and attestation, service TDs, debug aspects etc.
→ **TDX Module** **TD Migration Architecture Specification**	[TD Migration Spec]	Architecture overview and specification for TD migration
TDX Module **ABI Reference Specification**	[TDX Module ABI Spec]	Detailed TDX module Application Binary Interface (ABI) reference specification, covering the entire TDX module architecture

This document is a work in progress and is subject to change based on customer feedback and internal analysis. This document does not imply any product commitment from Intel to anything in terms of features and/or behaviors.

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

1.2. Document Organization

The document has two main sections:

- Section 1 contains an introduction to the document, overview of TD Migration, scenarios and requirements.
- Section 2 contains the Intel TDX Module Migration architecture

The detailed reference specification of TD Migration data structures and interface functions is provided in the [TDX Module ABI Spec].

1.3. Glossary

For a complete TDX module glossary, see the [TDX Module Base Spec].

Table 1.2: Intel TDX Module Glossary for TD Migration

Acronym	Full Name	New for TDX	Description
MigTD	**Migration TD**	Yes	A specific type of **Service TD** used to provide Live Migration capability for TD VMs. A Migration TD extends the TCB of the serviced tenant TD.

Acronym	Full Name	New for TDX	Description
MSK	**Migration Session Key**	Yes	AES-GCM-256 key generated by the source MigTD and shared with the destination MigTD (protected by the Migration Transport key). This key helps protect the TD private data and is used for export and import of the TD confidential assets.
MTK	**Migration Transport Key**	Yes	Authenticated Diffie-Helman negotiated symmetric key generated after mutual attestation of the MigTDs and is used to help protect the transport of the Migration Session Key from the source to the destination platform.

1.4. Notation

See the [TDX Module Base Spec].

1.5. References

1.5.1. Intel Public Documents

Table 1.3: Intel Public Documents

Reference	Document	Version & Date
Intel SDM	Intel® 64 and IA-32 Architectures Software Developer's Manual	June 2021
ISA Extensions	Intel® Architecture Instruction Set Extensions and Future Features Programming Reference	May 2021

1.5.2. Intel TDX Public Documents

Table 1.4: Intel TDX Public Documents

Reference	Document	Version & Date
TDX Whitepaper	Intel Trust Domain Extensions Whitepaper	August 2020
Intel TDX Spec	Intel® Architecture Trust Domain Extensions (TDX) Specification	Rev. 1.0, August 2020
MKTMEi Spec	Intel® Architecture Memory Integrity Specification	Rev. 1.0, March 2020
TDX Module Base Spec	Intel TDX Module Base Architecture Specification	September 2021
TDX Module ABI Spec	Intel TDX Module ABI Spec Reference Specification	September 2021

1.5.3. Non-Intel Public Documents

Table 1.5: Non-Intel Public Documents

Reference	Document	Version & Date
AES-256-GCM	NIST Special Publication 800-38D: Recommendation for Block Cipher Modes of Operation: Galois/Counter Mode (GCM) and GMAC	November 2007

2. TD Migration Overview

For an overview of TDX, refer to the [TDX Module Base Spec].

2.1. Introduction

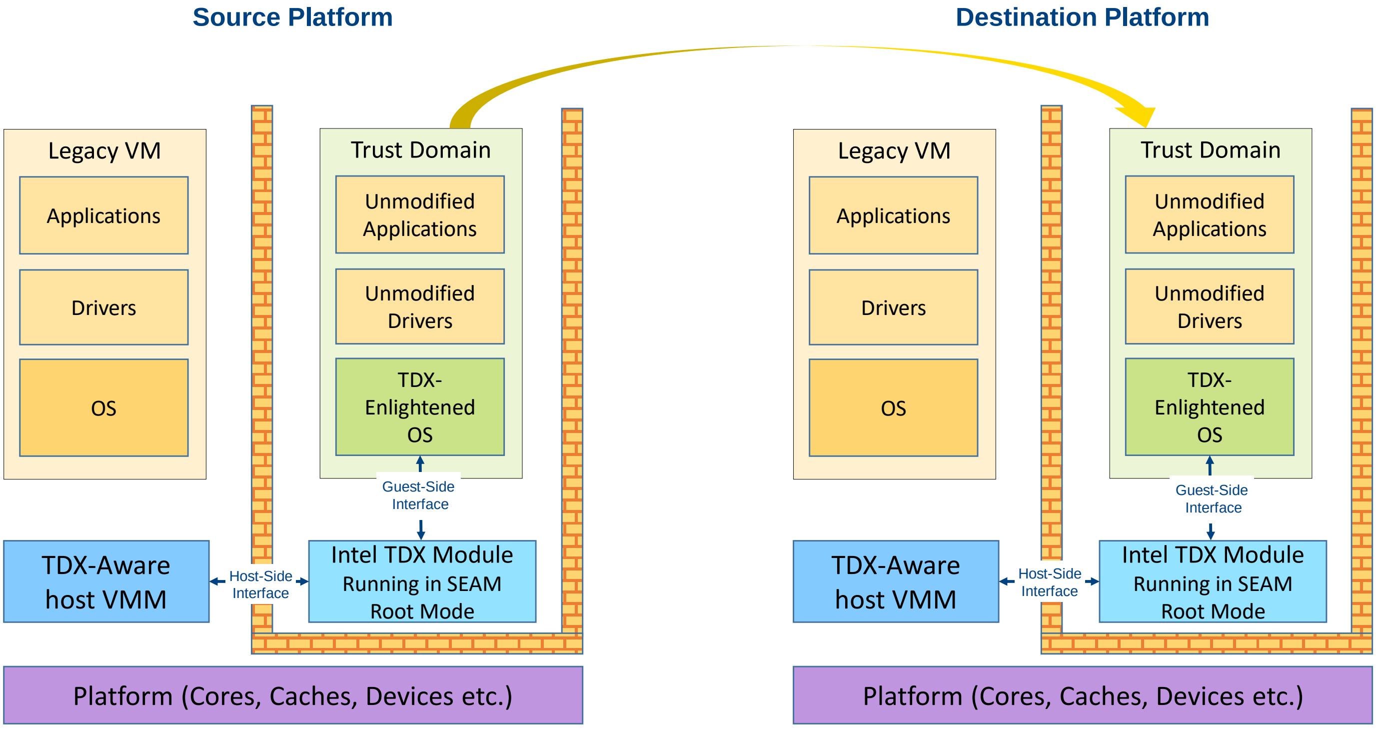

Figure 2.1: TD Migration

Analogous to legacy VM migration, a cloud-service provider (CSP) may want to relocate/migrate an executing Trust Domain from a **source TDX platform** to a **destination TDX platform** in the cloud environment. A cloud provider may use TD migration to meet customer SLA, while balancing cloud platform upgradability, patching and other serviceability requirements. Since a TD runs in a CPU mode which helps protect the confidentiality of its memory contents and its CPU state from any other platform software, including the hosting Virtual Machine Monitor (VMM), this primary security objective must be maintained while allowing the TD resource manager, i.e., the host VMM to migrate TDs across compatible platforms. The TD typically may be assigned a different HKID (and will be always assigned a different ephemeral key) on the destination platform chosen to migrate the TD.

In this specification, the TD being migrated is called the **source TD**, and the TD created as a result of the migration is called the **destination TD**. An extensible **TD Migration Policy** is associated with a TD that is used to maintain the TD's security posture. The TD Migration policy is enforced in a scalable and extensible manner using a specific type of **Service TD** called the **Migration TD (a.k.a. MigTD)** (introduced in the Figure 2.2 below) – which is used to provide services for migrating TDs.

The TD Live Migration process (and the Migration TD) does not depend on any interaction with the TD guest software operating inside the TD being migrated.

2.2. TD Migration Scenarios

This section describes the usage scenarios addressed by this specification (and those explicitly out of scope). This specification documents the TD Migration functionality from a Live Migration (scenario described below) perspective. Cold Migration and other scenarios described below are effectively subset scenarios that are software managed via the Intel TDX module interface functions in this specification.

2.2.1. Cold migration

Cold migration with destination known and resumed such that both ends must be alive to do the handoff

- TD image is <u>suspended</u> during migration and resumed after a duration >> TCP timeout

- Useful for rolling upgrades/patch + rebooting servers (non-reboot patches can be done without migrating the TD), capacity planning and load balancing.

2.2.2. Live Migration

Live migration with destination known and resumed such that both ends must be alive to do the handoff

- TD executing during migration and paused for duration << TCP timeout
- Customer SLA, capacity planning/load balancing

A TD may be live migrated more than once using multiple sessions.

2.2.3. Image Snapshot and Jumpstart

Pre-built Image/Jumpstart with destination unknown and TD image stored for an indeterminate amount of time.

This usage has additional platform security requirements that are not comprehended in this specification. Example use cases are saving checkpoints of TDs such that TD may be pre-loaded into memory. Alternate implementations to satisfy this usage are possible. E.g., the TD could un-hibernate the image itself. **This scenario/use case is out of scope for this specification**.

2.3. Components Involved in TD Migration

The main components involved in TD migration are:

- The TD being migrated
- Host VMMs on the source and destination platforms
- Migration TDs on the host and destination platforms

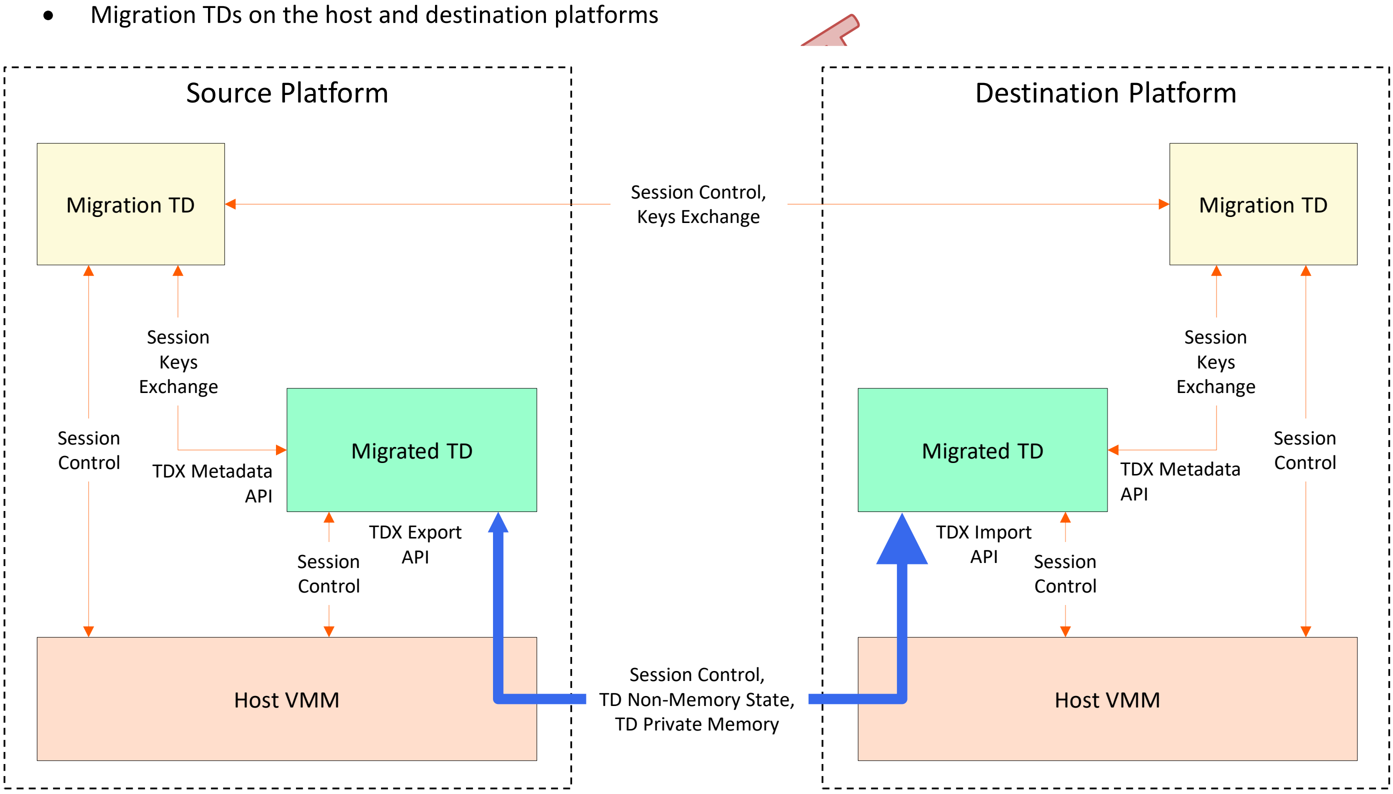

Figure 2.2: Component Involved in TD Migration

A **Migration TD (MigTD)** is used to evaluate potential migration sources and targets for adherence to the TD Migration Policy. The TD Migration policy enumerates TDX platform TCB requirements as well as acceptable destination Migration TD TCB levels.

If TCB levels are acceptable, the Migration TDs on the source and destination platforms securely exchange unique per-session **Migration Session Keys (MSKs)** which are used to migrate assets of a specific TD. The MigTD on each side reads an encryption key generated by the TDX module, as TD metadata, and transfers it to the MigTD on the other side. That MigTD writes the key, as TD metadata, as the decryption key for its side.

The host VMM may bind a MigTD to one or more TDs being migrated. For details, see [TDX Module Base Spec]'s Service TDs chapter. Since the MigTD is in the TCB of the TD being migrated, a MigTD must be pre-bound to the target TD being migrated before the target TD measurement is finalized. The MigTD lifecycle does not have to be coincidental with the target TD – the MigTD may be instantiated when required for Live Migration, but it must be bound to the target TD before Live Migration can begin, and must be operational until the migration session keys has been successfully programmed for the target TD being migrated. The MigTD measurements are in included in the target TD's attestation information structures.

The host VMM, via the Intel TDX Module, is responsible for export/import of the TD content, and the transport of the protected TD content to the destination platform.

2.4. Migrated Assets

The table below shows the TD assets that are migrated. Metadata includes TD-scope and VCPU-scope non-memory state (such as control state, CPU register state etc.) and memory attributes (such as GPA and access permissions). Metadata is not migrated as-is; it is serialized into a migration format and re-created on the destination platform.

Table 2.1: Migrated TD Assets

TD Asset	Where Held	Export Functions	Import Functions
Immutable Non-Memory State (Metadata)	TDX module global TDR TDCS	TDH.EXPORT.STATE.IMMUTABLE	TDH.IMPORT.STATE.IMMUTABLE
Mutable Non-Memory State (Metadata)	TDCS TDVPS	TDH.EXPORT.STATE.TD TDH.EXPORT.STATE.VP	TDH.IMPORT.STATE.TD TDH. IMPORT.STATE.VP
Memory State and Metadata	TD private pages Secure EPT	TDH.EXPORT.MEM	TDH. IMPORT.MEM

2.5. Guest TD Migration Life Cycle Overview

2.5.1. Pre-Migration

2.5.1.1. *Intel TDX Module Enumeration*

The host VMM calls the TDH.SYS.RD or TDH.SYS.RDALL interface function to enumerate Intel TDX Module functionality and learns from the TDX_FEATURES that the Intel TDX Module supports TD Migration. The host VMM learns details of TD migration capabilities and service TD capabilities from the other fields.

2.5.2. Reservation and Session Setup

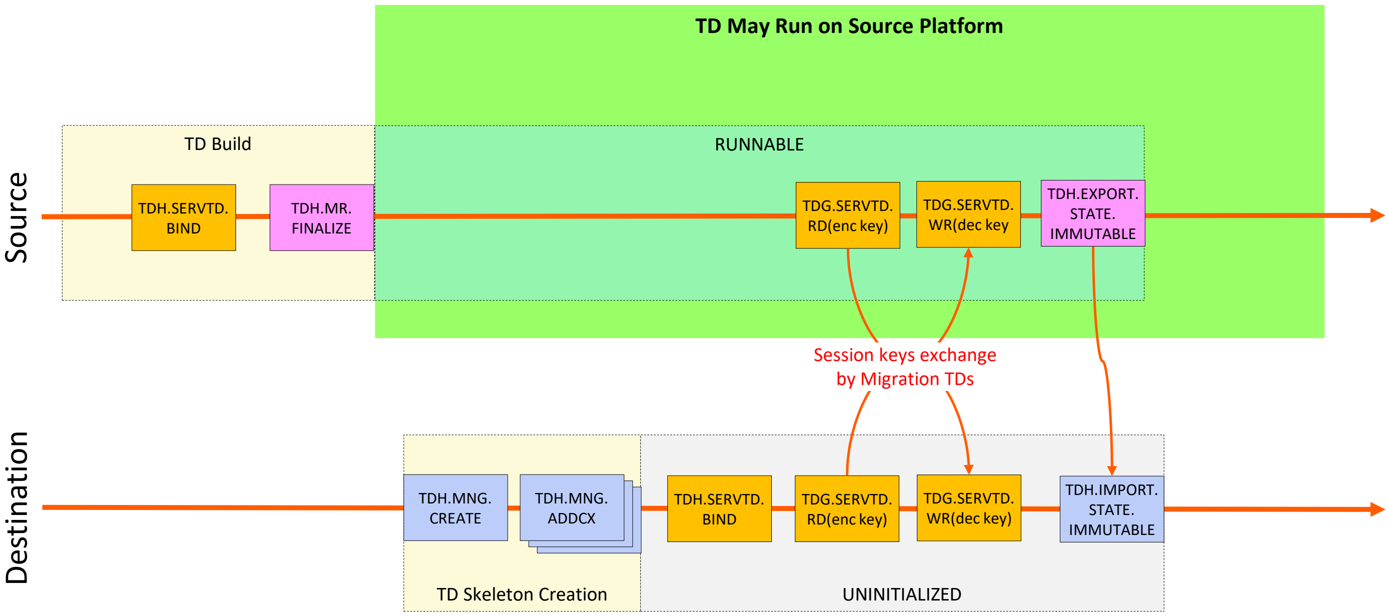

Figure 2.3: Migration TD Binding and Session Setup

2.5.2.1. *Guest TD Build, Migration TD Binding and TD Execution on the Source Platform*

The source TD build and execution process is described in the [TDX Module Base Spec]. To be migratable, the TD may be initialized, using the TDH.MNG.INIT function, with **ATTRIBUTES.MIGRATABLE** bit set to 1.

Before a migration session can begin, the host VMM on the source platform must use TDH.SERVTD.BIND to bind a Migration TD to the source TD. The MigTD may read selected metadata fields of the source TD, e.g., its ATTRIBUTES and XFAM configuration, to be used in evaluating a migration policy.

2.5.2.2. *Guest TD Initial Build on the Destination Platform*

Same as a legacy TD build process, the host VMM creates a new guest TD by using the TDH.MNG.CREATE interface function. This destination TD is setup as a “template” to receive the state of the Source Guest TD. The host VMM programs the HKID and the HW-generated encryption key assigned to the TD into the MKTME encryption engines using the TDH.MNG.KEY.CONFIG interface function on each package. The host VMM can then continue to build the TDCS by adding TDCS pages using the TDH.MNG.ADDCX interface function.

Once the destination TDCS is built and before TD import can begin, the VMM on the destination platform must use TDH.SERVTD.BIND to bind a Migration TD to the destination TD.

2.5.2.3. *Migration Session Key Negotiation*

The Migration session keys are an ephemeral AES-256-GCM keys used for confidentiality and integrity protection of the TD private state exported from the source platform and imported on the destination platform, and for integrity protections of the migration session control protocol. TD shared memory state is migrated by the untrusted host VMM per legacy methods – the same network transport may be used for both by the host VMM. The Migration Session Key (MSK) is established by a Migration TD which is responsible for evaluation of the Migration policy for the TD being migrated.

The migration TDs executing on the source and destination platforms use a TD-quote-based mutual authentication protocol to create a VMM-transport-agnostic session between them. The Migration TDs negotiate a protected transport session (using Diffie-Hellman exchange). Using this protected transport session, the migration policy can be evaluated by the Migration TD.

The Service TD binding mechanism supported by the TDX module allows the Migration TD to access target TD metadata – specifically the Migration session keys. The MigTD can access the TD metadata using TDG.SERVTD.RD/WR* guest-side interface functions.

The TDX modules on both the source and destination platforms generate ephemeral **migration session encryption keys**. The Migration TDs on each side uses the Service TD interface read that key and securely transfer it to the peer Migration TD, which uses the Service TD interface to write it as the **migration session decryption key**.

After this point, the host VMM can invoke TDX Module functions such as TDH.EXPORT.* to export state at the source platform and TDH.IMPORT.* to import TD state at the destination platform. The protocol is described in Ch. 5.5 in detail.

2.5.3. In-Order Memory Migration Phase

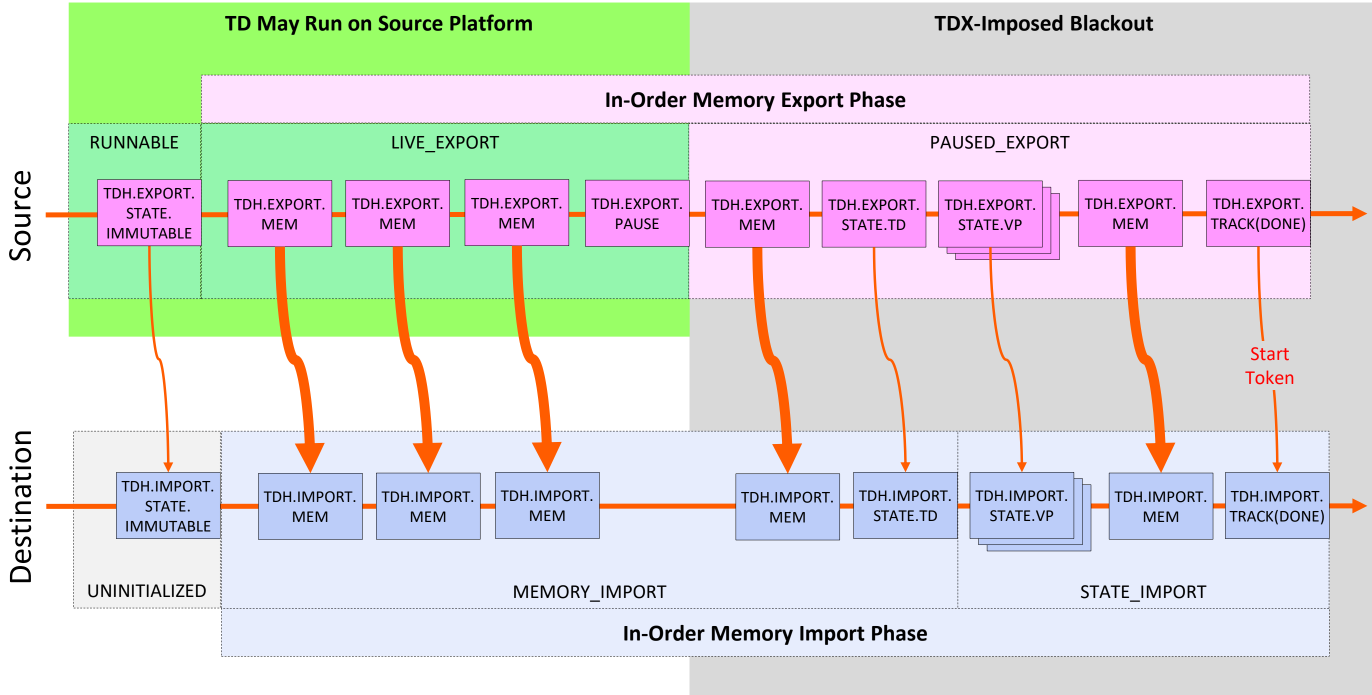

Figure 2.4: In-Order Migration Phase

2.5.3.1. *TD Global Immutable Metadata (Non-Memory State) Migration*

Intel TDX Module protects the confidentiality and integrity of a guest TD global state. Control structures, which hold guest TD metadata, are not directly accessible to any software (besides the Intel TDX Module) or devices. These structures are stored encrypted and integrity-protected in memory with the TD private key and managed by Intel TDX Module interface functions.

Immutable metadata is the set of TD state variables that are set by TDH.MNG.INIT, may be modified during TD build but are never modified after the TD's measurement is finalized using TDH.MR.FINALIZE. Some of these state variables control how the TD and its memory is migrated. Therefore, the immutable TD control state is migrated before any of the TD memory state is migrated.

TD immutable state is exported via the TDH.EXPORT.STATE.IMMUTABLE interface function and imported on the destination platform via the TDH.IMPORT.STATE.IMMUTABLE interface function. TD global immutable state migration is described in Ch. 7.

2.5.3.2. *Iterative Pre-Copy of Memory State*

2.5.3.2.1. Migration Considerations for TD Private Memory

Intel TDX helps protect guest TD state in private memory from a malicious VMM, using MKTME (memory encryption and integrity protection) and the Intel TDX Module. The Intel TDX Module performs ephemeral key id management to enforce the TDX security objectives. Memory encryption is performed by encryption engines that reside at each memory controller, with no software access (including the TDX module) to the ephemeral keys. The memory encryption engine holds a table of encryption keys, in the Key Encryption Table (KET). The encryption key selected for memory transactions is based on a Host Key Identifier (HKID) provided with the memory access transaction.

The Intel TDX Module API functions enable the host VMM to manage HKID assignment to guest TDs, configure the memory encryption engines etc., while assuring proper operation to maintain TDX's security objectives. The host VMM also does not have access to the TD encryption keys.

TD Migration does not migrate the HKIDs – a free HKID is assigned to the TD created on the destination platform to receive migratable assets of the TD from the source platform. All TD private memory is protected during transport from the source platform to the destination platform using an intermediate encryption performed using AES-GCM 256 using the MSK negotiated via the Migration TDs on the source and destination platform. On the destination platform the memory is encrypted via the destination ephemeral key as it is imported into the destination platform memory assigned to the destination TD. The import operation on the destination TDX module verifies and decrypts the TD private data using the MSK; it uses the MKTME engine to encrypt (and integrity protect) while writing it to memory using the destination TD HKID.

During live migration, the source TD is allowed to modify private memory (until the source TD is paused by the host VMM to complete the last phase of migration). To allow this, TD private memory is migrated over a set of **Migration Epochs**. Migration epochs enforce TD Live migration security property **S4: CSP must not be able to operate the destination TD on any stale state from the source TD**. The host VMM may also instantiate multiple **migration streams** for memory state transfer (for example to leverage multiple host hardware threads) – as long as the security invariants are not violated. TD Private memory migration is described in Ch. 8 in detail.

Shared memory assigned to the TD is migrated using legacy mechanisms used by the host VMM.

Encryption-based memory protection is described in the [MKTME PAS] and the ISA is described in the [Intel TDX PAS]. TD migration has no change to TD key management when the Migration TD uses an independent HKID.

2.5.3.2.2. Migration Considerations for EPT Structures

Guest Physical Address (GPA) space is divided into private and shared sub-spaces, determined by the SHARED bit of GPA. The CPU translates shared GPAs using the Shared EPT, which resides in host VMM memory, and is directly managed by the host VMM, same as with legacy VMX. The CPU translates private GPAs using a separate Secure EPT. Secure EPT pages are encrypted and integrity protected with the TD's ephemeral private key.

As there is no guarantee of allocating the same physical memory addresses to the TD being migrated on the destination platform, **the memory used for Secure EPT structures is not migrated across platforms**. Hence, the VMM must invoke the TDX module's TDH.MEM.SEPT.* interface functions on the destination platform to re-create the private GPA mappings on the destination platform (per the assigned HPAs). The Intel TDX module uses the cryptographically protected exported meta-data (generated via TDH.EXPORT.MEM) to verify and enforce (via the TDH.IMPORT.MEM) that the Secure EPT security properties from the source platform are rec-created correctly as TD private memory contents are migrated, thus preventing remap attacks during migration. TD private memory migration is described in Ch. 8 in detail.

