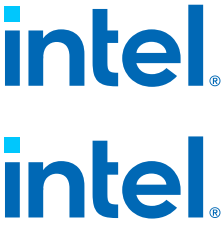


Intel® TDX Module Architecture Specification: TD Migration

~~348550-005US~~

~~October 2024~~

2025-09-01T00:00:00

.

Notices and Disclaimers

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

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

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

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

No product or component can be absolutely secure.

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

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

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

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

~~Section 1: Introduction and Overview~~

Table of Contents

SECTION 1:
TD MIGRATION INTRODUCTION AND OVERVIEW

5

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 TD Partitioning Architecture Specification	[TD Partitioning Spec]	Architecture overview and specification for TD Partitioning
TDX Module Interrupt Virtualization Architecture Specification	[Interrupt Virtualization Spec]	Architecture overview and specification for interrupt virtualization
TDX Module TDX Connect Specification	[TDX Connect Spec]	Architecture overview and specification for TDX Connect
TDX Module ABI Reference Tables	[TDX Module ABI Tables]	A set of ~~JSON format~~ files detailing TDX module Application Binary Interface (ABI)
TDX Module TDX Connect ABI Reference Specification	[TDX Connect ABI Spec]	Detailed TDX module Application Binary Interface (ABI) reference specification, covering the TDX connect architecture
TDX Module ABI Reference 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

~~Section 1: Introduction and Overview~~

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
	Blackout Period	No	The period when the guest TD does not run anymore on the source platform and does not run yet on the destination platform.
	Cold Migration	No	Migration usage mode where the TD does not run during the migration session.
	In-Order Phase	Yes	The first phase of the TD migration session, where a strict order of memory export vs. memory import is maintained.
	Live Migration	No	Migration usage mode where the TD runs during most of the migration session.
	Migration Bundle	Yes	The basic unit of migrated information, composed of headers, metadata and/or migrated data.
	Migration Commitment	Yes	The act of committing the migration, disallowing the TD from running on the source platform and allowing it to run on the destination platform.
	Migration Epoch	Yes	A mechanism used to enforce ordering across multiple concurrent streams. Any specific page can be migrated only once per epoch.
	Migration Policy	No	Policy enforced by the Migration TDs, e.g., based on the source and destination platform properties.
	Migration Stream	Yes	A sequential stream of migration bundles, where order is enforced.
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.
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.
	Out-Of-Order Phase	Yes	The last phase of the TD migration session, where a strict order of memory export vs. memory import is not maintained.
	Post-Copy	No	Migration usage mode where part of the TD memory image is migrated after the TD is allowed to run on the destination platform.
	Pre-Copy	No	Migration usage mode where (most of) the TD memory image is migrated before the TD is allowed to run on the destination platform.
TCP	**Transmission Control Protocol**	No	Transmission Control Protocol (TCP) is a communications standard that enables application programs and computing devices to exchange messages over a network. It is designed to send packets across the internet and ensure the successful delivery of data and messages over networks.

1.4. Notation

See the [TDX Module Base Spec].

1.5. References

1.5.1. Intel Public Documents

See the [TDX Module Base Spec].

1.5.2. Intel TDX Public Documents

See the [TDX Module Base Spec].

1.5.3. Non-Intel Public Documents

Table 1.3: 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

~~Section 1: Introduction and Overview~~

2. TD Migration Overview

Unreleased Feature: Some of the text in this section is related to Non-Blocking Export, a feature which has not been released yet at the time of writing of this document. Details related to that feature serve as a preview and are subject to change.

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

2.1. Introduction

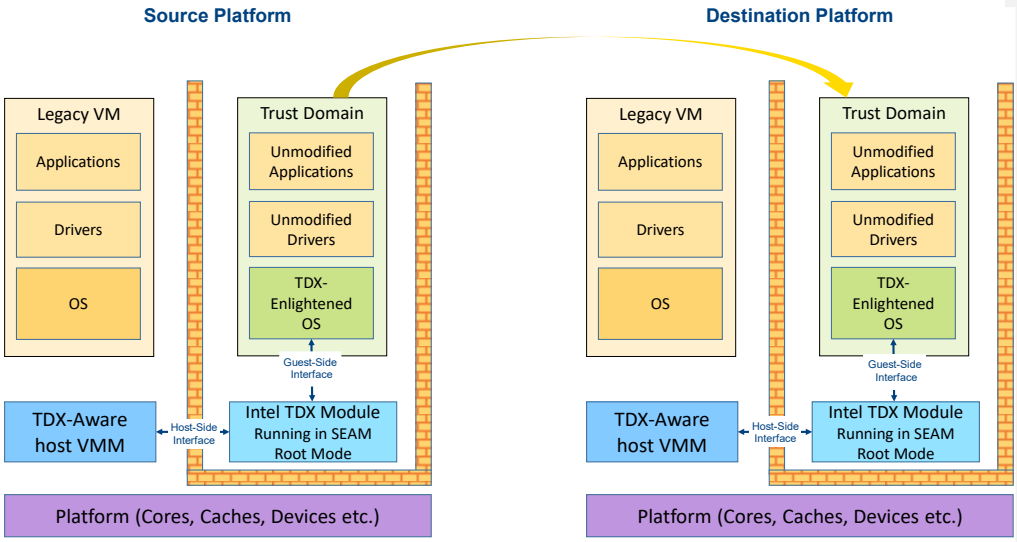


Figure 2.1: TD Migration

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~~is configured with an HKID ~~(and will be always assigned a different ephemeral key)~~ on the destination platform ~~chosen to migrate~~which is independent of its HKID on the ~~TD~~source platform and is associated with a different ephemeral key.

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**~~ Both source and destination ~~**known and resumed such that both ends**~~platforms must be alive ~~**to do the handoff**~~during migration.
- The TD ~~image~~ is suspended during migration ~~and resumed after a duration >>~~. Blackout time is typically longer than TCP timeout, which may cause remote connections to the TD to break up.

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

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

2.2.2. Live Migration

- ~~**Live migration with**~~ Both source and destination ~~**known and resumed such that both ends**~~platforms must be alive ~~**to do the handoff**~~during migration.
- The TD continues executing during migration ~~and~~. It is paused for ~~duration <<~~a short blackout time, typically shorter than TCP timeout, so remote connections to the TD should not break up.

- ~~Customer~~Live migration may be useful for supporting customer SLA requirements, capacity planning~~/~~ and load balancing.

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

2.2.3. Image Snapshot and Jumpstart~~—~~ **(Not Supported)**

- ~~**Pre-built Image/Jumpstart with destination unknown and**~~Destination need not be alive during export, when the migration image is prepared.
- Source need not be alive during import, when the migration image is loaded.
- The TD image may be 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** and is not supported by it.

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~~

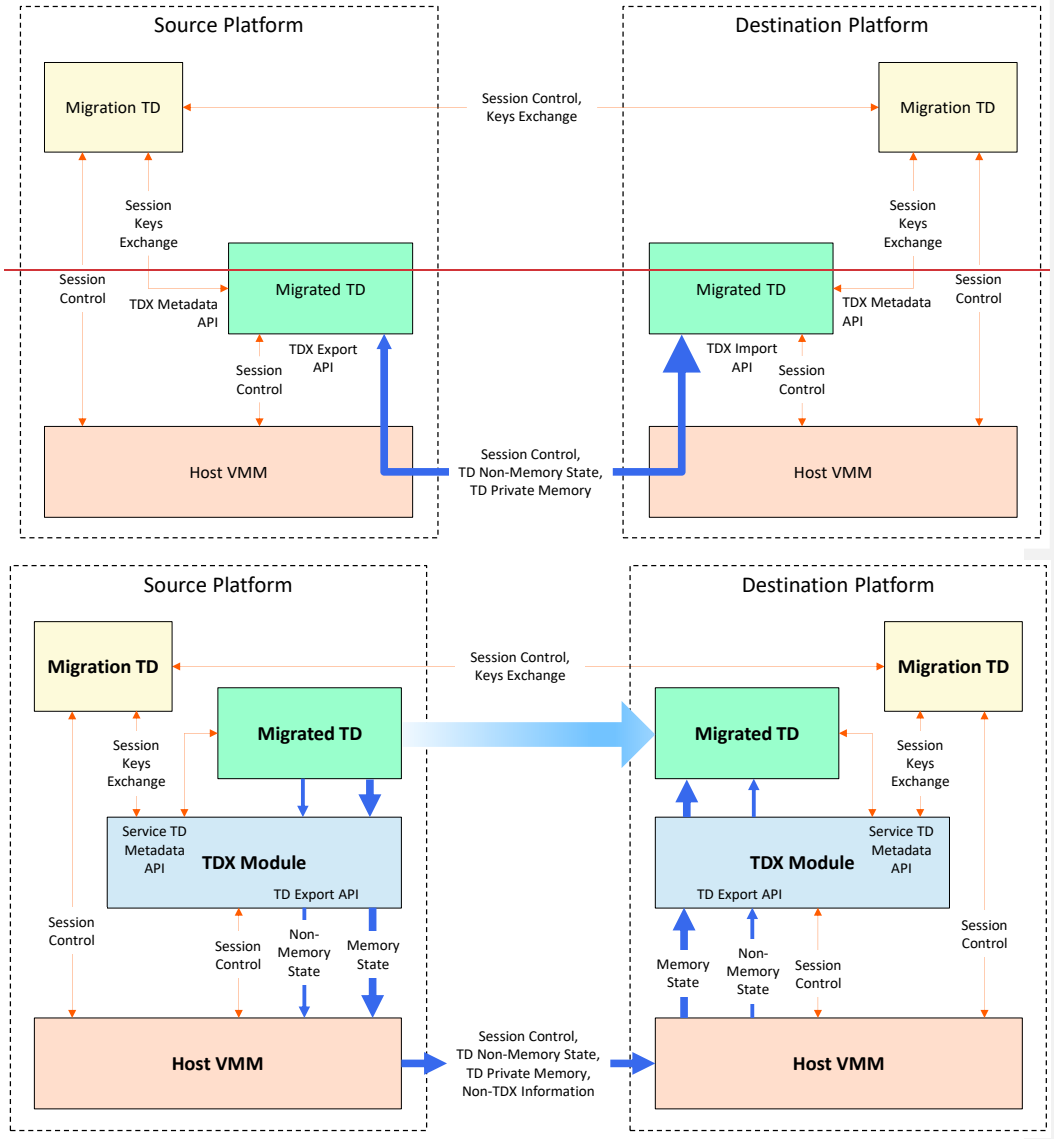


Figure 2.2: ~~Component~~Components Involved in TD Migration

~~A~~ Migrated TD

The migrated TD's role is passive. It is not directly aware of it being migrated.

Migration TD (MigTD) ~~is used~~

Migration TDs (MigTD) exist on the source and destination platforms. Their main role is to implement a **migration policy** and evaluate ~~potential~~ migration sources and ~~targets~~destinations for adherence to ~~the TD Migration Policy. The TD Migration policy enumerates~~that policy. The migration policy may enumerate TDX platform TCB requirements ~~as well as~~ , platform features and acceptable destination Migration TD TCB ~~levels~~requirements.

~~If TCB levels are acceptable, the Migration TDs on the source and destination platforms~~The MigTDs securely exchange unique per-session **Migration Session ~~Keys (~~Key (MSK)** pair. The 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 securely transfers it to the MigTD on the other side~~. That MigTD writes the key, as TD metadata,~~, where it is written as the decryption key for its side.

~~The host VMM may bind a MigTD to one or more~~Migration TDs ~~being migrated. For~~implement and use the TDX module Service TD protocol (for details, see [TDX Module Base Spec]'s Service TDs chapter~~. Since the MigTD~~). The host VMM may bind a MigTD to one or migrated TDs. The MigTD is in the TCB of the ~~TD being~~ migrated~~, a~~ TD and its measurements are included in the migrated TD's attestation information. Thus, the MigTD must be pre-bound to the ~~target TD being~~ migrated TD before ~~the target TD~~ that TD's measurement is finalized. The MigTD lifecycle does not have to be coincidental with the ~~target~~migrated TD – the only requirement is that the MigTD ~~may be instantiated when required for Live Migration, but it~~ must be bound to the ~~target~~migrated TD before ~~Live Migration~~ 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.~~exchanged.

Host VMM

The host ~~VMM~~VMMs on the source and destination sides orchestrate and manage the migration session, via ~~the Intel TDX Module, is~~ their respective TDX modules and MigTDs. They are responsible for ~~export/import~~exporting and importing of the ~~TD content, and~~ TD's memory and non-memory state, via the TDX module, and for the transport of that state between the source and the destination platforms.

TDX Module~~the protected~~

The TDX module implements a set of TD migration primitives to implement and enforce the security of migration session control, TD private memory migration and TD ~~content to the destination platform.~~ non-memory state migration.

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.~~2.5.1. Reservation and Session Setup

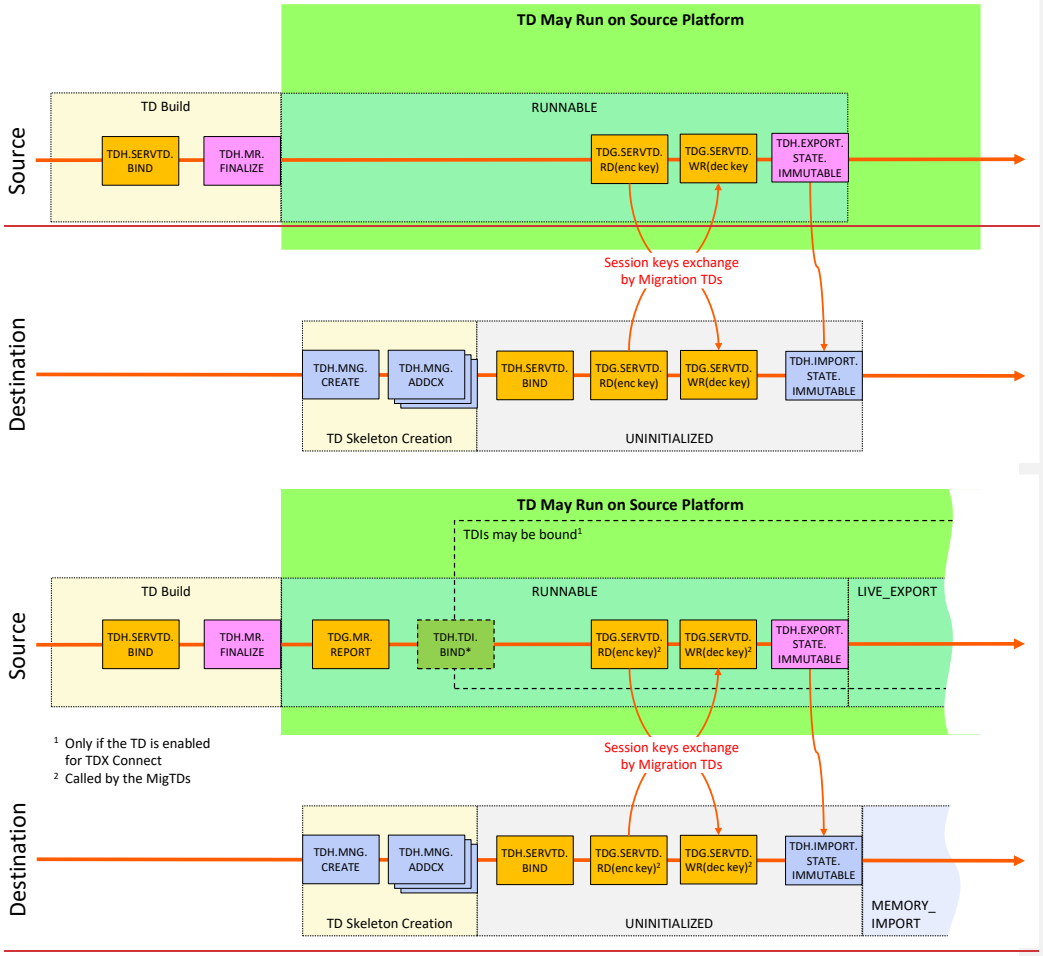


Figure 2.3: Migration TD Binding and Session Setup

~~2.5.2.1.~~2.5.1.1. *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. The bound MigTD is reflected in the migratable TD's TDREPORT.

~~2.5.2.2.~~2.5.1.2. *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~~ As with any TD build, the host VMM ~~programs~~configures the TD's private HKID ~~and the HW-generated encryption key assigned~~(which is not related to the ~~TD into~~HKID used on the ~~MKTME encryption engines~~source platform) 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. Once migration succeeds, the MigTD bound at the destination will be reflected in the migratable TD's TDREPORT.

~~2.5.2.3.~~2.5.1.3. *Migration TDs Session ~~Key Negotiation~~Establishment*

~~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~~Prior to starting any TD migration session, the migration TDs on the source and destination platforms need to create a secure connection between them. This connection may be made to support one or more migration sessions. Migration TDs may be written by any vendor; thus, details may vary. Typically, 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 typically negotiate a protected transport session (using Diffie-Hellman exchange). Using this protected transport session, the migration policy can be evaluated by the Migration ~~TD~~TDs.

2.5.1.4. *Migration Session Key and Protocol Version Exchange*

The migration session keys are ephemeral AES-256-GCM keys used for confidentiality and integrity protection of the migrated TD private state, 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 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.~~

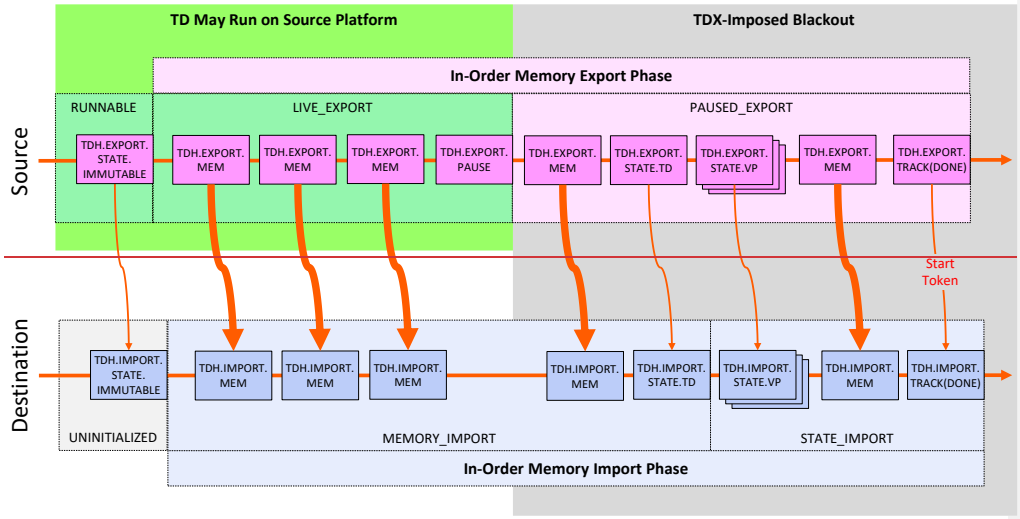
~~The TDX~~The Migration Session Keys (MSKs) are exchanged between the TDX modules on both ~~the source and destination platforms generate~~sides with the help of the Migration TDs.

The TDX module on each side generates an ephemeral **migration session encryption ~~keys~~key**. The Migration TDs on each side uses the Service TD ~~interface~~metadata read ~~that~~function (TDG.SERVTD.RD) to read the encryption key and securely transfer it to the peer Migration TD, which uses the Service TD ~~interface~~metadata write function (TDG.SERVTD.WR) to write it as the **migration session decryption key**.

In addition to the decryption keys, the Migration TDs on both sides write the migration protocol version to be used by the TDX modules.

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.~~2.5.2. In-Order Memory Migration Phase



The TD migration session has two phases: in-order and out-order. Those terms are defined in the context of TDX module enforcement of memory import ordering vs. memory export ordering. In-order enforcement is required during the live migration phase, until the source TD is paused and the blackout period begins. During this phase, as long as the source TD pages are mutable, order enforcement is essential to help ensure the migrated memory image is correct.