Even though Secure EPT structures are not migrated, the source SEPT structures track the state of the mappings when a page is exported and then modified by the TD OS in the pre-copy stage. The TD OS may be allowed to modify such a page and the TDX module enforces that the modified and previously exported page is re-exported by the source host VMM and re-imported by the destination host VMM.

2.5.3.3. *Source TD Stop and Final Non-Memory State Migration*

Following pre-copy of TD private memory, the host VMM must pause the source TD for a brief period (also called the blackout period) so that the VMM may export the final control state (for all VCPUs and for the TD overall). The VMM initiates this via TDH.EXPORT.PAUSE, which checks security pre-conditions and prevents TD VCPUs from executing any more. It then allows export of final (mutable) TD non-memory state.

2.5.3.4. *In-Order Memory State Re-Migration Completion*

Any memory state that has been migrated must be up to date when the in-order phase completes. If a memory page had been migrated and its content was later updated by the running TD, it must be re-migrated. The TDX Module enforces this using the commitment protocol described in 2.5.6 below.

2.5.3.5. *TD-Scope and VCPU-Scope Mutable Non-Memory State Migration*

TD mutable non-memory state is a set of source TD state variables that might have changed since it was finalized via TDH.MR.FINALIZE. Immutable non-memory state exists for the TD scope (as part of the TDR and TDCS control structures) and the VCPU scope (as part of the TDVPS control structure).

Mutable TD state is exported by TDH.EXPORT.STATE.TD (per TD) and TDXPORT.STATE.VP (per VCPU) and imported by TDH.IMPORT.STATE.TD and TDH.IMPORT.STATE.VP respectively. This is described in Ch. 7.

2.5.4. Out-Of-Order Memory Migration Phase

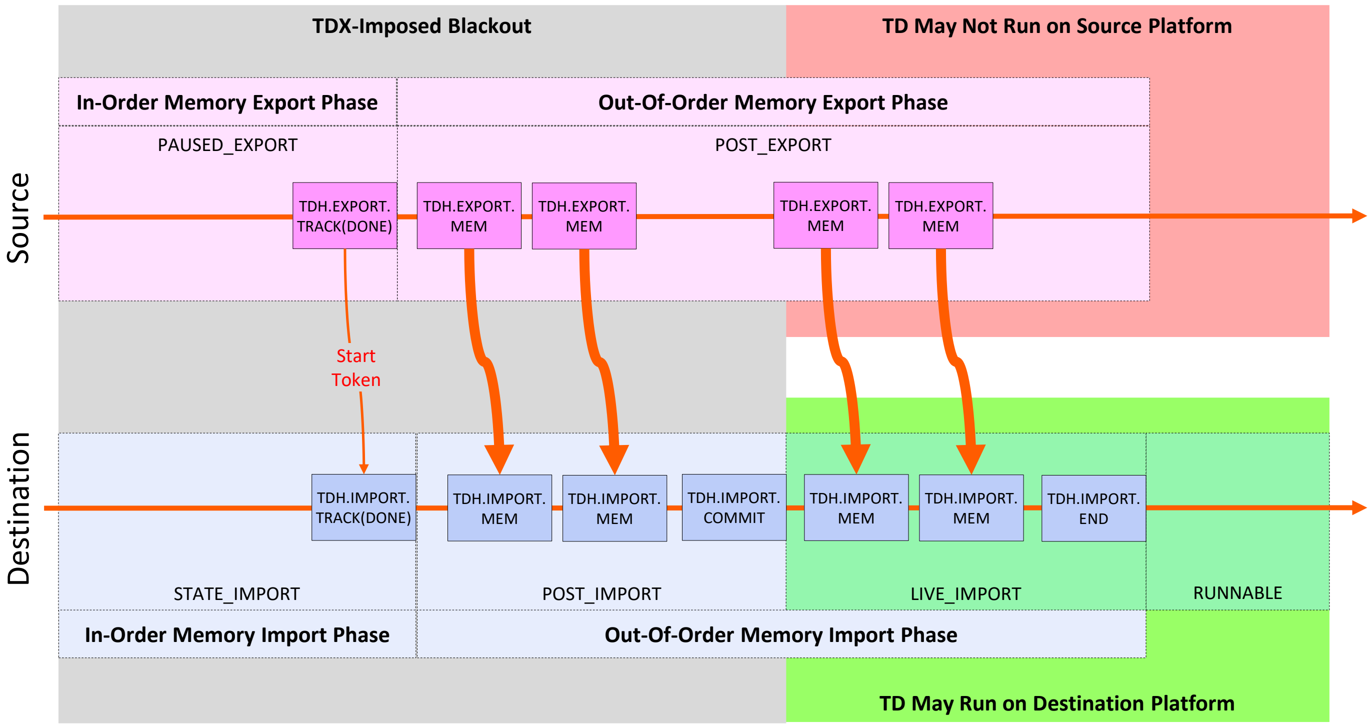

Figure 2.5: Out-Of-Order Migration Phase

2.5.4.1. *Migration of Memory State and Commitment of Import*

Memory pages that have not been migrated during the in-order phase may be migrated during the out-of-order phase. Since the memory state on the source platform does not change at this stage, the order of migration is not enforced.

The host VMM on the destination platform commits the import by calling TDH.IMPORT.COMMIT or TDH.IMPORT.END. At that point, the TD may run on the destination platform and may not run on the source platform.

2.5.4.2. *Post-Copy of Memory State*

In some live migration scenarios, the host VMM may stage some memory state transfer to occur lazily after the destination TD has started execution. In this case, the host VMM will be required to fetch the required pages as accesses occur by the destination TD – this order of access is indeterminate and will likely differ from the order in which the host VMM has queued memory state to be transferred.

In order to support that on-demand model, the order of memory migration during this **post-copy stage** is not enforced by TDX. The host VMM may implement multiple migration queues with multiple priorities for memory state transfer. For example, the host VMM on the source platform may keep a copy of each encrypted migrated page until it receives a confirmation from the destination that the page has been successfully imported. If needed, that copy can be re-sent on a high priority queue. Another option is, instead of holding a copy of exported pages, to call TDH.EXPORT.MEM again on demand.

Also, to simplify host VMM software for this model, the TDX module interface functions used for memory import in this post-copy stage return additional informational error codes to indicate that a stale import was attempted by the host VMM to account for the case where the low latency import operation for a GPA superseded the import from the higher latency import queue.

2.5.5. Aborted Private Memory Migration

In a live migration scenario, an error may cause CSP orchestration to abort an active TD live migration session. In such a scenario, the host VMM on the source platform may pro-actively initiate an abort via TDH.EXPORT.ABORT. It may also respond to an about token received from the destination platform, where it may be generated by TDH.IMPORT.ABORT for a late abort (after pre-copy has been completed). In both scenarios, the host VMM must reset the state of the SEPT state of exported pages on the source platform, using TDH.EXPORT.RESTORE.

2.5.6. Commitment

The commitment protocol is enforced by the Intel TDX Module to help ensure that a host VMM cannot violate the security objectives of TD Live migration – for example, both the destination and source TD must not continue to execute after live migration of the source TD to a destination TD, even if an error causes the TD migration to be aborted.

This protocol is enforced via the following TDX Module interface functions:

- On the source platform, TDH.EXPORT.PAUSE starts the blackout phase of TD live migration and TDH.EXPORT.TRACK ends the blackout phase of live migration (and marks the end of the transfer of TD memory pre-copy, mutable TD VP and mutable TD global control state). TDH.EXPORT.TRACK generates a MSK-based cryptographically-authenticated start token to allow the destination TD to become runnable. On the destination platform, TDH.IMPORT.TRACK – which consumes the cryptographic start token, allows the destination TD to be un-paused.
- In error scenarios, the migration process may be aborted proactively by the host on the source platform via TDH.EXPORT.ABORT before a start token was generated; if a start token was already generated (i.e. pre-copy completed), the destination platform can generate an abort token using TDH.IMPORT.ABORT which generates an abort token which may be consumed by TDH.EXPORT.ABORT by the source TD platform TDX Module to abort the migration process and again allows the source TD to become runnable again.

The detailed operations are described in Ch. 5.5.

2.6. Impact of Migration on Measurement and Attestation

TD measurement is extended for the MigTD bound to the TD being migrated, and the ATTRIBUTES.MIGRATABLE bit is part of the TD attestation. For details, see the [TDX Module Base Spec].

2.7. Intel TDX Module Managed Control Structures affected by Migration

Intel TDX Module manages a set of control structures that are not directly accessible to untrusted host software. The control-structures are protected in memory using encryption and integrity (with TDX private keys). Most control structures are in memory assigned to the TD by the host VMM. The following table describes the impact of Migration on the TD control structures. See the detailed definition of these structures in the [TDX Module ABI Spec].

Table 2.2: TDX-Managed Control Structures

Scope	Name	Meaning	Migration Impact
Platform	KOT	Key Ownership Table	None
	PAMT	Physical Address Metadata Table	PAMT.BEPOCH is used to hold migration epoch information
Guest TD	TDR	Trust Domain Root	None
	TDCS	Trust Domain Control Structure	TD ATTRIBUTES field has a new MIGRATABLE Security attribute that must be set for a TD to be migratable. Some state is initialized (same as legacy) and some state is imported via TDH.IMPORT.STATE.IMMUTABLE and TDH.IMPORT.STATE.TD. TDCS has new Migration stream context structures associated with the TD setup via TDH.MIG.STREAM.CREATE.
	SEPT	Secure EPT	SEPT entry state is much extended to support tracking of memory export and import by the TDX module.

Scope	Name	Meaning	Migration Impact
	TDINFO_ STRUCT	TD measurement	A new field SERVTD_HASH is added, see the [TDX Module Base Spec] for details.
Guest TD VCPU	TDVPS	Trust Domain Virtual Processor State	Some state is initiated and some state Imported via Intel TDX Module API TDH.IMPORT.STATE.VP for migrating VCPU control state

There are new data structures introduced for TD Migration that are generated by the TDX-module (and managed by the VMM).

Table 2.3: TDX-Generated Control Structures

Name	Meaning	Migration impact
MBMD	Migration Bundle Metadata	Common header and type information for migrated information
GPA_LIST	GPA list of migrated pages	List of GPAs and associated attributes, used for memory migration and related memory operations
MIGRATION_BUFFER_ LIST	Migration buffer list	List of migration buffers provided by the host VMM to hold migration data

2.8. Intel TDX Module TD Migration Interface Functions Overview

See the [TDX Module Base Spec]'s overview chapter.

3. TD Migration Software Flows

This chapter summarizes the software flows used for TD migration using Intel TDX Module interface functions.

3.1. Typical TD Migration Flow Overview

Source Platform				Destination Platform
1. Create Worker Threads, disable device optimizations				4. Receive memory layout and device resource information
2. Save memory layout and device resource information				5. Allocate guest memory and reserve resources on devices
3. Send memory layout and resource information to the destination host	Send Migration Information	→	Receive Migration Information	
				6. Setup worker process and signal the completion to source host
	TDH.SERVTD.BIND		TDH.SERVTD.BIND	
	Establish DH session between MigTDs using QE	←→	Establish DH session between MigTDs using QE	
7. MigTDs exchange session keys, version information etc.	TDG.SERVTD.RD, TDG.SERVTD.WR	←→	TDG.SERVTD.RD TDG.SERVTD.WR	8. MigTDs exchange session keys, version information etc.
	TDH.EXPORT.STATE. IMMUTABLE	→	TDH.IMPORT.STATE. IMMUTABLE	
	Memory Pre-Export		Memory Pre-Import	
	TDH.EXPORT.BLOCKW			
	TDH.MEM.TRACK			
9. Run multiple passes of memory walker to transfer dirty guest-memory pages	TDH.EXPORT.MEM	→	TDH.IMPORT.MEM	10. Import TD memory contents
	TDH.EXPORT.UNBLOCKW			
	TDH.EXPORT.TRACK	→	TDH.IMPORT.TRACK	
	TDH.EXPORT.PAUSE			
	TDH.EXPORT.STATE.TD	→	TDH.IMPORT.STATE.TD	
11. Export mutable state	TDH.EXPORT.STATE.VP	→	TDH.IMPORT.STATE.VP	12. Import mutable state
	TDH.EXPORT.TRACK(start token)	→	TDH.IMPORT.TRACK(start token)	
13. Export memory	TDH.EXPORT.MEM	→	TDH.IMPORT.MEM	14. Import memory
	15. (optional) Send end of export indication	→	16. (optional) Receive end of export indication	
			TDH.IMPORT.COMMIT	
				17. Resume VPs
		←	18. (optional) Request post-copy of pages	
19. (optional) Export memory (post-copy)	TDH.EXPORT.MEM	→	TDH.IMPORT.MEM	
			TDH.IMPORT.END	
	Receive migration success indication	←	Send migration success indication	
	Indicate success to VM Management		Indicate success to VM Management	
20. Tear-down source TD				

Figure 3.1: Typical TD Migration Flow

3.2. Successful TD Export

The following sequence is typically used to export a TD from a source platform.

Table 3.1: Typical TD Export Sequence

Migration Phase	Step	Description	Plurality	TDX Module Interface Functions
In-Order Export	**Start of Export Session**	VMM initializes MigTD (as a non-migratable TD) and binds it to the source TD.	Once	TDH.SERVTD.BIND
		VMM/orchestration sets up transport session between source and destination MigTD. MigTDs setup their own protected channel.	Once	
		MigTDs reads the session encryption key, version information and other metadata from the source TD and sends it to the MigTD on the destination.	Once	TDG.SERVTD.RD
		MigTD receives the session encryption key from the MigTD on the destination, and writes it as the session decryption key to the source TD. It may also write the migration version.	Once	TDG.SERVTD.WR
		VMM starts the export session and exports immutable state creating a state migration bundle.	Once	TDH.EXPORT.STATE.IMMUTABLE
	Live Memory Export	Host VMM blocks a set of pages for writing.	Multiple	TDH.EXPORT.BLOCKW
		Host VMM increments the TD's TLB epoch	Once per migration epoch	TDH.MEM.TRACK
		Host VMM starts migration epoch and creates epoch token migration bundle; a page can be exported once per epoch.	Once per migration epoch	TDH.EXPORT.TRACK(epoch token)
		Host VMM exports, re-exports or cancels the export of TD private pages and creates a memory migration bundle.	Multiple	TDH.EXPORT.MEM
		TD write attempt to write to page blocked for writing results in an EPT violation. The host VMM unblocks the page; if already exported, it will need to be re-blocked and re-exported.	Multiple	TDH.EXPORT.UNBLOCKW
	Mutable Non-Memory State Export	VMM pauses the source TD	Once	TDH.EXPORT.PAUSE
		VMM exports mutable TD-scope state and creates a state migration bundle.	Once	TDH.EXPORT.STATE.TD
		VMM exports mutable VCPU-scope state and creates a state migration bundle.	Per VCPU	TDH.EXPORT.STATE.VP
Out-Of-Order Export	**Cold Memory Export**	Host VMM starts the out-of-order export phase and creates a start token migration bundle.	Once	TDH.EXPORT.TRACK(start token)

Migration Phase	Step	Description	Plurality	TDX Module Interface Functions
		Host VMM exports TD private pages and creates a memory migration bundle.	Multiple	TDH.EXPORT.MEM
TD Teardown	**End**	Host VMM gets success notification from the destination platform, terminates the export session and tears down the TD on the source platform	Once	TDH.MNG.VPFLUSHDONE TDH.PHYMEM.CACHE.WB TDH.MNG.KEY.FREEID TDH.PHYMEM.PAGE.RECLAIM TDH.PHYMEM.PAGE.WBINVD

3.3. Successful TD Import

The following sequence is typically used to import a TD to a destination platform.

Table 3.2: Typical TD Import Sequence

Migration Phase	Step	Description	Plurality	TDX Module Interface Functions
In-Order Import	**Start of Import Session**	VMM creates the destination TD's skeleton	Once	TDH.MNG.CREATE TDH.MNG.KEY.CONFIG TDH.MNG.ADDCX
		VMM initializes MigTD (as a non-migratable TD) and binds it to the destination TD.	Once	TDH.SERVTD.BIND
		VMM/orchestration sets up transport session between source and destination MigTD. MigTDs setup their own protected channel.	Once	
		MigTDs reads the session encryption key, version information and other metadata from the destination TD and sends it to the MigTD on the source.	Once	TDG.SERVTD.RD
		MigTD receives the session encryption key from the MigTD on the source, and writes it as the session decryption key to the destination TD. It may also write the migration version.	Once	TDG.SERVTD.WR
		VMM starts the import session and imports immutable state with a state migration bundle received from the source platform.	Once	TDH.IMPORT.STATE.IMMUTABLE
	Pre-Copy Memory Import	Host VMM builds the Secure EPT by allocating physical pages.	Multiple	TDH.MEM.SEPT.ADD
		Host VMM imports TD private pages with a memory migration bundle received from the source platform.	Multiple	TDH.IMPORT.MEM
		Host VMM starts migration epoch with an epoch token migration bundle received from the source platform; a page can be imported once per epoch.	Once per migration epoch	TDH.IMPORT.TRACK(epoch token)

Migration Phase	Step	Description	Plurality	TDX Module Interface Functions
	Mutable TD-scope and VCPU-scope non-memory state import	VMM imports mutable TD-scope state with a state migration bundle received from the source platform.	Once	TDH.IMPORT.STATE.TD
		VMM creates VCPU.	Per VCPU	TDH.VP.CREATE
		VMM allocates physical pages for the VCPU's TDVPS.	Multiple per VCPU	TDH.VP.ADDCX
		VMM imports mutable VCPU-scope state with a state migration bundle received from the source platform.	Per VCPU	TDH.IMPORT.STATE.VP
Out-Of-Order Import	**Pre-Copy Memory Import**	Host VMM starts the out-of-order import phase with a start token migration bundle received from the source platform.	Once	TDH.IMPORT.TRACK(start token)
		Host VMM builds the Secure EPT by allocating physical pages.	Multiple	TDH.MEM.SEPT.ADD
		Host VMM imports TD private pages with a memory migration bundle received from the source platform.	Multiple	TDH.IMPORT.MEM
	Post-Copy Memory Import	Host VMM commits the import session, allowing the TD to run on the destination platform.	Once	TDH.IMPORT.COMMIT
		Host VMM can execute the TD as usual. Memory can be imported on demand.	Per VCPU	TDH.VP.ENTER
		On EPT violation, host VMM requests a page import from the source platform.	Multiple	N/A
		Host VMM builds the Secure EPT by allocating physical pages.	Multiple	TDH.MEM.SEPT.ADD
		Host VMM imports TD private pages with a memory migration bundle received from the source platform.	Multiple	TDH.IMPORT.MEM
	End	Host VMM terminates the import session	Once	TDH.IMPORT.END

3.4. TD Import Abort

The following sequences are typically used to abort an import of TD to a destination platform, if an error is detected.

3.4.1. TD Import Abort During the In-Order Import Phase

Table 3.3: Typical TD Import Sequence Abort During In-Order Input

Migration Phase	Step	Description	Plurality	TDX Module Interface Functions
In-Order Import				
		Host VMM builds the Secure EPT by allocating physical pages.	Multiple	TDH.MEM.SEPT.ADD

Migration Phase	Step	Description	Plurality	TDX Module Interface Functions
	Pre-Copy Memory Import (Failed)	Host VMM imports TD private pages with a memory migration bundle received from the source platform. TDH.IMPORT.MEM returns an error status indicating a failed import session.	Multiple	TDH.IMPORT.MEM
	Abort Token Transmission (Optional)	VMM creates an abort token and transmits it to the source platform.	Once	TDH.IMPORT.ABORT
TD Teardown	**End**	Host VMM terminates the import session and tears down the TD on the destination platform	Once	TDH.MNG.VPFLUSHDONE TDH.PHYMEM.CACHE.WB TDH.MNG.KEY.FREEID TDH.PHYMEM.PAGE.RECLAIM TDH.PHYMEM.PAGE.WBINVD

3.4.2. TD Import Abort During the Out-Of-Order Import Phase

Table 3.4: Typical TD Import Sequence Abort During Out-of-Order Input

Migration Phase	Step	Description	Plurality	TDX Module Interface Functions
In-Order Import				
Out-Of-Order Import	**Memory Import (Failed)**			
		Host VMM imports TD private pages with a memory migration bundle received from the source platform. TDH.IMPORT.MEM returns an error status indicating a failed import session.	Multiple	TDH.IMPORT.MEM
	Abort Token Transmission	VMM creates an abort token and transmits it to the source platform.	Once	TDH.IMPORT.ABORT
TD Teardown	**End**	Host VMM terminates the import session and tears down the TD on the destination platform	Once	TDH.MNG.VPFLUSHDONE TDH.PHYMEM.CACHE.WB TDH.MNG.KEY.FREEID TDH.PHYMEM.PAGE.RECLAIM TDH.PHYMEM.PAGE.WBINVD

5 ## 3.5. TD Export Abort

The following sequence is typically used to export a TD from a source platform, if the export is aborted.

3.5.1. Export Abort During the In-Order Export Phase

Table 3.5: Typical TD Export Sequence Abort During In-Order Export

Migration Phase	Step	Description	Plurality	TDX Module Interface Functions

Migration Phase	Step	Description	Plurality	TDX Module Interface Functions
In-Order Export	**Start of Export Session**	VMM starts the export session and exports immutable state creating a state migration bundle.	Once	TDH.EXPORT.STATE.IMMUTABLE
	Export Abort	Host VMM aborts the export session.	Once	TDH.EXPORT.ABORT
Source TD Run and Restore		Host VMM may run and manage the source TD	Multiple	TDH.VP.ENTER etc.
		Host VMM restores SEPT entries to their normal non-export state	Multiple	TDH.EXPORT.UNBLOCKW TDH.EXPORT.RESTORE

3.5.2. Export Abort During the Out-Of-Order Export Phase

Table 3.6: Typical TD Export Sequence Abort During Out-Of-Order Export

Migration Phase	Step	Description	Plurality	TDX Module Interface Functions
In-Order Export				
Out-Of-Order Export				
	Export Abort	Host VMM receives an abort token from the destination platform and abort the export session.	Once	TDH.EXPORT.ABORT
Source TD Run and Restore		Host VMM may run and manage the source TD	Multiple	TDH.VP.ENTER etc.
		Host VMM restores SEPT entries to their normal non-export state	Multiple	TDH.EXPORT.UNBLOCKW TDH.EXPORT.RESTORE

5

SECTION 2:
TD MIGRATION ARCHITECTURE SPECIFICATION

4. Migration TD

This chapter provides a short introduction to the Migration TD and its roles. For details of the Intel reference MigTD, refer to the [MigTD Spec].

4.1. Migration Policy

A Migration TD implements a migration policy; details are beyond the scope of this document. Inputs for migration policy evaluation are:

- TDX module attestation information, obtained using TDG.MR.REPORT.
- TDX module capabilities enumeration, including migration protocol version, obtained using TDG.SYS.RD/RDALL.

4.2. TD Configuration by a Migration TD

Note: The text in this section applies to the next release of the TDX module.

A Migration TD that is bound to a certain target TD can configure the virtual value of CPU Family/Model/Stepping that the target TD can enumerate by executing CPUID(1). This is designed to enable migration between dissimilar platforms, in case Family/Model/Stepping information is important (e.g., if the TD software uses it to enumerate CPU functionality that is not enumerated in the standard ways using CPUID and RDMSR).

To configure the virtual value of CPUID(1).EAX:

- The target TD must still be in its build phase and its measurement must not have been finalized by TDH.MR.FINALIZE.
- The MigTD must be bound to the target TD by the host VMM (using TDH.SERVTD.BIND).
- The MigTD can write a new value of virtual CPUID(1).EAX using TDG.SERVTD.WR. It needs to provide the correct field ID for writing EBX/EAX values of CPUID(0), and a write mask of 0x00000000FFFFFFFF. For metadata field ID structure details refer to the [ABI Spec].
- If the configured Family and Model values are the same as those of the underlying CPU, then the configured stepping ID may not be higher than the underlying CPU's stepping ID.

4.3. Migration Version Setup

Before starting a migration session, the MigTDs on the source and destination should agree on migration protocol version that is supported by both sides. To do so, each MigTD can read the following fields using TDG.SYS.RD:

- MIN_EXPORT_VERSION
- MAX_EXPORT_VERSION
- MIN_IMPORT_VERSION
- MAX_IMPORT_VERSION

The MigTD on each side then should write the MIG_VERSION to the target TD using TDG.SERVD.WR.

4.4. Migration Keys Exchange

1. The MigTD on each side reads the migration encryption key (MIG_ENC_KEY), which is randomly generated by the TDX module, using TDG.SERVTD.RD.
2. The MigTD on each side sends the key value over a secure channel to the peer MigTD on the other side.
3. The MigTD on each side writes the received key value as the migration decryption key (MIG_DEC_KEY), using TDG.SERVTD.WR.

4.5. Example Migration Session Establishment

The goal is to stablish secure transport channel between Intel TDX Modules and MigTDs on both sides across compatible platforms and reserve resources for the migration session.