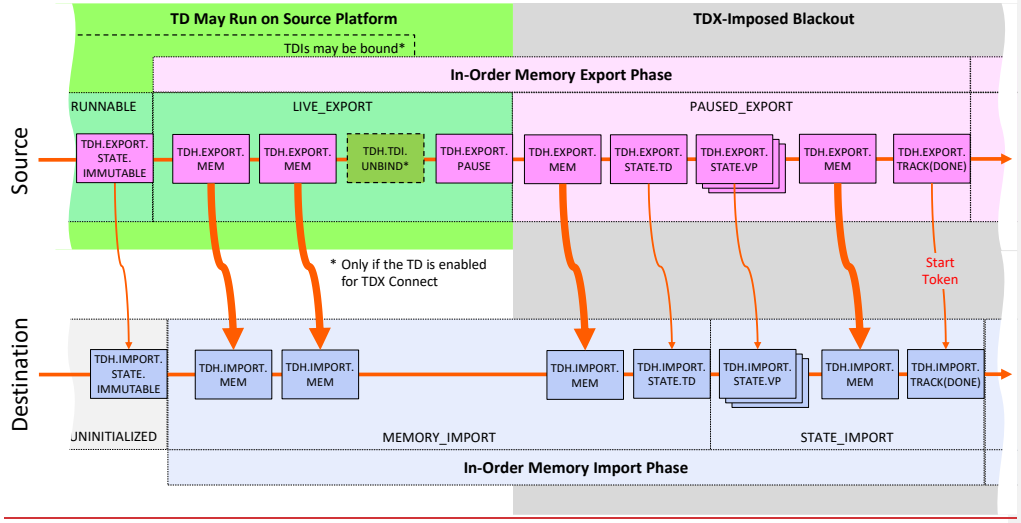


Figure 2.4: In-Order Migration Phase

~~2.5.3.1.~~2.5.2.1. *TD ~~Global~~ Scope Immutable Metadata (Non-Memory State) Migration*

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

Immutable metadata is the set of TD-scope state variables that are set by TDH.MNG.INIT, which 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.~~2.5.2.2. *Iterative Pre-Copy of Memory State*

During the in-order migration phase, the VMM aims to balance TD availability with progressing the migration work. This is generally done through an iterative pre-copy of TD memory pages and their in-order import on the destination platform. The details of TD private memory migration covered in Ch. 8; this section explains its key concepts.

~~2.5.3.2.1.~~2.5.2.2.1. Migration Considerations for TD Private Memory

Memory Migration vs. Memory Encryption

TD Migration does not migrate the TD's private HKID nor the keys used to encrypt its private memory. The TD's private assets are migrated using the migration session keys as described above. The host VMM on the destination platform assigns a new free HKID and the TDX module generates new memory encryption keys.

Memory Modification Tracking and Iterative Migration

During live migration, the running source TD and, if enabled for TDX Connect, its bound TDIs may write to memory. A memory page that has been exported and later modified must be re-migrated in order to ensure the consistency of the migrated memory image. To enforce that, the TDX module employs a page modification tracking mechanism. Two export modes may be supported:

- **Write-blocking export** is based on blocking memory for writing, to detect TD attempts to modify memory that has been exported.
- **Non-blocking export** (if supported by the TDX module) detects memory modifications based on EPT Dirty bit indication set by the hardware.

Migration Streams

To utilize multiple LPs and improve migration bandwidth, the host VMM may create multiple **migration streams** for concurrently transferring memory state. Migration streams are described in 5.4.

Migration Epochs~~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.~~

Each migration stream enforces import vs. export ordering within the stream. To enforce ordering across multiple concurrent streams, migration epochs are used. Any specific page can be migrated only once per epoch. Migration epochs are described in 6.1.4.

~~2.5.3.2.2.~~2.5.2.2.2. 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~~ trees 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~~not migrated but rebuilt on the destination platform~~, the~~ using memory ~~used for Secure EPT structures is not~~page attributes, which are migrated ~~across platforms. Hence, the~~as metadata along with page contents. To do this, the host VMM ~~must invoke~~calls the TDX ~~module's~~module's TDH.MEM.SEPT~~.* interface functions on the destination platform~~.ADD to build the Secure EPTs and TDH.IMPORT.MEM to ~~re-create the private GPA mappings on the destination platform (per the assigned HPAs). The Intel TDX module~~import memory pages. TDH.IMPORT.MEM 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~~ page metadata (including GPA, Read, Write and Execute attributes, and PENDING state) to ensure Secure EPT properties from the source platform are ~~rec created correctly as TD private memory contents are migrated, thus~~accurately recreated, 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.~~

Migrating shared memory is outside the scope of TDX. Shared memory assigned to the TD can be migrated by the host VMM using legacy mechanisms.

~~2.5.3.3.~~2.5.2.3. *Source TD ~~Stop~~Pause and Final Non-Memory State Migration*

Following pre-copy of TD private memory, the host VMM must detach any connected TDIs (applicable for TDX Connect) and 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.~~2.5.2.3.1. ~~In-Order~~ Final Non-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.~~

~~2.5.3.5.1.1.1.1.~~ ~~*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).

Once the source TD is paused, the host VMM exports the final (mutable) TD non-memory state, for the TD as a whole and for each VCPU.

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

2.5.2.3.2. In-Order Memory State 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 or attributes were later updated by the running source TD or by a TDI, it must be re-migrated. If the TD is enabled for TDX Connect, then all its private memory must be migrated.

The TDX Module enforces this using the commitment protocol described in 2.5.4 below.

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

~~2.5.4.~~2.5.3. Out-Of-Order Memory Migration Phase

Figure 2.5: Out-Of-Order Migration Phase

~~2.5.4.1.~~2.5.3.1. *Migration of Memory State and Commitment of Import*

~~Memory~~If TDX connect is not enabled for the migrated TD, 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~~import vs. export is not enforced.

The host VMM on the destination platform commits the import by calling TDH.IMPORT.COMMIT (if post-copy is to be used) 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.~~2.5.3.2. *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~~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~~using 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.~~

Post-copy migration is not supported if the TD is enabled for TDX Connect.

~~2.5.6.~~2.5.4. Migration 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~~either the destination ~~and~~or source TD ~~must~~, but not ~~continue to~~both, may 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(DONE) 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(DONE) generates a ~~MSK-based cryptographically authenticated~~secure **start token** ~~to allow~~, which indicates that the TD will not run on the source platform and allows the destination TD to become runnable. On the destination platform, TDH.IMPORT.TRACK ~~, which~~ consumes the ~~cryptographic~~ start token~~,~~. TDH.IMPORT.COMMIT (in case post-copy is used) or TDH.IMPORT.END commits the migration and allows the TD to run on the destination ~~TD to be un-paused~~.

2.5.5. ~~In error~~ Migration Abort

There are two abort scenarios~~, the~~:

Source Initiated Abort: In a live migration ~~process~~ scenario, an error may ~~be aborted proactively by the host~~ cause migration orchestration to abort the migration session while pre-copy is in progress, i.e., the source TD may still run or export blackout started, but no start token has yet been

generated. In such a scenario, the host VMM on the source platform may initiate an abort via TDH.EXPORT.ABORT ~~before~~.

- **Destination Initiated Abort:** If the pre-copy is complete and a start token ~~was~~has been generated~~; if a start token was already generated (i.e. pre-copy completed)~~, abort must be initiated by the host VMM on the destination platform. The TD may be re-allowed to run on the source platform unless the import session has been committed by TDH.IMPORT.COMMIT or TDH.IMPORT.END. To do that, the host VMM can generate an **abort token** using TDH.IMPORT.ABORT. The abort token, which ~~generates an abort token which~~indicates that the TD will not run on the destination platform, is sent to the source platform where it 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~~.

In both scenarios, the host VMM on the source platform must restore the SEPT state of exported pages on the source platform, using TDH.EXPORT.RESTORE or TDH.MEM.SCAN.RANGE(EXPORT_RESTORE) (if supported).

The detailed operations are described in Ch. 6.

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.~~
	~~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

2.8.2.7. Intel TDX Module TD Migration Interface Functions Overview

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

3. TD Migration Software Flows

Unreleased Feature: Some of the text in this section is related to Non-Blocking Export, a feature which has not been released yet at the time of writing of this document. Details related to that feature serve as a preview and are subject to change.

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

3.1. Typical TD Migration Flow Overview (Write-Blocking Based Export)

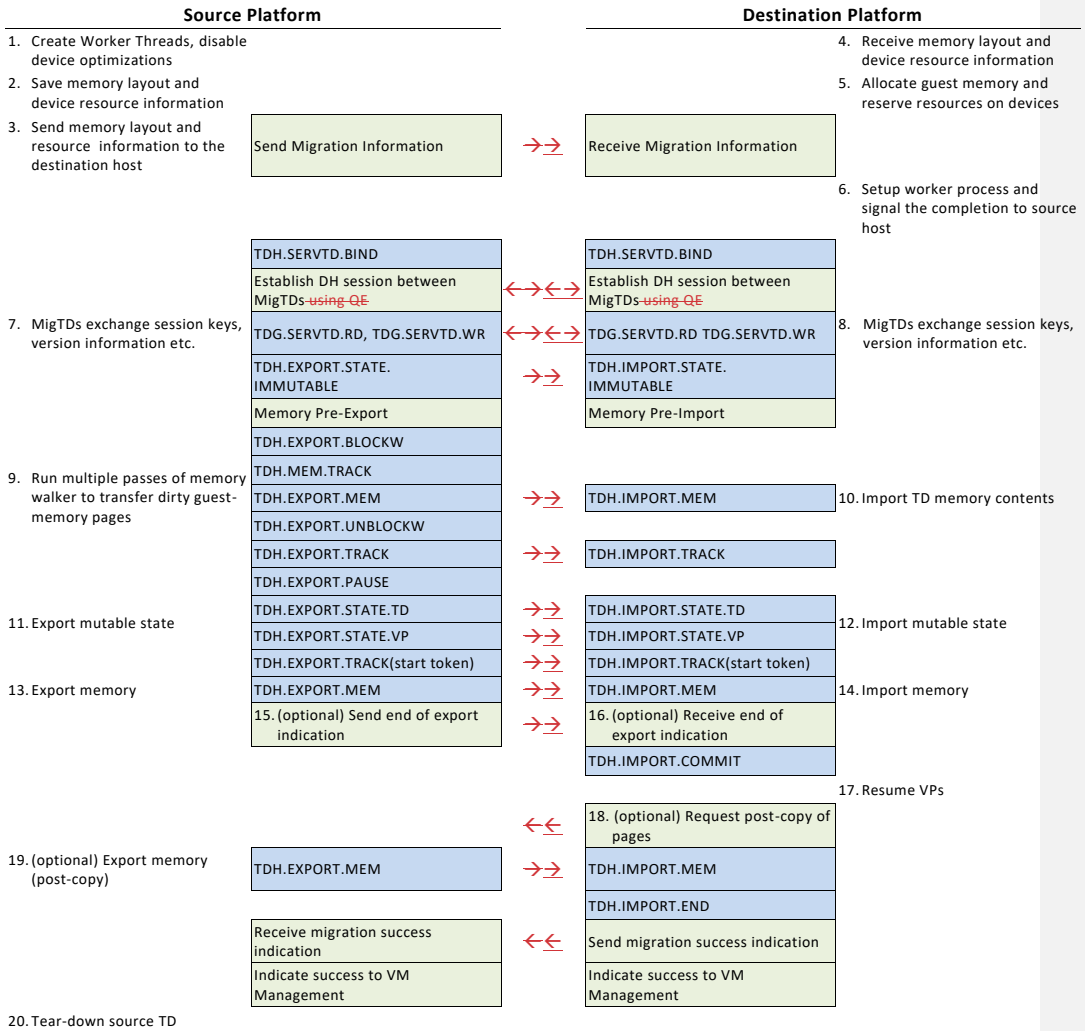


Figure 3.1: Typical TD Migration Flow (Write-Blocking Export)

3.2. Typical TD Migration Flow Overview (Non-Blocking Export)

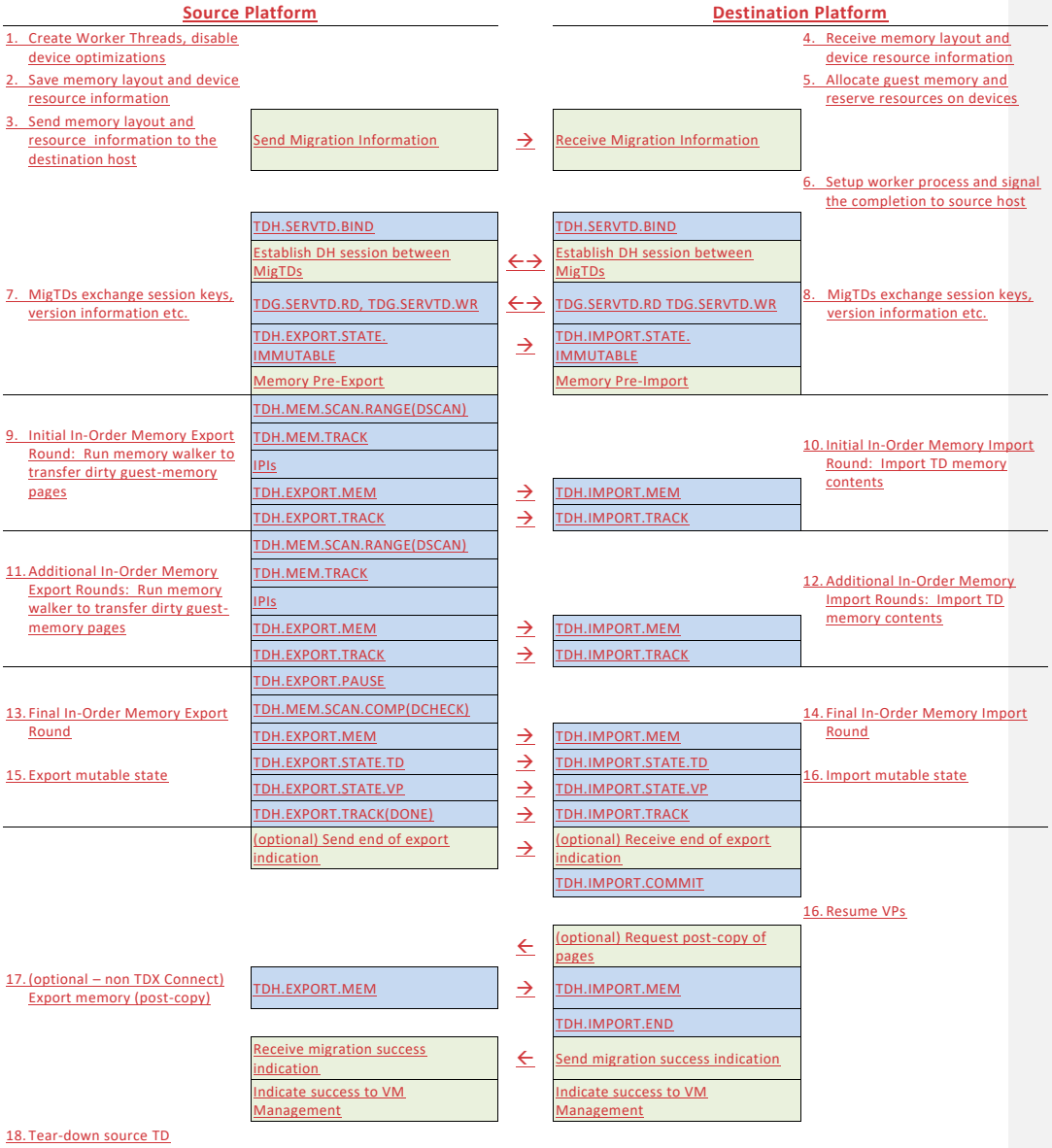


Figure 3.2: Typical TD Migration Flow (Non-Blocking Export)

~~3.2.~~3.3. Successful ~~TD~~Write-Blocking Based Export

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

Table 3.1: Typical Write-Blocking Based 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~~set up 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~~. MigTD 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)
		Host VMM exports TD private pages and creates a memory migration bundle.	Multiple	TDH.EXPORT.MEM

Migration Phase	Step	Description	Plurality	TDX Module Interface Functions
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

3.4. Successful Non-Blocking Export

The following sequence is typically used to export a TD from a source platform using the non-blocking method.

Table 3.2: Typical Non-Blocking TD Export Sequence

Migration Phase	Step	Description	Plurality	TDX Module Interface Functions
In-Order Export	**Start of Export Session**	Same as above		
	Live Memory Export (Initial Round)	Host VMM scans memory for page export candidates.	Multiple	TDH.MEM.SCAN.RANGE(DSCAN)
		Host VMM increments the TD's TLB epoch.	Once per migration epoch	TDH.MEM.TRACK
		Host VMM exports TD private pages reported by TDH.MEM.SCAN and creates memory migration bundles.	Multiple	TDH.EXPORT.MEM
	Live Memory Export (Optional Additional Rounds)	Host VMM scans memory for page export candidates.	Multiple	TDH.MEM.SCAN.RANGE(DSCAN)
		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 reported by TDH.MEM.SCAN and creates a memory migration bundle.	Multiple	TDH.EXPORT.MEM
	Final Memory Export and 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
		Host VMM scans memory for page export candidates.	Multiple	TDH.MEM.SCAN.COMP(DCHECK)
		Host VMM exports, re-exports or cancels the export of TD private pages reported by TDH.EXPORT.SCAN and creates a memory migration bundle.	Multiple	TDH.EXPORT.MEM

Migration Phase	Step	Description	Plurality	TDX Module Interface Functions
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)
		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.~~3.5. Successful Import

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

Table 3.3: 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~~set up 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~~. MigTD 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

Migration Phase	Step	Description	Plurality	TDX Module Interface Functions
		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)
	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.~~3.6. 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.~~3.6.1. TD Import Abort During the In-Order Import Phase

5 **Table 3.4: Typical TD Import Sequence Abort During In-Order Input**

Migration Phase	Step	Description	Plurality	TDX Module Interface Functions

Migration Phase	Step	Description	Plurality	TDX Module Interface Functions
In-Order Import	**Pre-Copy Memory Import (Failed)**	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. 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.~~3.6.2. TD Import Abort During the Out-Of-Order Import Phase

Table 3.5: 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

~~3.5.~~3.7. 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.~~3.7.1. Export Abort During the In-Order Export Phase

Table 3.6: Typical TD Export Sequence Abort During In-Order Export

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.~~3.7.2. Export Abort During the Out-Of-Order Export Phase

Table 3.7: 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

SECTION 2:
TD MIGRATION ARCHITECTURE SPECIFICATION

4. Migration TD, Migration Policy and the Extended TCB

This chapter describes the role of the Migration TD, the migration policy it implements and the extended TCB for migrated TDs. For details of the Intel reference MigTD, refer to the [MigTD Spec].

4.1. Extended TCB and the Migration Policy

The TCB of a migrated TD is extended to include the Migration TD bound to it. The Migration TD builds trust relationships with ~~to~~ other Migration TDs, and through them, with other platforms and their component. Thus, it can be said that **a migrated TD's effective TCB extends to the whole migration pool of platforms**, each with its own Migration TD, TDX modules etc.

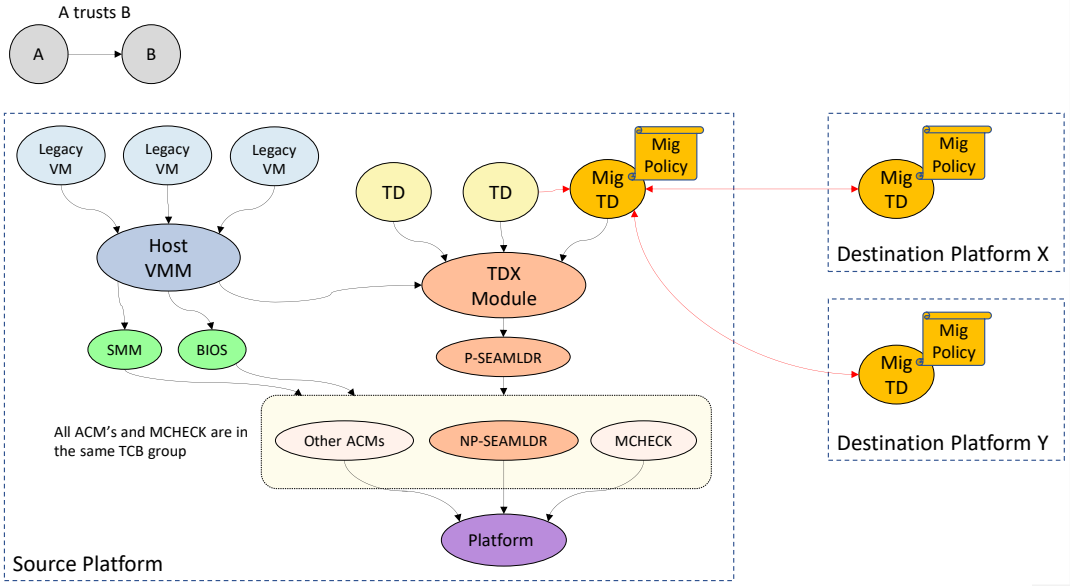


Figure 4.1: Trust Relationship with Migration TDs

This extended TCB is controlled by the **migration policy** implemented by the Migration TD. TDX does not enforce any specific migration policy; this is up to the migration TD's writer.

4.2. Attestation of the Migration TD and its Migration Policy

A migrated TD's attestation reflects the current platform on which it is running. I.e., calling TDG.MR.REPORT generates a TDREPORT_STRUCT with CPUSVN and TEE_TCB_INFO for the current platform. However, there is no requirement to re-attest a TD after migration. The initial attestation, as well as attestation done at any time during the migrated TD's lifetime, contains the information about the TD's extended TCB, provided as a measurement of the Migration TD and its migration policy.

A Migration TD is a private case of a **Service TD**, which is described in the [Base Spec]. As such, a hash of the bound Migration TD's TDREPORT_STRUCT is included in the migrated TD's TDREPORT_STRUCT (as the SERVTD_HASH field). The service TD protocol allows, at binding time, to select which field of the TDREPORT_STRUCT is included in the calculation of SERVTD_HASH.

The static components of the Migration TD, i.e., code and data added and measured during its build, are measured by its MRTD. This may include, for example, a baseline migration policy. Migration TD writers are expected to measure the configurable part of the migration policy and its parameters, i.e., any change that can be made after the Migration TD build was finalized, using one or more of the RTMRs (run time measurement registers) provided by the TDX architecture via the TDG.MR.RTMR.EXTEND interface function. Those RTMRs should be included in the SERVTD_HASH calculation. Thus, the migration policy is attested as part of the Migration TD, and as such, as part of the migrated TD.

SERVTD_HASH is not migrated; it is recalculated at the beginning of an import session (TDH.IMPORT.STATE.IMMUTABLE) to reflect the Migration TD which is bound at the destination.

4.3. Inputs to the Migration TD's Migration Policy Evaluation

The following data can be read and evaluated by the Migration TD, as part of its migration policy evaluation. The migration TD may reflect such inputs in an RTMR (using TDG.MR.RTMR.EXTEND) so that it becomes part of any migrated TD's attestation.

System-Scope Information Provided by TDG.SYS.RD*

System-scope (platform and TDX module) information can be read using TDG.SYS.RD or TDG.SYS.RDALL. Examples of such information are:

- TDX features supported on this platform (e.g., TDX_FEATURES0 field)
- Supported TD features (e.g., ATTRIBUTES_FIXED0/1, XFAM_FIXED0/1, CPUID_CONFIG_VALUES and IA32_ARCH_~~CAPABILITIED~~CAPABILITIES_CONFIG_MASK fields)
- TD Migration protocol features (e.g., MIN/MAX_EXPORT/IMPORT_VERSION fields)

The complete list of fields enumerated by TDG.SYS.RD* is provided in the [ABI Spec].

System-Scope Information Provided by TDG.MR.REPORT

The Migration TD may call TDG.MR.REPORT to get the TDREPORT_STRUCT, which contains information about the platform (CPUSVN) and the TDX module (TEE_TCB_INFO).

Migration Policy Configuration

The Migration TD's migration policy may be configured by, e.g., the host VMM.

4.4. Migrated TD Information Provided by TDG.SERVD.RD

~~A~~ Migration ~~TD~~TDs on both sides may read migrated TD information using TDG.SERVD.RD, ~~in order~~ to decide whether that TD can be migrated to a specific destination platform. Note that similar checks are also done by the TDX module on import, so there's no strict requirement for the Migration TD to do them.

Examples of such information are:

- ATTRIBUTES_FIXED0/1
- XFAM_FIXED0/1
- GPAW
- CPUID_VALUES

4.5. Migration Protocol Version Setup

The migration protocol supports versioning, to allow for future updates.

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 migrated TD using TDG.SERVD.WR.

4.6. Migration Session Keys (MSKs) 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. The TDX module generates a new key on each such read operation.
2. The MigTD on each side sends the key value over a secure channel to the peer MigTD on the other side.

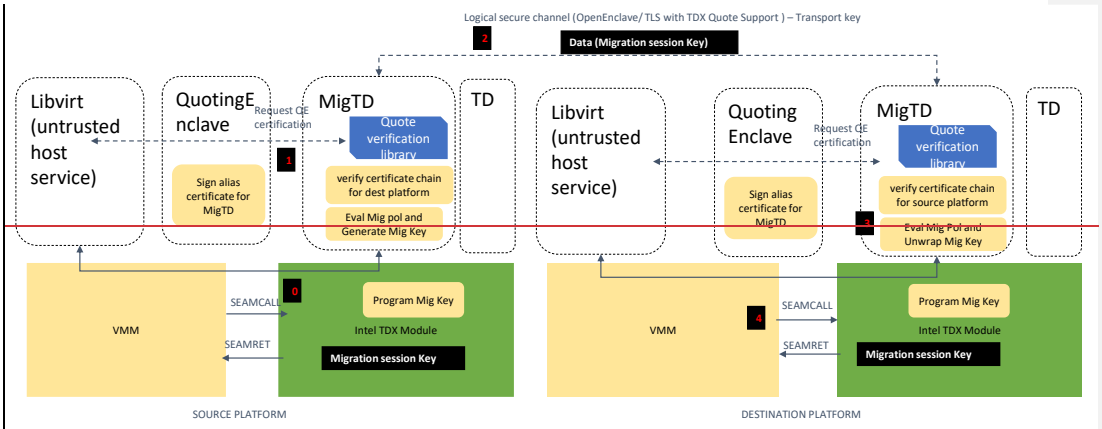
Note: **A MigTD must never send the same key value to more than one peer MigTD**. Failing to do so may expose TDX to an attack where the same TD is migrated to more than one destination. The MigTD should always read a new MIG_ENC_KEY value using TDG.SERVTD.RD if it needs to resend it.

3. The MigTD on each side writes the received key value as the migration decryption key (MIG_DEC_KEY), using TDG.SERVTD.WR.

4.7. Example Migration Session Establishment

Note: The example below illustrates migration session establishment and doesn't necessarily imply actual implementation.

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



~~**Figure 4.2: MigTD Transport Security Setup**~~

1. On the source platform~~1~~, TD-s to be migrated is created with MIGRATABLE attribute.~~ TD-s is built using the legacy process.~~
2. On the source platform~~1~~, MigTD-s is created and built.
3. On the source platform~~1~~, MigTD-s evaluates the platform and TDX module information using TDG.SYS.RD* and TDG.MR.REPORT and reflects its updated migration policy in the MigTD's RTMR[3], using TDG.MR.RTMR.EXTEND.
4. The host VMM on the source platform ~~1 instantiates~~ binds MigTDs to TD-s via an unsolicited ~~service~~Service TD binding using TDH.SERVTD.BIND:
   4.1. MigTD-s requests the host VMM to be bound to TD-s, using TDG.VP.VMCALL.
   4.2. The VMM invokes TDH.SERVTD.BIND to bind TD0 to MigTD-s. As a result, the MigTD's TDREPORT fields' hash is included in the migrated TD's TDREPORT (as SERVTD_HASH).
   4.3. ~~The VMM communicates~~ TDH.SERVTD.BIND returns the migrated TD's binding handle~~, target~~ and TD_UUID ~~and other binding parameters~~. The host VMM communicates them to MigTD-s.
5. The host VMM on the source platform ~~1 finalized~~finalizes the build of TD-s using TDH.MR.FINALIZE.
6. TD-s executes on the source platform~~1~~.
7. Similar to MigTD-s above, on the destination platform~~2~~, MigTD-d is created and built. It evaluates ~~is~~its platform and TDX module and reflects ~~in~~the updated migration policy in RTMR[3].
8. The ~~cloud orchestration~~migration orchestrator triggers a migration of TD-s from the source platform ~~1~~ to the destination platform~~2~~.

~~Section 2: TD Migration Architecture Specification~~

9. 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.

~~10.~~ The ~~host VMM initiates the~~ migration ~~process for the TD-s:~~

~~10.1.~~10. ~~The VMM creates~~orchestrator requests the source and destination Mig TDs to establish a ~~network transport~~secure session ~~(nonce) with the destination platform (destination of TD~~ and negotiate the migration~~) and requests a quote from the MigTD-d on destination platform~~ policy agreement.

~~10.2. 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).~~

~~10.3.~~11. MigTD-s verifies the MigTD-d quote ~~using a Quote Verification Library in the MigTD-s~~ and establishes ~~via Diffie-Helman a~~a secure transport key for the session with the destination platform (and vice-versa).

~~11.~~12. MigTD-s authenticates the Migration Policy and evaluates it ~~per~~vs. the capabilities (SVN etc.) of the destination platform (learnt via the quote) for the specified live migration session.

~~12.~~13. On the destination platform~~-2~~, a destination TD-d skeleton is created via legacy process.

~~13.~~14. On the destination platform~~-2~~, the host VMM may ~~binds~~bind MigTD-d to TD-d using TDH.SERVTD.BIND.

~~14.~~15. Migration Keys Exchange:

~~14.1.~~15.1. MigTD-s reads the ~~Migration Forward Key (as the~~ source migration encryption key~~) from TD-s~~ using TDG.SERVTD.RD. This key is used as the forward migration session key, to encrypt information exported by the TDX module.

~~14.2.~~15.2. MigTD-s sends the ~~Migration Forward Key~~forward migration session key to MigTD-d.

~~14.3.~~15.3. On the destination platform, MigTD-d writes the ~~Migration Forward Key (~~forward migration session key, as the migration decryption key~~) to TD-d,~~ using TDG.SERVTD.WR.

~~14.4.~~15.4. MigTD-d reads the ~~Migration Backward Key (as the~~ destination migration encryption key~~) from TD-d~~ using TDG.SERVTD.RD. This key is used as the backward migration session key, to encrypt information sent by the TDX module on the destination.

~~14.5.~~15.5. MigTD-d sends the ~~Migration Backward Key~~backward migration session key to MigTD-s.

~~14.6.~~15.6. On the ~~source~~destination platform, MigTD-~~s~~d writes the ~~Migration Backward Key (~~backward migration session key, as the migration decryption key~~) to TD-s,~~ using TDG.SERVTD.WR.

~~15.~~16. 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. Common 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

A migration bundle is the basic unit of information transported between the source and destination platforms. This section describes the generic migration bundle structure. Private memory migration uses an enhanced format, described in 0.

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)**. The host VMM may add its own untrusted metadata to the migration bundle, for managing the migration. Migration bundle transport is the responsibility of untrusted software and is out of the scope of this specification.

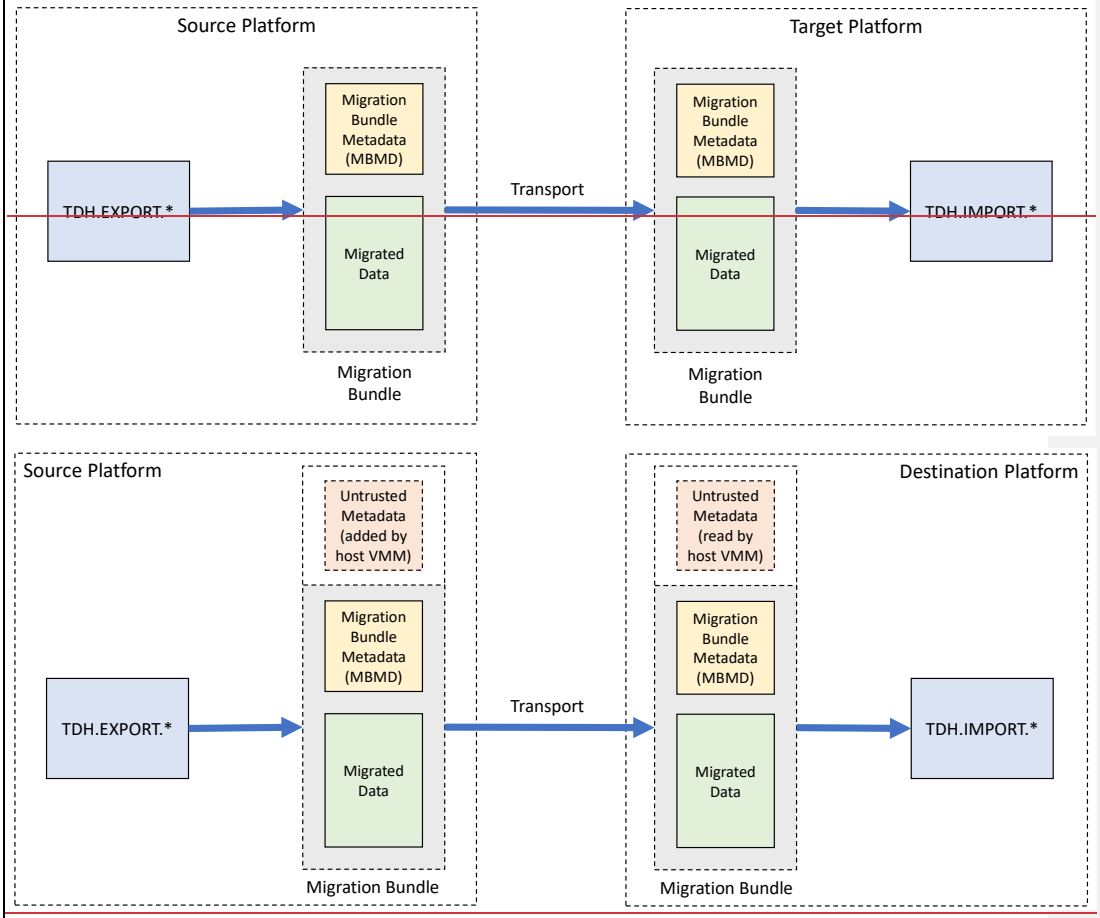


Figure 5.1: Migration Bundle

5.1.2. Migration Data

Migration data contains either TD private memory contents or TD non-memory state. It confidentially 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~~beyond 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~~by the host VMM.

5.1.3. Migration Bundle Metadata (MBMD)

A migration bundle metadata (MBMD) structure provides metadata for ~~an associated~~the migrated data in the migration ~~data~~bundle. 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~~. The host VMM can read the MBMD; this is required for the VMM to perform the required operations. E.g., the host VMM on the destination platform reads the MBMD to decide which import function to call.

The MBMD 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~~Monotonically 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].

5.1.4. Untrusted Metadata

The host VMM may add its own metadata to the migration bundle, e.g., to support its implementation of the migration protocol. This metadata is untrusted and is not used by the TDX module.

5.2. Export and Import Functions Interface

~~Export~~ On each invocation, export and import functions operate on a single migration bundle ~~at a time, which belongs to~~ and 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.~~5.2.1. Migrating a Multi-Page Migration Bundle

Migration bundles may consist of multiple pages of migrated information. 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~~buffers 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. The required number of migration pages per TDH.EXPORT.* function is enumerated by the TDX module. See the [ABI Spec] for details.

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.

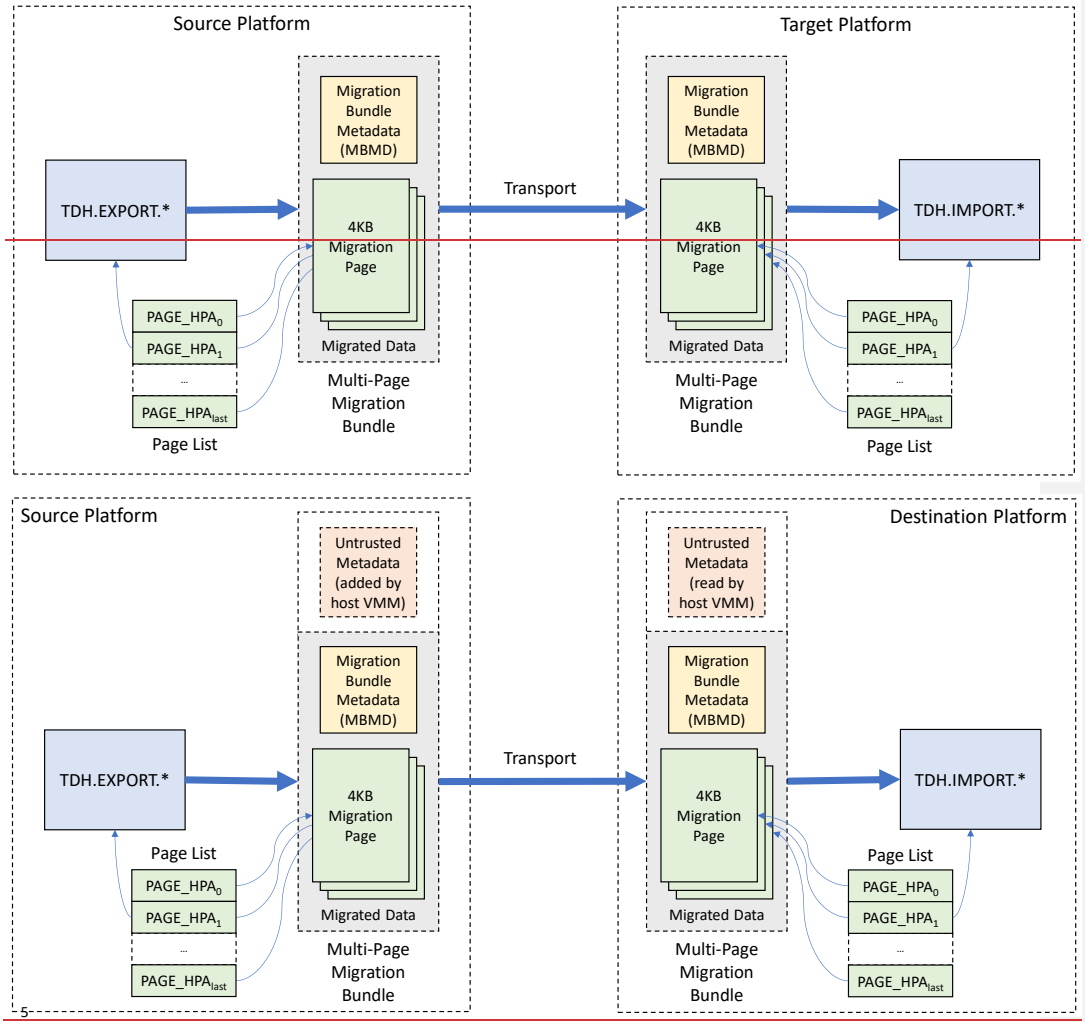


Figure 5.2: Migrating a Multi-Page Migration Bundle

~~5.2.3.~~5.2.2. Migration Functions Interruptibility

TDH.EXPORT.* and TDH.IMPORT.* functions may take relatively long time to execute. This is especially true for ~~2MB page~~memory migration~~ and multiple~~, which can process up to 512 4KB ~~page migration.~~pages. To avoid latency issues, such functions may be **interruptible** and ~~**restartable**~~**resumable**. 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).
- As described later, migration functions that work with a Migration Stream use Migration Stream Context (MIGSC). If an external event is pending, the functions store their context in the proper MIGSC and ~~returns~~return 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. A migration function which is called as a resumption of an interrupted operation 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.