Section 2: TD Migration Architecture Specification

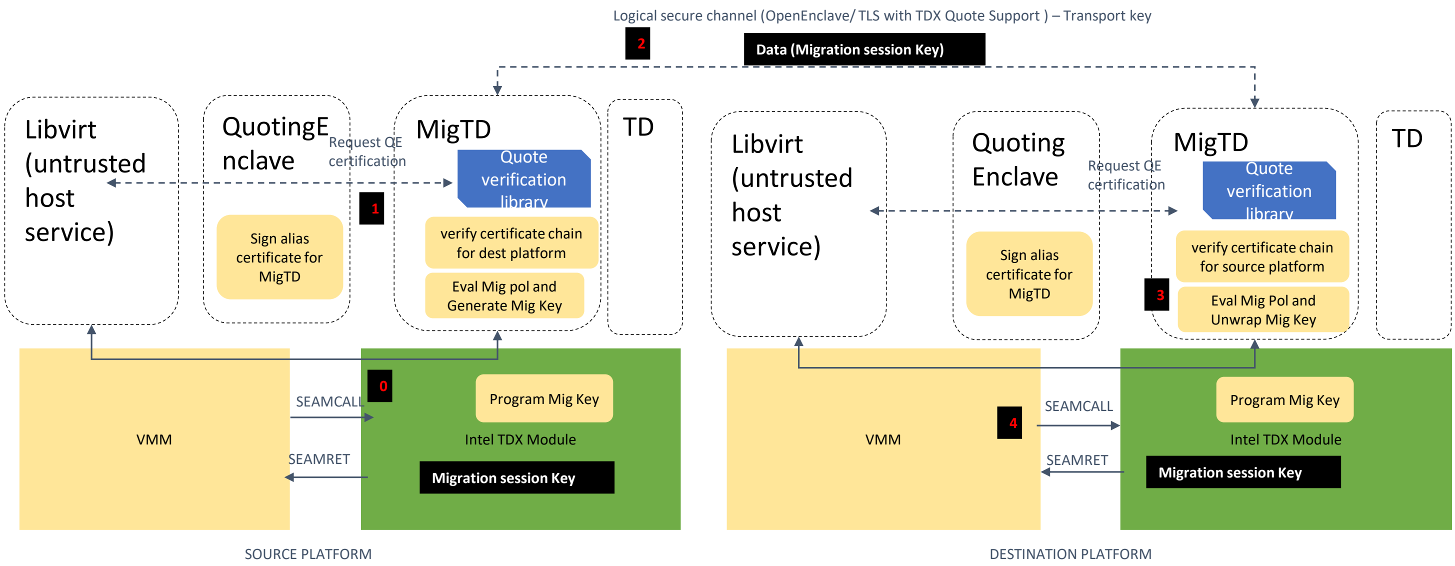

Figure 4.1: MigTD Transport Security Setup

1. On platform 1, TD-s to be migrated is created with MIGRATABLE attribute. TD-s is built and executed using the legacy process.
   1.1. The VMM may pre-bind MigTD-s to TD-s using TDH.SERVTD.PREBIND.
2. The cloud orchestration triggers a migration of TD-s from platform 1 to platform 2.
3. The host VMM on platform 1 instantiates MigTD-s and binds it to TD-s via an **unsolicited** service TD binding using TDH.SERVTD.BIND.
   3.1. MigTD-s requests the host VMM to be bound to TD-s, using TDG.VP.VMCALL.
   3.2. The VMM invokes TDH.SERVTD.BIND to bind TD0 to MigTD-s.
   3.3. The VMM communicates the binding handle, target TD_UUID and other binding parameters to MigTD-s.
4. MigTD-s may read selected metadata of TD-s (e.g., its ATTRIBUTES and XFAM) using TDG.SERVTD.RD, to use as an input to the migration policy evaluation. The readable metadata fields are listed in the [TDX Module ABI Spec].
5. The VMM initiates the migration process for the TD-s:
   5.1. The VMM creates a network transport session (nonce) with the destination platform (destination of TD migration) and requests a quote from the MigTD-d on destination platform.
   5.2. The VMM notifies MigTD-s of a new session providing quote for MigTD-d (from destination platform); in response MigTD-s invokes TDG.MR.REPORT and requests a QUOTE from the host VMM (to be sent to source platform).
   5.3. MigTD-s verifies the MigTD-d quote using a Quote Verification Library in the MigTD-s and establishes via Diffie-Helman a transport key for the session with destination platform (and vice-versa).
6. MigTD-s and MigTD-d can enumerate their respective TDX module properties (e.g., what migration version is supported, what CPU functionality is supported) using TDG.SYS.RD/RDALL. They can exchange this information in order to ensure compatibility.
7. MigTD-s authenticates the Migration Policy and evaluates it per the capabilities (SVN etc.) of the destination platform (learnt via the quote) for the specified live migration session.
8. On platform 2 a destination TD-d skeleton is created via legacy process.
   8.1. The VMM may pre-bind MigTD-d to TD-d using TDH.SERVTD.PREBIND.
9. Migration Keys Exchange:
   9.1. MigTD-s reads the Migration Forward Key (as the migration encryption key) from TD-s using TDG.SERVTD.RD.
   9.2. MigTD-s sends the Migration Forward Key to MigTD-d.
   9.3. On the destination platform MigTD-d binds with TD-d and writes the Migration Forward Key (as the migration decryption key) to it using TDG.SERVTD.WR.
   9.4. MigTD-s reads the Migration Backward Key (as the migration encryption key) from TD-s using TDG.SERVTD.RD.
   9.5. MigTD-d sends the Migration Backward Key to MigTD-s.
   9.6. On the source platform MigTD-s writes the Migration Backward Key (as the migration decryption key) to TD-s using TDG.SERVTD.WR.
10. The host VMM on the source platform can now initiate the state export via TDH.EXPORT* SEAMCALLs and import state via TDH.IMPORT* SEAMCALLs

5. TD Migration Common Mechanisms

This chapter describes the infrastructure used by all Import/Export APIs to migrate TD private memory and metadata.

5.1. Migration Bundles

This section describes the generic migration bundle structure. Private memory migration uses an enhanced format, described in 8.2.6.

5.1.1. Overview

TD information is transported from the source platform to the destination platform in **migration bundles**. A migration bundle consists of **migration data**, which may span one or more 4KB pages or one 2MB page, and **migration bundle metadata (MBMD)**. Migration bundle transport is the responsibility of untrusted software and is out of the scope of this specification.

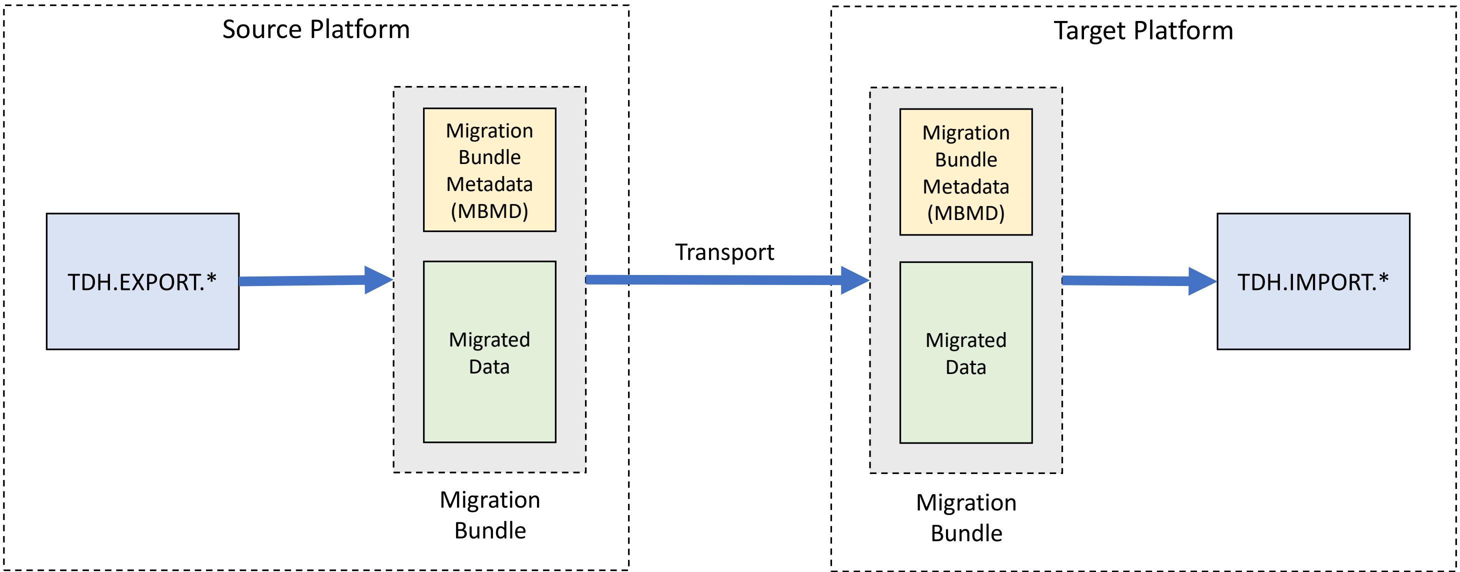

Figure 5.1: Migration Bundle

5.1.2. Migration Data

Migration data contains either TD private memory contents or TD non-memory state. It is confidentiality-protected using AES-GCM with the TD migration key and a running migration session counter. Migration data is integrity-protected by its associated MBMD. For encryption details, see 5.3.

Note: Migration of shared memory pages is the responsibility of untrusted software and is out of the scope of this specification.

In memory, migration data occupies one or more 4KB shared memory pages, or one 2MB shared memory page, managed be the host VMM.

5.1.3. Migration Bundle Metadata (MBMD)

A migration bundle metadata (MBMD) structure provides metadata for an associated migration data. In memory, MBMD resides in a shared page, managed by the host VMM, and must be naturally aligned. An MBMD is not confidentiality protected, but it provides integrity protection for itself and for its associated migration data.

The MBMD structure consists of a fixed header and a per-type variable part. The header contains the following fields:

SIZE: Overall size of the MBMD structure, in bytes

MIG_VERSION: Migration protocol version

MB_TYPE: The type of information being migrated

MB_COUNTER: Per-stream migration bundle counter

MIG_EPOCH: Migration epoch number

MIGS_INDEX: Index of the migration stream

IV_COUNTER: Monotonously increasing counter, used as a component in the AES-GCM IV

The last field of each MBMD is an AES-256-GCM MAC over other MBMD fields and other associated migration data (migration pages).

The detailed MBMD definition is provided in [TDX Module ABI Spec].

5.2. Export and Import Functions Interface

Export and import functions operate on a single migration bundle at a time, which belongs to a specific migration stream.

5.2.1. Overview of Migration Data Format in Memory

While in memory, a migration bundle always contains a single MBMD. Optional migration data can be stored in multiple 4KB migration buffer pages.

5.2.2. Migrating a Multi-Page Migration Bundle

To export a multi-page migration bundle, the host VMM on the source platform prepares a set of migration buffer pages and a buffer for an MBMD in shared memory. The required number of migration pages per TDH.EXPORT.* function is enumerated by TDH.SYS.INFO. The host VMM provides the MBMD's HPA and a list of HPA pointers to the migration pages as an input to the TDH.EXPORT* function.

To import a multi-page migration bundle, the host VMM on the destination platform prepares the set of migration pages and the MBMD, as received from the source platform, in shared memory. The host VMM provides the MBMD's HPA and a list of HPA pointers to the migration pages as an input to the TDH.IMPORT* function.

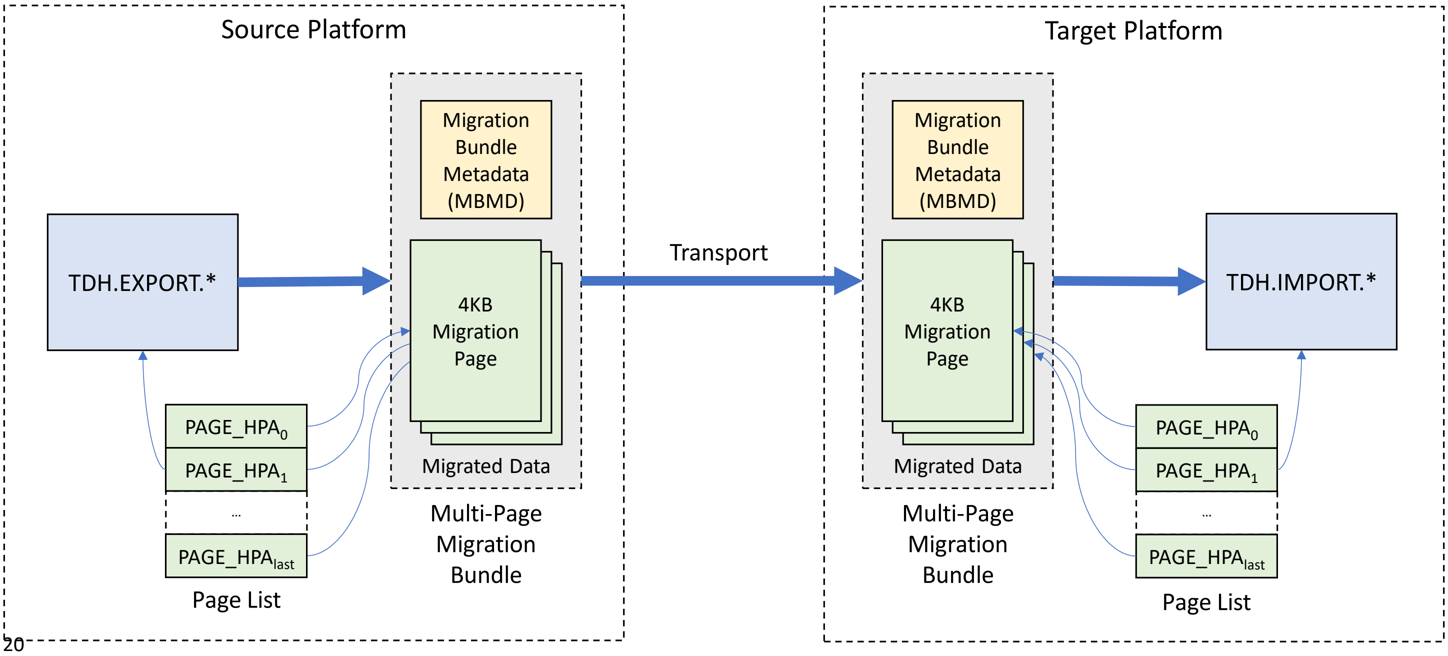

Figure 5.2: Migrating a Multi-Page Migration Bundle

5.2.3. Migration Functions Interruptibility

TDH.EXPORT.* and TDH.IMPORT.* functions may take relatively long time to execute. This is especially true for 2MB page migration and multiple 4KB page migration. To avoid latency issues, such functions may be **interruptible** and **restartable**. This is supported as follows:

- TDH.EXPORT.* and TDH.IMPORT.* functions are designed to synchronously check for a pending external event by reading MSR_INTR_PENDING (once after every pre-determined number of cycles, chosen to be smaller than the maximum allowed cycle latency).

- If an external event is pending, the functions store their context in the proper MIGSC and returns with a TDX_INTERRUPTED_RESUMABLE completion status.
- The host VMM is expected to call the TDH.EXPORT.* or TDH.IMPORT.* function again with the same set of inputs until the operation is completed successfully (completion status is TDX_SUCCESS) or some error occurs (completion status indicates an error).
- An input flag indicates whether the invocation of a TDH.EXPORT.* or TDH.IMPORT.* function starts a new operation (and possibly aborts an interrupted one) or resumes an interrupted operation.

5.3. Cryptographic Protection for Migration Data

5.3.1. Encryption Algorithm

TD migration uses AES in Galois/Counter Mode (GCM) to transfer state between the source and destination platform platforms. Per [AES-256-GCM] definitions, the TD data private memory or non-memory state temporarily held in the CPU cache during TDH.EXPORT.* forms the "Plaintext", and some of the MBMD fields form the "Additional Authenticated Data". The "Plaintext" is encrypted using a Migration key (described below). The MAC size, also known as t, as defined in [AES-256-GCM], must be 128 bits.

The Initialization Vector (IV) is 96 bits. It is composed as described below. Since 64 bits will never wrap around in practice, this helps ensure a unique counter for each stream.

Table 5.1: Components of the 96-bits IV

Bits	Size	Name	Description
63:0	64	IV_COUNTER	Starts from 1, incremented by 1 every time AES-GCM is used to encrypt data and/or generate a MAC for a migration bundle. The counter is incremented even if the data is discarded and not used for migration.
79:64	16	MIGS_INDEX	Stream index (see 5.3)
95:80	16	RESERVED	Set to 0

5.3.2. Migration Session Keys

Two migration session keys are used, one in each direction:

- The TDX module on the source platform generates a **migration session forward key** for encrypting migration bundles by the source TDX module and decrypting them by the destination TDX module.
- The TDX module on the destination platform generates a **migration session backward key** for encrypting migration bundles by the destination TDX module and decrypting them by the source TDX module.

Each of the MigTDs on the source and the destination platforms reads the key generated on their side, known as the **migration encryption key**, from the TDX module's target TD's metadata, and transfer it on a secure like to its peer MigTD on the other side of the migration session. The peer MigTD then writes the key, to be used as the **migration decryption key**, to the TDX module's target TD's metadata. TD metadata read and write used the Service TD protocol, as described in the [TDX Module Base Spec].

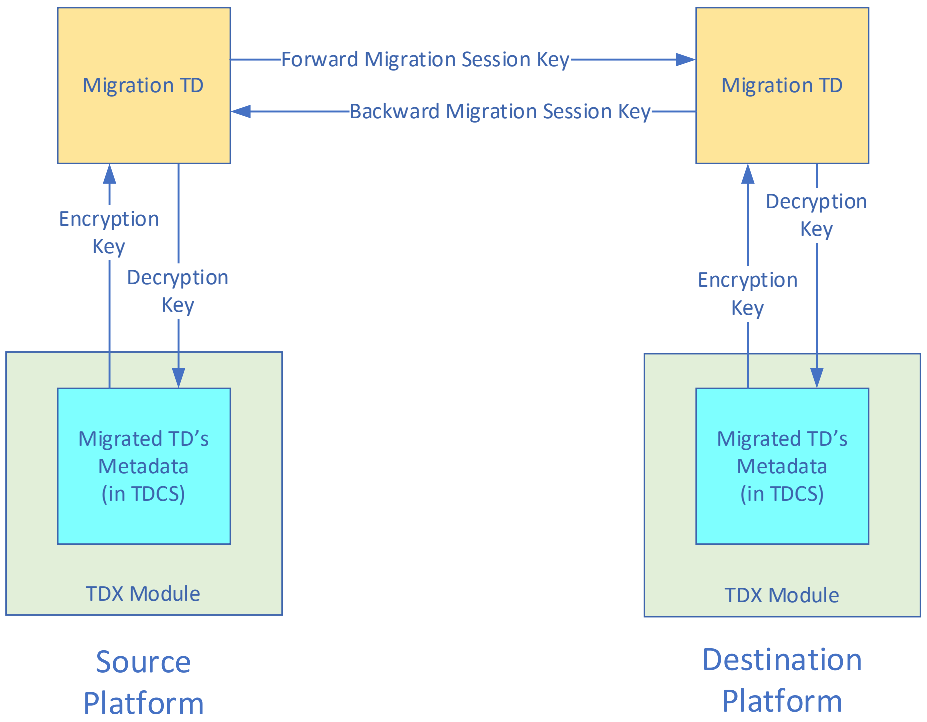

Figure 5.3: Migration Session Keys Exchange

The migration keys properties are as follows:

- The key strength is 256 bits.
- A new encryption key is generated by the TDX module on TD creation (TDH.MNG.CREATE) and at the beginning of each migration session (TDH.*PORT.STATE.IMMUTABLE).
- The encryption key is read by the MigTD from the migrated TD's metadata using the Service TD metadata protocol, as described in the [TDX Module Base Spec].
- The MigTD transfer the encryption key to its peer MigTD, over a secure channel both MigTDs have created.
- The decryption key is written by the MigTD to the migrated TD's metadata using the Service TD metadata protocol, as described in the [TDX Module Base Spec].
- The keys are accessible only by the MigTD and the Intel TDX Module.
- On migration session start by TDH.*PORT.STATE.IMMUTABLE, the Intel TDX module copies the encryption and decryption keys into working keys that are used throughout the session.
- The keys are used by TDH.EXPORT.*/TDH.IMPORT.* to control the migration session and migrate TD memory and non-memory. The migration stream AES-GCM protocol requires that state is migrated in-order between the source and destination platform. This helps guarantee the order within each migration stream.
- The keys are **destroyed** when a TD holding them is torn down, or when new keys are generated or programmed.

5.4. Migration Streams and Migration Queues

Migration stream is a **TDX concept**. Multiple streams allow multi-threaded, concurrent export and import, and enable the Intel TDX Module to enforce proper ordering of migration bundles during the in-order phase where this is essential.

Migration queue is a **host VMM concept**. Multiple queues allow QoS and prioritization. E.g., Post-copy of pages on demand (triggered by an EPT violation on the destination platform) may have a higher priority than other post-copy of pages. To avoid head-of-line blocking by waiting in the same queue as lower priority pages, a separate high priority queue can be used by the host VMM.

From the migration streams and migration queues perspective, a migration session is divided into two main phases:

- **In-order**, where the source TD may run, and its memory and non-memory state may change. During the in-order phase, the order of memory migration is critical. A newer export of the same memory page must be imported after an older export of the same page. Furthermore, for any memory page that has been migrated during the in-order phase, the most up-to-date version of that page must be migrated before the in-order phase ends. **In the in-order phase, one or more migration streams are mapped to each migration queue**.
- **Out-of-order**, where the source TD does not run, and its memory and non-memory state may not change. During out-of-order, the order of memory migration is not important – except that migration bundles exported during the

in-order phase can't be imported during the out-of-order phase. Furthermore, the host VMM may assign exported pages (even multiple copies of the same exported page) to different priority queue. This is used, e.g., for on-demand migration after the destination TD starts running.

The **start tokens**, generated by TDH.EXPORT.TRACK and verified by TDH.IMPORT.TRACK, serve as markers to indicate the end of the in-order phase and start of the out-of-order phase. They are used to implement a rendezvous point, enforcing all the in-order state (across all streams) to have been imported before the out-of-order phase starts and the destination TD may execute.

5.3 describes how the stream context held by the source and the destination platforms and the MBMD fields included in each migration bundle are used to construct the non-repeated AES-GCM IV. Note also that the same stream queues can be used for both in-order and out-of-order. The semantic use of the queues is up to the host VMM.

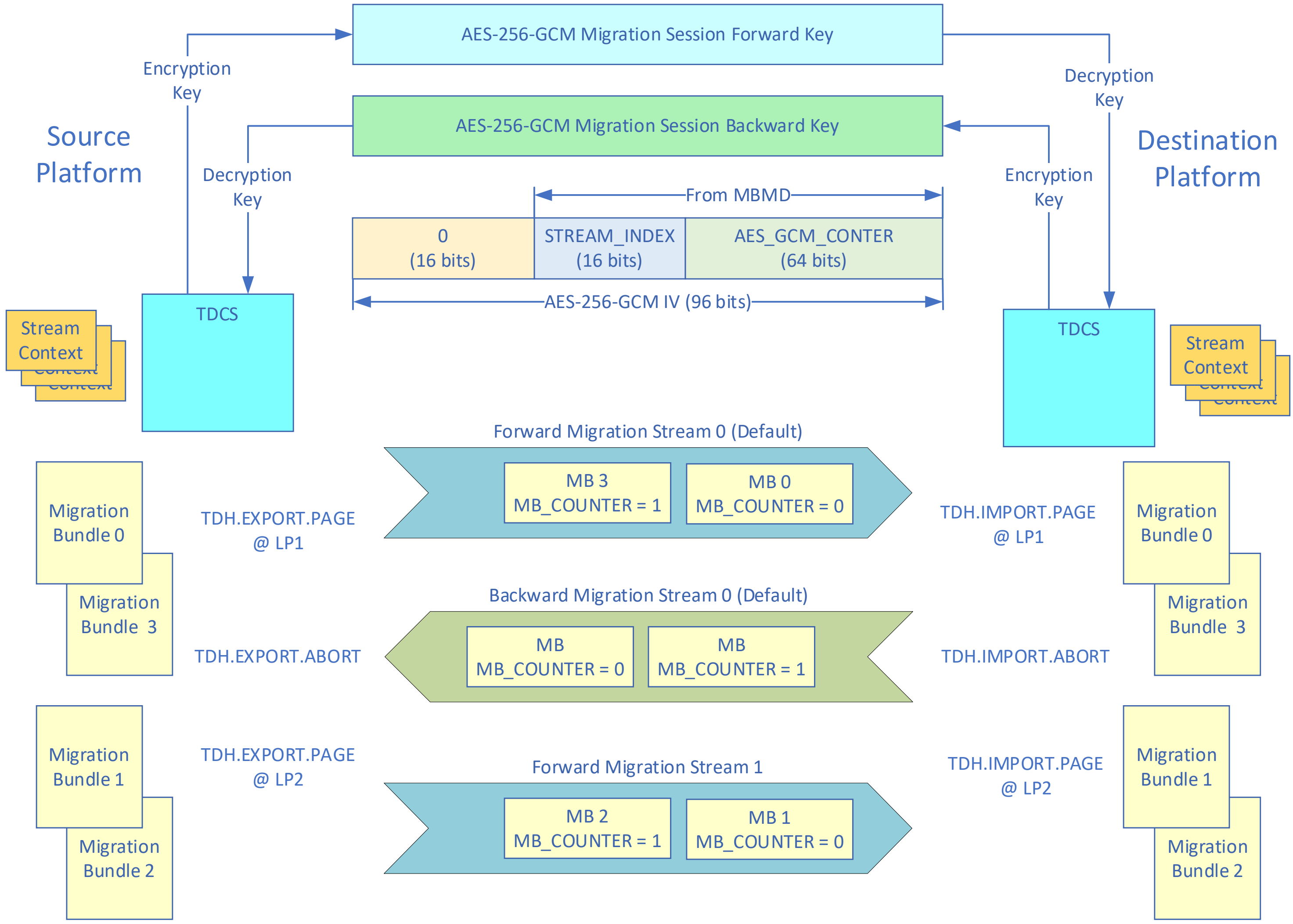

Figure 5.4: Migration Streams

Migration streams have the following characteristics:

- Within each stream, state is migrated in-order. This is enforced by the MB_COUNTER field of MBMD.
- Export or import operations using a specific migration stream must be serialized. Concurrency is supported only between streams.
- The host VMM should use the same stream index to import memory on the destination TD (which should be in MEMORY_IMPORT, STATE_IMPORT or RUNNABLE state). This is enforced by TDH.IMPORT.MEM.
- Non-memory state can only be migrated once; there is no override of older migrated non-memory state with a newer one. Ordering requirements (e.g., TD-scope non-memory state must be imported before VCPU non-memory state) are enforced by the lifecycle state machine, as described in 6.2.
- The maximum number of forward streams is implementation dependent:
  - Each stream requires context space allocation.
  - Stream ID requires a field in the MBMD header.

  The maximum number of forward streams is enumerated by TDH.SYS.RD*.

- There is one backward stream.

5.5. Measurement and Attestation

5.5.1. TD Measurement Registers Migration

TDs have two types of measurement registers:

MRTD: Static measurement of the TD build process and the initial contents of the TD. This state is migrated as part of the global immutable state of the TD (via TDH.EXPORT.STATE.IMMUTABLE and TDH.IMPORT.STATE.IMMUTABLE).

RTMR: An array of general-purpose measurement registers, available to the TD software for measuring additional logic and data loaded into the TD at runtime. Since this measurement covers dynamic state beyond the static state and can be extended by TD software via TDG.MR.RTMR.EXTEND, hence, this state is migrated only during the blackout period, as part of the TD's mutable state (via TDH.EXPORT.STATE.TD and TDH.IMPORT.STATE.TD).

All TD measurements are reflected in TD attestations.

5.5.2. TD Measurement Reporting Changes

The TDINFO structure is enhanced to include hashes of Service TDs' TDINFO; for TD migration, the applicable Service TD is the Migration TD. Refer to the [TDX Module Base Spec] for a discussion of Service TDs.

5.5.3. TD Measurement Quoting Changes

To create a remotely verifiable attestation, the TDREPORT_STRUCT must be converted into a Quote signed by a certified Quote signing key, as described in the [TDX Module Base Spec].

TDREPORT_STRUCT is HMAC'ed using an HMAC key unique to each platform and accessible only to the CPU. This protects the integrity of the structure and can only be verified on the local platform via the SGX ENCLU(EVERIFYREPORT2) instruction. TDREPORT_STRUCT cannot be sent off platforms for verification; it first must be converted into signed Quotes.

If a report is generated by TDG.MR.REPORT on the source platform, but the is migrated to a destination platform, the local HMAC key is different and hence the EVERIFYREPORT2 on the migrated TDG.MR.REPORT is expected to fail. The TD software is typically unaware of being migrated. It is expected to retry the TDG.MR.REPORT operation if it fails.

5.5.4. TCB Recovery and Migration

The TDX architecture has several levels of TCB:

- CPU HW level, which includes microcode patch, ACMs, and PFAT
- Intel TDX Module
- Attestation Enclaves, which include the TD Quoting Enclave and Provisioning Certification Enclave

The TCB Recovery story is different for each level. The existing TCB Recovery model for CPU SW level items applies similarly to TDX and SGX and requires a restart of the platform to take effect. The Intel TDX Module can be unloaded and reloaded to reflect an upgraded Intel TDX Module. The enclaves can be upgraded at runtime, but if the PCE is upgraded, a new certificate must be downloaded.

5.6. TDX Control Structure Updates

This section discusses updates and additions to the global and TD-scope control structures.