5.3. Cryptographic Protection ~~for~~of 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~~its side, known as the **migration encryption key**, from the TDX module's ~~target~~migrated TD's metadata, and ~~transfer~~transfers it ~~on~~using a secure ~~like~~connection 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~~migrated TD's metadata. TD metadata read and write ~~used~~use the Service TD protocol, as described in the [TDX Module Base Spec].

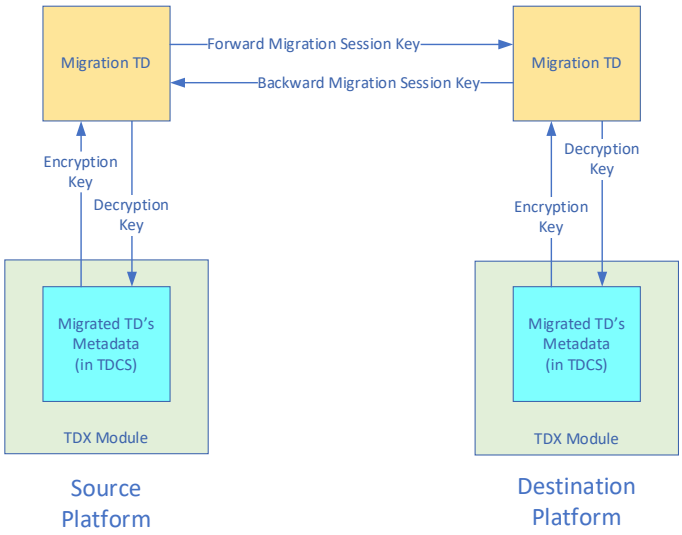


Figure 5.3: Migration Session Keys Exchange

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~~transfers 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 the start of 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 a 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(DONE) 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.

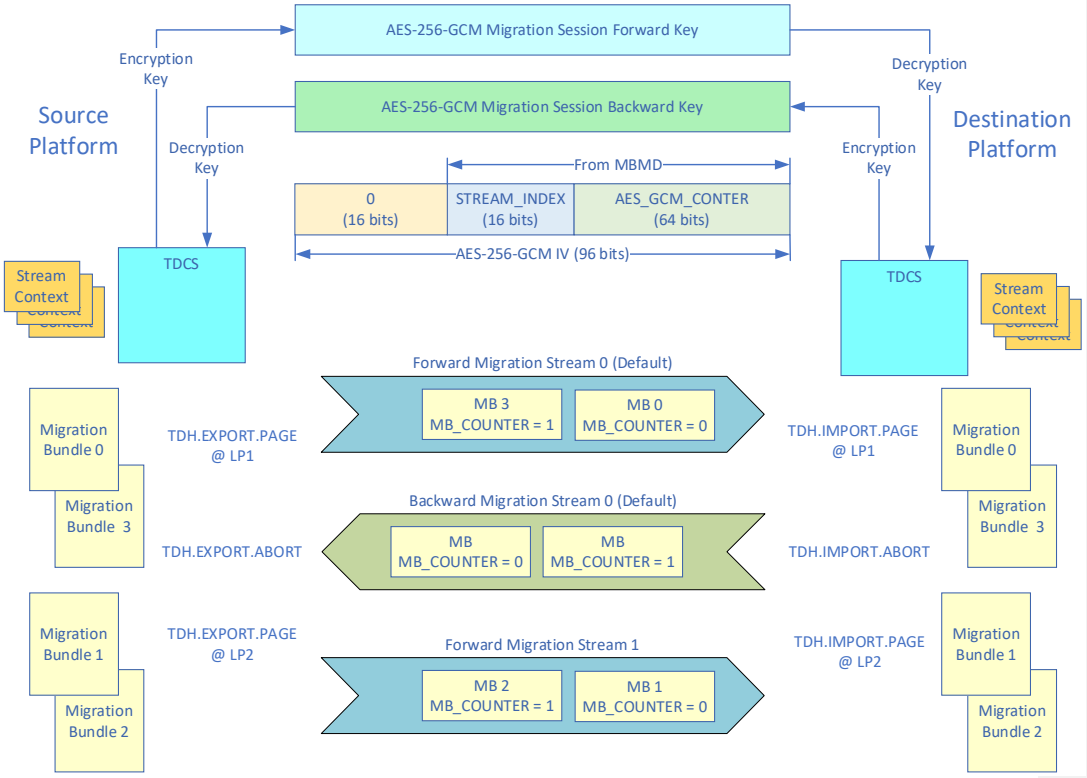


Figure 5.4: Migration Streams

Migration streams have the following characteristics:

- Within each stream, the 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.
- There is one backward stream.

- The maximum total number of forward and backward migration streams is enumerated by MAX_MIGS, readable TDH.SYS.RD*.

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 TD 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~~ etc.
- 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~~Structures Support of TD Migration

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

~~Section 2: TD Migration Architecture Specification~~

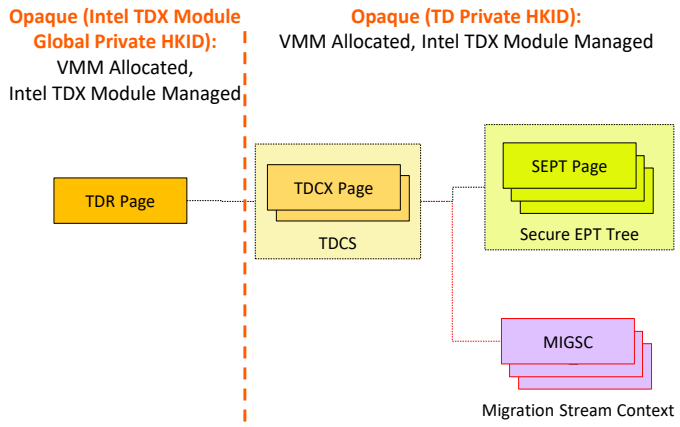


Figure 5.5: TD Scope Control Structures Overview

5.6.1. Per TD: TDCS

As described in 6.2, to support 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].

5.6.3.5.6.1. 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; it 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

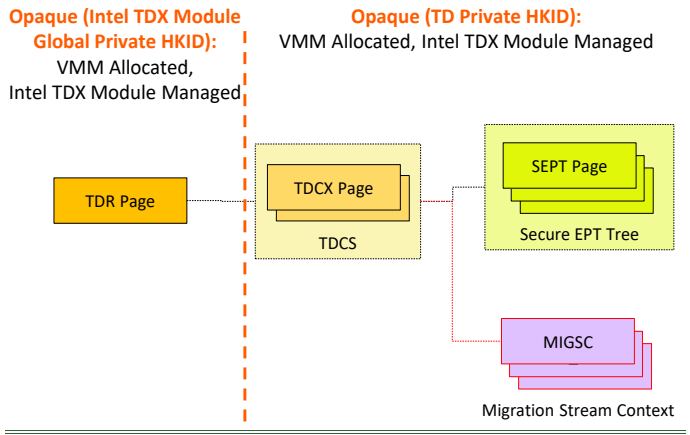


Figure 5.5: TD-Scope Control Structures Overview

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~~On the destination platform, 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 a migration protocol version.

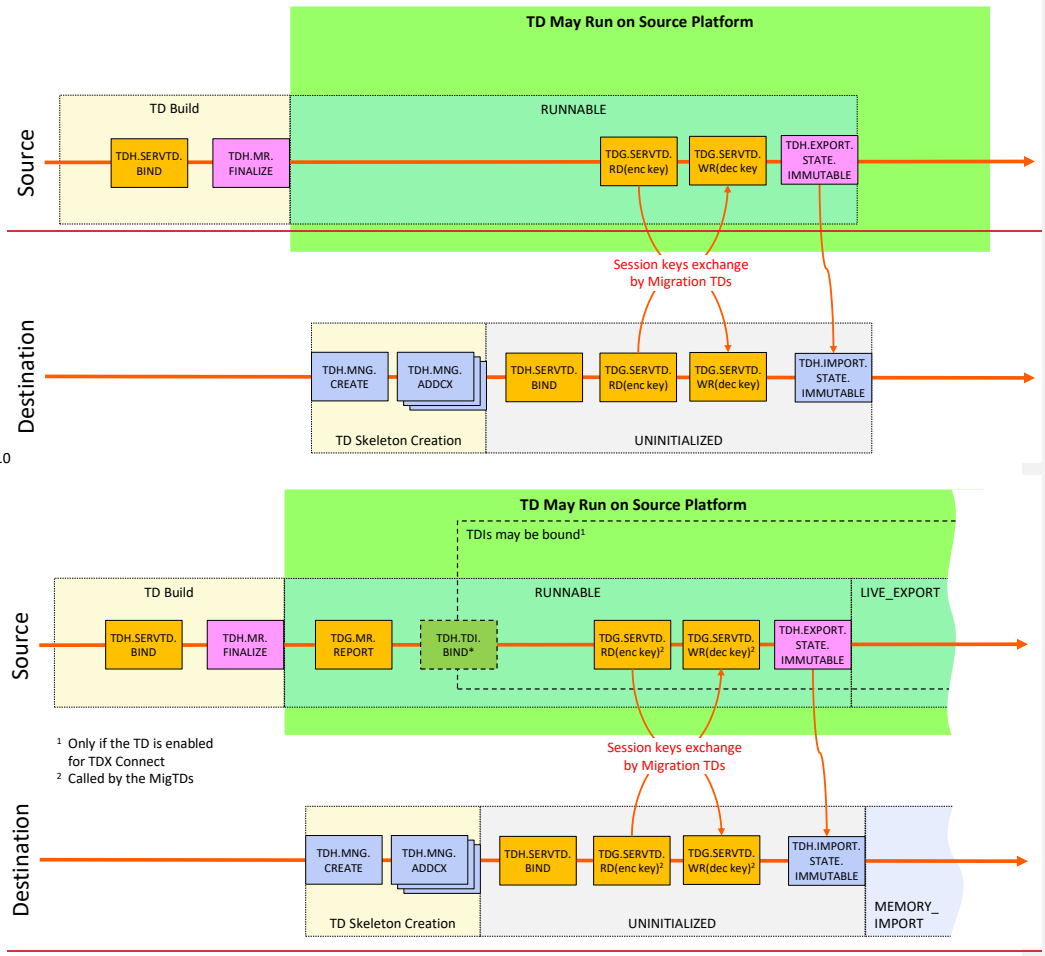


Figure 6.1: Pre-Migration

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 6.2.4.2 below.
- Phases of the export and import sessions

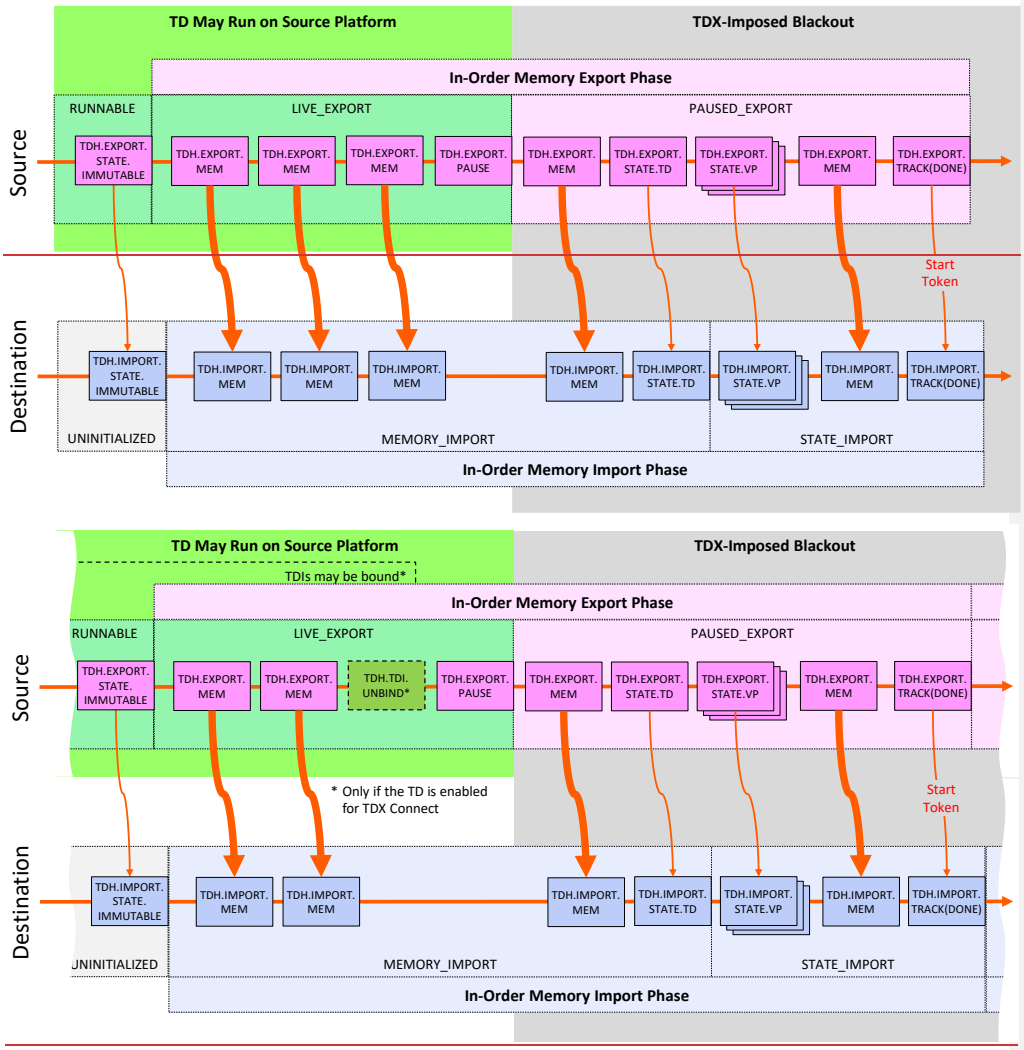


Figure 6.2: Migration Session In-Order Phase (Success Case)

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. If the TD is configured with TDX Connect enabled, the TDX module checks that all non-migratable assets have been released (e.g., MMIO mappings, TDIs).

TDH.EXPORT.TRACK, (DONE), when invoked on the source platform, verifies proper in-order export. ~~TD~~:

- The TD's non-memory state must have been exported~~, and if~~.
- If any TD private page has been exported, the latest version of that page must have been exported.
- If the TD is configured with TDX Connect enabled, all TD private pages must have been exported.

TDH.EXPORT.TRACK(~~done~~DONE) ends ~~that~~the in-order memory export phase and creates a **start token** migration bundle that is transmitted by the host VMM to the destination platform, where TDH.IMPORT.TRACK(~~done~~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~~can no longer 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~~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.

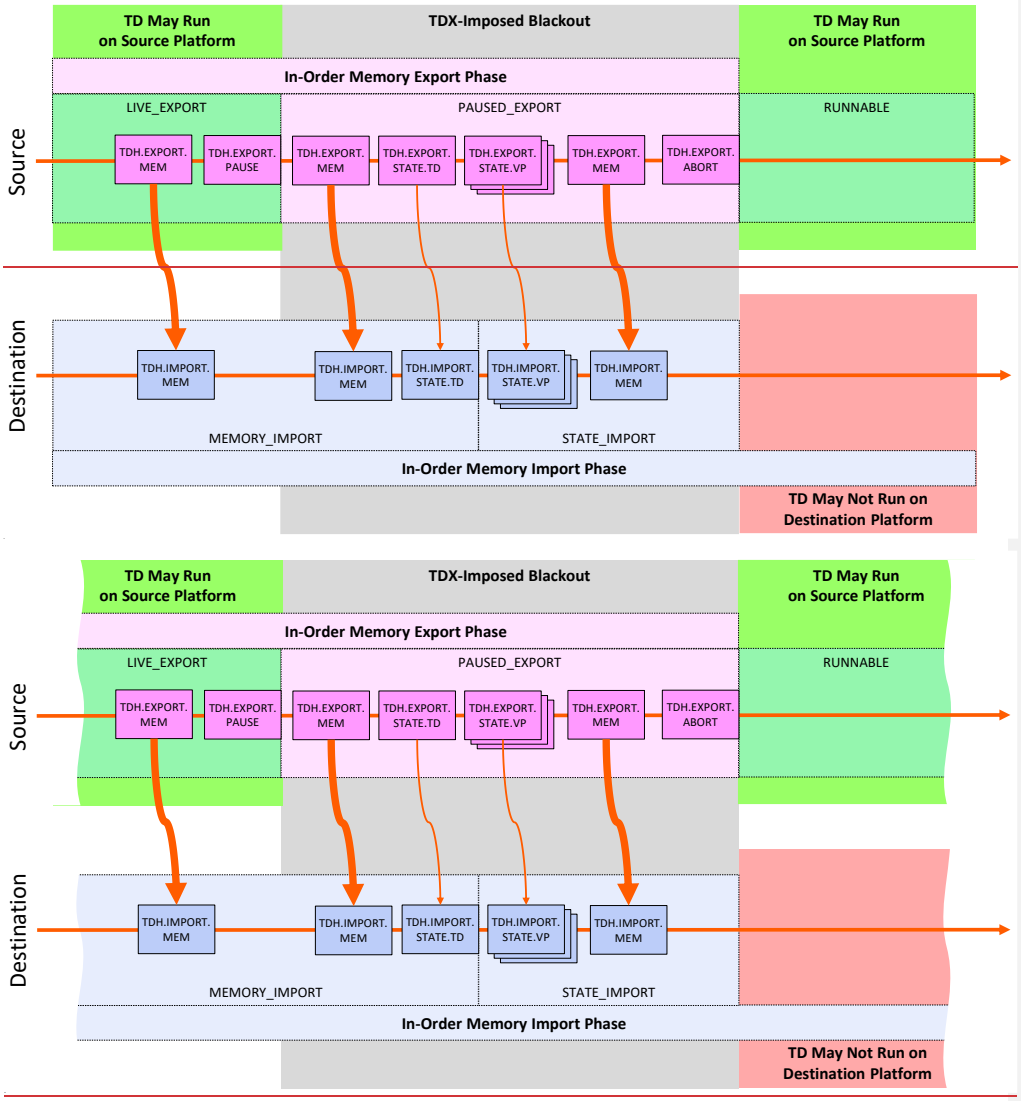


Figure 6.4: Migration Session Control Overview (Abort During the In-Order Phase)

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~~can 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.