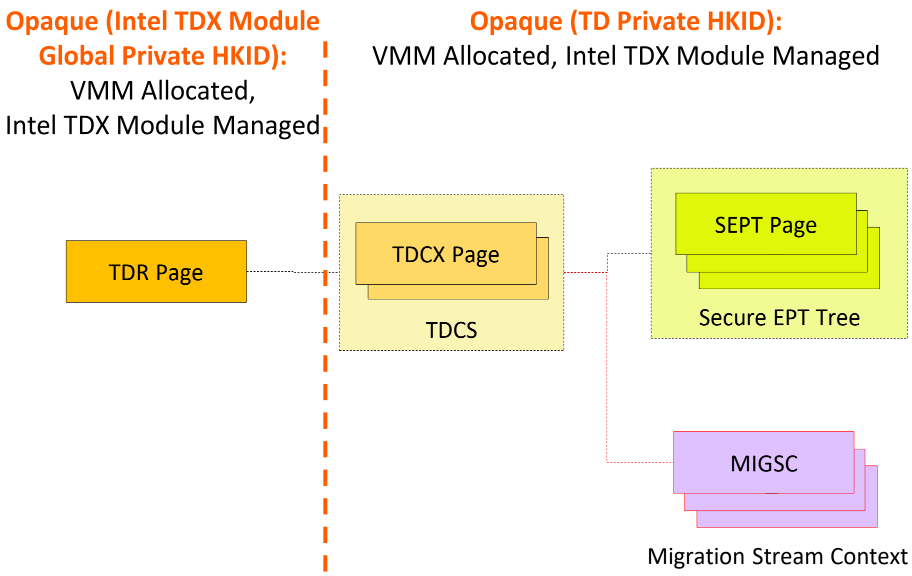

Figure 5.5: TD-Scope Control Structures Overview

5.6.1. Per-TD: TDCS

As described in 6.2, TD migration happens when both the source and destination TDs have their ephemeral encryption keys assigned and configured, and the TDCS pages have been allocated. Thus, all migration-related metadata is stored in TDCS. There is no change to TDR. TDCS details are described in the [TDX Module ABI Spec].

5.6.1.1. *Updates to Existing TDCS Fields*

TDCS is updated with the following migration-related fields:

- The ATTRIBUTES field has a new MIGRATABLE bit.
- TD state variables are enhanced to support the updated TD Operation State Machine and the new TD Migration State Machine, described in 6.2.
- Service TD context array supports, among other, the Migration TD. Service TDs are described in the [TDX Module Base Spec].

5.6.1.2. *TDCS Migration-Related Fields*

The following TDCS fields hold per-TD migration context. The detailed specification is provided in the [TDX Module ABI Spec].

Table 5.2: TDCS Migration Context High Level Definition

Field	Description
MIG_KEY_SET	Set when a new MIG_KEY is written, cleared when the MIG_KEY is copied to MIG_WORKING_KEY
EXPORT_COUNT	Counts the number of times this TD has been exported, included aborted export sessions. Incremented at the beginning of each export session (TDH.EPORT.STATE.IMMUTABLE).
IMPORT_COUNT	Counts the number of times this TD has been imported. Incremented by TDH.IMPORT.COMMIT.
MIG_EPOCH	Migration epoch Starts from 0 on migration session start, incremented by 1 on each epoch token. A value of 0xFFFFFFFF indicates out-of-order phase.
BW_EPOCH	Blocking-for-write epoch Holds the value of TD_EPOCH at last time TDH.EXPORT.BLOCKW blocked a page for writing.

Field	Description
TOTAL_MB_COUNT	The total number of migration bundles exported or imported during the current migration sessions
MIG_KEY	Migration key, as written by the Migration TD
MIG_WORKING_KEY	Migration working key, copied from MIG_KEY at the beginning of a migration session and used throughout the session
MIG_VERSION	Migration protocol version, as written by the migration TD
MIG_WORKING_VERSION	Migration working protocol version, copied from MIG_VERSION at the beginning of a migration session and used throughout the session
DIRTY_COUNT	Counts of the number of pages that must be re-exported, because their contents have been modified since they have been exported, before a start token may be generated
MIG_COUNT	Counts the number of SEPT entries that need to be cleaned up after an aborted migration
NUM_MIGS	Number of Migration Stream Context (MIGSC) pages that have been allocated
MIGSC_LINKS	An array of links to MIGSC pages. Each entry contains the following information: **MIGSC_HPA:** Physical address of the MIGSC page (without the HKID bits) **INITIALIZED:** A boolean flag, indicating that the migration stream has been initialized.

5.6.2. Secure EPT

Secure EPT entry structure is updated as follows:

- Multiple TDX-specific state bits, to support the many new states required for TD private memory migration (see Chapter 8).
- Host-side entry lock bit, to support concurrent migration operations on separate Secure EPT entries without exclusively locking the whole Secure EPT tree.

Secure EPT entry details are provided in the [TDX Module ABI Spec].

5.6.3. MIGSC: Migration Stream Context

Migration streams are defined in 5.3.

MIGSC (Migration Stream Context) is an opaque control structure that holds migration stream context. MIGSC occupies a single 4KB physical page, and is created using the TDH.MIG.STREAM.CREATE function. MIGSC can only be created if a migration session is not in progress.

Most of the migration stream context is initialized at the beginning of a new migration session by TDH.EXPORT.STATE.IMMUTABLE or TDH.IMPORT.STATE.IMMUTABLE.

Table 5.3: Migration Stream Context High Level Definition

Field	Description	Initial Value
IV_COUNTER	Monotonously incrementing 64b counter, used as a component in the AES-GCM IV IV_COUNTER is incremented near the beginning of any TDX module function that creates a migration bundle, before it is used for composing an IV. This is done to avoid reusing the IV value in case of a failure.	0

Field	Description	Initial Value
NEXT_MB_COUNTER	Transmitted migration bundle counter (64b) On export, incremented by 1 after every successful MBMD generation. After transitioning to the out-of-order phase by TDH.EXPORT.TRACK, bit 63 is set to 1.	0
EXPECTED_MB_COUNTER	Expected received migration bundle counter (64b) Applicable only on the import side, during the in-order phase before a start token has been received on this stream. A received MBMD's MB_COUNTER must be equal or higher than EXPECTED_MB_COUNTER. During the out-of-order phase, the imported MBMD's MB_COUNTER is not compared to EXPECTED_MB_COUNTER. Instead, its bit 63 is checked to be 1.	0
AES_GCM_CONTEXT	Implementation-dependent AES-GCM context	N/A
INTERRUPTED_STATE	The state of interrupted TDH.EXPORT.* or TDH.IMPORT.* interface function. Includes the following: • Valid flag • Interrupted function's leaf number • Interrupted function's input operands • Interrupted function's MBMD • Other state the needs to be saved across interruptions and resumptions.	N/A

6. Migration Session Control and State Machines

This chapter discusses the TD migration session control, state machine and messaging protocol.

6.1. Overview

6.1.1. Pre-Migration

Prior to starting a migration session, the following should have happened:

- The TD has been created as a skeleton (control structure pages only) on the destination platform.
- Migration TDs should be bound as service TDs on both source and destination platforms.
- Migration TDs must have exchanged the migration session keys and decided on migration version.

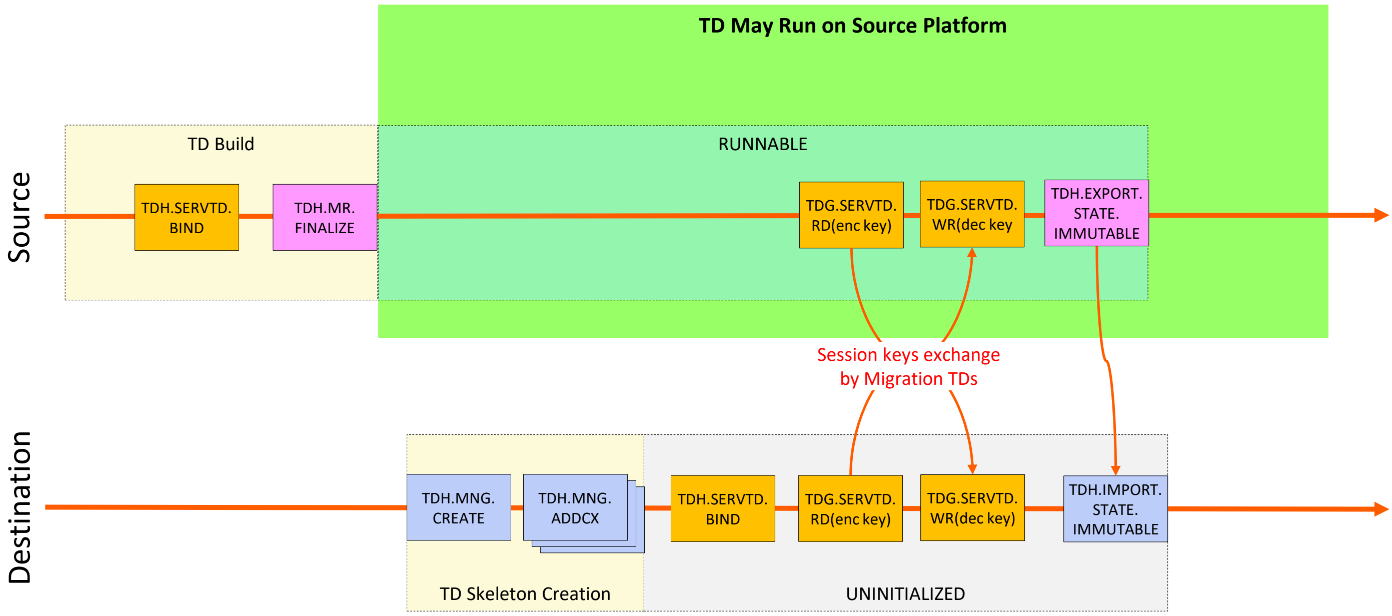

Figure 6.1: Pre-Migration

6.1.2. Successful Migration Session

Figure 6.3 below shows an overview of a successful migration session. This figure shows the following:

- Migration the session control TDX Module interface functions (TDH.EXPORT.PAUSE, TDX.EXPORT.TRACK etc.)
- States of the TD Operation State Machine (RUNNABLE, LIVE_EXPORT etc.) are also shown. The state machine itself is discussed in 6.2.2 below.
- Phases of the export and import sessions

Section 2: TD Migration Architecture Specification

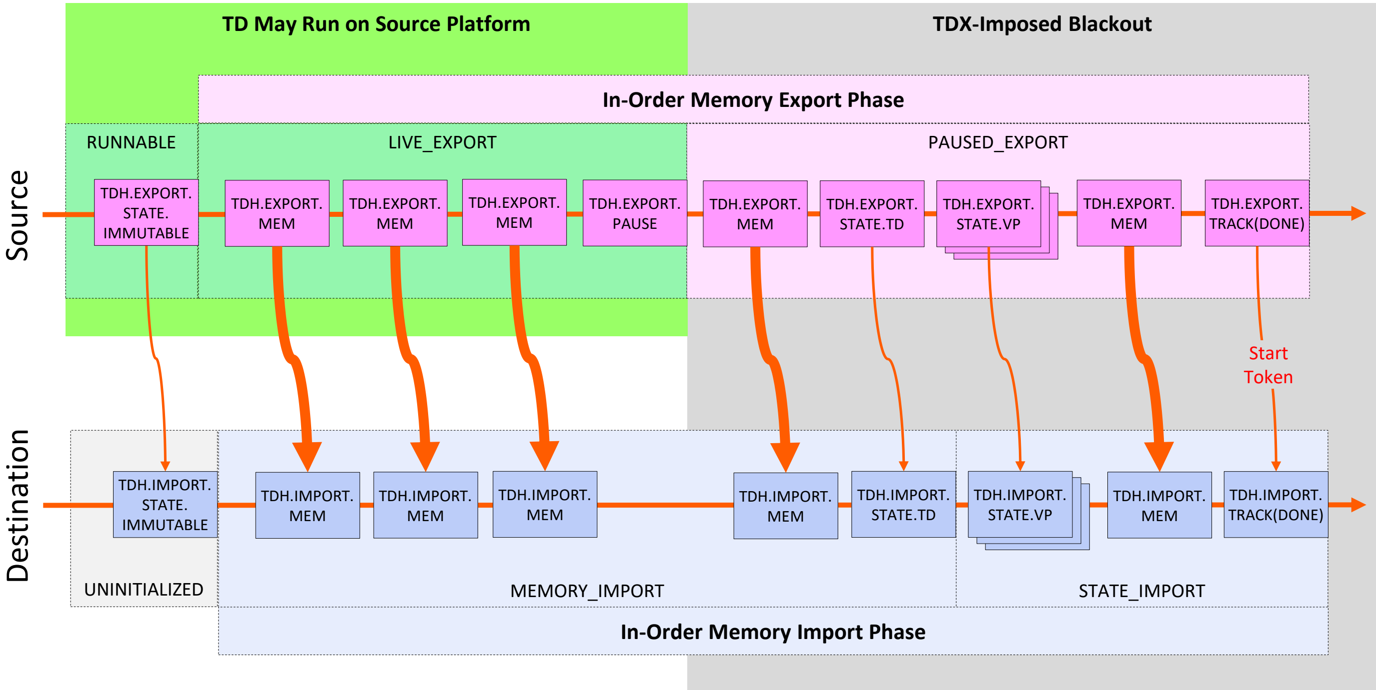

Figure 6.2: Migration Session In-Order Phase (Success Case)

On the source platform, an export session's in-order export phase starts with the host VMM invoking the TDH.EXPORT.STATE.IMMUTABLE function. This function creates a migration bundle that is transmitted by the host VMM to the destination platform, where TDH.IMPORT.STATE.IMMUTABLE is invoked to start the import session's in-order import phase.

TDH.EXPORT.PAUSE pauses the TD on the source platform and starts the TDX-imposed blackout period.

TDH.EXPORT.TRACK, when invoked on the source platform, verifies proper in-order export. TD state must have been exported, and if any TD private page has been exported, the latest version of that page must have been exported. TDH.EXPORT.TRACK(done) ends that phase and creates a **start token** migration bundle that is transmitted by the host VMM to the destination platform, where TDH.IMPORT.TRACK(done) is invoked to end the in-order import phase. See also the discussion of migration epochs in 6.1.4 below.

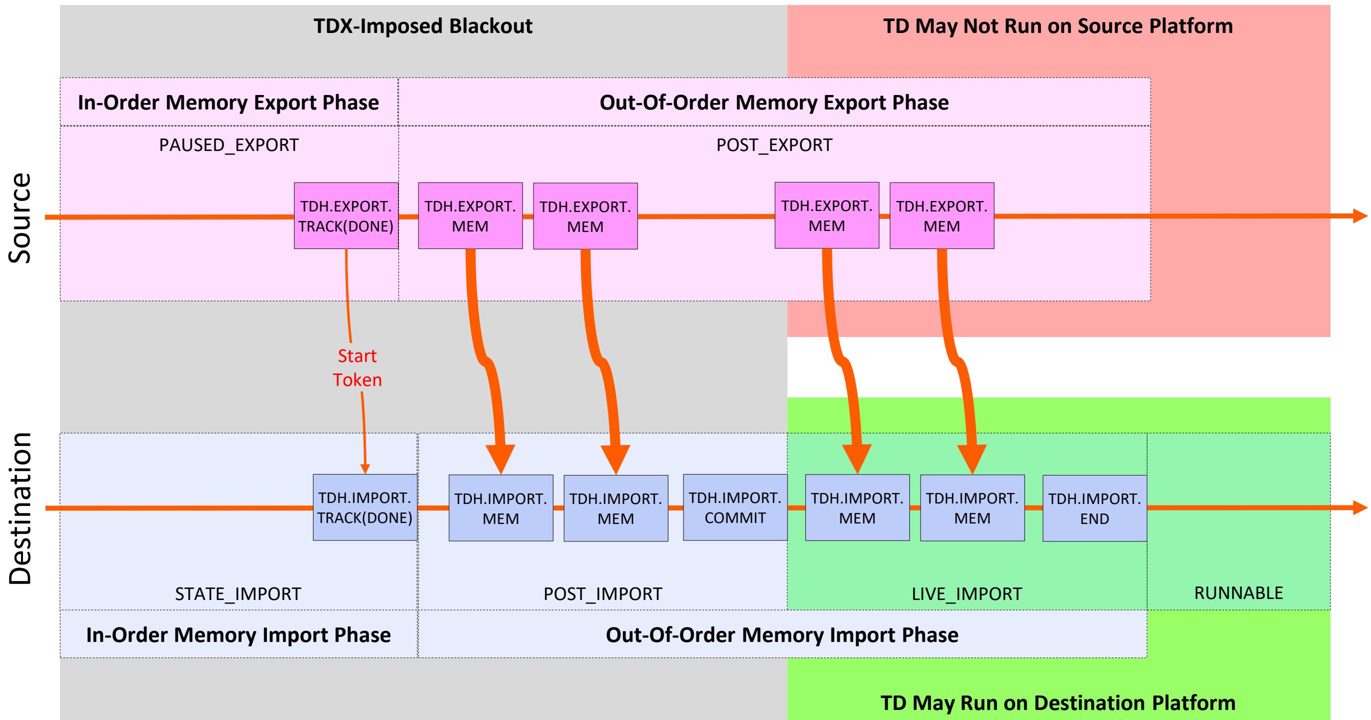

Figure 6.3: Migration Session Out-Of-Order Phase (Success Case)

TDH.IMPORT.COMMIT, invoked on the destination platform, commits the migration session and enables the TD to run on it, while memory import may continue. This also helps ensure that the TD will not run on the source platform, since an abort token will not be generated.

Optionally, TDH.IMPORT.END, invoked on the destination platform, commits the migration session and enables the TD to run on it if not already done by TDH.IMPORT.COMMIT. TDH.IMPORT.END ends the migration session; memory import is no longer allowed.

6.1.3. Aborted Migration Session

6.1.3.1. *Abort during the In-Order Phase*

Figure 6.4 below shows a case where a migration session is aborted during the in-order migration phase. TDH.EXPORT.ABORT, invoked by the host VMM, terminates the export session and enables the TD to resume running on the source platform. By design, the TD should not be able to run on the destination platform – it is up to the host VMM to free up any resource allocated there.

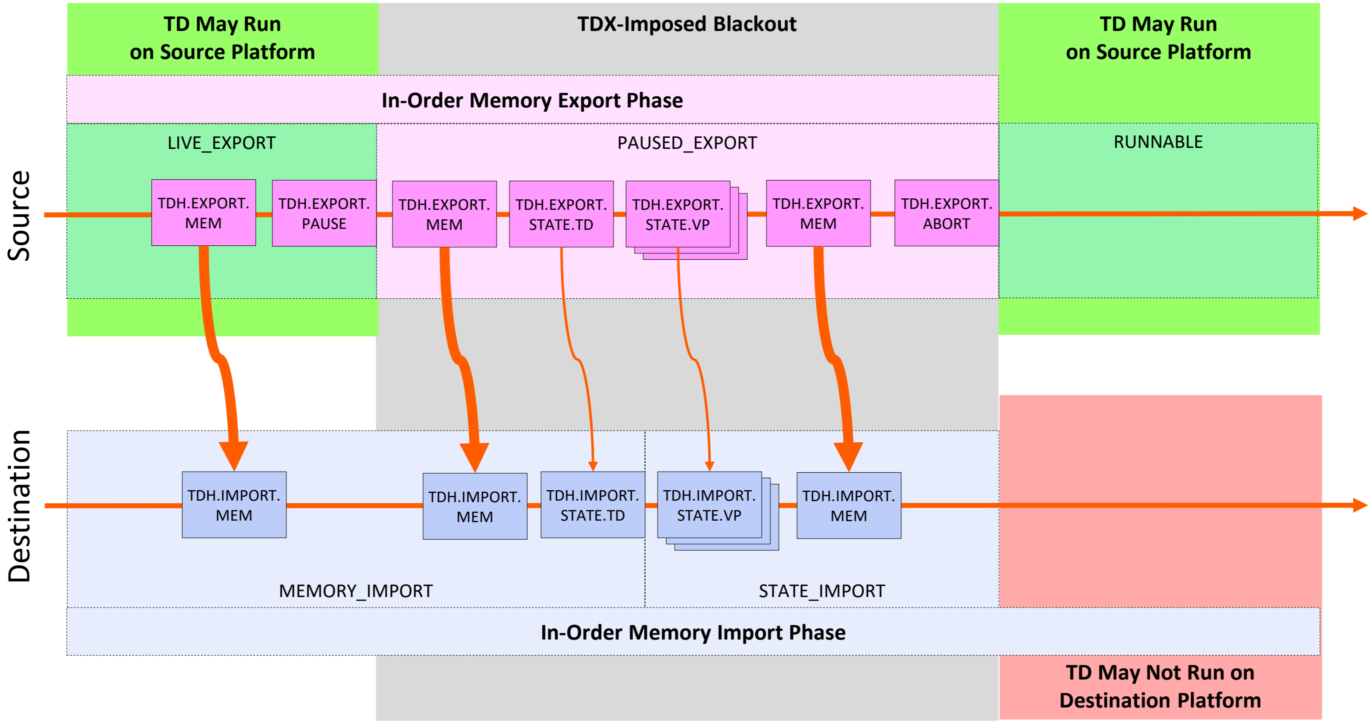

Figure 6.4: Migration Session Control Overview (Abort During the In-Order Phase)

6.1.3.2. *Abort during the Out-Of-Order Phase*

Figure 6.5 below shows a case where a migration session is aborted during the out-of-order migration phase. TDH.IMPORT.ABORT is invoked by the host VMM on the destination platform. This function terminates the import session and puts the TD in a state where, by design, it should not be able to run – it is up to the host VMM to free up any resource allocated there. TDH.IMPORT.ABORT also creates an abort token, which is transmitted by the host VMM back to the source platform.

On the source platform, the host VMM invokes TDH.EXPORT.ABORT, which checks the validity of the abort token and enables the TD to resume running.

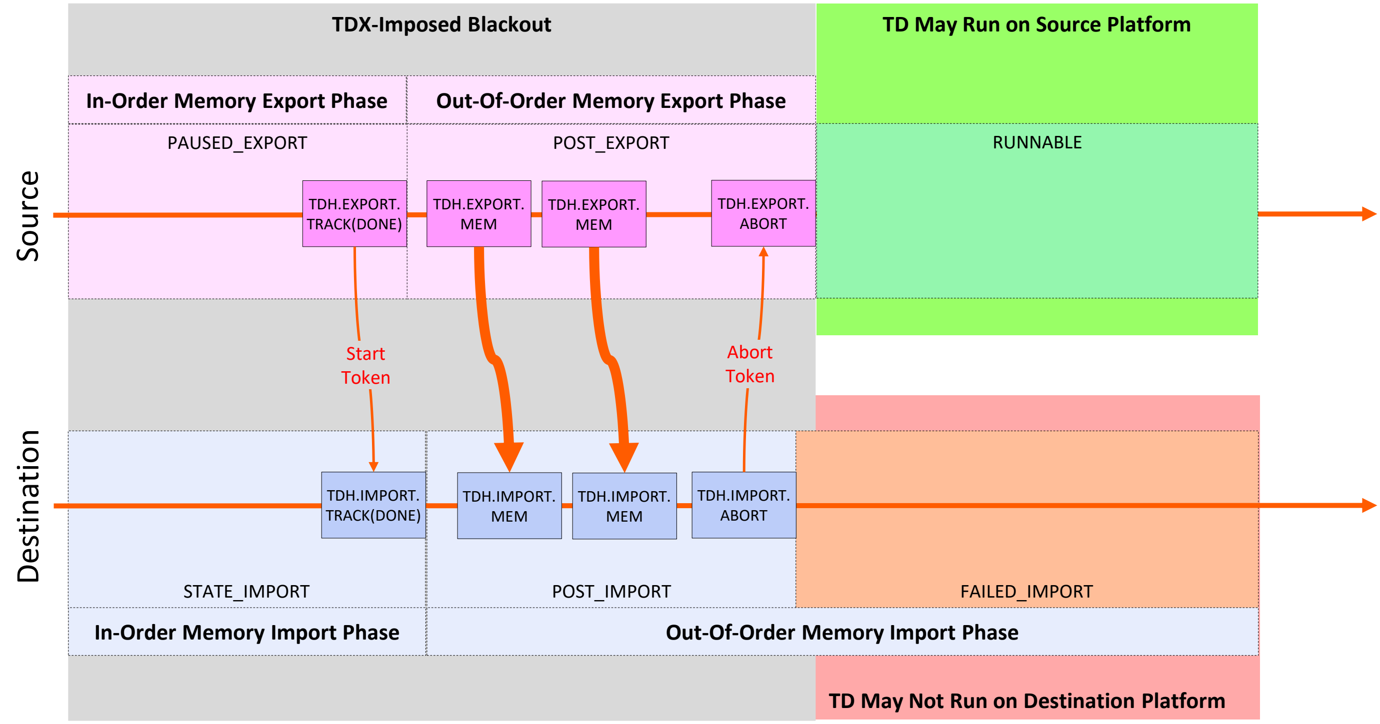

Figure 6.5: Migration Session Control Overview (Abort During the Out-Of-Order Phase)

6.1.4. Migration Epochs

To help maintain proper ordering of migrated page versions, the in-order migration phase is divided to multiple migration epochs. A specific page can only be migrated once per migration epoch. This is detailed in 8.5.1.3. TDH.EXPORT.TRACK starts a new export epoch and creates an **epoch token** migration bundle that is transmitted by the host VMM to the destination platform, where TDH.IMPORT.TRACK is invoked to a new import epoch. The last invocations of TDX.EXPORT.TRACK and TDX.IMPORT.TRACK, with a parameter indicating that the in-order phase is done, start the out-of-order export and import phases respectively.

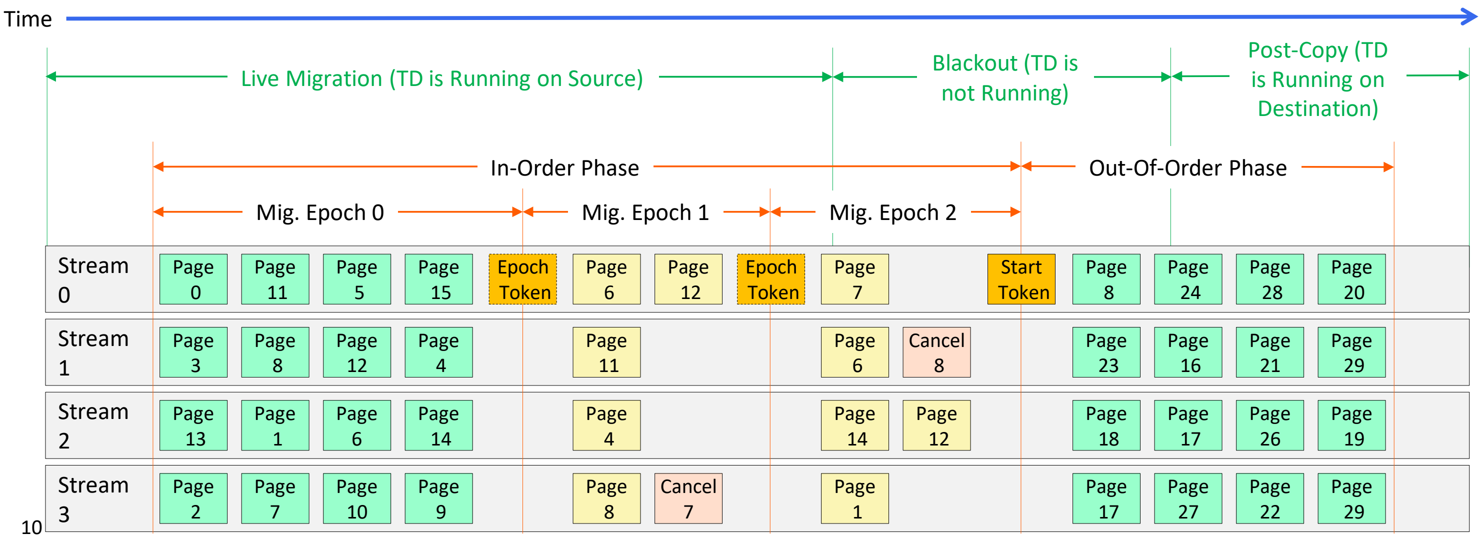

Figure 6.6: Migration Epochs Overview

6.2. *TD Migration State Machines*

6.2.1. Overview

The whole TD migration process happens within the TD_KEYS_CONFIGURED state of the TD life cycle state machine, where an HKID has been assigned to the TD and the keys have been configured on the hardware. As a reminder, the TD life cycle state diagram is shown in Figure 6.7 below. For details, see the [TDX Module Base Spec].