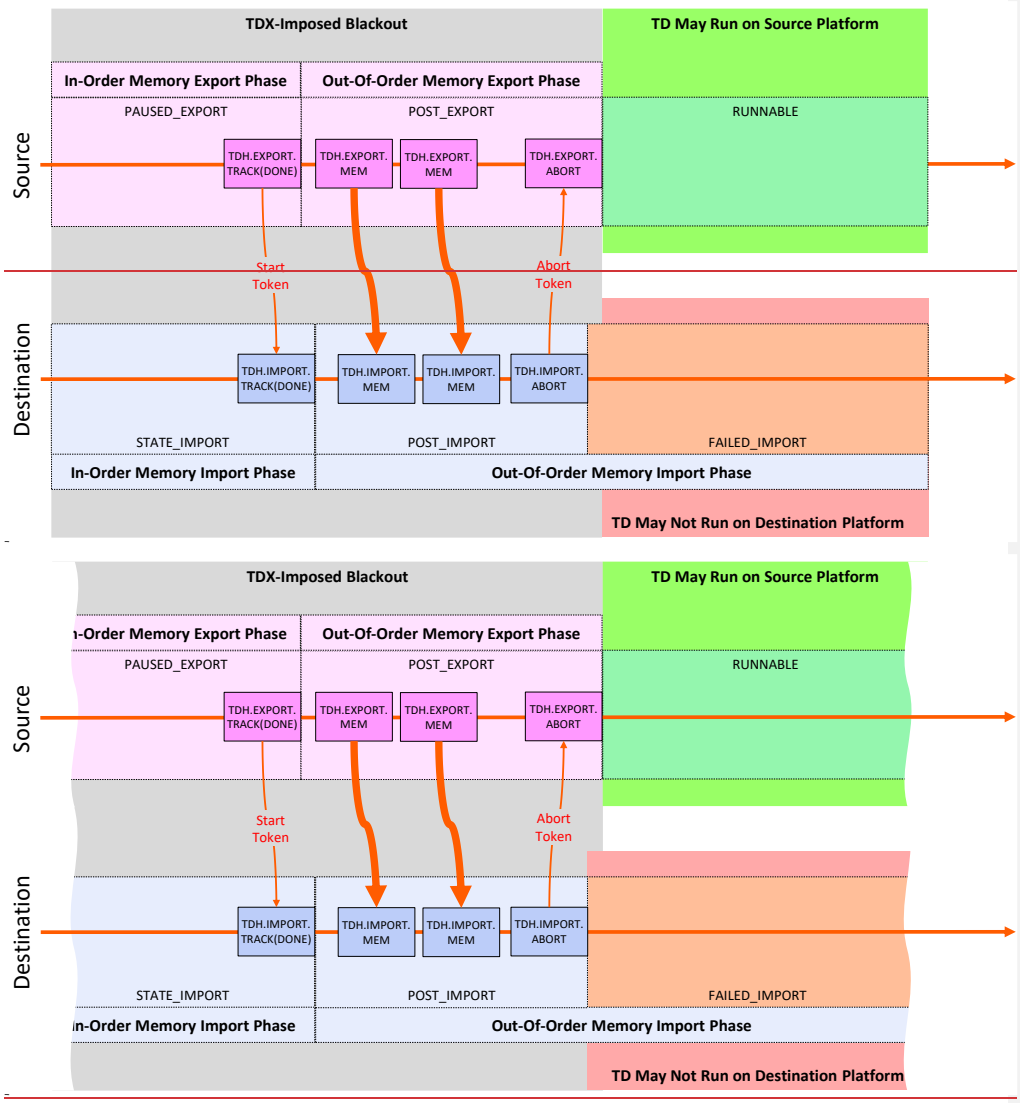


Figure 6.5: Migration Session Control Overview (Abort During the Out-Of-Order Phase)

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.~~

As described in 5.4, 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.

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**, generated on the source platform by TDH.EXPORT.TRACK, 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, check 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.

Notes

- Do not confuse TDH.MEM.TRACK (which is used for TLB tracking) with TDH.EXPORT.TRACK (which rendezvous all migration streams).
- Migration epoch is a TDX concept. It roughly corresponds to **migration round** (or **migration pass**) which is a usage concept.

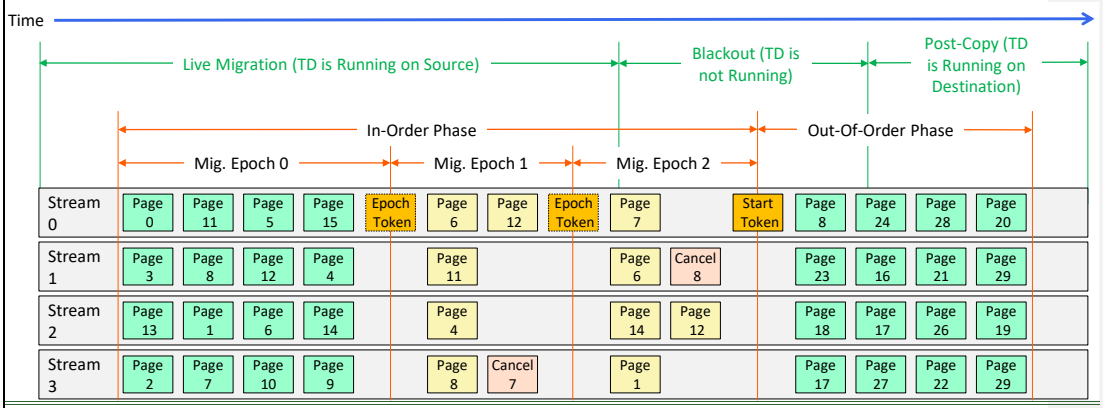
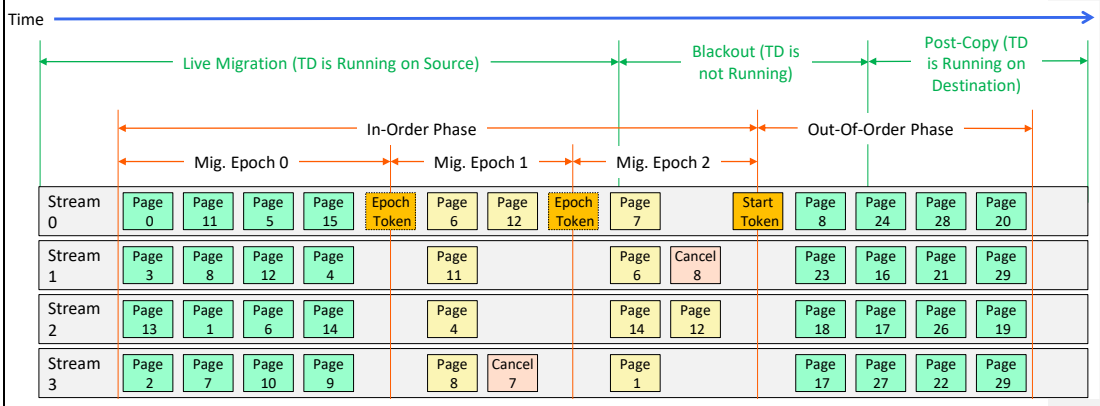


Figure 6.6: Migration Epochs Overview

6.2. Migration Session Control

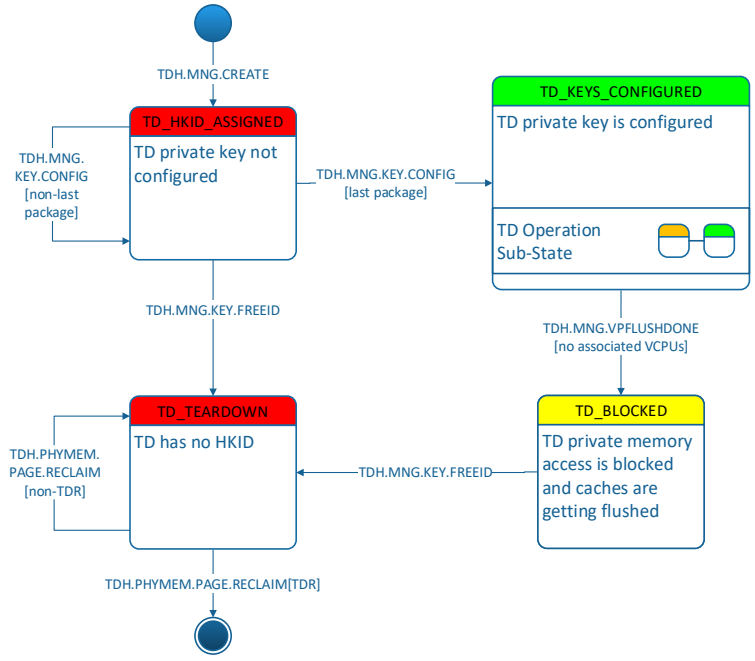


~~Figure 6.6: Migration Epochs Overview~~

~~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].~~

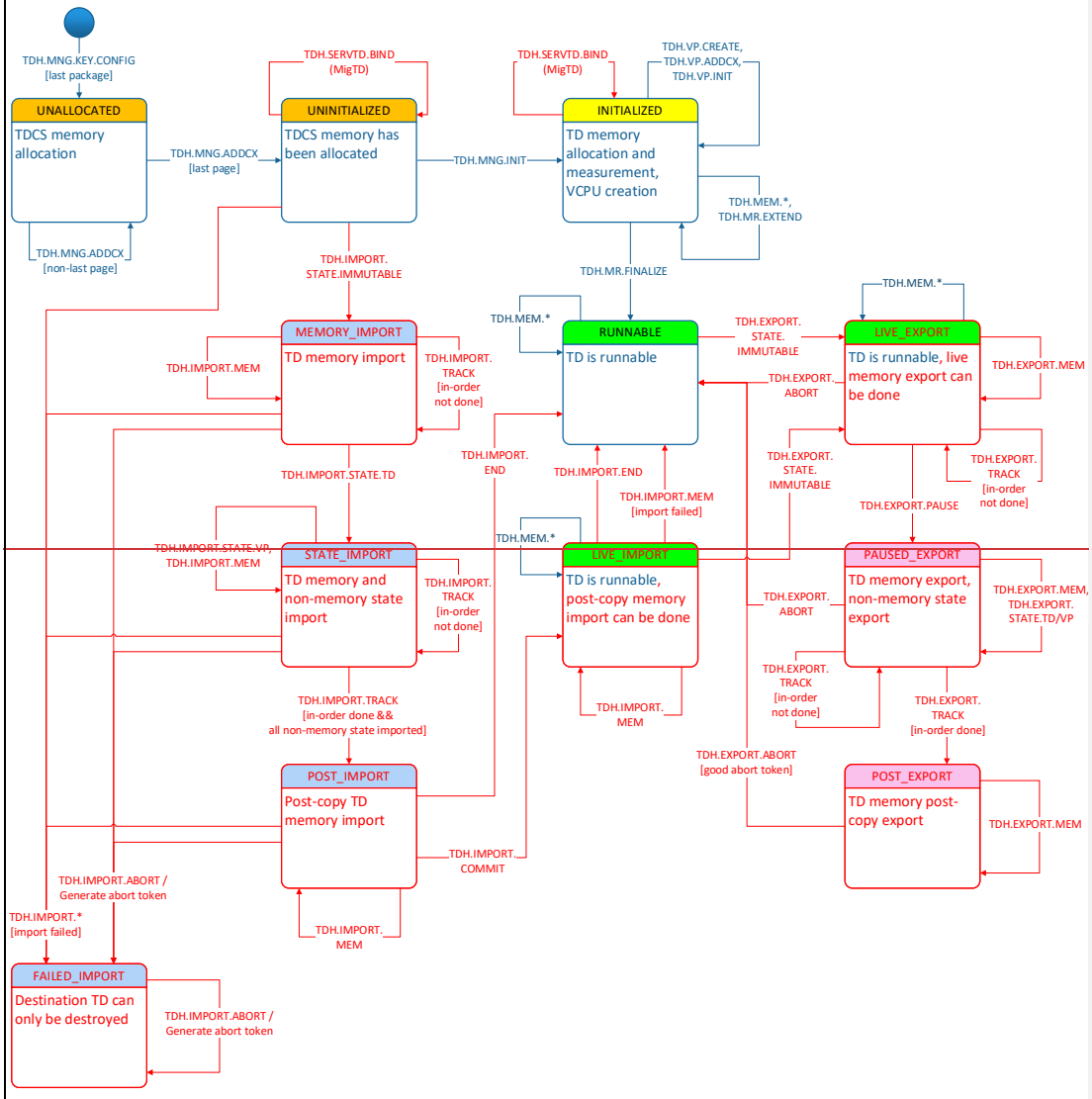


~~Figure 6.7: TD Life Cycle State Diagram~~

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

~~6.2.2.1.1.1.1. 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.~~6.2.1. 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.~~6.2.2. 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 non-memory 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. For TDX Connect, all non-migratable assets have been released (e.g., MMIO mappings, TDIs). 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~~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~~If the TD has not been configured with TDX Connect enabled, memory export (TDH.EXPORT.MEM) may continue~~,~~; this is required 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.~~6.2.3. 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~~import failure in the LIVE_IMPORT state terminates the import session; it transitions the TD's OP_STATE to the RUNNABLE state, where the TD can continue running normally. ~~Abort~~An 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.

6.2.4. Details: Migration State Machine

This section provides a detailed view of the migration session control state machine. Details may be of interest to host VMM programmers who require a deeper understanding of TD Migration.

6.2.4.1. *Details: Reminder: TD Lifecycle State Machine*

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].

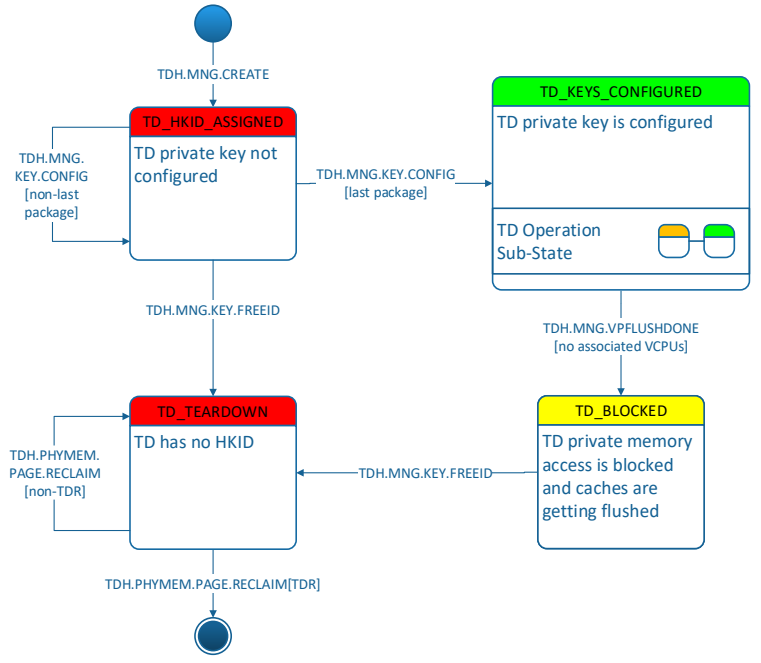


Figure 6.7: TD Life Cycle State Diagram

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

6.2.4.2. *Details: 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.

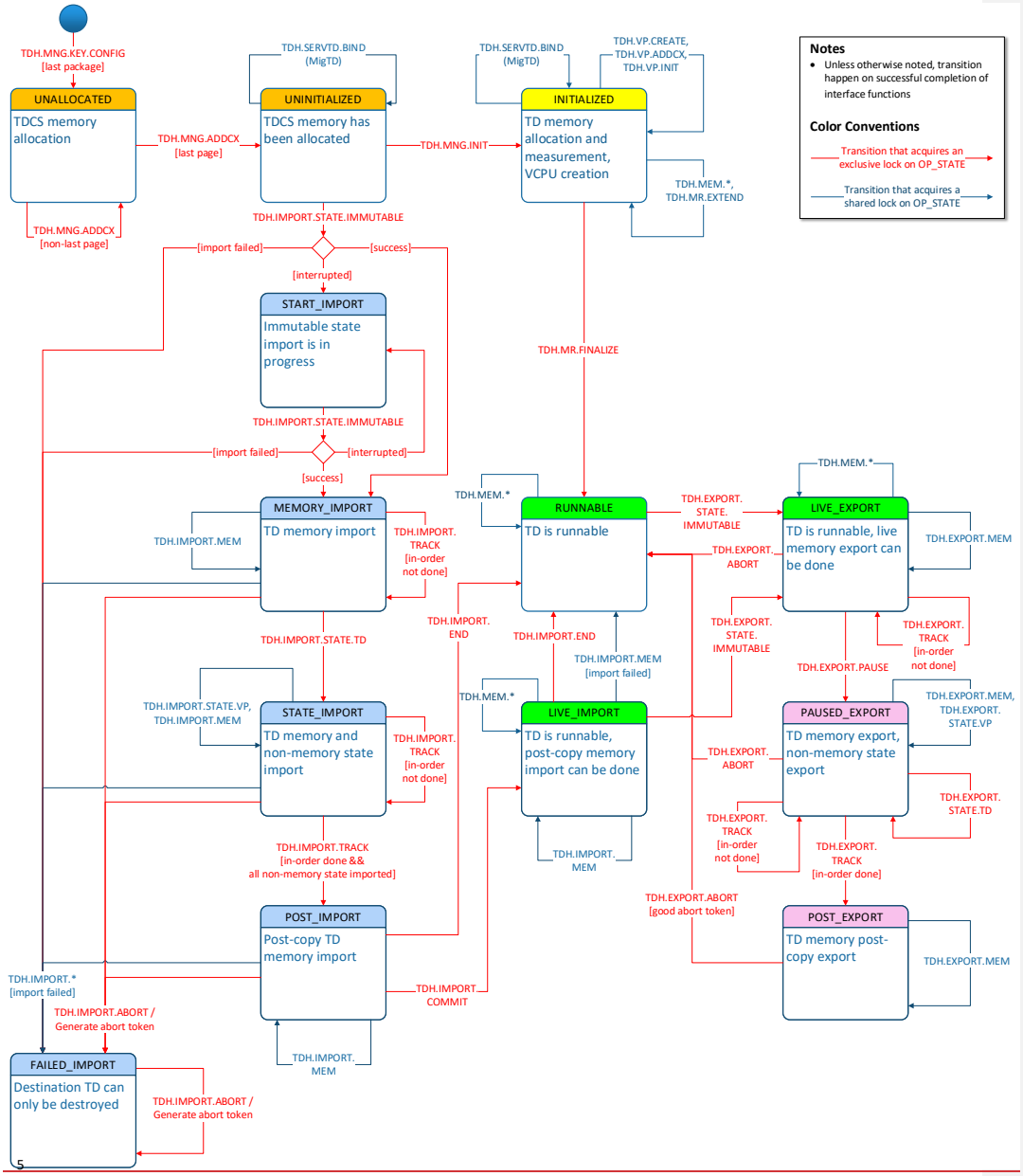


Figure 6.8: TD Operation State Machine (Sub-States of TD_KEYS_CONFIGURED)

6.2.6. Details: 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).

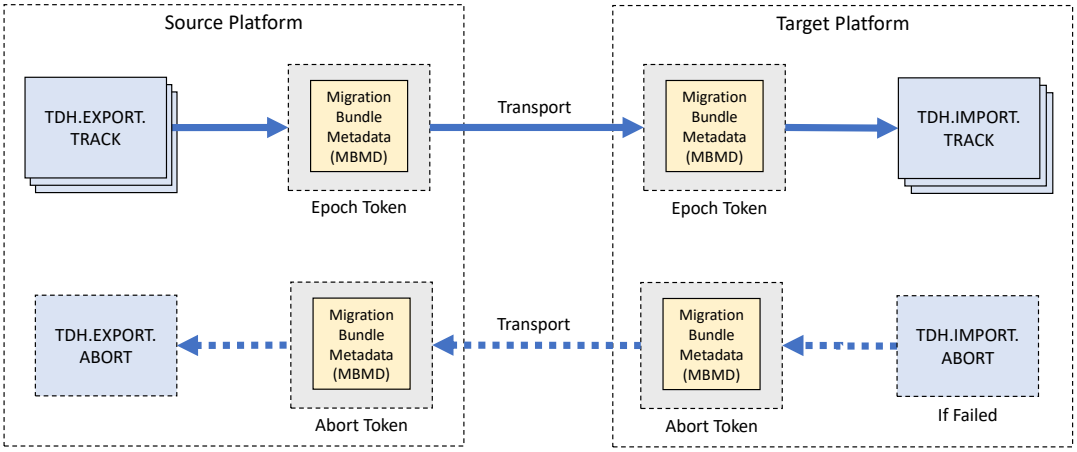
~~6.2.7.~~6.2.4.3. *Summary*

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	• Export session started. • 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).
START_IMPORT	Destination	• Import session started. • TDH.IMPORT.STATE.IMMUTABLE has been interrupted.
MEMORY_IMPORT	Destination	• Immutable non-memory state (TDH.EXPORT.STATE.IMMUTABLE) has been imported. • TD memory can be imported.
STATE_IMPORT	Destination	• Mutable non-memory TD-scope state (TDH.EXPORT.STATE.TD) has been imported. • 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

~~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**~~

Migration tokens are transmitted from the source platform to the destination platform and vice versa as part of the migration session control.

An **epoch token** is generated by TDH.EXPORT.TRACK. ~~it~~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.

Migration tokens are formatted as Migration Bundles, with only an MBMD. Its format is defined in the [TDX Module ABI Spec].