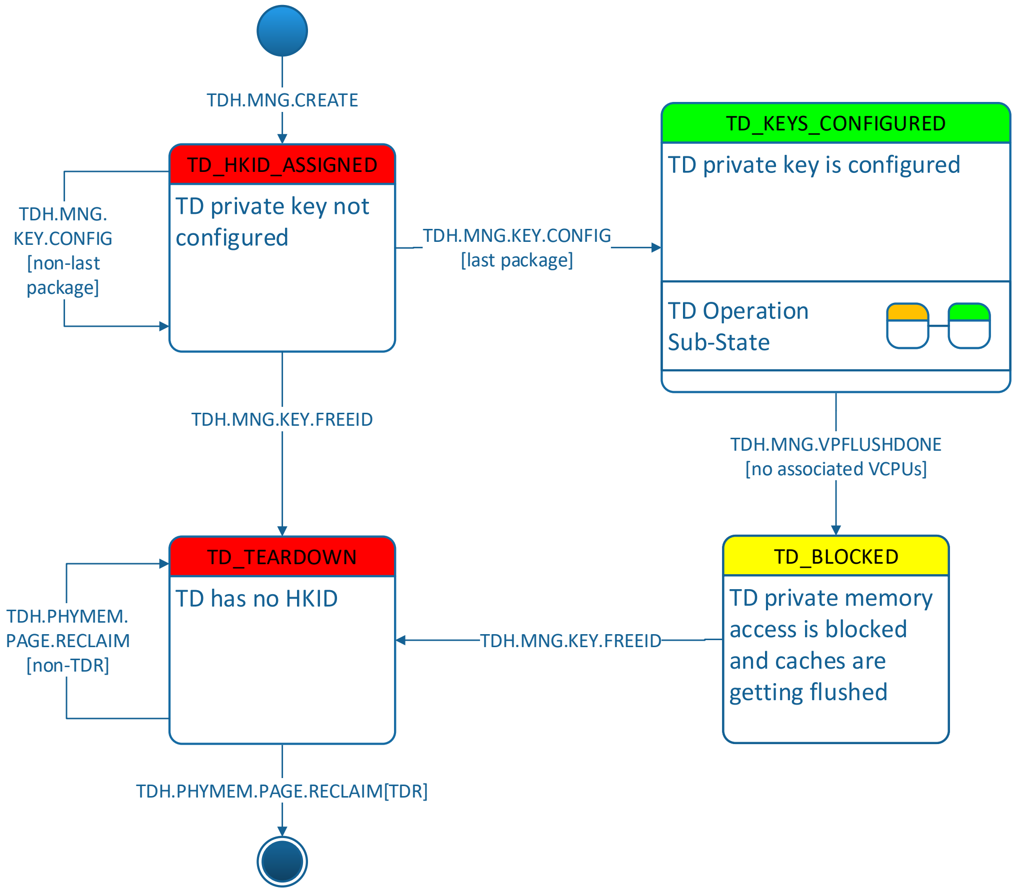

Figure 6.7: TD Life Cycle State Diagram

Within the TD_KEYS_CONFIGURED state, a secondary-level **TD Operation state machine** controls the overall TD operation, including migration.

6.2.2. OP_STATE: TD Operation State Machine

The TD Operation state machine is shown in Figure 6.8 below. The baseline state machine is extended with new migration-related states and transitions, highlighted in red text and lines. The export states are highlighted in purple and the import states are highlighted in blue.

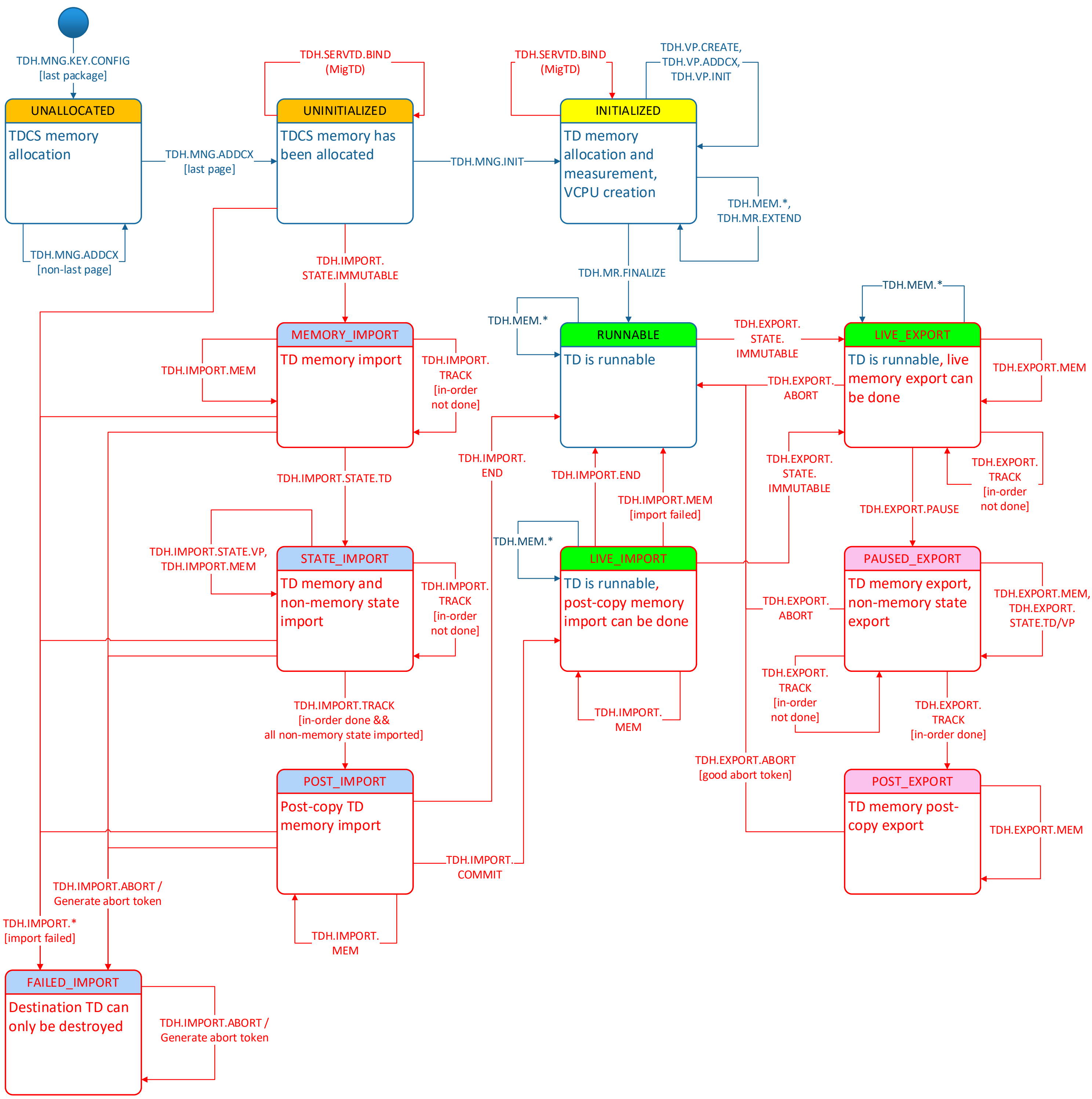

Figure 6.8: TD Operation State Machine (Sub-States of TD_KEYS_CONFIGURED)

6.2.3. Migration TD Binding and Migration Key Assignment

Migration TD binding (using TDH.SERVTD.BIND) must happen before a migration session can start. This may happen during TD build, before the measurement has been finalized (by TDH.MR.FINALIZE). Alternatively, pre-binding (using TDH.SERVTD.PREBIND) can be done during TD build, and actual binding can happen later. On the destination platform migration TD binding and TD import must happen before the TD is initialized (by TDH.MNG.INIT).

Migration key assignment, done by TDG.SERVTD.WR, may happen at any time after migration TD binding, except during the PAUSED_EXPORT and POST_EXPORT states. A new migration key must be written for any migration session.

6.2.4. Export Side (Source Platform)

To begin an export session, the TD's OP_STATE must either be RUNNABLE, indicating that its measurement has been finalized (by TDH.MR.FINALIZE), or LIVE_IMPORT, indicating that this TD has been previously imported.

An export session begins with immutable TD state export (using TDH.EXPORT.STATE.IMMUTABLE). This function copies the migration key to a working migration key. It then starts the **in-order export phase**. It transitions the OP_STATE to LIVE_EXPORT, allowing the source TD to continue running normally while private memory is being exported.

TDH.EXPORT.PAUSE transitions the source TD's OP_STATE into the PAUSED_EXPORT state. In this state, TD private memory and TD state modification are prevented. None of the TD VCPUs may be running (i.e., in TDX non-root mode), and no host-side (SEAMCALL) function is allowed to change any TD non-memory state that is to be exported. Memory export (via TDH.EXPORT.MEM etc.) may still continue. Per-TD and per-VCPU mutable control state are exported using TDH.EXPORT.STATE.TD and TDH.EXPORT.STATE.VP respectively.

At any time, the export may be aborted by the host VMM using TDH.EXPORT.ABORT, which returns the source TD to the RUNNABLE state, where it can continue to run normally. No abort token is required at this phase since no start token has been generated and the destination TD, by design, should not be able to run.

Note: TDH.EXPORT.STATE.TD is expected to be called by the exporting host VMM prior to TDH.EXPORT.STATE.VP, but this is only enforced on the import side.

TDH.EXPORT.TRACK(done) generates a **start token** which the host VMM transmits to the destination VMM. It transitions the source TD OP_STATE into the POST_EXPORT state, starting the **out-of-order export phase**. Memory export (TDH.EXPORT.MEM) may continue, to support the out-of-order stage of the TD live migration.

In the TD Migration Session POST_EXPORT state, a TDH.EXPORT.ABORT with a valid **abort token**, received from the destination VMM, indicates that the TD, by design, should not be able to run on the destination platform. It terminates the export session and returns the source TD to the RUNNABLE state, where it can continue to run normally.

The host VMM can start tearing down the source TD at any time, by ensuring that no VCPU is associated with an LP (i.e., by executing TDH.VP.FLUSH for all VCPUs) and issuing TDH.MNG.VPFLUSHDONE. Typically, it will do so in the after it gets a notification from the destination platform that import has been successful.

6.2.5. Import Side (Destination Platform)

Migration TD binding (using TDH.SERVTD.BIND) and migration key assignment (using TDG.SERVTD.WR) must happen in the UNINITIALIZED state, where TDCS memory has already been allocated but the destination TD has not been initialized yet. This is required since the destination TD is going to be initialized by importing immutable state from the source TD.

TDH.IMPORT.STATE.IMMUTABLE starts the **in-order import phase**. It initializes the destination TD's TDCS with imported immutable state and transitions the destination TD's OP_STATE into MEMORY_IMPORT. In this state, TD private memory can be imported using TDH.IMPORT.MEM etc.

TDH.IMPORT.STATE.TD imports the per-TD mutable state and transitions the destination TD's OP_STATE into STATE_IMPORT. In this state, mutable VCPU state can be imported using TDH.IMPORT.STATE.VP. TD private memory import also continues.

Upon executing TDH.IMPORT.TRACK with a valid **start token** as operand, the destination TD's OP_STATE transitions into the POST_IMPORT state, starting the **out-of-order import phase**. Memory import (a.k.a. **post-copy**) may continue, but pages can only be imported if their GPA is free (i.e., the Secure EPT state is FREE).

An import failure up to this point, e.g., improper sequence of page import vs. alias import, or executing TDH.IMPORT.TRACK with a bad start token received from the source platform, transitions the TD's OP_STATE to the FAILED_IMPORT state. In addition, the host VMM can explicitly abort the import by using TDH.IMPORT.ABORT. In the FAILED_IMPORT state, the TD is designed not to run; it can only be torn down. TDH.IMPORT.ABORT generates an **abort token**, which can be transmitted to the source platform.

TDH.IMPORT.COMMIT transitions the destination TD's OP_STATE transitions into the LIVE_IMPORT state. In this state, the destination TD may run normally. Out-of-order memory import may continue as long as the destination TD is in the LIVE_IMPORT state. An input failure in the LIVE_IMPORT terminates the import session; it transitions the TD's OP_STATE to the RUNNABLE state, where the TD can continue running normally. Abort token can no longer be generated.

TDH.IMPORT.END ends the import session and transitions the destination TD's OP_STATE into the RUNNABLE state. This transition is optional if TDH.IMPORT.COMMIT has already been executed; it removes any limitations on TD memory management that exist during the out-of-order import phase.

A new export session (TDH.EXPORT.STATE.IMMUTABLE) terminates a previous out-of-order import.

6.2.6. OP_STATE Concurrency Considerations

6.2.6.1. *Export Side*

The following export functions typically result in an OP_STATE transition. They need to run while the source TD may be running, therefore they acquire a **shared** lock on the source TD (via its TDR page). To avoid concurrent execution with other export functions that may result in an OP_STATE transition, they acquire an **exclusive** lock on OP_STATE itself and must be serialized by the host VMM:

- TDH.EXPORT.STATE.IMMUTABLE
- TDH.EXPORT.ABORT

The following export functions typically result in an OP_STATE transition. The source TD does not run when the execute. They acquire an **exclusive** lock on the source TD (via its TDR page). This implicitly locks OP_STATE itself. These interface functions must be serialized by the host VMM:

- TDH.EXPORT.PAUSE
- TDH.EXPORT.TRACK

The following export functions do not result in an OP_STATE transition, but they depend on OP_STATE not changing during their execution. They acquire a **shared** lock on the source TD (via its TDR page) and a **shared** lock on OP_STATE itself.

- TDH.EXPORT.BLOCKW
- TDH.EXPORT.UNBLOCKW
- TDH.EXPORT.MEM
- TDH.EXPORT.STATE.TD
- TDH.EXPORT.STATE.VP
- TDH.MIG.STREAM.CREATE

6.2.6.2. *Import Side*

The following import functions typically result in an OP_STATE transition. They acquire an **exclusive** lock on the destination TD (via its TDR page) and must be serialized by the host VMM:

- TDH.IMPORT.STATE.IMMUTABLE
- TDH.IMPORT.STATE.TD
- TDH.IMPORT.TRACK
- TDH.IMPORT.COMMIT
- TDH.IMPORT.END
- TDH.IMPORT.ABORT

The following import functions do not typically result in an OP_STATE transition, but they may depend on OP_STATE not changing during their execution. To maximize import performance, they are designed to be executed concurrently on multiple LPs. These interface function acquire a **shared** lock on the destination TD (via its TDR page):

- TDH.IMPORT.MEM
- TDH.IMPORT.STATE.VP

However, all TDH.IMPORT.* functions may result in a transition to the FAILED_IMPORT state. For example, a TDH.IMPORT.MEM on one LP might transition to FAILED_IMPORT, while a concurrent TDH.IMPORT.MEM on another LP might still be in progress. The architecture is designed to harmlessly support this case; the FAILED_IMPORT state has no direct output transition – the destination TD can only be torn down, starting with TDH.MNG.VPFLUSHDONE which acquires an exclusive lock on the TD (via its TDR) and thus helps ensure that any TDH.IMPORT.* are not in progress.

Similarly, TDH.IMPORT.MEM may result in a transition to the RUNNING state, in case of an import error. The architecture is designed to harmlessly support this case; transitions out of the RUNNING state either acquire an exclusive lock on the OP_STATE or acquire an exclusive lock on the TD (via its TDR).

6.2.7. Summary

Table 6.1: OP_STATE Sub-States of TD_KEYS_CONFIGURED

Sub-State	Source / Destination	Description
UNALLOCATED	Both	TDCS memory is being allocated.
UNINITIALIZED	Both	• TDCS is pending initialization. • On the destination platform, migration TD binding and migration key assignment must happen in this state.
INITIALIZED	Source	• TD is being built. Memory is added and measured. VCPUs are created. • Migration TD binding may happen in this state.
RUNNABLE	Both	• TD can run.
LIVE_EXPORT	Both	• TD can run. • Immutable non-memory state (TDH.EXPORT.STATE.IMMUTABLE) has been exported. • Live memory can be exported (TDH.EXPORT.MEM etc.).
PAUSED_EXPORT	Source	• No TD VCPU may run. • TD memory can't be written. • Memory can be exported. • Mutable non-memory state is being exported.
POST_EXPORT	Source	• Start token has been generated on all streams that were active. • Mutable control state has been exported. • No TD VCPU may run. • TD memory can't be written. • Memory can be exported (post-copy).
MEMORY_IMPORT	Destination	• TD memory can be imported.
STATE_IMPORT	Destination	• TD memory can be imported. • TD VCPU non-memory state can be imported
POST_IMPORT	Destination	• TD memory can be imported (post-copy).
LIVE_IMPORT	Destination	• TD VCPUs may run. • TD memory can be imported (post-copy).
FAILED_IMPORT	Destination	• Destination TD will not run.

6.3. Migration Tokens

5 A migration token is formatted as a Migration Bundle, with only an MBMD. Its format is defined in the [TDX Module ABI Spec].

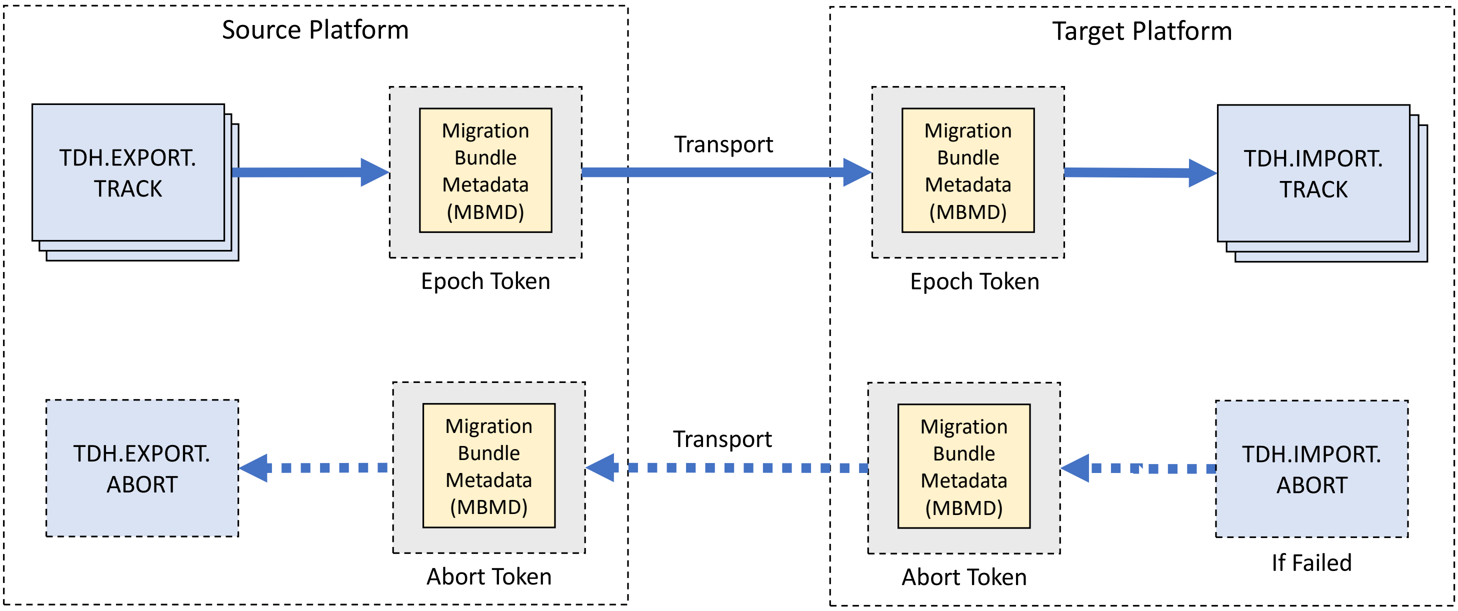

Figure 6.9: Migration Tokens

An **epoch token** is generated by TDH.EXPORT.TRACK. it serves as a separator between migration epochs. A **start token** is a special version of an epoch token which starts the out-of-order phase. The start token helps ensure that no newer version of any memory page exported prior to the start token exists on the source platform.

The **abort token** is generated by TDH.IMPORT.ABORT on the destination platform if import fails for any reason. It helps ensure that the TD will not run on the destination platform, and therefore may be restored on the source platform.

6.4. Migration Protocol Versioning

6.4.1. Introduction

Migration protocol version number is provided as part of the MBMD header. Migration protocol changes may require migration version increment and may impact source and destination compatibility. For example, this may happen due to:

- Incompatible MBMD format changes
- New values of MBMD fields
- New memory migration variants (e.g., support of aliases for VM nesting)
- Incompatible migration session state machine changes

Non-memory state (metadata) migration changes may also require migration version increment. For example, this may happen due to:

- New exported metadata fields
- New values or format of exported metadata fields

6.4.2. Enumeration of Supported Migration Versions

TDX module enumeration version support using global metadata fields that can be read by the host VMM (TDH.SYS.RD) and MigTD (TDG.SYS.RD).

On export, the TDX module can work with MIG_VERSION in the range specified by the following metadata fields:

MAX_EXPORT_VERSION: Maximum value of migration version supported for export

MIN_EXPORT_VERSION: Minimum value of migration version supported for export

For example, a module may be updated to support version X for new memory migration formats for VM Nesting. But it may be written to export using version X-1 if a non-VM-Nesting TD is exported to an older module.

On import, the TDX module understands MIG_VERSION in the range specified by the following:

MAX_IMPORT_VERSION: Maximum value of migration version supported for export

MIN_IMPORT_VERSION: Minimum value of migration version supported for import

For example, if the module supports version X that was created to support new memory migration formats for VM Nesting, it can still understand version X-1 that doesn't use the new formats.

6.4.3. Setting the Migration Protocol Version for a Migration Session

Migration protocol version to be used for a migration session is set the MigTD before the session starts. MigTDs at the source and destination platform enumerate export and import versions support respectively, and decide on the version that is compatible between the platforms, to be used for the migration session. TD-scope metadata field MIG_VERSION is writable by the MigTD using TDH.SERVTD.WR. At the start of the migration session, the TDX module copies MIG_VERSION to an internal WORKING_MIG_VERSION that is used throughout the session.

6.5. Export Session Control Functions

This section provides an overview of the export session control functions. A detailed description is provided in [TDX Module ABI Spec].

6.5.1. TDH.EXPORT.STATE.IMMUTABLE (Session Control Aspects)

TDH.EXPORT.STATE.IMMUTABLE starts an export session. It also exports the TD's immutable state – that functionality is discussed in 7.3.1.

The host VMM is expected, prior to invoking TDH.EXPORT.STATE.IMMUTABLE, to create enough migration streams contexts using TDH.MIG.STREAM.CREATE.

Inputs

- Source TD handle: the TDR page HPA
- MBMD HPA
- Migration Page List HPA and size
- Migration stream index
- Number of migration streams that will be used during the in-order phase

Pre-Conditions

- TD is runnable
- A new migration key has been set

Operation (Session Control Aspects Only)

1. Start the export session in-order phase.
2. Export a TD Immutable State MBMD. The MBMD details the number of migration streams that will be used during the in-order phase and the maximum number of migration streams.

6.5.2. TDH.EXPORT.PAUSE

TDH.EXPORT.PAUSE pauses the source TD starts the TDX-enforced blackout period. This operation is local to the source platform and is not communicated to the destination platform.

Inputs

- Source TD handle: the TDR page HPA

Outputs

- Success or failure status

Pre-Conditions

- TD immutable state has been exported.
- All TD VCPUs have stopped executing and no other TD-specific SEAMCALL is running.

Operation

- Prevent further TD entries and host-side functions that may modify the TD state.

6.5.3. TDH.EXPORT.TRACK

TDH.EXPORT.TRACK starts a new migration epoch. It generates an epoch token. If so requested, it starts the out-of-order phase and generates a start.

Inputs

- Source TD handle: the TDR page HPA
- Migration stream index

Outputs

- Epoch token migration bundle

Pre-Conditions

- The export session is in progress, but its out-of-order phase has not begun yet.
- If a start token is to be generated:
  For any page that has been exported so far, an up-to-date version of that page has been exported.

  Note: TDH.EXPORT.TRACK does not check that all non-memory state has been exported. This is checked on the destination platform by TDH.IMPORT.TRACK.

Operation

1. Start a new migration epoch.
2. Create an epoch token migration bundle which includes only an MBMD.

6.5.4. TDH.EXPORT.ABORT

TDH.EXPORT.ABORT aborts the export session. If invoked during the in-order export phase, after the TD has been paused, it re-enables the TD to run on the source platform. If invoked during the out-of-order phase, it consumes an abort token received from the destination platform; if that token is correct it enables the TD to run on the source platform.

Inputs

- Source TD handle: the TDR page HPA
- Optional: migration bundle containing the abort token received from the destination platform.

Pre-Conditions

- An export session is in progress.
- Export has not been committed by TDH.EXPORT.COMMIT.

Operation

- Terminate the export session: Invalidate all migration contexts for source TD.
- If the export session is in the in-order phase, re-enable the TD.
- If the export session is in the out-of-order phase, check the abort token. If valid, re-enable the TD.

6.6. Import Session Control Functions

This section provides an overview of the import session control functions. A detailed description is provided in [TDX Module ABI Spec].

6.6.1. TDH.IMPORT.STATE.IMMUTABLE (Session Control Aspects)

TDH.IMPORT.STATE.IMMUTABLE consumes a TD immutable state migration bundle and starts the import session in-order phase. It also imports the TD’s immutable state – that functionality is discussed in 7.4.1.

The host VMM is expected, prior to invoking TDH.IMPORT.STATE.IMMUTABLE, to parse the received MBMD, determine the number of migration streams required and assure that enough migration stream contexts have been created using TDH.MIG.STREAM.CREATE.

Inputs

- Destination TD handle: the TDR page HPA
- MBMD HPA
- Migration Page List HPA and size

Pre-Conditions

- TDCS been allocation but not initialized.
- A new migration key has been set.

Operation (Session Control Aspects Only)

Start the import session in-order phase.

Any failure aborts the operation and marks the TD as IMPORT_FAILED; it will not run.

6.6.2. TDH.IMPORT.TRACK

TDH.IMPORT.TRACK consumes an epoch token received from the source platform and starts a new epoch. If a start token is received, TDH.IMPORT.TRACK starts the import session in-order phase.

Any failure aborts the operation and marks the TD as IMPORT_FAILED; it will not run.

Inputs

- Destination TD handle: the TDR page HPA
- Migration stream index
- Migration bundle containing the epoch token

Outputs

- None

Pre-Conditions

- The import session is in progress, but its out-of-order phase has not begun yet.
- The start token migration bundle is imported successfully.
- If a start token is received, all mutable state must have been imported.

Operation

1. Starts a new migration epoch.
2. If a start token has been received, start the out-of-order import phase.

6.6.3. TDH.IMPORT.COMMIT

TDH.IMPORT.COMMIT enables the TD to run.

Inputs

- Destination TD handle: the TDR page HPA

Outputs

- None

Pre-Conditions

- The import session out-of-order phase is in progress.

Operation

- Set the TD’s OP_STATE to LIVE_IMPORT, allowing it to run.

6.6.4. TDH.IMPORT.END

TDH.IMPORT.END ends the import session.

Inputs

- Destination TD handle: the TDR page HPA

Outputs

- None

Pre-Conditions

- The import session out-of-order phase is in progress.

Operation

- Set the TD's OP_STATE to RUNNABLE, ending the import session and allowing the TD to run.