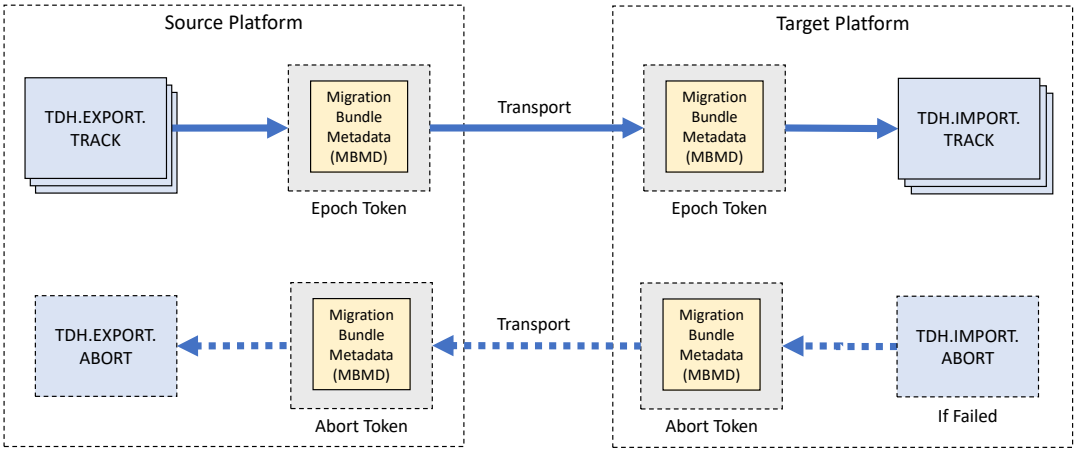


Figure 6.9: Migration Tokens

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~~enumerates supported migration versions 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~~import

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~~The 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~~. They 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~~Migration Session Control Functions Summary

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 (~~Table 6.2: Migration 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 ControlInterface 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.~~

Name	Description	Preconditions
TDH.EXPORT.STATE.IMMUTABLE	Start an export session. This function exports the TD's immutable state – that functionality is discussed in Ch. 7.	• TD is runnable. • A new migration key has been configured. • Enough migration stream contexts have been created using TDH.MIG.STREAM.CREATE.
TDH.EXPORT.PAUSE	Pauses the source TD and starts the TDX-enforced blackout period. This operation is local to the source platform and is not communicated to the destination platform.	• TD immutable state has been exported. • All TD VCPUs have stopped executing and no other TD-specific SEAMCALL is running.
TDH.EXPORT.TRACK	Starts a new migration epoch and generate an epoch token. If so requested, starts the out-of-order phase and generates a start token.	• The export session is in progress, but its out-of-order phase has not begun yet. • See the discussion of export completeness checks in Ch. 8.
TDH.EXPORT.ABORT	Aborts the export session.	• An export session is in progress. • If invoked during the out-of-order phase, the abort token received from the destination platform must be correct.
TDH.IMPORT.STATE.IMMUTABLE	Start an export session. This function exports the TD's immutable state – that functionality is discussed in Ch. 7.	• The TDCS been allocation but not initialized. • A new migration key has been configured. • Enough migration streams contexts have been created using TDH.MIG.STREAM.CREATE.
TDH.IMPORT.TRACK	Starts a new migration epoch based on an imported epoch token. If the token is a start token, starts the out-of-order phase.	• The import session is in progress, but its out-of-order phase has not begun yet. • The epoch token migration bundle is imported successfully. • If a start token is received, all mutable state must have been imported.
TDH.IMPORT.COMMIT	Enable the TD to run.	• The import season's out-of-order phase is in progress.
TDH.IMPORT.END	Ends the import session.	• The import season's out-of-order phase is in progress.

Name	Description	Preconditions
TDH.IMPORT.ABORT	Aborts the import session (if not already aborted) and generates an abort token.	• The import session is in progress. • The TD is not allowed to run yet (OP_STATE is not LIVE_IMPORT).

•

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.

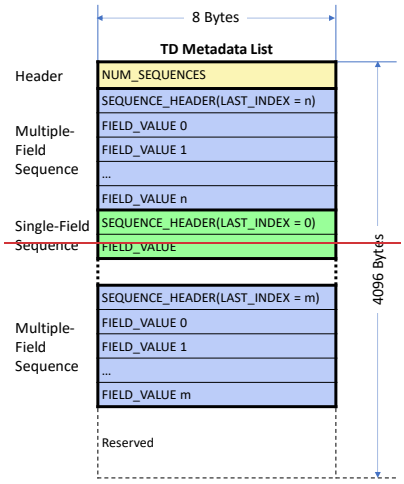
7.1. TD Non-Memory State Migration Operation

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. 6.

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~~); it 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~~VP calls 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 calls 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~~

7.2. Expected Configuration by the Host VMM

The host VMM is expected to configure a migratable TD in a way that will be compatible with the set of possible destination platforms and their Intel TDX module configurations. For example:

- The configured TD ATTRIBUTES and XFAM bits should be supported by all destination platforms and their Intel TDX module configurations.
- The configured virtual CPUID values should be supported by all destination platforms and their Intel TDX module configurations.

~~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~~Migration Functions Summary

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

~~TDH.EXPORT.STATE.IMMUTABLE exports~~This section provides a short summary of the ~~TD's immutable~~non-memory state ~~as a multi-page~~ migration ~~bundle. It also starts the export session – that functionality is described in 6.5.1.~~ interface functions. A detailed ~~description of TDH.EXPORT.STATE.IMMUTABLE~~specification 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.~~ Table 7.1: Non-Memory State ~~Import~~Migration Interface 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**~~

Name	Description	Preconditions
TDH.EXPORT.STATE.IMMUTABLE	Export the TD's immutable state as a multi-page migration bundle. This function starts the export session; that functionality is described in Ch. 6.	• TD is runnable. • A new migration key has been configured.
TDH.EXPORT.STATE.TD	Export the TD-scope mutable state as a multi-page migration bundle.	• The export session is in the in-order phase and the TD has been paused.
TDH.EXPORT.STATE.VP	Export the VCPU-scope mutable state as a multi-page migration bundle.	• The export session is in the in-order phase and the TD has been paused.
TDH.IMPORT.STATE.IMMUTABLE	Import the TD's immutable state as a multi-page migration bundle. This function starts the import session; that functionality is described in Ch. 6.	• TD has not been initialized. • A new migration key has been set.
TDH.IMPORT.STATE.TD	Import the TD's mutable state as a multi-page migration bundle.	• TD immutable state has been imported.
TDH.IMPORT.STATE.VP	Imports a VCPU mutable state as a multi-page migration bundle.	• TDVPS pages have been allocated by the host VMM, but the VCPU has not been initialized. • TD-scope state has been imported.

- ~~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.~~

8. TD Private Memory Migration

Unreleased Feature: Some of the text in this section is related to Non-Blocking Export, a feature which has not been released yet at the time of writing of this document. Details related to that feature serve as a preview and are subject to change.

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

8.1. Overview

8.1.1. In-Order and Out-of-Order Migration

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.

~~***8.2.1.1. Achieving Memory Migration Security Objectives***~~

~~**8.2.1.1.1. 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.~~

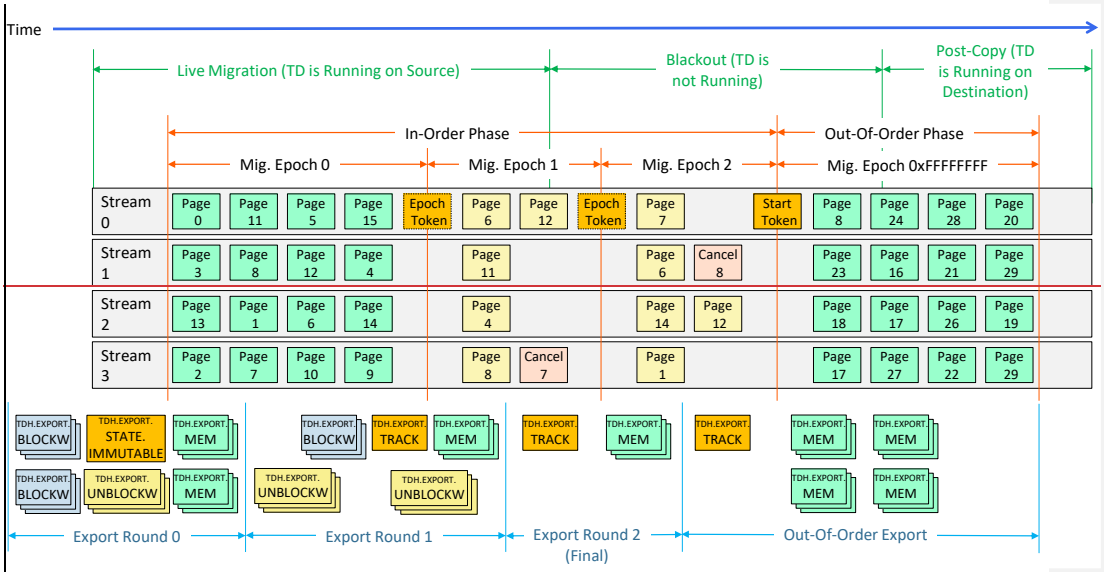
~~These security properties are maintained for a TD during migration with some additional functionality afforded to allow for live migration.~~

- ~~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.~~

~~Section 2: TD Migration Architecture Specification~~



Note: Even if the TDX module supports post-copy with non-blocking export, post-copy is only allowed if TDX Connect is not enabled for the TD.

8.1.2. Write-Blocking Export vs. Non-Blocking Live Export

During live migration, the TDX module tracks exported page modification to enforce consistent memory image migration. The TDX module supports two modes of memory live export:

Write Blocking: Memory pages are blocked for writing before being exported. A guest TD attempt to modify exported memory results in an EPT violation TD exit. The host VMM is expected to unblock the page and later block it again and re-export it. TDX module support for write-blocking based export is enumerated by TDX_FEATURES0.TD_MIGRATION (bit 0), readable by TDH.SYS.RD*.

Non-Blocking: Memory pages are not blocked for writing. If a memory page is modified by the guest TD, DMA or by the TDX module, its SEPT entry's Dirty bit is set. The host VMM is expected to call a TDX module interface function that scans the TD's GPA space for dirty pages, and re-export those pages. TDX module support for non-blocking export is enumerated by TDX_FEATURES0.NON_BLOCKING_EXPORT (bit 41), readable by TDH.SYS.RD*.

The export mode is a TDX module configuration parameter, selected by the host VMM via a setting as part of the TDX module initialization sequence.

- By default, write blocking export is used.
- Non-blocking export can be configured during first-time module initialization (by TDH.SYS.CONFIG).
- On TD-preserving TDX module update, the export mode may be updated from the default write blocking mode to non-blocking mode, if no export session has been used before.

8.2. Conventions: SEPT Entry State Diagrams Color Coding

This chapter contains multiple SEPT entry state diagrams, which use the following color-coding conventions.

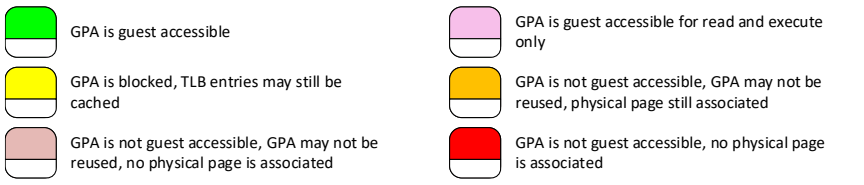


Figure 8.1: ~~Migration Epochs~~Secure EPT Entry State Diagrams Color Conventions

~~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.~~

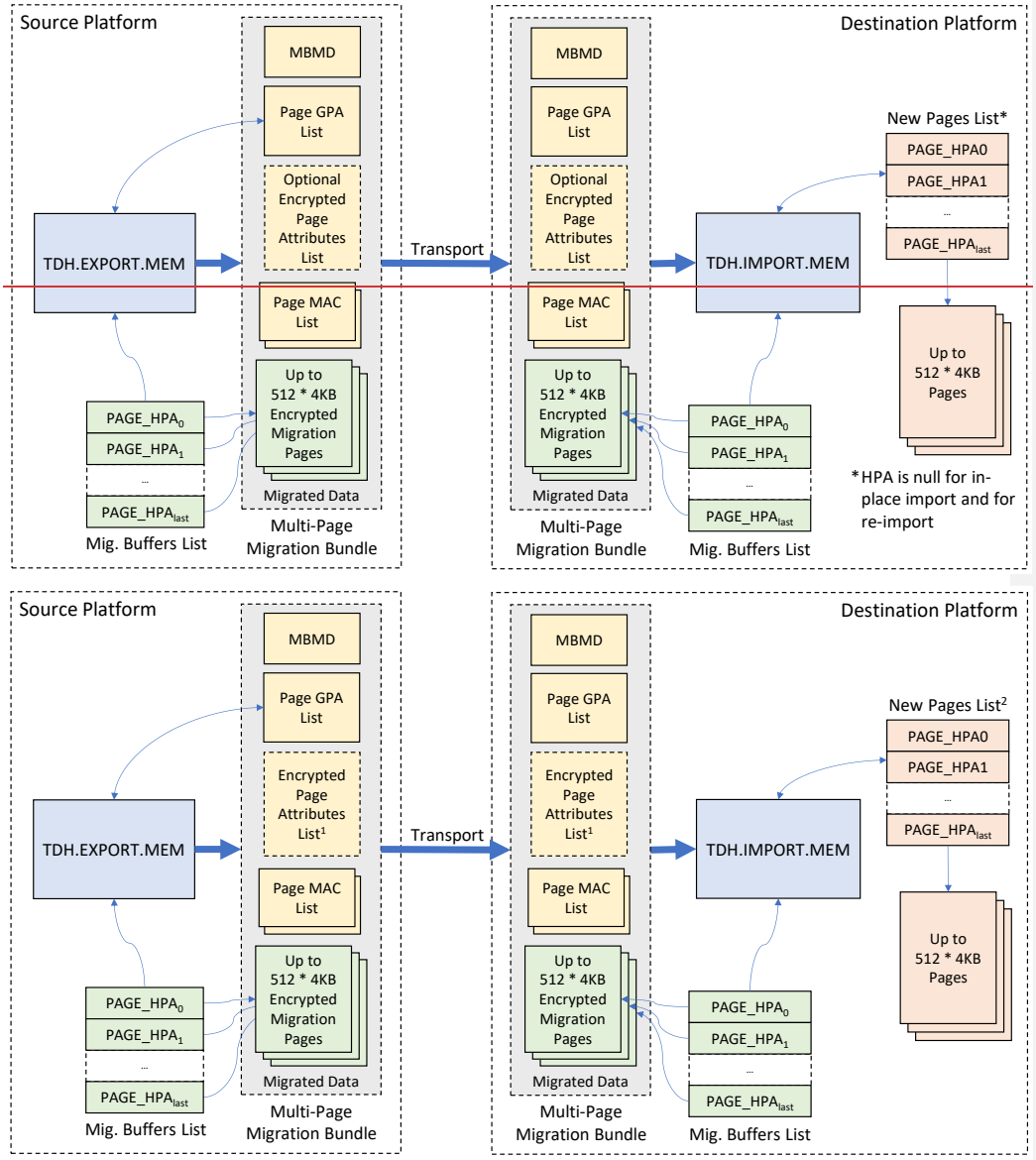
~~8.2.6.1.1.1. Out-Of-Order Phase and Its Usage for Post-Copy~~

~~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~~Bundles

8.3.1. Overview



[1] Page Attributes List is required for partitioned TDs.
[2] Page List HPA is null for in-place import and for re-import.

Figure 8.2: Private Memory Migration

~~Section 2: TD Migration Architecture Specification~~

~~Contrary to~~Unlike 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~~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~~monotonically 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.

8.3.2. GPA List

~~A~~As shown in the example in the diagram above, 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, TDH.MEM.SCAN.COMP and TDH.MEM.SCAN.RANGE.

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

Table 8.1: GPA List Entry Abstract Definition

Field		Usage for Page Migration	Encrypted?	Details
~~Type~~	~~4KB~~ Page Size	4KB	No	
GPA		GPA bits 51:12	No	
State		MAPPED or PENDING	No	
Operation		NOP, MIGRATE, REMIGRATE or CANCEL	No	
Attributes		Page attributes for each L2 VM	Yes	Optional, provided in a separate list (see below)
MAC		Integrity protection for the above fields and for the migrated page content	N/A	Provided in a separate list

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

The GPA list is MAC-protected but is not encrypted. This allows the host VMM on the destination to parse the GPA list in order to prepare for calling TDH.IMPORT.MEM; e.g., build the Secure EPT to map the imported pages.

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

8.3.3. ~~Optional~~ Page Attributes List (Required for Partitioned TDs)

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~~entry is extended with L2 page attributes. These are provided in a separate page attributes~~.~~

Inserted Cells
Inserted Cells
Deleted Cells
Inserted Cells

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

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

. The page attributes list is encrypted and MAC-protected.

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

8.3.4. Private Memory Migration Buffer

~~Migration buffer~~The migration buffer holds the encrypted migrated page content; thus, it is included only if the page metadata indicates a MIGRATE or a REMIGRATE request, and the page is MAPPED. The migration buffer is allocated by the host VMM and resides in shared memory. The migration buffer is encrypted and MAC-protected.

~~8.4. TD Private Memory Export~~

On first time import of a page (MIGRATE request), the host VMM can select in-place import: the migration buffer becomes the TD private page which holds the imported and decrypted content.

8.4. Write-Blocking Based Memory Export

This section describes private memory export using the write-blocking method.

8.4.1. Host VMM Perspective

This section describes write-blocking based export from the host VMM perspective. This is a simplified view; a more detailed view is discussed later.

~~8.4.1.~~8.4.1.1. *Typical Write-Blocking Export ~~Round~~Session*

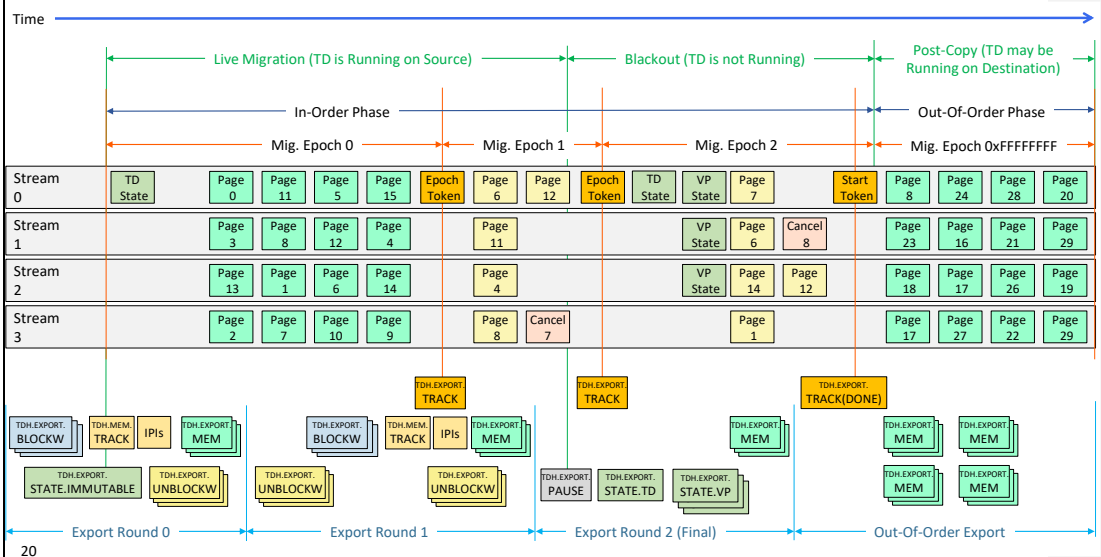


Figure 8.3: Typical Write-Blocking Based Export Session

Typical expected usage divides the export session ~~to~~into **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.

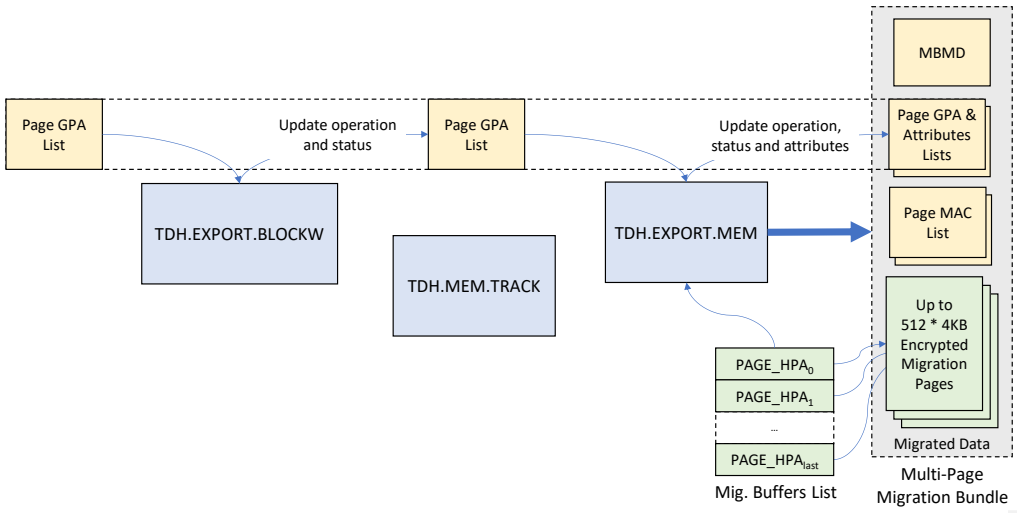
1.2. Invoke TDH.MEM.~~TARCK~~TRACK.
1.3. Issue IPIs to ensure TD re-entry on all VCPUs and TLB invalidation.
2. Start a new **migration epoch** by invoking TDH.EXPORT.TRACK.
3. Invoke TDH.EXPORT.MEM with a list of pages.
   3.1. If a page is being exported, mark its list entry as MIGRATE.
   3.2. If a page has been exported before but ~~need~~needs 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.

~~***8.4.2.1.1.1.1. 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.~~



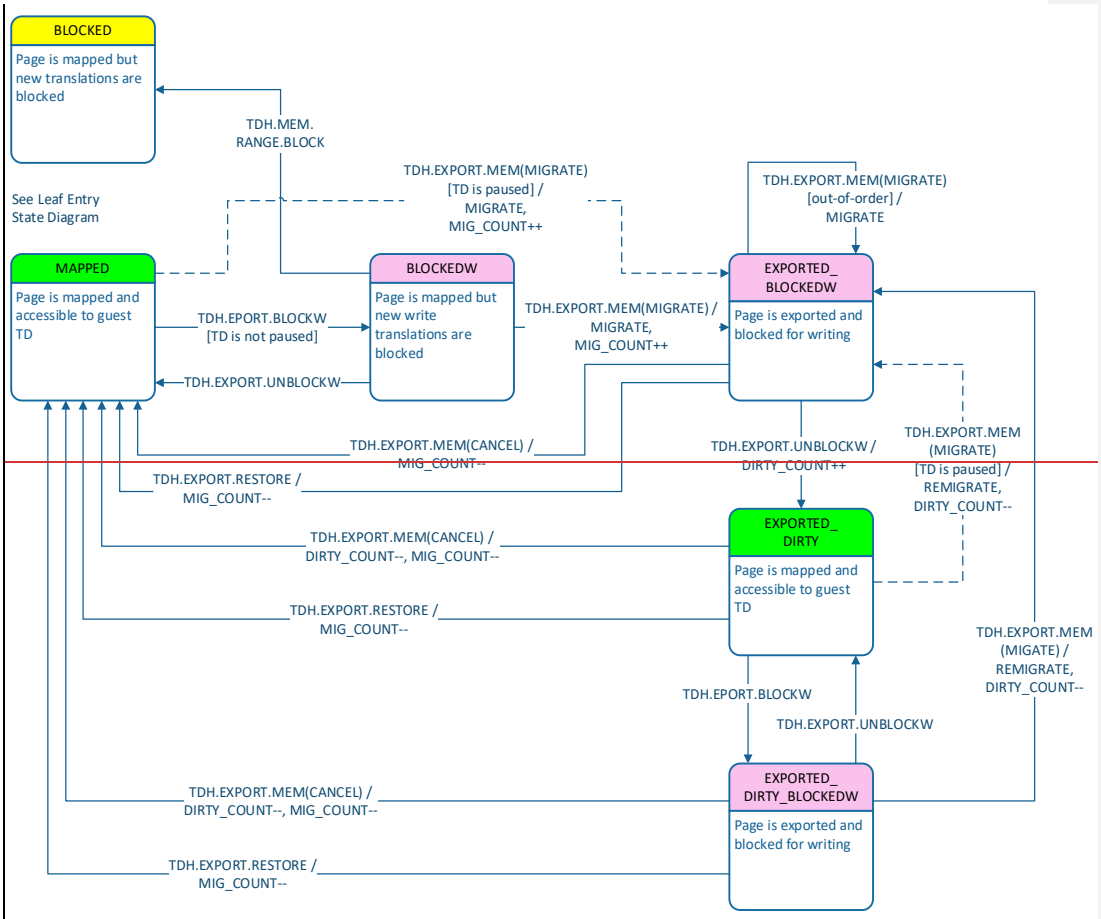
~~**Figure 8.3: Typical Memory Export Round and the GPA List**~~

~~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.~~

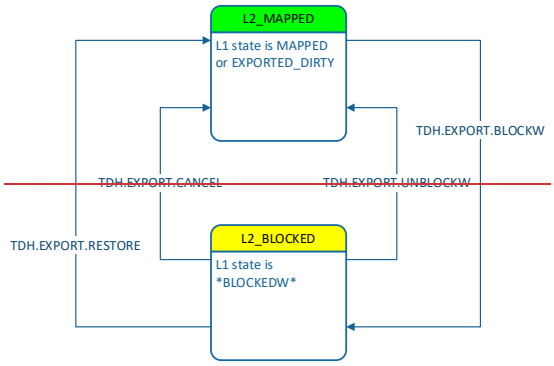
~~Section 2: TD Migration Architecture Specification~~



~~**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.~~8.4.1.2. *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~~tracked by the TDX module. 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~~exist 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.~~8.4.1.3. *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 8.8 below.

~~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.~~8.4.1.4. *Unblocking for Write, Tracking Dirty Pages and Re-Exporting*

~~8.4.7.1.~~8.4.1.4.1. 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.~~8.4.1.4.2. 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. (e.g., the page is BLOCKED).

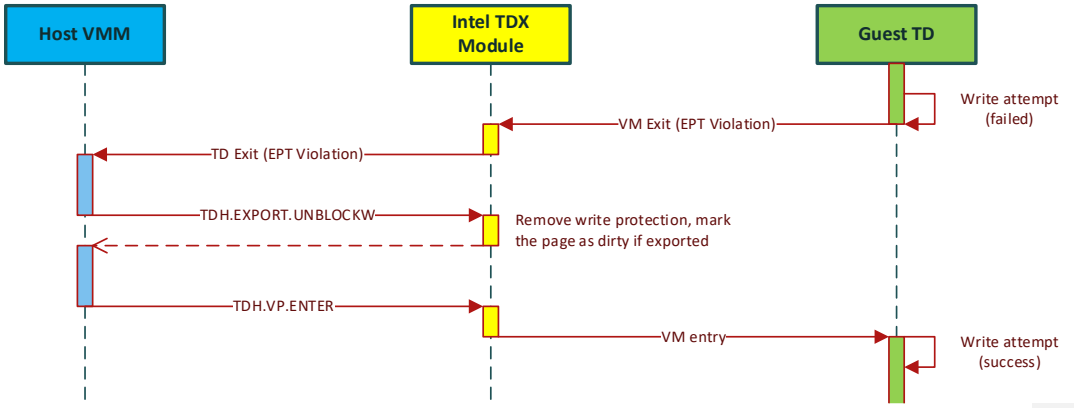


Figure 8.4: Typical Sequence for Unblocking a Page on Guest TD Write Attempt (Write-Blocking Export)

To enable access to the page, the host VMM is expected to execute TDH.EXPORT.UNBLOCKW and then resume the TD VCPU by TDH.VP.ENTER.

- 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 to MAPPED 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.~~below.

~~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.~~

8.4.1.5. *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 (see later) 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.

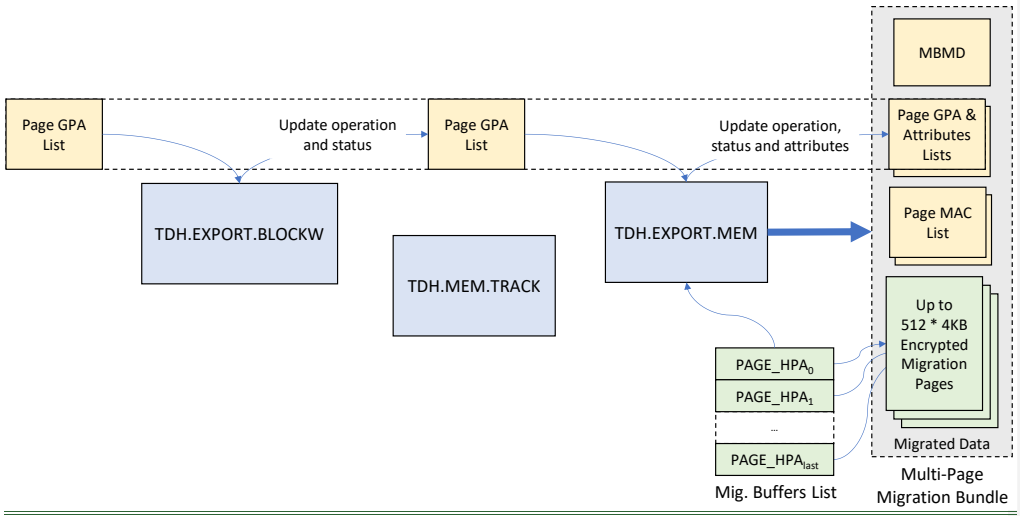


Figure 8.5: Typical Write-Blocking Based Memory Export Round and the GPA List

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

~~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.~~

~~**8.4.8.1.1.1. *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.~~

~~Section 2: TD Migration Architecture Specification~~

~~8.4.10.~~8.4.1.6. *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.

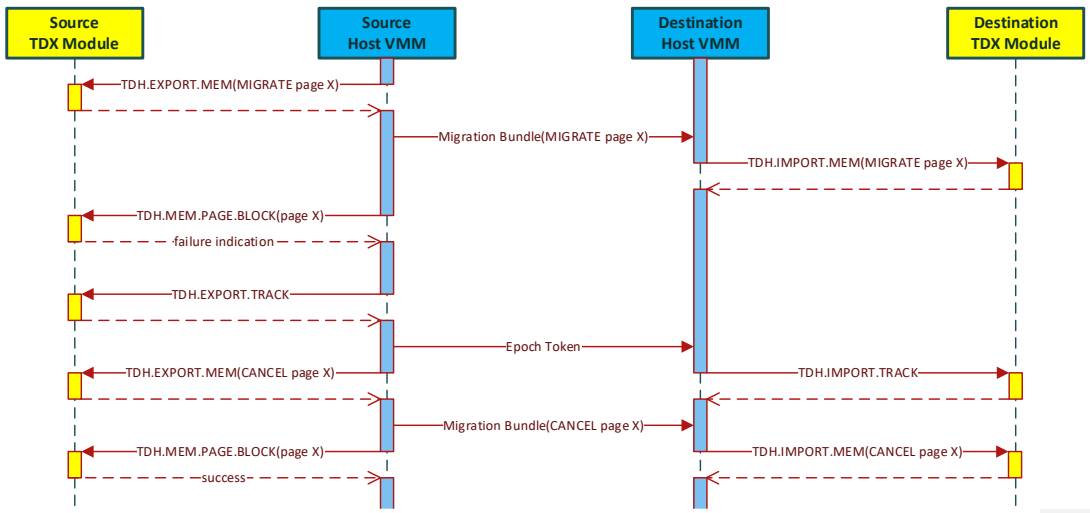
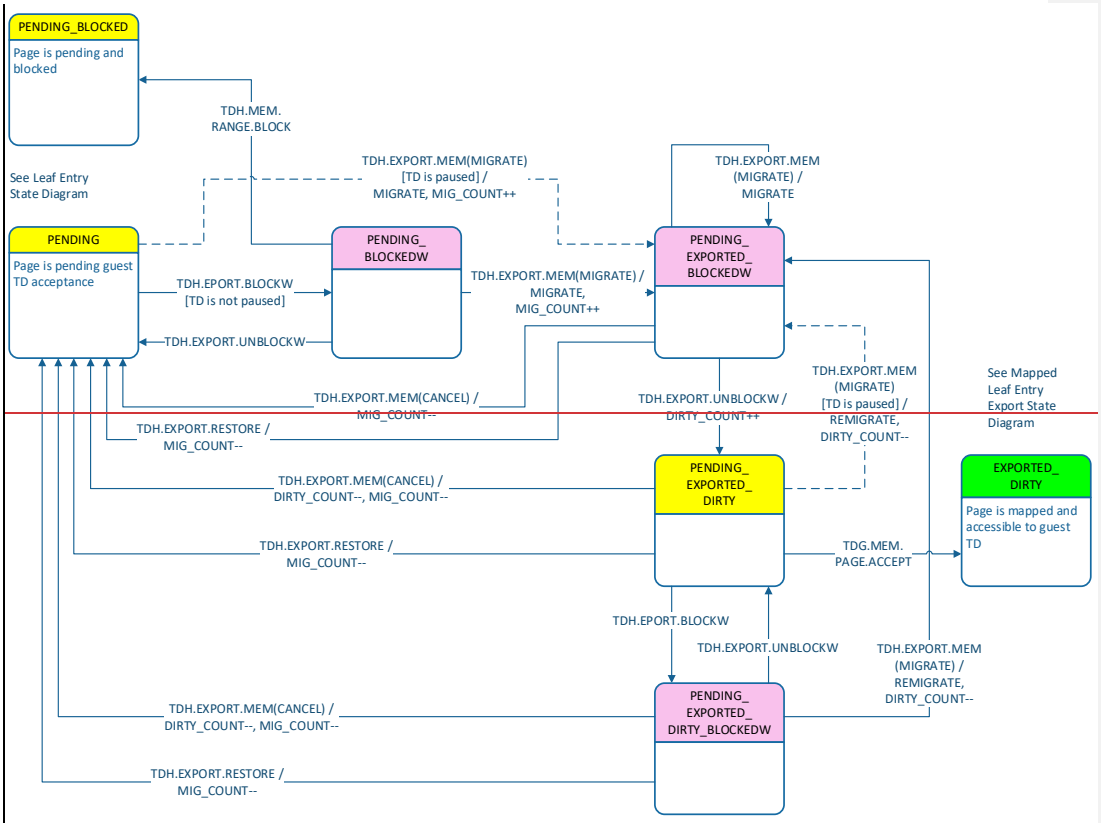


Figure 8.6: Typical Sequence for Cancelling a Page Export (Write-Blocking Export)

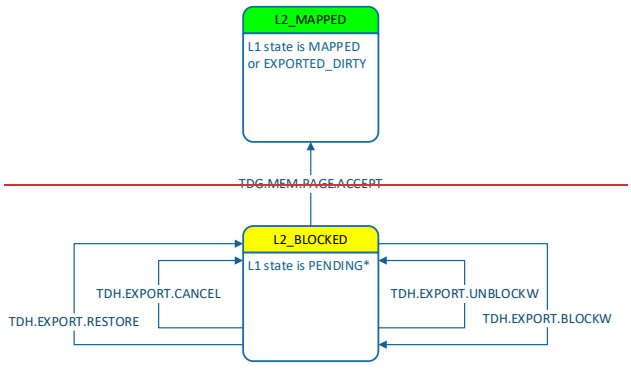
~~8.4.11.1.1.1. *Exporting Pending Pages*~~

8.4.1.7. *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~~a PENDING page by TDG.MEM.PAGE.ACCEPT at any time. ~~Thus, TDH.EXPORT.MEM~~If supported, the guest may ~~export~~release a ~~pending~~MAPPED page~~. This is indicated~~ by ~~the GPA list entry~~TDG.MEM.PAGE.RELEASE, converting it to a PENDING page. The page content of a PENDING page is not exported, 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 8.4.1.4 above.

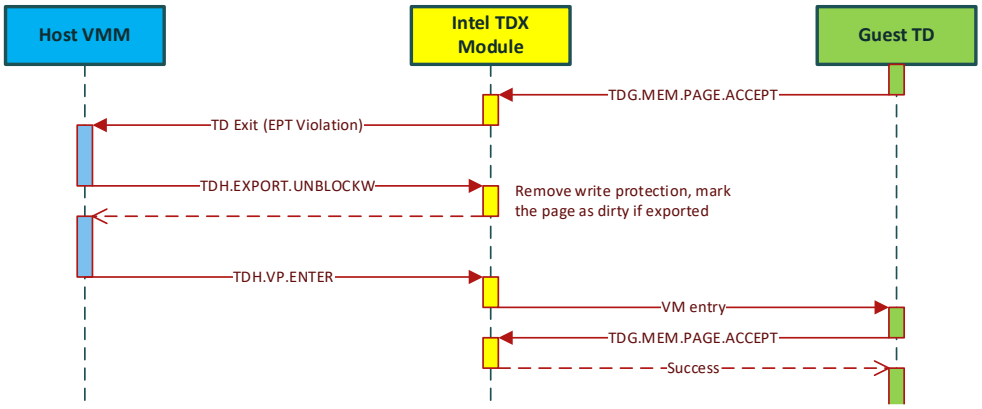


Figure 8.7: Typical Sequence for Unblocking a PENDING Page on TDG.MEM.PAGE.ACCEPT Attempt (Write-Blocking Export)

8.4.1.8. *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 to 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.

~~8.4.12.~~8.4.1.9. *SEPT Cleanup after Export Abort*

~~Following~~After 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~~ The TDX module holds a TDCS~~:~~

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

exported pages that require cleanup. This field is readable by the host VMM, using TDH.MNG.RD. The counter is initialized to 0. To start a new migration session, its value must be 0.

8.4.2. Details of Write-Blocking Based Export

This section provides a detailed view of write-blocking based export. Details may be of interest to host VMM programmers who require a deeper understanding of TD Migration.

8.4.2.1. *Details: L1 SEPT Leaf Entry Partial State Diagram for Mapped Page Export*

Figure 8.8 below shows a partial SEPT entry state diagram for exporting mapped pages. The following sections describe the details. The color-coding convention used in this diagram is described in 8.2.

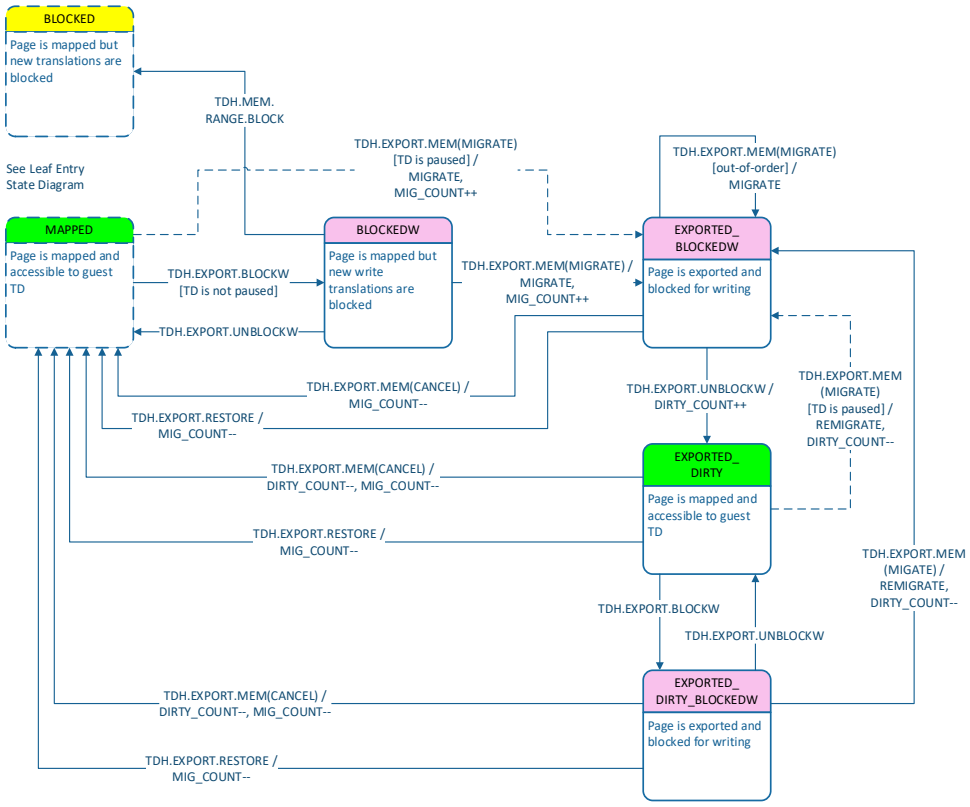


Figure 8.8: Partial L1 SEPT Leaf Entry State Diagram for Mapped Page Write-Blocking Based Export

8.4.2.2. *Details: L1 SEPT Leaf Entry Partial State Diagram for Pending Page Export*

The figure below shown the partial state diagram for PENDING page export.