6.6.5. TDH.IMPORT.ABORT

TDH.IMPORT.ABORT aborts the import session (if not already aborted) and generates an abort token. The TD will not run.

Inputs

- Destination TD handle: the TDR page HPA

Outputs

- Migration bundle containing the abort token

Pre-Conditions

- The import session is in progress.
- The TD is not allowed to run yet (OP_STATE is not LIVE_IMPORT).

Operation

- Generate an abort token MBMD.
- Set the OP_STATE to FAILED_IMPORT. In this state it can only be torn down.

7. TD Non-Memory State Migration

This chapter discusses all non-memory state migration, immutable and mutable.

TD-scope non-memory state resides in control structures TDR and TDCS. TD VCPU state resides (while the VCPU is not running) in TDVPS, which includes the TD VMCS. This chapter discusses how non-memory state migration is migrated.

7.1. TD Non-Memory State Migration Operation

7.1.1. Non-Memory State Migration Data

Non-memory state migration data is used for migrating immutable state, at the beginning of the migration process, by TDH.EXPORT.STATE.IMMUTABLE and TDH.IMPORT.STATE.IMMUTABLE, and for migrating mutable state, at the end of the migration process, by TDH.EXPORT.STATE.TD, TDH.IMPORT.STATE.TD, TDH.EXPORT.STATE.VP and TDH.IMPORT.STATE.VP.

The non-memory state is migrated in a way that abstracts the actual TD control structure format, allowing that format to remain implementation-dependent and vary between the source and destination platforms. To support that, each of the state migration data's 4KB pages contains a **TD metadata list**, composed of multiple **field sequences**. Each field sequence contains a **sequence header** and one or more **field values**.

Metadata abstraction is discussed in the [TDX Module Base Spec].

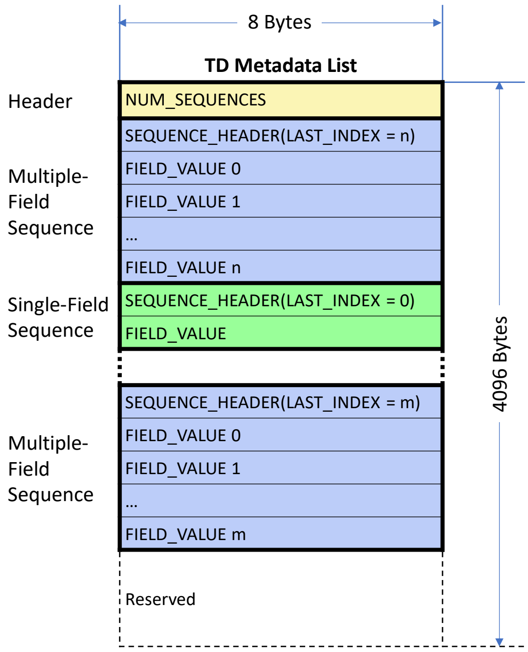

Figure 7.1: Non-Memory State Migration Page

Metadata fields are exported in order of their field codes. This enables easy identification of missing required fields on import.

7.1.2. Non-Memory State MBMD

The MBMD for each non-memory state migration bundle contains the following type-specific fields:

- **Metadata type**: Immutable TD-scope metadata or mutable L1 VCPU-scope metadata
- VM index and VCPU index (if applicable).

Details of the non-memory state MBMD are defined in the [TDX Module ABI Spec].

7.1.3. Immutable vs. Mutable TD State

In the scope of TD migration, **immutable** state is defined as any TD state that may not change after TD build, i.e., after TD measurement has been finalized (by TDH.MR.FINALIZE).

Migrated immutable state includes the following:

- Platform-scope immutable state required so that the TDX module on the destination platform can verify compatibility. Namely, it includes the **source TDX module's version information**.
- TD-scope immutable state of the source TD

Immutable TD state export and import functions (TDH.EXPORT.STATE.IMMUTABLE, TDH.IMPORT.STATE.IMMUTABLE) start the migration session. Migration session control is discussed in Ch. 5.5.

Migrated mutable state includes the following:

- TD-scope mutable state
- VCPU-scope mutable state

Mutable TD state export is done after the TD has been paused (by TDH.EXPORT.PAUSE), and helps ensure that the state will not change anymore until TD export completes. TDH.EXPORT.STATE.TD exports TD-scope mutable state, followed by multiple, per-VCPU TDH.EXPORT.STATE.VPs which export VCPUs mutable state.

Mutable TD state import must begin with TD-scope state import (by TDH.IMPORT.STATE.TD), followed by multiple, per-VCPU TDH.IMPORT.STATE.VP which import VCPUs state.

7.2. State Migration Rules

7.2.1. General State Export Rules

Only state that may be used for import on the destination platform is exported from the source platform. State that is never imported, or that is not in use based on the TD configuration (ATTRIBUTES, XFAM and CPUID configuration), is not exported. For example:

- KeyLocker state is not exported if ATTRIBUTES.KL is not set to 1.
- Only the defined VAPIC page fields are exported.

There may be exceptions where state is exported even if it's not explicitly imported. This may be done for possible future compatibility or for simplicity of export. For example:

- TDX module version information – this information is exported so a future TDX module may examine it on import, to take some action due to possible incompatibility or bug of the exporting TDX module.

7.2.2. General State Import Rules

In addition to the immutable or mutable classification, non-memory state can be classified as **migrated** and/or **initialized**. For migrated each state component, import may be **mandatory** or **optional**. For optionally migrated state, a default initial value must be specified.

Each import function verifies that all the applicable mandatory state has been imported and initializes the default values for state components that have not been imported.

7.2.3. Immutable State Import Rules

TD immutable state is verified by TDH.IMPORT.STATE.IMMUTABLE against destination platform capabilities and Intel TDX module version, capabilities and configuration. The checks are similar, but not identical, to the TD_PARAMS checks done on the source platform by TDH.MNG.INIT. For example:

- TD ATTRIBUTES bits must be compatible with the destination platform and Intel TDX module configuration.
- Any XFAM bit that was set on the source platform by TDH.MNG.INIT must be supported by the destination platform.
- Virtual CPUID configuration is calculated on the source platform by TDH.MNG.INIT. This configuration is exported and checked on import to be compatible with the destination platform. If a certain CPUID leaf or sub-leaf is not virtualized on the source platform (i.e., its execution by the TD results in a #VE), it is also not virtualized on the destination platform – even if it would be virtualized on that platform if the TD was created there. CPUID virtualization, including fine-grained virtualization of sub-features, is described in the [TDX Module Base Spec].

Immutable state that can be regenerated on the destination platform is not imported. For example:

- The TD's MSR exit bitmaps are generated by TDH.IMPORT.STATE.IMMUTABLE, like the way they are generated by TDH.MNG.INIT on the source platform.

7.2.4. Mutable State Import Rules

Intel SDM, Vol. 3, 26.3.1	Checks on the Guest State Area
Intel SDM, Vol. 3, 26.8	VM-Entry Failures During or After Loading Guest State

7.2.4.1. *Imported State Verification*

Mutable non-memory state is verified by TDH.IMPORT.STATE.TD and TDH.IMPORT.STATE.VP against destination platform capabilities and Intel TDX module configuration. For example:

- CR0 and CR4 values are verified using the same rules used for CR0 and CR4 virtualization, respectively. This is required because the values of the IA32_VMC_CR*_FIXED* MSRs may be different between the platforms. For details, see the [TDX Module Base Spec].
- CPUID virtualization state, originally calculated on TD initialization based on configuration and the capabilities of the source CPU, is verified versus the capabilities of the destination CPU.

7.2.4.2. *Handling State that is Not Verified on Import*

In some cases, immutable VCPU state is difficult to verify during TDH.IMPORT.STATE.VP. This may include, for example:

- Guest MSR state saved in TDVPS
- Guest state saved in TD VMCS

In those cases, the TDX Module reports the incompatibility on TDH.VP.ENTER using CPU compatibility checks, as follows:

- The Intel TDX module gracefully handles WRMSR errors, i.e., #GP(0), that occur during the TDH.VP.ENTER flow when loading guest MSR values. In this case, the Intel TDX module marks the TD as FATAL and TDH.VP.ENTER terminates with an error code.

  Note: This is different from the current TDX 1 behavior, but may be implemented in TDX 1 due to a requirement to support host VMM writes of guest MSRs for debug TDs.

- The Intel TDX module gracefully handles guest state checks that fail during VM entry. In this case, the CPU behavior is like a VM exit, with the exit reason indicating VM-entry failure due to invalid guest state, MSR loading or a machine-check event. In this case, the Intel TDX module marks the TD as FATAL and TDH.VP.ENTER terminates with an error code.

  Note: This is different from the current TDX 1 behavior but may be implemented in TDX 1 due to a requirement to support host VMM writes of guest MSRs for debug TDs.

7.2.4.3. *State Initialized or Calculated on Import*

Many of the TD VMCS execution controls that control the host VMM interaction with the guest TD are reset to their initial state on import. These include, for example:

- Posted interrupt execution controls (see the [TDX Module Base Spec])

Other state is calculated on import. For example:

- The virtual TSC value, sampled and exported by TDH.EXPORT.STATE.TD, is used by TDH.IMPORT.STATE.TD to calculate a new TSC offset so that the virtual TSC value will continue as a monotonously incrementing value on the destination platform. For details see the [TDX Module Base Spec].

7.3. *Non-Memory State Export Functions*

7.3.1. TDH.EXPORT.STATE.IMMUTABLE (State Export Aspects)

TDH.EXPORT.STATE.IMMUTABLE exports the TD's immutable state as a multi-page migration bundle. It also starts the export session – that functionality is described in 6.5.1.

A detailed description of TDH.EXPORT.STATE.IMMUTABLE is provided in the [TDX Module ABI Spec].

Inputs

- Source TD handle: the TDR page HPA
- MBMD HPA
- Migration Page List HPA and size
- Migration stream index
- Number of migration streams that will be used during the in-order phase

Pre-Conditions

- TD is runnable
- A new migration key has been set

Operation (State Export Aspects Only)

1. Export the TD's immutable state:
   1.1. Serialize and encrypt the TD exported immutable state into the multi-page migration data buffer.
   1.2. Update the MBMD with the MB type, stream index, metadata type and the MAC.

7.3.2. TDH.EXPORT.STATE.TD

TDH.EXPORT.STATE.TD exports the TD-scope mutable state as a multi-page migration bundle.

A detailed description of TDH.EXPORT.STATE.TD is provided in the [TDX Module ABI Spec].

Inputs

- Source TD handle: the TDR page HPA
- MBMD HPA
- Migration Page List HPA and size
- Migration stream index

Pre-Conditions

- The export session is in the in-order phase and the TD has been paused

Operation

1. Export the TD's mutable state:
   1.1. Serialize and encrypt the TD exported mutable state into the multi-page migration data buffer.
   1.2. Update the MBMD with the MB type, stream index, metadata type and the MAC.

7.3.3. TDH.EXPORT.STATE.VP

TDH.EXPORT.STATE.VP exports the VCPU-scope mutable state as a multi-page migration bundle.

A detailed description of TDH.EXPORT.STATE.VP is provided in the [TDX Module ABI Spec].

Inputs

- Source VCPU handle: the TDVPR page HPA
- MBMD HPA
- Migration Page List HPA and size
- Migration stream index

Pre-Conditions

- The export session is in the in-order phase and the TD has been paused

Operation

1. Export the VCPU's mutable state:
   1.1. Serialize and encrypt the VCPU exported mutable state into the multi-page migration data buffer.
   1.2. Update the MBMD with the MB type, stream index, metadata type and the MAC.

7.4. Non-Memory State Import Functions

7.4.1. TDH.IMPORT.STATE.IMMUTABLE (State Import Aspects)

TDH.IMPORT.STATE.IMMUTABLE imports the TD's immutable state as a multi-page migration bundle. It also starts the import session – that functionality is described in 5.5.

A detailed description of TDH.IMPORT.STATE.IMMUTABLE is provided in the [TDX Module ABI Spec].

Inputs

- Destination TD handle: the TDR page HPA
- MBMD HPA
- Migration Page List HPA and size

Pre-Conditions

- TD has not been initialized
- A new migration key has been set

Operation (State Import Aspects Only)

1. Initialize TDCS default values.
2. Read MBMD into an internal buffer.
3. To save internal buffer space, the steps below can be done on, e.g., one import data page at a time:
   3.1. Decrypt the TD immutable state from the multi-page migration data buffer into a temporary buffer.
   3.2. De-serialize the imported fields, verify, then set TDR and TDCS fields based on the imported values.
4. Verify the calculated MAC versus the value read from the MBMD.
5. Verify that all required fields have been imported.

Any verification failure aborts the operation and marks the TD as IMPORT_FAILED; it will not run.

7.4.2. TDH.IMPORT.STATE.TD

TDH.IMPORT.STATE.TD imports the TD's mutable state as a multi-page migration bundle.

A detailed description of TDH.IMPORT.STATE.TD is provided in the [TDX Module ABI Spec].

Inputs

- Destination TD handle: the TDR page HPA
- MBMD HPA
- Migration Page List HPA and size

Pre-Conditions

- TD immutable state has been imported

Operation

1. Read MBMD into an internal buffer.
2. To save internal buffer space, the steps below can be done on, e.g., one import data page at a time:
   2.1. Decrypt the TD mutable state from the multi-page migration data buffer into a temporary buffer.
   2.2. De-serialize the imported fields, verify, then set TDR and TDCS fields based on the imported values.
3. Verify the calculated MAC versus the value read from the MBMD.
4. Verify that all required fields have been imported.
5. Allow VCPU state import: Set TDCS.OP_STATE to STATE_IMPORT.

Any verification failure aborts the operation and marks the TD as IMPORT_FAILED; it will not run.

7.4.3. TDH.IMPORT.STATE.VP

TDH.IMPORT.STATE.VP imports a VCPU mutable state as a multi-page migration bundle.

A detailed description of TDH.IMPORT.STATE.VP is provided in the [TDX Module ABI Spec].

Inputs

- Destination VCPU handle: the TDVPR page HPA
- MBMD HPA
- Migration Page List HPA and size

Pre-Conditions

- TDVPS pages have been allocated by the host VMM, but the VCPU has not been initialized.
- TD-scope state has been imported

Operation

1. Initialize the TDVPS (including TD VMCS) default values.
2. Read MBMD into an internal buffer.
3. To save internal buffer space, the steps below can be done on, e.g., one import data page at a time:
   3.1. Decrypt the VCPU mutable state from the multi-page migration data buffer into a temporary buffer.
   3.2. De-serialize the imported fields, verify, then set TDVPS (including TD VMCS) fields based on the imported values.
4. Verify the calculated MAC versus the value read from the MBMD.
5. Verify that all required fields have been imported.

Any verification failure aborts the operation and marks the TD as IMPORT_FAILED; it will not run.

Section 2: TD Migration Architecture Specification

8. TD Private Memory Migration

This chapter described how Intel TDX Module manages TD private memory and guest-physical (GPA) address translation meta-data migration.

8.1. Overview

TD private memory migration can happen in the in-order migration phase and out-of-order migration phase.

During the in-order phase, the host VMM may implement **live migration pre-copy**, by exporting memory content (using TDH.EXPORT.MEM etc.) while the TD is running (TDCS.OP_STATE is LIVE_EXPORT). This is not enforced by TDX; the host VMM may implement **cold migration** by avoiding memory export until the TD is paused.

During the out-of-order phase, the host VMM may implement **post-copy** by allowing the TD to run on the destination platform (using TDH.IMPORT.COMMIT). This is not enforced by TDX; the host VMM can first complete all memory migration before allowing the TD to run, yet benefit from the simpler and potentially higher performance operation supported during the out-of-order phase.

8.2. Achieving Memory Migration Security Objectives

8.2.1. General

The key security design goal for TD Private memory migration is to ensure integrity and freshness of the TD private memory at the destination TD after migration – this helps ensure that a malicious VMM cannot execute the TD after migration with any stale or modified data.

Integrity of memory includes the memory contents as well as the guest physical to host physical mapping and attributes that control TD access to private memory.

Using PAMT and Secure EPT, Intel TDX Module enforces the following properties for TD private GPA accesses:

Unique TD Association: A physical page used as a TD private page, Secure EPT page or a control structure can only be assigned to single guest TD.

Unique GPA Mapping: A TD private page or a Secure EPT page can be mapped at most by single guest TD GPA.

These security properties are maintained for a TD during migration with some additional functionality afforded to allow for live migration.

- Private TD pages and Secure EPT entries (for partitioned TDs, this includes L2 Secure EPT entries) are initialized in a single operation (via TDH.IMPORT.MEM) for pages migrated using TDH.EXPORT.MEM. Like the pre-conditions for the non-migration TDH.MEM.PAGE.ADD, the parent Secure EPT entry must be free (unmapped).
- On the source platform, a private page may be mapped as non-writable (a.k.a. blocked for writing) to allow for the page contents to be exported. For partitioned TDs, this any L2 mapping of a page is also mapped as non-writable. Following from previous security requirements, this mapping update also requires TLB tracking to help ensure that no active writable cached GPA address translations exist to the to-be-migrated GPA range.
- For 1GB and 2MB pages, secure EPT mapping demotion (to a 4KB page size) is required as a pre-condition to exporting contents of a page for migration.
- The Migration key used for exporting and important TD memory and CPU state is distinct from keys used for other operations such as Paging.

8.2.2. Migration Epochs: Usage of Stale Memory Copies due to Mis-Ordering

Running the destination TD with a stale copy of a memory page, because an older copy of a page was imported after a newer copy of that page, is prevented by the migration epochs mechanism. Within each migration stream, proper ordering is maintained by the migration bundle counter (MB_COUNTER) of each MBMD. However, there is no intrinsic guarantee of ordering across migration streams.

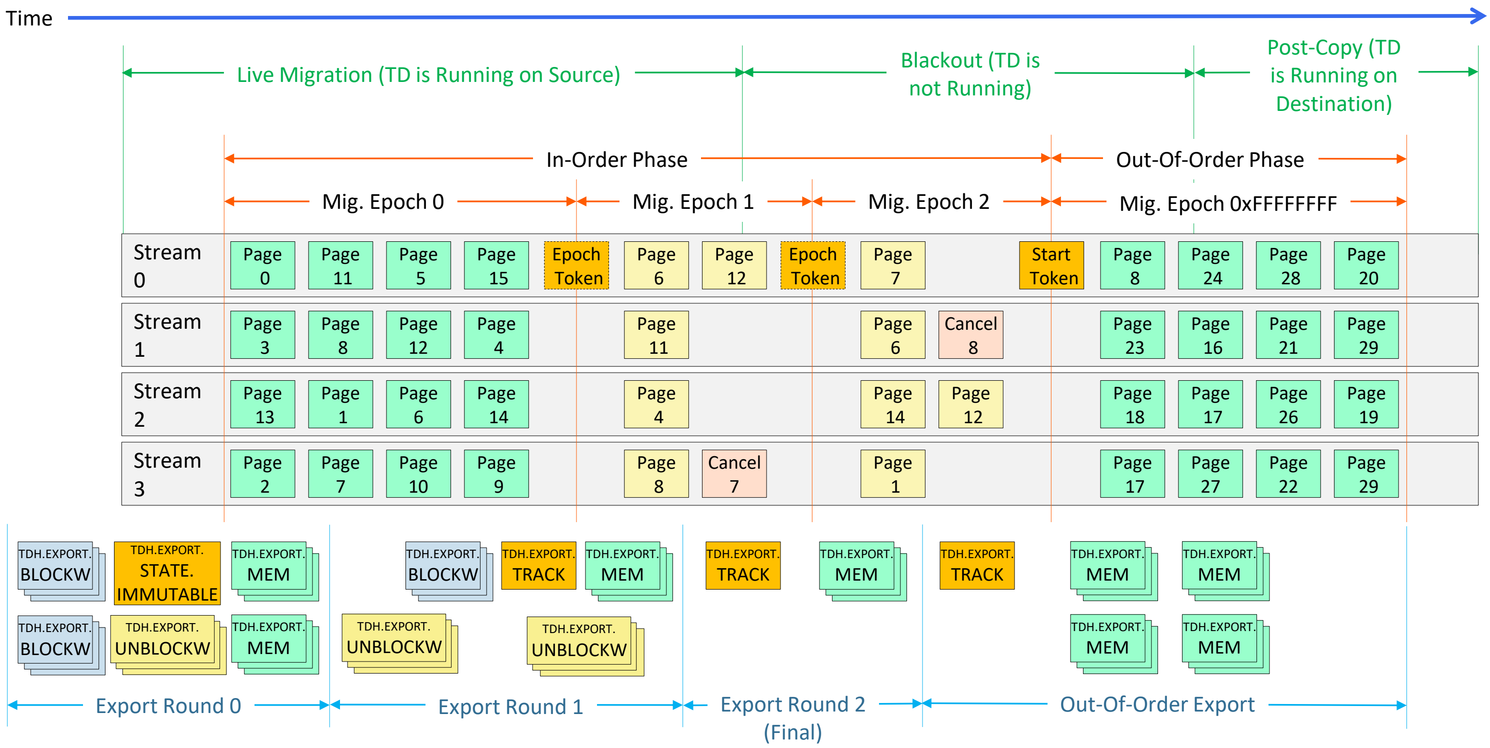

Figure 8.1: Migration Epochs

To help ensure overall ordering, the migration session is divided to **migration epochs**. A given page can only be imported, or its import can be cancelled, once per migration epoch. An **epoch token** serves as an epoch separator. It provided the total number of migration bundles exported so far. This helps TDH.IMPORT.TRACK, which imports the epoch token, checks that all migration bundles of the previous epoch have been received. No migration bundle of an older epoch may be imported.

The **start token**, which starts the out-of-order phase, is a special version of the epoch token. Epoch number 0xFFFFFFFF indicates the out-of-order phase.

Note: Migration epoch is a TDX concept. It roughly corresponds to **migration round** (or **migration pass**) which is a usage concept.

8.2.3. Preventing Usage of Stale Memory Copies due to Failure to Import

Running the destination TD with a stale copy of a memory page, because the target VMM failed to import a newer copy of a page, is prevented as follows:

Newer page state can only be generated before the source TD is paused (by TDH.EXPORT.PAUSE). Assume for example that two versions (v1 and v2) of the same page were exported, but the destination platform's VMM only imports the older version (v1), withholding the newer one (v2).

The in-order phase commitment protocol is designed to ensure that the export will fail, and the destination TD will not run. TDH.EXPORT.TRACK with an in-order-done parameter generates a start token that is dependent of the exact export sequence; it checks that no unexported newer versions of previously exported pages remain. The start token is verified by TDH.IMPORT.TRACK; the out-of-order migration phase may start, and the destination TD may run only if the start token verifies correctly. For migration session control details see Ch. 6.

8.2.4. Preventing Usage of Stale Memory Copies due to Failure to Export

Running the destination TD with a stale copy of a memory page, because the source VMM failed to export a newer copy of a page, is prevented as follows:

Assume for example that the source VMM exported an older version (v1) of page but never exported a newer version (v2) of that page. In this case, generating a start token by TDH.EXPORT.TRACK is prevented. A counter of dirty pages (TDCS.DIRTY_COUNT) is accumulated by the TDX module at source platform. If the value of that counter is not 0, then TDH.EXPORT.TRACK fails. See 8.4.7.3 for details.

8.2.5. Preventing Usage of Stale Memory GPA Mapping and Attributes

The destination TD is preventing from running with a copy of a memory page with stale GPA mapping, access permissions and other attributes (for partitioned TD, this included L2 mapping) as follows:

- GPA mappings and attributes are migrated together with their respective pages.
- Any change to GPA mappings or attributes is considered a change to the page and requires re-migration.

8.2.6. Out-Of-Order Phase and Its Usage for Post Copy

In the in-order import phase the VMM can import page for GPA addresses that are free, and it may also reload newer versions of pages to previously imported and present GPA addresses. In the out-of-order import phase, import is only allowed to non-present GPA addresses. At this stage, all memory state on the source platform is designed to be immutable, and the latest version of all pages exported so far will be imported. Thus, the order of out-of-order import is not relevant – except that memory content exported during the in-order phase can't be imported during the out-of-order phase. This allows using a separate migration stream for high-priority, low-latency updates, e.g., to implement **post-copy** by allowing the TD to run and migrate memory pages on demand at a high priority, based on EPT violation.

8.3. GPA Lists and Private Memory Migration Bundle

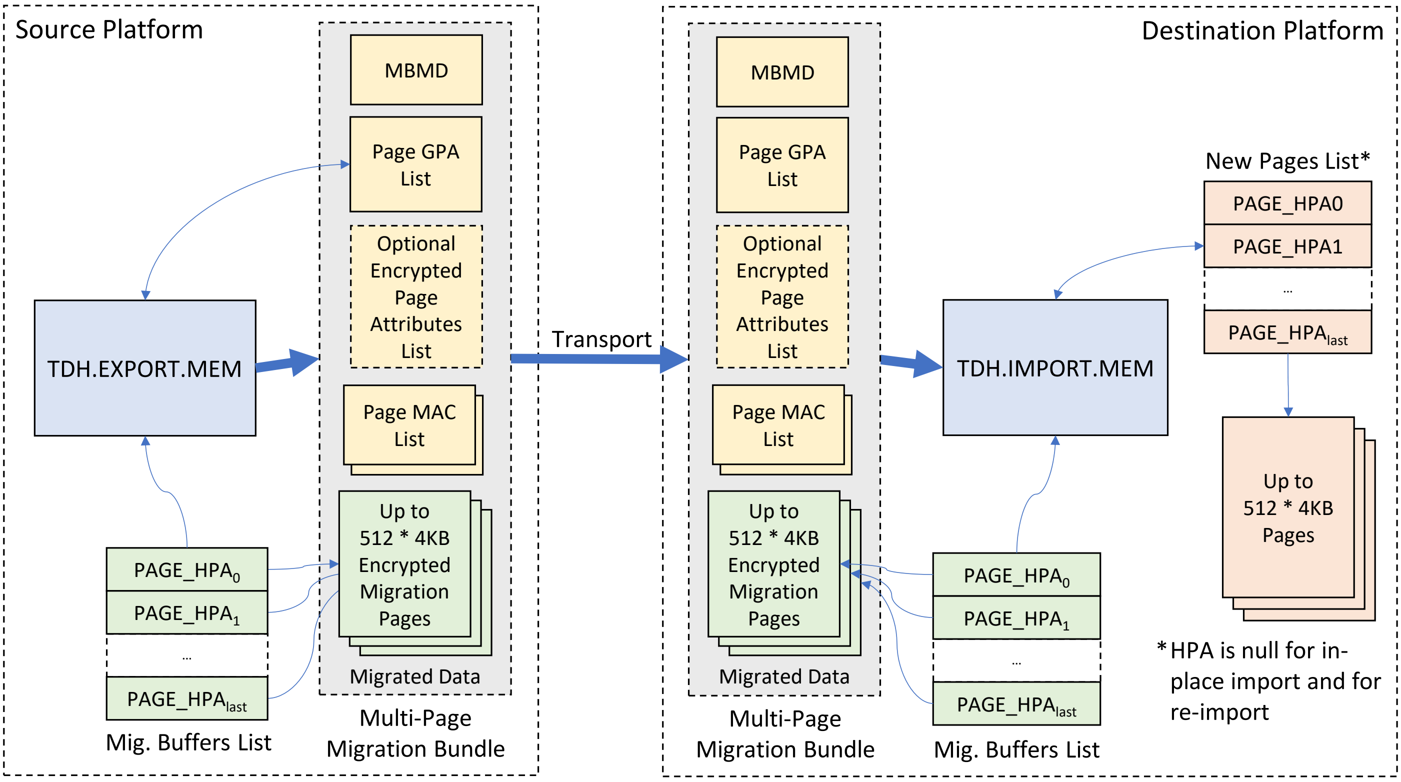

Figure 8.2: Private Memory Migration

Contrary to the generic migration bundle structure described in 5.1, private memory migration bundle is composed of multiple MAC-protected components:

- MAC-protected MBMD
- For each 4KB page:
  - Encrypted and MAC-protected 4KB migration buffer
  - MAC-protected page GPA and additional metadata
  - Optionally, for partitioned TDs, encrypted and MAC-protected page attributes

This structure allows the export and import functions to process the MBMD and each page and its metadata separately, avoiding the need to perform SEPT walks twice and to hold intermediate SEPT entry states. The separate parts of the migration bundle are cryptographically bound together as follows:

- A per-stream monotonously incrementing IV_COUNTER and the migration steam index are used for calculating the AES-GCM IV value, as described in 5.3.

- This is first done for the migration bundle's MBMD MAC.
- For each page, the IV_COUNTER is incremented by 1 and a new IV value is calculated and used for the page metadata MAC.
- The MBMD specifies the number of pages migrated by the migration bundle. This helps check that the whole migration bundle is imported on the destination platform.

8.3.1. GPA List

A GPA list is used as part of the private memory migration bundle. It is also used as an input and output of multiple memory migration interface functions: TDH.EXPORT.BLOCKW, TDH.EXPORT.MEM, TDH.EXPORT.RESTORE and TDH.IMPORT.MEM.

A GPA list contains up to 512 entries, each containing the following information:

Table 8.1: GPA List Entry Abstract Definition

Field	Page Migration
Type	4KB Page
GPA	GPA bits 51:12
State	MAPPED or PENDING
Operation	NOP, MIGRATE, REMIGRATE or CANCEL

A detailed definition of the GPA list is provided in the [TDX Module ABI Spec].

A single GPA list entry, a separate page MAC list entry and an optional separate page attributes list entry compose the page metadata.

8.3.2. Optional Page Attributes List

If the migrated TD is partitioned (TDCS.NUM_L2VMS is greater than 0), a page attributes list is used to extend the GPA list with the page attributes.

A page attributes list contains the same number of entries as the GPA list. Each entry contains 3 sub-entries, one for each L2 VM, each containing the following information:

- A flag indicating that the page attributes for this L2 VM are valid.
- Migratable page attributes: R, W, Xs, Xu, SSS, VGP, PWA, SVE

A detailed definition of the page attributes list is provided in the [TDX Module ABI Spec].

8.3.3. Private Memory Migration Buffer

Migration buffer is included only if the page metadata indicates a MIGRATE or a REMIGRATE request, and the page is MAPPED. it contains the encrypted content of the migrated page.

8.4. TD Private Memory Export

8.4.1. Typical Export Round

Typical expected usage divides the export session to **export rounds** (or **passes**). An export round may have the following steps:

1. If the TD has not been paused by TDH.EXPORT.PAUSE, ensure TLB shootdown:
   1.1. Invoke TDH.EXPORT.BLOCKW with a list of pages to be exported.
   1.2. Invoke TDH.MEM.TARCK.
   1.3. Issue IPIs to ensure TD re-entry on all VCPUs and TLB invalidation.
2. Start a new **migration epoch** by invoking TDH.EXPORT.TRACK.
3. Invoke TDH.EXPORT.MEM with a list of pages.
   3.1. If a page is being exported, mark its list entry as MIGRATE.

3.2. If a page has been exported before but need to be removed, promoted or demoted, cancel its migration by marking its list entry as CANCEL.

Note: In the example above, steps 1 and 2 need to be performed before step 3, but there is no strict requirement for the order of step 2 vs. step 1.

8.4.2. Using the same GPA List for TDH.EXPORT.BLOCKW and TDH.EXPORT.MEM

TDH.EXPORT.BLOCKW and TDH.EXPORT.MEM use GPA lists with compatible formats. This allows the same list to be used for blocking and exporting memory, as follows:

1. The host VMM on the source platform may prepare a GPA list with MIGRATE and CANCEL commands, and provide it as input to TDH.EXPORT.BLOCKW.
2. TDH.EXPORT.BLOCKW will attempt to block pages whose command is MIGRATE, and update the GPA list depending on the success of the operation.
3. The same GPA list can be provided as input to TDH.EXPORT.MEM, which then updates it depending on the success of the operation and whether this is a first time export (MIGRATE) or re-export (REMIGRATE), and adds page attributes information.

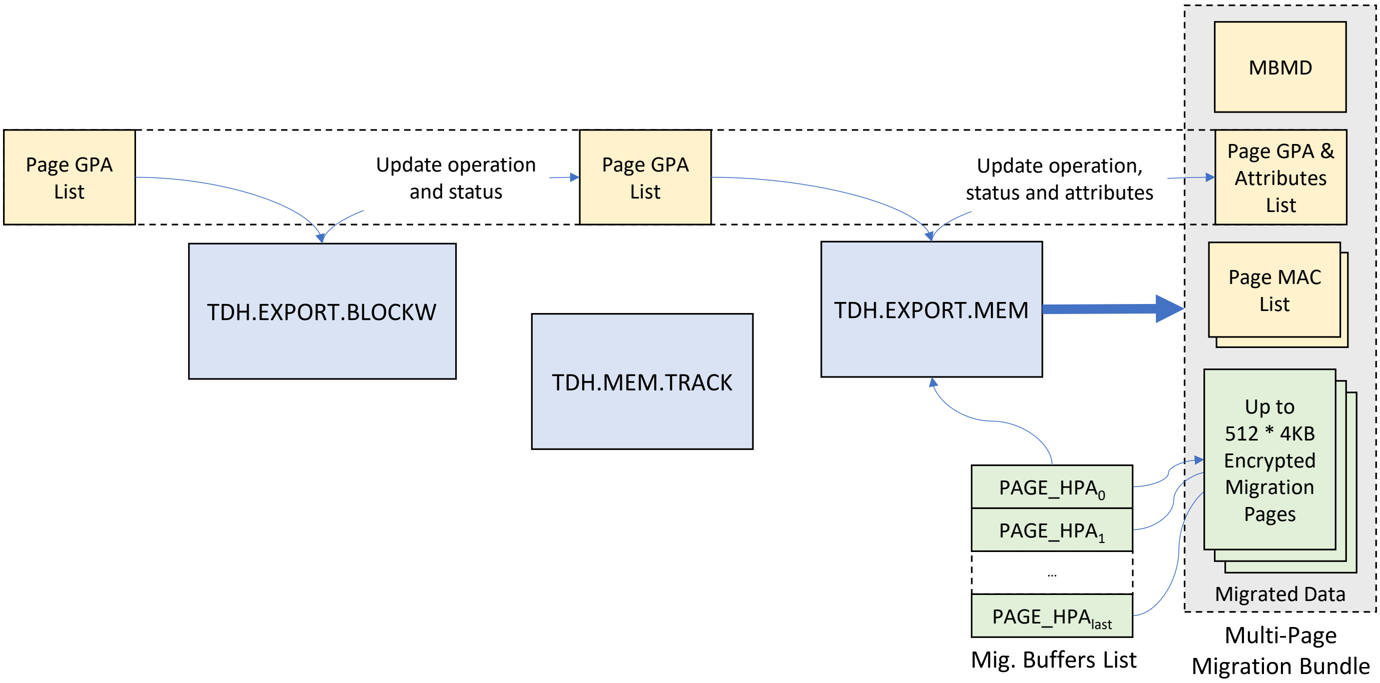

Figure 8.3: Typical Memory Export Round and the GPA List

A detailed definition of the GPA list is provided in the [TDX Module ABI Spec].

8.4.3. SEPT Leaf Entry Partial State Diagram for Mapped Page Export

Figure 8.4 below shows a partial SEPT entry state diagram for exporting mapped pages. The following sections describe the details.

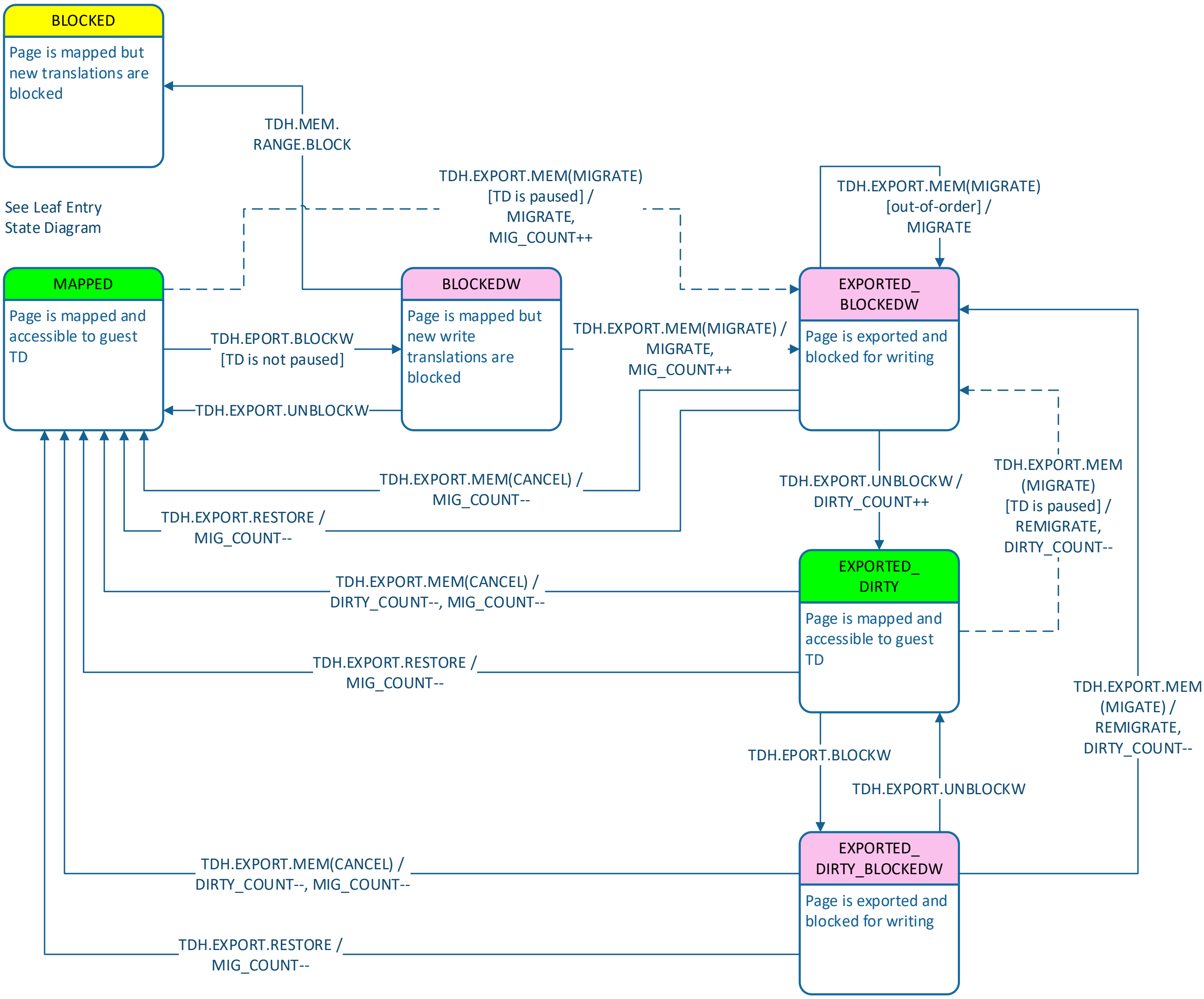

Figure 8.4: Partial SEPT Leaf Entry State Diagram for Mapped Page Export

8.4.4. L2 SEPT Leaf Entry Partial State Diagram for Mapped Page Export

For partitioned TDs, migration of page L2 attributes is done together with the page itself; it is controlled by the L1 SEPT entry state. Thus, the L2 SEPT entry state machine is quite simple, and is designed to control access to the page by the L2 VM. Specifically, there are no additional states and transitions for page export. When a page is blocked for writing (L1 SEPT state is one of the *BLOCKEW* states), the L2 SEPT entry state becomes L2_BLOCKED.

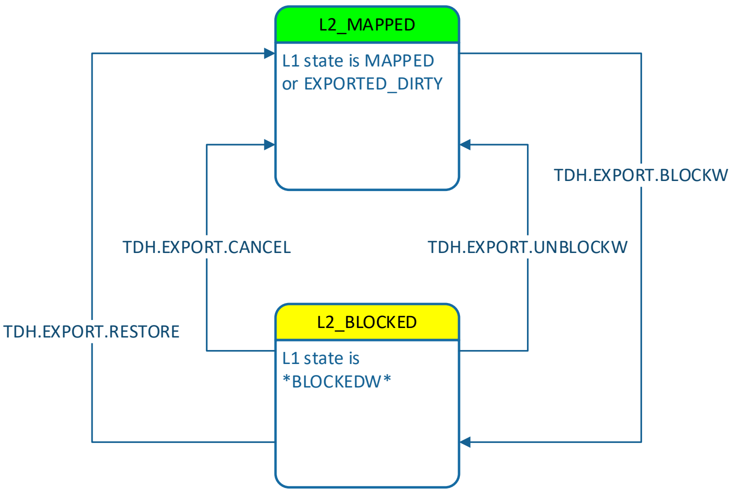

Figure 8.5: L2 Secure EPT Leaf Entry Partial State Diagram for Mapped Page Export

8.4.5. Live Export: Blocking for Writing, TLB Tracking and Exporting a Page

During the live export phase (when TDCS.OP_STATE is LIVE_EXPORT), exporting a private memory page requires that page modification must be prevented. This includes:

- Page content
- Page attributes
- For partitioned TDs, L2 page attributes

To achieve this, the page's L1 SEPT entry and any L2 SEPT entries must be blocked for writing by TDH.EXPORT.BLOCKW.

If the TD has not been paused, the host VMM must execute the TLB tracking sequence below, which together with the checks done by TDH.EXPORT.MEM helps ensure that no cached TLB entries that have been created before blocking for writing are left.

1. Execute TDH.EXPORT.BLOCKW on each page to be exported, blocking subsequent creation of writable TLB translations to that page. Note that cached translations may still exits at this stage.
2. Execute TDH.MEM.TRACK, advancing the TD's epoch counter.
3. Send an IPI (Inter-Processor Interrupt) to each RLP (Remote Logical Processor) on which any of the TD's VCPUs is currently scheduled.
4. Upon receiving the IPI, each RLP will TD exit to the host VMM.

At this point the blocked pages are considered tracked for export. Even though some LPs may still hold writable TLB entries to the target GPA ranges, those are designed to be flushed on the next TD entry. Normally, the host VMM on each RLP will treat the TD exit as spurious and will immediately re-enter the TD.

5. Export each page using TDH.EXPORT.MEM.

8.4.6. Exporting a Page after the Source TD is Paused

After the source TD is paused, no blocking is required since the TD is not running. This reduces the amount of work that needs to be done by the host VMM during the TD's blackout period. This is shown in the dashed transitions in Figure 8.4 above.

If the export session is aborted by TDH.EXPORT.ABORT, some LPs may still hold stale TLB entries exported pages. To help ensure they are flushed on the next TD entry, TDH.EXPORT.PAUSE advances the TD's epoch counter, similarly to TDH.MEM.TRACK.

8.4.7. Unblocking for Write, Tracking Dirty Pages and Re-Exporting

8.4.7.1. *Overview*

During the live export phase (when TDCS.OP_STATE is LIVE_EXPORT), the source TD may attempt to write a page that has been blocked for writing, or to modify the page attributes (for partitioned TDs, this includes L2 attributes). The TDX migration architecture allows the host VMM to unblock the page. The Intel TDX module tracks such pages as "dirty". All dirty pages must be re-exported by the host VMM for the in-order migration phase to be completed. This assures that either the latest version of a page has been exported by the time the source TD is paused, or that page has not been exported at all.

8.4.7.2. *Unblocking for Write and Re-Exporting a Page*

If the source TD attempts to write to a page that has been blocked for writing a TD exit will occur, indicating an EPT violation due to a write attempt to a non-writable page.

Note: No indication is directly provided to the host VMM whether this page is blocked for writing by TDH.EXPORT.BLOCKW or whether writing is disabled due to some other reason.

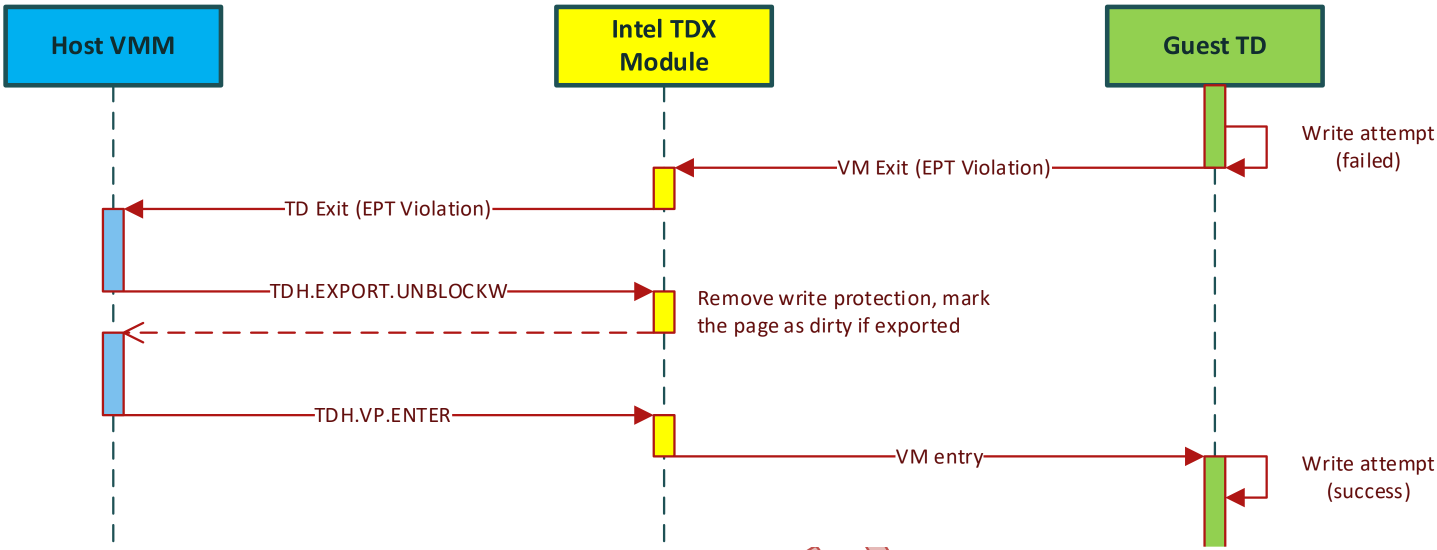

Figure 8.6: Typical Sequence for Unblocking a Page on Guest TD Write Attempt

To enable access to the page, the host VMM is expected to execute TDH.EXPORT.UNBLOCKW.

- If the page has not yet been exported, TDH.EXPORT.UNBLOCKW restores its SEPT entry's original MAPPED state.
- If the page has been exported, TDH.EXPORT.UNBLOCKW updates its SEPT state to EXPORTED_DIRTY. This state is similar from the guest TD's memory access perspective to MAPPED, but it indicates that the page is dirty and needs to be re-exported.
- For partitioned TDs, if the page has any L2 mappings, TDH.EXPORT.UNBLOCKW unblocks their L2 SEPT entries by restoring their W bit value.

The host VMM re-exports the page by TDH.EXPORT.BLOCKW, TLB tracking and TDH.EXPORT.MEM as described in 0 above.

8.4.7.3. *TDCS.DIRTY_COUNT: TD-Scope Dirty Page Counter*

TDCS.DIRTY_COUNT is TD-scope dirty page counter.

- DIRTY_COUNT is cleared when a new migration session begins (by TDH.EXPORT.STATE.IMMUTABLE).
- DIRTY_COUNT is incremented when a page that has previously been exported in the current session is unblocked for writing by TDH.EXPORT.UNBLOCKW.
- DIRTY_COUNT is decremented when a newer version of a page, which has previously been exported in the current session, it exported by TDH.EXPORT.MEM.

For successful start token generation by TDH.EXPORT.TRACK, the value of the DIRTY_COUNT must be 0, indicating that all pages exported so far have their newest pages exported. At this point, since the source TD is paused, no newer versions of any page can be created, and the destination TD can start execution. Private pages which has not been exported yet in the current session may still be remaining for post copy export. Note that exported pages may not have been transported yet. The start token MBMD's TOTAL_MB field verification enforces that all exported state has been imported (in-order) on destination – see the [TDX Module ABI Spec] for details.

8.4.8. Re-Exporting a Non-Dirty Page

In the out-of-order phase, where strict migration order is not enforced, the host VMM may re-export a previously exported page even if it has not been unblocked for writing and its contents have not been modified.

This allows a page can be re-exported and transferred to the destination platform over a high-priority stream. This helps reduce destination TD latency while waiting for a page to be imported.

Such an operation is tagged MIGRATE, not REMIGRATE, in the exported GPA list. This is because the exact same version of the page is being exported.

8.4.9. Interruptible Memory Export

TDH.EXPORT.MEM may export up to 512 4KB pages. To keep its latency within reasonable limit, the function is designed to be interruptible. TDH.EXPORT.MEM can only be interrupted after completing the export of each page. If TDH.EXPORT.MEM detects that an interrupt is pending, it saves its intermediate state and returns with a proper status indication. The host VMM is expected to re-invoke TDH.EXPORT.MEM to complete the export operation.

Intermediate state is saved as part of the migration stream context that has been used for the interrupted TDH.EXPORT.MEM. Upon invocation, TDH.EXPORT.MEM checks to see if an intermediate state has been saved, and if so, it checks that it is being invoked with the same input arguments as last time when it was interrupted.

8.4.10. Prohibited Operations on Exported Pages and Export Cancellation

Once a page has been exported during the current export session, it can't be blocked, removed, promoted, demoted or relocated. This prevents the destination platform from using a stale copy of that page.

In order to perform such memory management operations on an exported page, the host VMM must first execute TDH.EXPORT.MEM indicating a CANCEL operation for the page. No migration buffer is required for this GPA list entry. The When the GPA list is processed on the destination platform by TDH.IMPORT.MEM, the previously migrated page is removed from the destination TD. TDH.EXPORT.MEM restores the page SEPT entry to its pre-export MAPPED or PENDING state.

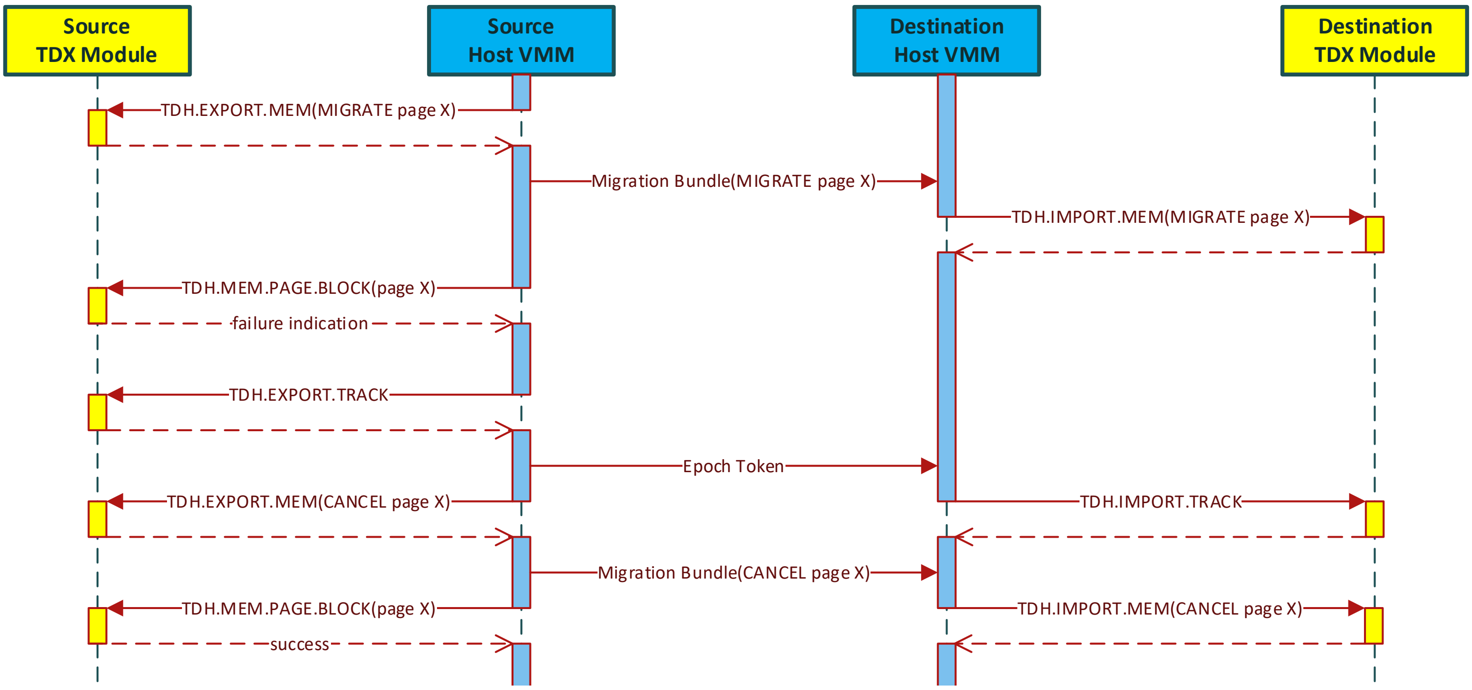

Figure 8.7: Typical Sequence for Cancelling a Page Export

8.4.11. Exporting Pending Pages

The host VMM is not directly aware if a page is in a PENDING state or not; the guest TD may accept the page by TDG.MEM.PAGE.ACCEPT at any time. Thus, TDH.EXPORT.MEM may export a pending page. This is indicated by the GPA list entry, and no migration buffer is used since the page content is not exported. The page attributes (including the optional page attributes list entry) are exported. On the destination platform, TDH.IMPORT.MEM creates the page in a PENDING state.

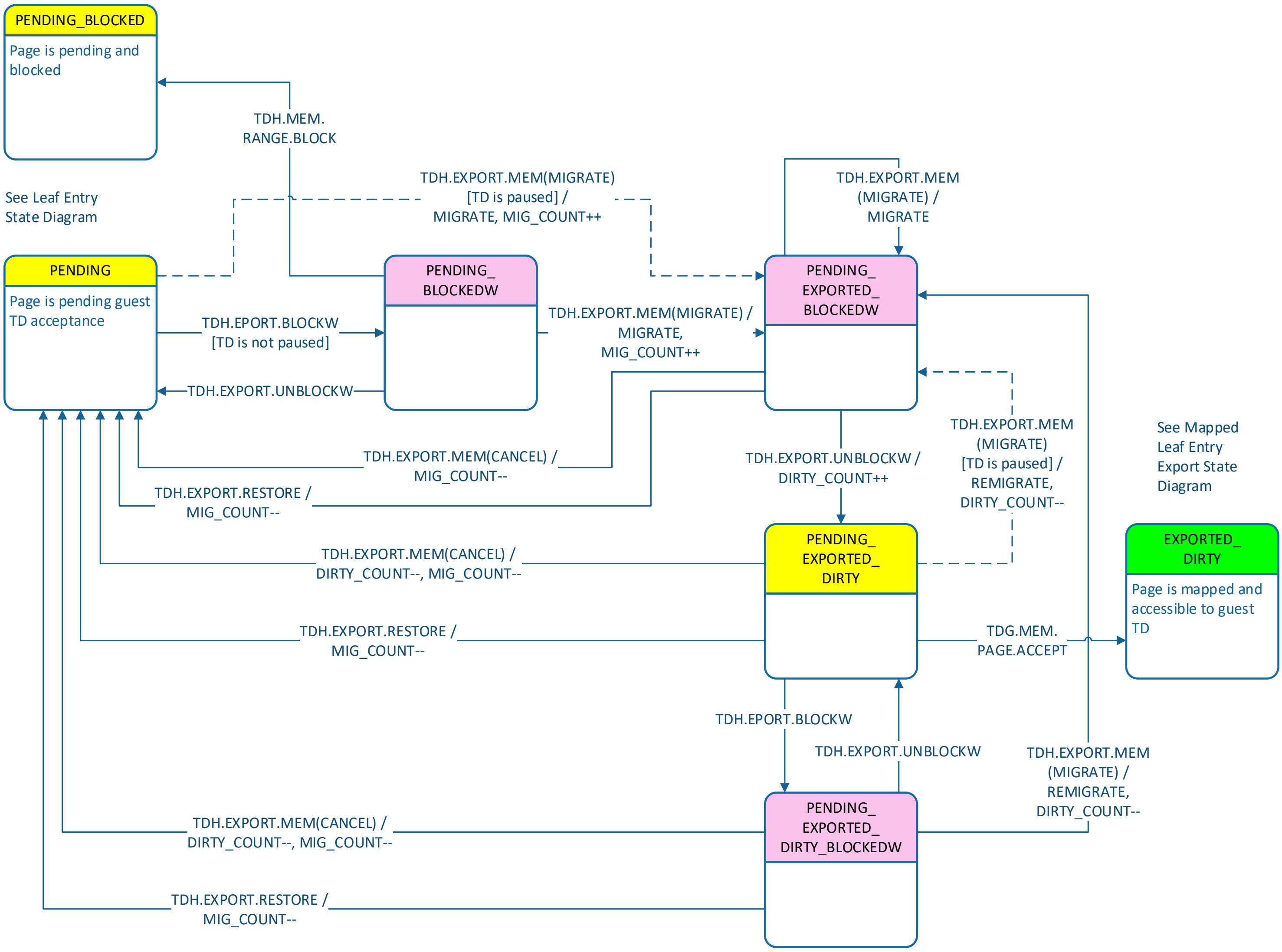

Figure 8.8: Partial SEPT Leaf Entry State Diagram for Pending Page Export

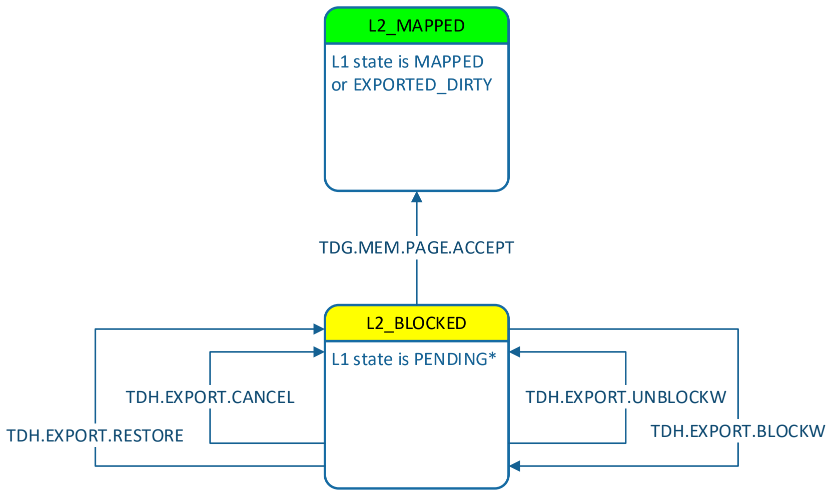

Figure 8.9: L2 Secure EPT Leaf Entry Partial State Diagram for Pending Page Export

If the guest TD accepts a pending page that has been exported, TDG.MEM.PAGE.ACCEPT results in an EPT violation. The host VMM is expected to call TDH.EXPORT.UNBLOCKW, which marks the page as PENDING_EXPORT_DIRTY, and resumes the guest TD. TDH.MEM.PAGE.ACCEPT then re-executes; it initialized the page and updates the SEPT state to mark the page as EXPORTED_DIRTY (where the page is mapped and accessible to the guest TD). The host VMM can then re-export the page, as described in 8.4.7 above.

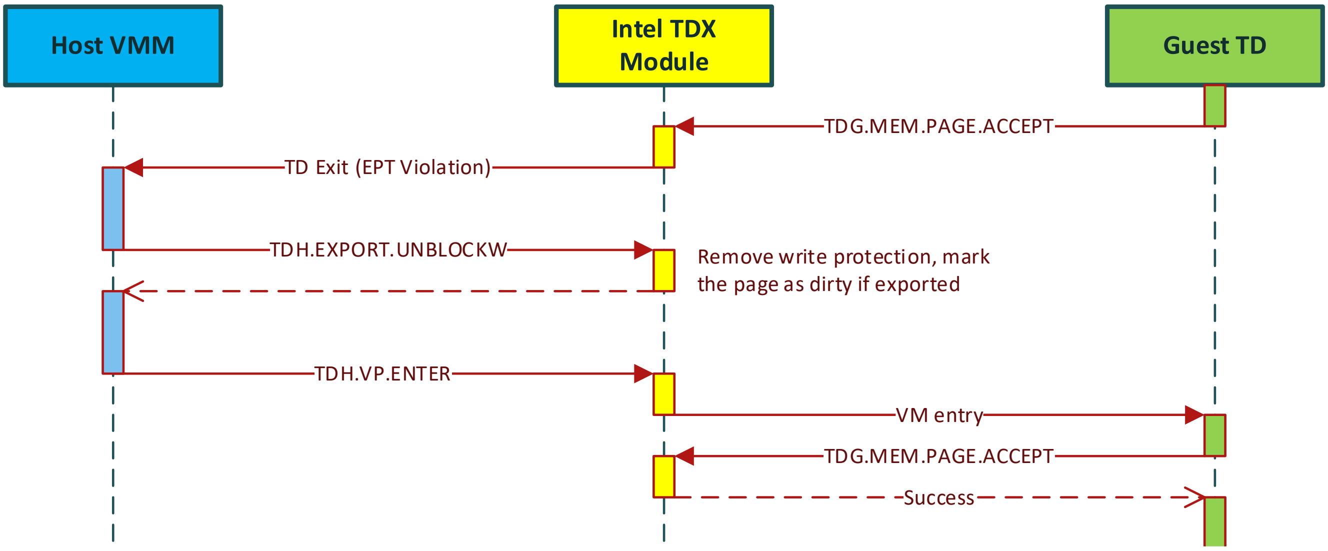

Figure 8.10: Typical Sequence for Unblocking a PENDING Page on TDG.MEM.PAGE.ACCEPT Attempt

8.4.12. SEPT Cleanup after Export Abort

Following an export session is aborted (by TDH.EXPORT.ABORT) the source TD is allowed to run. However, SEPT entries and, for partitioned TDs, L2 SEPT entries, that have been modified during the aborted export session keep their state. Such SEPT entries must be cleaned up by the host VMM before memory management operations are allowed on them, and/or before a new export session is attempted, as follows:

- Cleanup of SEPT entries that have been blocked for writing is done by TDH.EXPORT.UNBLOCKW (if the page is to be written) or TDH.RANGE.BLOCK (is the page is to be blocked for some memory management operation).
- Cleanup of SEPT entries that have been exported is done by TDH.EXPORT.RESTORE.

To track and enforce proper cleanup, the following counter is maintained in TDCS:

- MIG_COUNT counts the number of leaf SEPT entries that need to be cleaned up. L2 SEPT entries (if any) are not counted.

The counter is initialized to 0. To start a new migration session, its value must be 0.

TDH.EXPORT.MEM increments MIG_COUNT for exported pages, and decrements MIG_COUNT for export cancels.

8.5. *TD Private Memory Import*

8.5.1. In-Order Import Phase

8.5.1.1. *Overview of In-order Import*

During the in-order import phase, a page may be imported multiple times. In addition, a page import may be cancelled. Ordering is maintained by the MBMD's MB_COUNTER and the requirement that a page can only be imported once per migration epoch.

Once out-of-order import phase begins, any pages that has been imported are designed to be up to date.

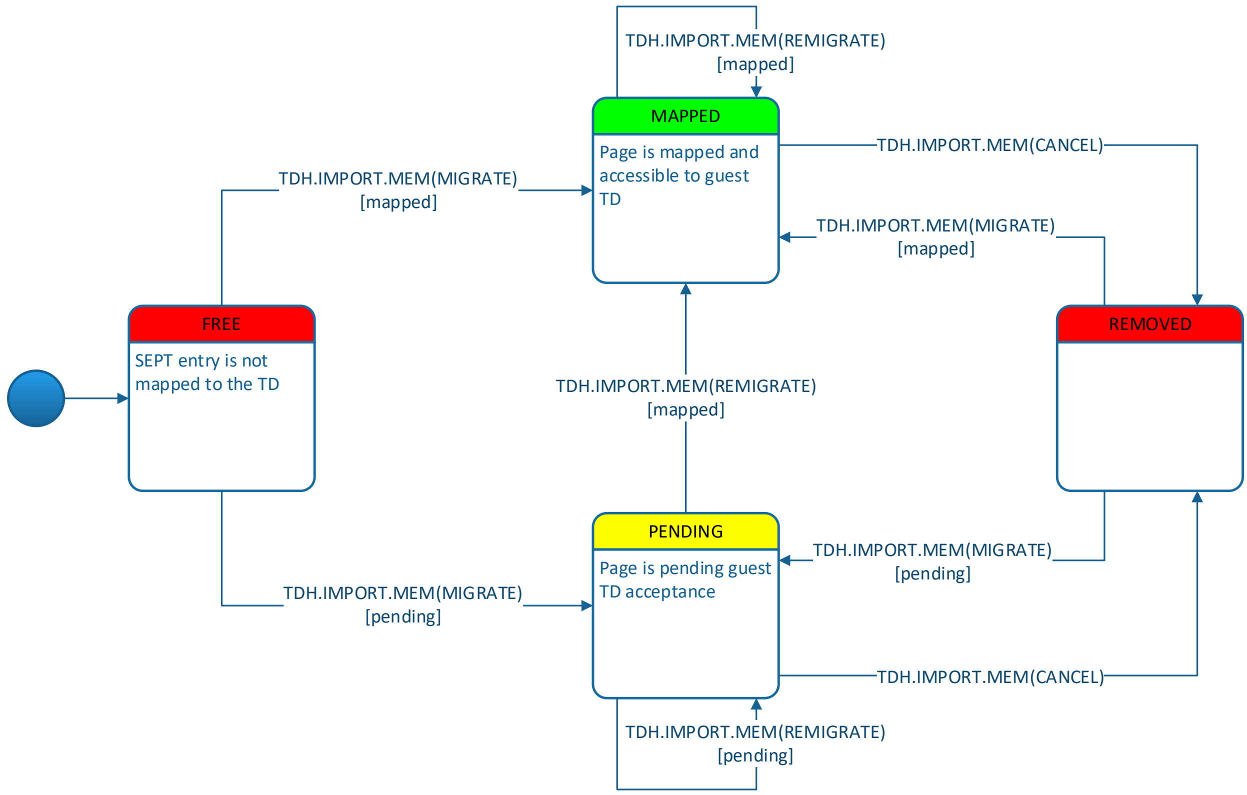

Figure 8.11: Partial SEPT Entry State Diagram for Page In-Order Import Phase

8.5.1.2. *Memory Management During In-Order Import*

8.5.1.2.1. TLB Tracking During In-Order Import

During the in-order import phase, no blocking and no TLB tracking is required, since the destination TD is not running yet.

8.5.1.2.2. Secure EPT Management During In-Order Import

Addition and removal of Secure EPT pages are allowed during the in-order phase – they are required as part of building the TD on the destination platform.

An SEPT page can only be removed if all its entries are FREE; specifically, it can't be removed if any entry state is REMOVED.

8.5.1.2.3. Page Management During In-Order Import

Page management operations are prohibited during the in-order import phase:

- Page addition by TDH.MEM.PAGE.ADD and TDH.MEM.PAGE.AUG
- Page removal by TDH.MEM.PAGE.REMOVE
- Page promotion and demotion by TDH.MEM.PAGE.PROMOTE and TDH.MEM.PAGE.DEMOTE
- Page relocation by TDH.MEM.PAGE.RELOCATE
- GPA range blocking and unblocking by TDH.MEM.RANGE.BLOCK and TDH.MEM.RANGE.UNBLOCK

8.5.1.3. *Enforcing a Single Import Operation per Migration Epoch*

When a page is imported during the in-order, the current migration epoch is recorded in the page's PAMT.BEPOCH field.

Note: TDH.MEM.RANGE.BLOCK, which is the only other interface function that writes to PAMT.BEPOCH, can't be invoked during in-order import.

Page re-import and import cancel operations compare the recorded migration epoch in the existing page's PAMT. For the import to succeed, it should be older than the current migration epoch.

When a page import is cancelled during the in-order, the physical page is removed but its SEPT entry is put into a REMOVED state, and the current migration epoch is recorded in the SEPT entry's HPA field. For partitioned TDs, any L2 SEPT entries that map the page become L2_FREE.

Page first-time import operation compares the recorded migration epoch in the existing page's SEPT entry. For the import to succeed, it should be older than the current migration epoch.

8.5.1.4. *Importing L2 Page Mappings During In-Order Import*

For partitioned TDs, L2 page mappings are imported as part of a page import, and follow these rules:

- When a page is imported or re-imported, its L2 page mappings, if any, are created or updated in the L2 SEPT leaf entry.
- When a page is re-imported, and a previously imported page mapping does not exist in the new import, the L2 SEPT leaf entry is set to L2_FREE.
- When a page import is cancelled, its L2 SEPT leaf entries, if any, are set to L2_FREE.

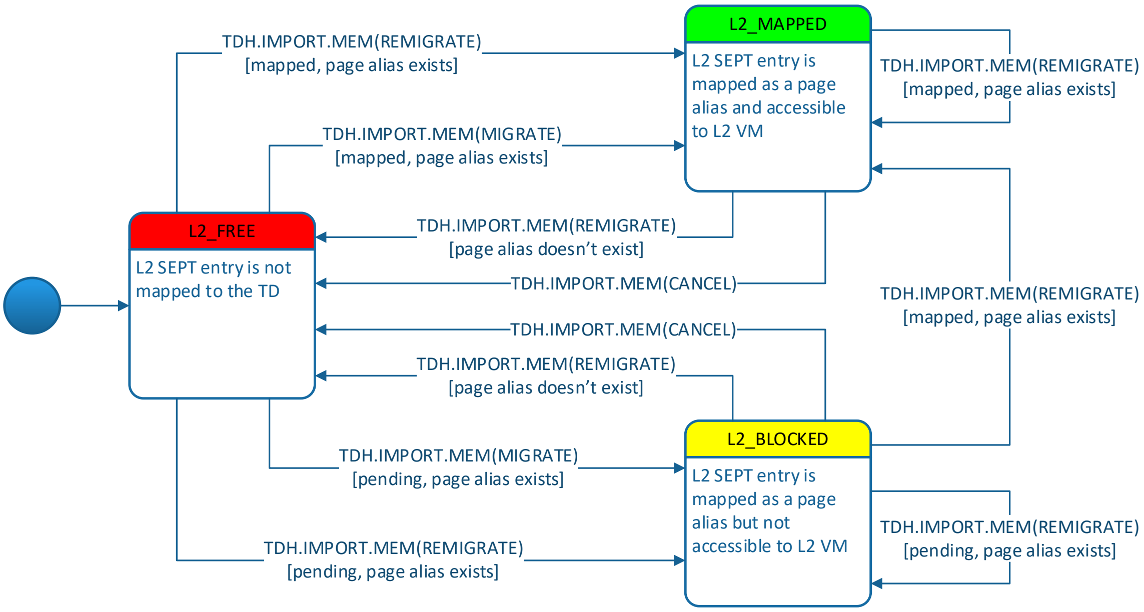

Figure 8.12: Partial L2 SEPT Leaf Entry State Diagram for L2 Page Mapping Import During the In-Order Phase

To import a page that has L2 pages mappings, the host VMM on the destination platform must have built the L2 SEPT for the applicable L2 VMs, using TDH.MEM.SEPT.ADD, down to the proper page mapping level.

8.5.2. Out-of-Order import Phase

8.5.2.1. *Overview of Out-of-Order Import*

When the out-of-order import phase begins, any pages that have been imported are designed to be up to date. A page may only be imported if its GPA mapping does not exist yet (SEPT entry's state is FREE). An import attempt of a page that has been imported before during the out-of-order phase is dropped but not considered an error; this is normally the result of the same page being migrated on a high-priority queue. Source memory state is immutable, so ordering is not required.

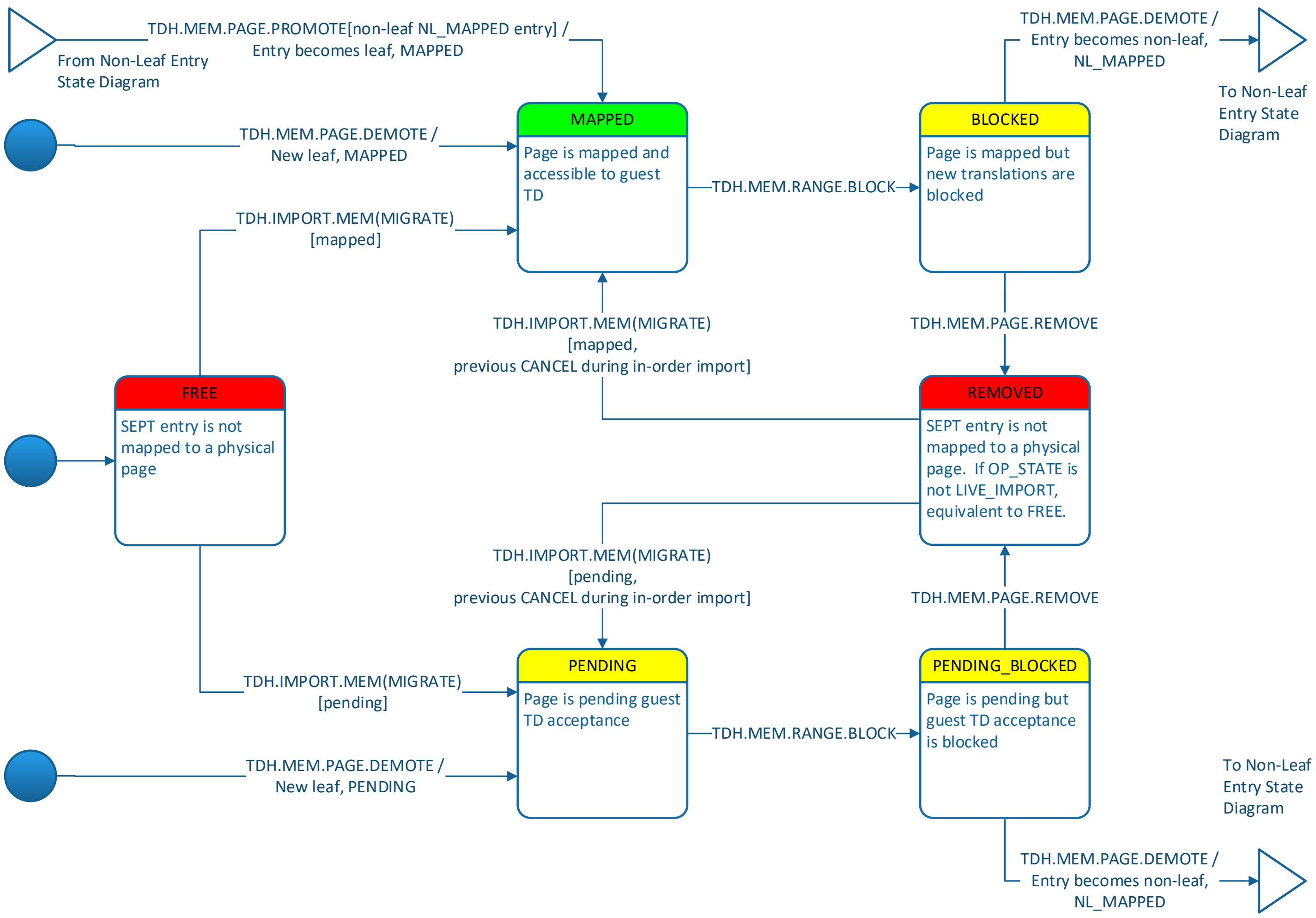

Figure 8.13: Partial SEPT Entry State Diagram for Page Out-of-Order Import Phase

8.5.2.2. *Memory Management During Out-of-Order Import*

8.5.2.2.1. TLB Tracking During Out-of-Order Import

During the out-of-order import phase, TLB tracking is required in the LIVE_IMPORT OP_STATE, since the TD may be running on the destination platform.

8.5.2.2.2. Secure EPT Management During Out-of-Order Import

Addition and removal of Secure EPT pages are allowed during the out-of-order phase – they are required as part of building the TD on the destination platform.

An SEPT page can only be removed if all its entries are FREE; specifically, it can't be removed if any entry state is REMOVED.

8.5.2.2.3. Page Addition During Out-of-Order Import

TDH.MEM.PAGE.ADD is prohibited but TDH.MEM.PAGE.AUG is allowed in the LIVE_IMPORT OP_STATE.

If a page was added locally (TDH.MEM.PAGE.AUG), this is equivalent to the VMM removing a page without coordinating with the TD, then adding a new page. The TD should not accept (TDG.MEM.PAGE.ACCEPT) such a page since from its point of view this is a page that already existed in its GPA space. The secure EPT entry state for the locally added page is PENDING, and if a page is imported to the same GPA, import will fail.

8.5.2.2.4. Promotion and Demotion During Out-of-Order Import

Page promotion and demotion are allowed during the out-of-order phase.

8.5.2.2.5. Page Removal During Out-of-Order Import

Page removal (TDH.MEM.PAGE.REMOVE) is allowed during the out-of-order import phase. However, the page's SEPT entry is not marked as FREE when the page is removed. Instead, the SEPT entry state is set to REMOVED. The REMOVED state is equivalent to the FREE state, except for the following limitations that apply as long as the TD is in the LIVE_IMPORT OP_STATE:

- Page import is not allowed to this GPA.
- Removal of the parent SEPT page is not allowed.

The above limitations prevent the following attack scenario:

1. The host VMM creates a copy of a migration bundle and saves it for later.
2. The host VMM import a page using the migration bundle.
3. The TD runs and modifies the imported page content.
4. The host VMM removes the page.
5. The host VMM attempt to re-import the page using the saved copy of the migration bundle.

If the host VMM succeeded in re-importing the page, it would have rolled back the page content. Remember we do not enforce order of import during the out-of-order phase. But setting the SEPT entry state to REMOVED when the page was removed prevents this attack.

8.5.2.2.6. Page Relocation During Out-of-Order Import

Page relocation is supported at any stage without any changes.

8.5.2.3. Importing L2 Page Mappings During Out-of-Order Import

During out-of-order import, L2 page mappings may be created as part of a page import. L2 SEPT leaf entries don't need the REMOVED state; thus, the L2 state transitions are very simple.

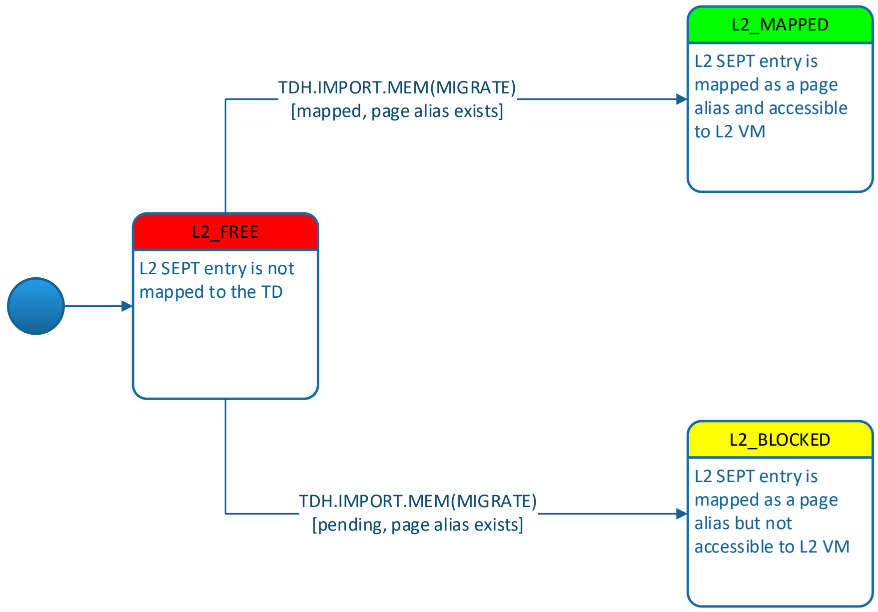

Figure 8.14: L2 SEPT Leaf Entry State Diagram for Page Out-of-Order Import Phase Partial

8.5.3. In-Place Import

In-place import repurposes the physical pages holding the imported data as private memory pages that hold the decrypted data. This saves the host VMM on the destination platform the need to allocate memory for the imported data, at the cost of a small fixed-sized intermediate buffer that needs to be held by Intel TDX Module, and some other complications. In-place import may be selected for each page imported for the first time, or following a previous CANCEL, but not for re-import of a new version of a previously imported page.

8.6. Secure EPT Concurrency Considerations

Note: OP_STATE related concurrency considerations are described in 6.2.6.

To support high performance migration, memory migration interface functions are allowed to run concurrently on multiple LPs. However, no concurrent operation is allowed on any single Secure EPT entry. Interface functions that work on specific Secure EPT entries acquire an exclusive lock to that entry.

8.7. Memory Migration Interface Functions

This section provides a short overview of the memory migration interface functions. A detailed specification is provided in [TDX Module ABI Spec].

8.7.1. TDH.EXPORT.BLOCKW

TDH.EXPORT.BLOCKW blocks a list of 4KB pages for writing, as a preparation for export. The function records the current value of TD_EPOCH in TDCS.BW_EPOCH.

Inputs

- Source TD handle: the TDR page HPA
- GPA list

Pre-Conditions

- Export session is in the in-order phase and the TD has not been paused yet

Operation

See the [TDX Module ABI Spec].

8.7.2. TDH.EXPORT.MEM

TDH.EXPORT.MEM exports, re-exports or sends an export cancellation request for a list of 4KB pages. A page may be PENDING (in which case no data is exported).

Inputs

- Source TD handle: the TDR page HPA
- GPA list
- MBMD HPA
- Migration buffers list HPA
- For a partitioned TD, a page attributes list HPA
- Page MAC list HPA
- Migration stream index

Pre-Conditions

- Export session is in progress

Operation

See the [TDX Module ABI Spec].

8.7.3. TDH.EXPORT.RESTORE

TDH.EXPORT.RESTORE restores a list of 4KB pages after an export abort.

Inputs

- Source TD handle: the TDR page HPA
- GPA list

Pre-Conditions

- Export session is not in progress

Operation

See the [TDX Module ABI Spec].

8.7.4. TDH.EXPORT.UNBLOCKW

TDH.EXPORT.UNBLOCKW unblocks a 4KB page that has been blocked for writing.

Inputs

- Source TD handle: the TDR page HPA
- Source GPA and level

Pre-Conditions

- Either an export session is in progress but committed export phase has not begun, or the TD is allowed to run

Operation

See the [TDX Module ABI Spec].

8.7.5. TDH.IMPORT.MEM

TDH.IMPORT.MEM exports, re-imports or cancels a previous import for a list of 4KB pages.

Inputs

- Destination TD handle: the TDR page HPA
- GPA list HPA
- For partitioned TDs, page attributes list HPA
- MBMD HPA
- Migration buffers list HPA
- Page MAC list HPA
- Migration stream index
- New TD pages list HPA

Pre-Conditions

- Import session is in progress

Operation

See the [TDX Module ABI Spec].