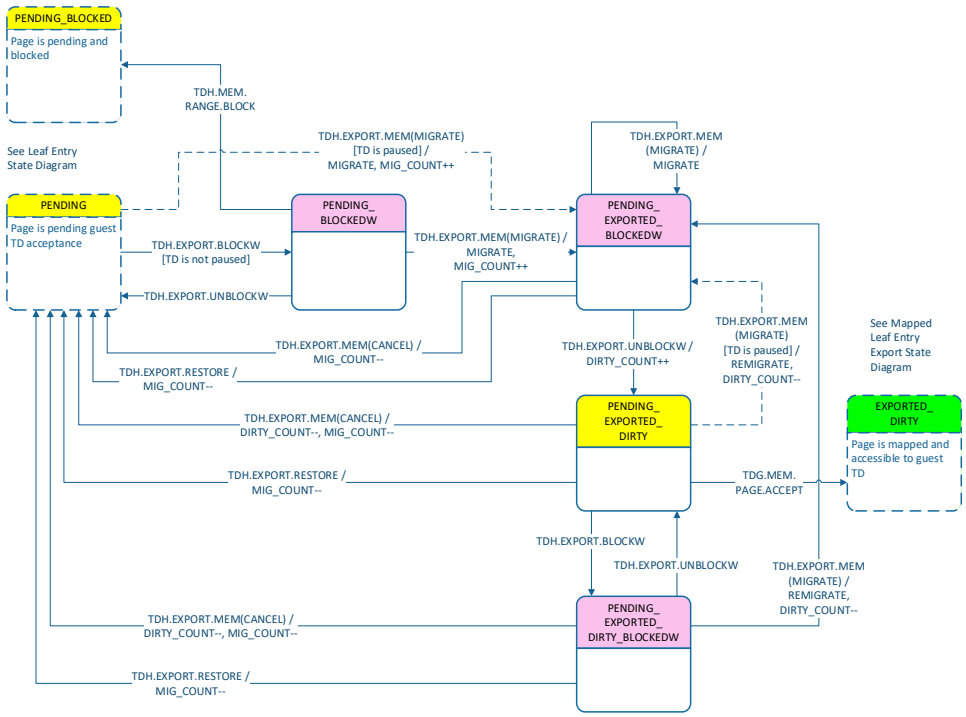


Figure 8.9: Partial L1 SEPT Leaf Entry State Diagram for Pending Page Write-Blocking Based Export

8.4.2.3. *Details: TDCS.DIRTY_COUNT: TD-Scope Dirty Page Counter*

TDCS.DIRTY_COUNT is a 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, is 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 have 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 the destination – see the [TDX Module ABI Spec] for details.

8.5. *Non-Blocking Memory Export*

Unreleased Feature:	Non-Blocking Export is a feature which has not been released yet at the time of writing of this document. Details related to that feature serve as a preview and are subject to change.

8.5.1. Host VMM Perspective

This section describes non-blocking export from the host VMM perspective. This is a simplified view; a more detailed view is discussed later.

8.5.1.1. *EPT Access and Dirty Bits Background*

Intel SDM, Vol. 3, 30.3.5 Accessed and Dirty Flags for EPT

The Dirty Bit

Non-blocking live export relies on detecting memory changes using the Secure EPT entry's Dirty bit (9). This bit is set by the h/w (CPU or IOMMU) in the leaf EPT entry for a certain GPA when it performs an EPT walk for translating that GPA for a write operation. The Dirty bit is sticky – the h/w may only set it to 1 but never clear it. The Dirty bit is only cleared by s/w – in case of Secure EPT, by the TDX module.

Implication of Address Translation Caching

The CPU and IOMMU cache page address translations and paging structures. An EPT entry's Dirty bit is only set during an EPT walk, when there is a need to do address translation, and the applicable entries are not cached. This means that even if the TDX module clears the Dirty bit of an SEPT entry, the h/w may not be aware of this since it holds a cached entry; it will not set the Dirty bit if there's a new write access. Since page modification detection relies on the correct value of the Dirty bit, there is a need to do a TLB shootdown, as described in the following sections. TLB shootdown is tracked by the TDX module as a prerequisite to page export, to help ensure secure operation.

8.5.1.2. *Memory Export Concept: Scan and Export*

Non-blocking export is based on the concept of scan and export. Export typically consists of multiple rounds, with the following steps per round:

1. Scan the TD's GPA space for memory export candidates.
2. Do a TLB shootdown.
3. Based on the scan results, prepare a list of pages and export them.

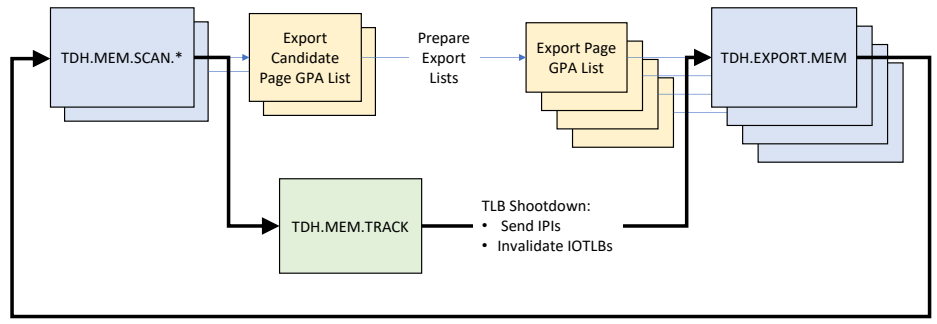


Figure 8.10: Memory Export Round Concept: Scan and Export

The host VMM is **not required** to track the state of the TD private pages. It can rely on the TDX Module memory scan functions to provide the required information.

8.5.1.3. *Conceptual, Simplified Page State Diagram*

Although the host VMM is not required to track the TD private page state, it is worthwhile to understand how the TDX module tracks page state using the Secure EPT state. This section provides a conceptual view of the page state machine, which serves as a simplified overview of the export operation from the host VMM's perspective.

Note: The actual SEPT entry state machine is more complex. It is described in the following sections and in the [Base Spec].

The following table lists the conceptual state machine's states and their handling by the memory scan functions (which are described later):

Table 8.2: Conceptual, Simplified Page States for Dirty Bit Based Export

State	Description	TDH.MEM.SCAN.*
NOT_EXPORTED	Pages that either were not exported during the current migration session, or that were exported but their export was later cancelled.	Page GPA is reported by TDH.MEM.SCAN.*, to let the host VMM know it needs to be exported.
EXPORTED	Pages that were exported. Pages may have their Dirty bit set due to content modifications by the CPU, DMA or the TDX module, or due to attributes modifications by the guest TD (e.g., TDG.MEM.PAGE.ACCEPT or TDG.MEM.PAGE.ATTR.WR).	If the page's SEPT entry's Dirty bit is set, TDH.MEM.SCAN.* clears the Dirty bit, sets the page state to EXPORTED_MODIFIED and reports the page GPA to let the host VMM know it needs to be re-exported.
EXPORTED_MODIFIED	Pages that were exported, and later either scanned and found to require re-export since their Dirty bit was set, or their attributes were changed by some TDX module memory management function.	TDH.MEM.SCAN.* reports the page GPA to let the host VMM know it needs to be re-exported.
EXPORT_CANCEL_REQUIRED	Pages that were exported but their export must be cancelled, due to memory management operations (such as TDH.MEM.RANGE.BLOCK) by the host VMM. **Note:** This conceptual state represents multiple states, shown in the detailed diagram later; those are required since different memory management operations are handled differently.	TDH.MEM.SCAN.* reports the page GPA to let the host VMM know it needs to be re-exported.
BLOCKED	When a page that was in an EXPORT_CANCEL_REQUIRED state due to blocking is exported (as a CANCEL operation), the page state becomes the normal BLOCKED state. **Note:** This conceptual state represents multiple states, shown in the detailed diagram later; those are required since different memory management operations are handled differently.	Page GPA is not reported by TDH.MEM.SCAN.RANGE(DSCAN) since it is not exported.

Note:
This is a conceptual diagram. State names are not actual SEPT state names. Some transitions are simplified or not shown.

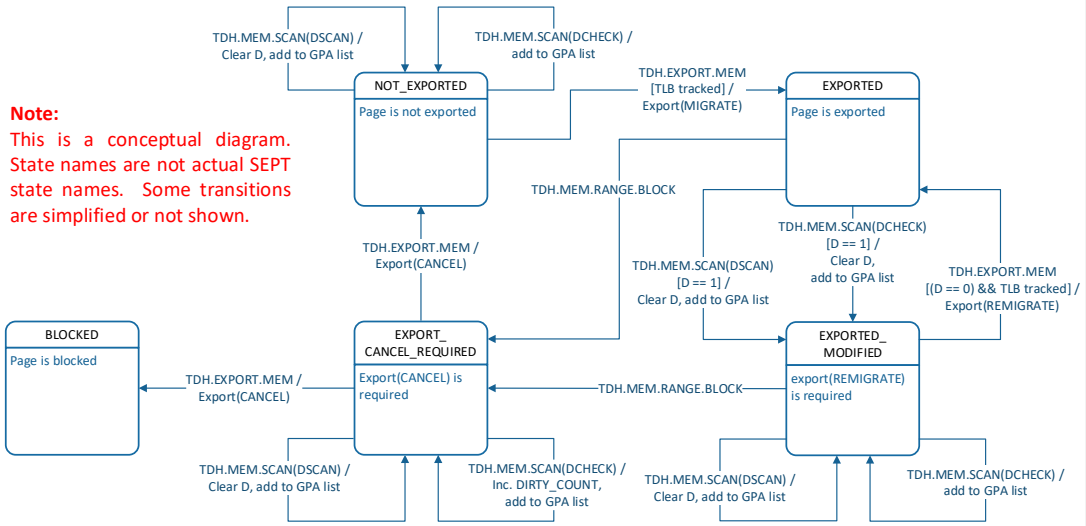


Figure 8.11: Conceptual Page State Diagram for Non-Blocking Export

8.5.1.4. *Scanning for Candidate Pages to Export or Re-export*

8.5.1.4.1. Overview

During the live export phase, memory pages may be modified asynchronously to the host VMM's operations, by the running TD or its bound TDIs. **The host VMM does not need to know the SEPT entry states**. It uses the export candidates list provided by the following interface functions:

TDH.MEM.SCAN.RANGE(DSCAN): This function is used iteratively by the host VMM during the LIVE EXPORT phase of TD migration to identify pages in the specified GPA range needing export. Identification is based on the SEPT entry's page state and the Dirty bit. The function returns a list of pages that the host VMM can use to prepare export requests (as an input to TDH.EXPORT.MEM).

TDH.MEM.SCAN.COMP(DCHECK): This function is used by the host VMM during the export blackout period, after the TD is paused and all TDIs are unbound. The function performs a comprehensive scan of the TD's GPA address space and identifies the remaining pages needing export and returns a page list. It ensures migration consistency by requiring that the whole GPA range will be scanned and maintaining a counter of pages that need to be exported. TDH.EXPORT.TRACK(DONE) then checks that those pages have indeed been exported, as a precondition for generating a start token.

Although memory export only supports 4KB page mapping, the scanning functions return information for 4KB, 2MB and 1GB page mapping sizes. The host VMM is responsible for demoting 2MB and 1GB pages before calling TDH.EXPORT.MEM.

The host VMM is expected to maintain an **export GPA list**:

- A GPA should be added to the list when it is reported by TDH.MEM.SCAN.*.
- A GPA should be removed from the list when it is successfully exported by TDH.EXPORT.MEM.

TDH.MEM.SCAN.* returns a list of export candidates regardless of whether pages in that list have been returned by a previous call to TDH.MEM.SCAN.*. If page X has been reported as an export candidate, and has not yet been exported, it may be reported again.

8.5.1.4.2. Using the Memory Scanning Functions

To help reduce the scan time, especially during the export blackout period, TDH.MEM.SCAN.* can be called concurrently on multiple LPs.

A comprehensive scan of the whole GPA range is done as follows:

- The host VMM needs to configure the comprehensive scan using TDH.MEM.SCAN.CONFIG. The host VMM can divide the TD's GPA address space into ranges, typically to optimize the scan for NUMA configurations. The configuration is not migrated with the TD; it needs to be redone if the TD is to be migrated again.
- The host VMM can call TDH.MEM.SCAN.COMP on multiple threads, specifying the pre-configured GPA range to scan by each instance (multiple instances can concurrently scan the same range).
- If the scan fails (e.g., there are blocked pages), the host VMM can reset the comprehensive scan using TDH.MEM.SCAN.RESET and then retry the scan.

For further details, see the [TDX Module ABI Spec].

8.5.1.5. *Typical Non-Blocking Export Session*

Typical expected usage divides the export session into **export rounds** (or **passes**). The diagram below shows a typical export session, which is composed of several **live migration export rounds** and a **final export round** during the blackout period, when the TD is not running. It also shows post-copy (which is not applicable if the TD is configured for TDX Connect).

The diagram below shows 4 migration streams and the usage of migration epochs to synchronize them. Migration epochs are defined in 6.1.4.

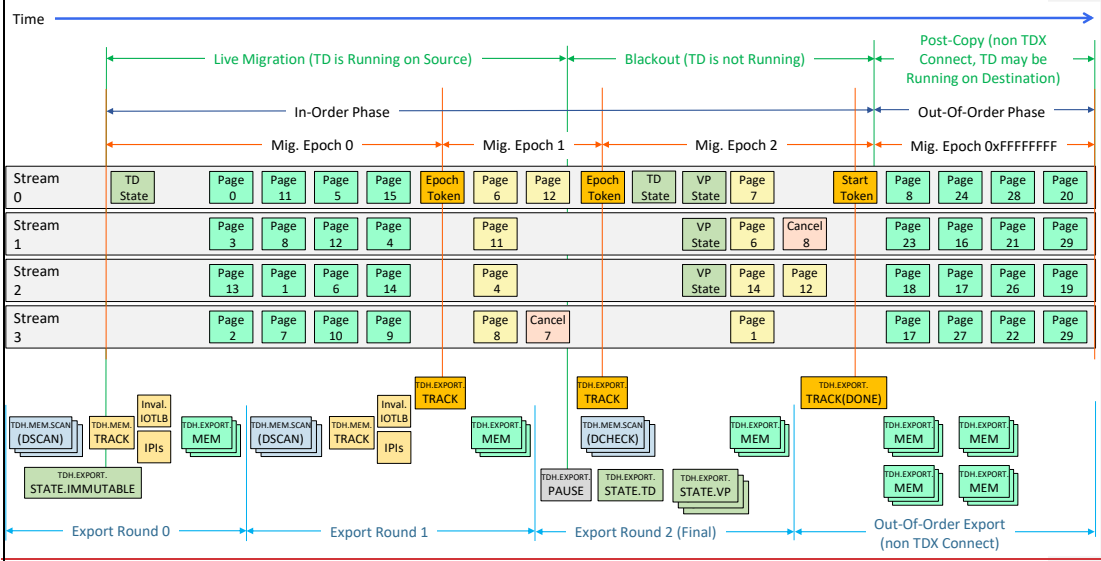


Figure 8.12: Typical Non-Blocking Export Session

8.5.1.5.1. Typical Non-Blocking Live Export Round

Note: Do not confuse TDH.MEM.TRACK (which is used for TLB tracking) with TDH.EXPORT.TRACK (which rendezvous all migration streams).

An export round during the live migration phase of the export session typically has the following steps:

1. Prepare a list of pages to be exported by calling TDH.MEM.SCAN.RANGE(DSCAN) one or more times.
   1.1. TDH.MEM.SCAN.RANGE(DSCAN) can be called concurrently on multiple LPs.
2. Do the TDX TLB (and IOTLB, if the TD is enabled for TDX Connect) shootdown sequence. This is required to ensure that, after the following export, the CPU or IOMMU will indeed set the Dirty bit if a page is written.
   2.1. Call TDH.MEM.TRACK.
   2.2. Issue IPIs (Inter-Processor Interrupts) to all RLP (Remote Logical Processors) on which any of the TD's VCPUs is currently scheduled. This causes TD re-entry and TLB (and IOTLB, if the TD has bound TDI) invalidation on all VCPUs.

3. Start a new **migration epoch** by invoking TDH.EXPORT.TRACK. This is required for the TDX module on the destination to ensure proper ordering on import.
4. Call TDH.EXPORT.MEM with a list of pages based on the export candidates list reported by TDH.MEM.SCAN.RANGE(DSCAN) above.
   4.1. The host VMM may choose not to include pages reported as blocked, since this is an interim state used as preparation to some memory management operation.
   4.2. TDH.EXPORT.MEM determines the required export operation (MIGRATE, REMIGRATE or CANCEL) based on each page state.
   4.3. TDH.EXPORT.MEM checks that the TLB shootdown above has indeed been done, using the TDX TLB tracking mechanism (see the [TDX Module Base Spec] for details).
   4.4. TDH.EXPORT.MEM can be called concurrently on multiple LPs, each exporting to a different migration stream.

Concurrency and Order during Live Export

- There is no strict ordering requirement between steps 2 and 3, as long as they precede step 4.
- TDH.MEM.SCAN.RANGE(DSCAN) can run concurrently with TDH.EXPORT.MEM, as long as different pages are processed. E.g., it is possible to scan a certain GPA range while exporting memory of a different GPA range.
- If TDH.MEM.SCAN.RANGE(DSCAN) encounters a busy SEPT entry due to a conflict with some other function, it skips that entry. The page will be processed by the next TDH.MEM.SCAN.RANGE(DSCAN) or TDH.MEM.SCAN.COMP(DCHECK) . For details, see the [ABI Spec].
- If TDH.EXPORT.MEM encounters a busy SEPT entry due to a conflict with some other function, it skips that entry. It writes the status into the GPA list; the host VMM may read that list and call TDH.EXPORT.MEM to export pages that have not been exported. For details, see the [ABI Spec].

8.5.1.5.2. Typical Final Export Round

The final export round ensures that the full up-to-date memory image (and non-memory state) is exported. It typically consists of the following steps:

1. Pause the TD using TDH.EXPORT.PAUSE. This requires no pages to be blocked and, if the TD is enabled for TDX Connects, no TDIs to be attached.
2. Export the TD's non-memory immutable state using TDH.EXPORT.STATE.TD and TDH.EXPORT.STATE.VP.
3. Prepare a list remaining of pages to be exported in the final round by calling TDH.MEM.SCAN.COMP(DCHECK) one or more times until the whole GPA range has been scanned.
   3.1. TDH.MEM.SCAN.CONFIG should be called prior to TDH.MEM.SCAN.COMP to configure the GPA ranges for scanning.
   3.2. TDH.MEM.SCAN.COMP(DCHECK) can be called concurrently on multiple LPs.
4. There is no need to do a TLB shootdown since the TD is paused and no TDI is bound to the TD.
5. Start a new **migration epoch** by invoking TDH.EXPORT.TRACK. This is required for the TDX module on the destination to ensure proper ordering on import.
6. Call TDH.EXPORT.MEM with a list of pages. The TDX module determines the required export operation (MIGRATE, REMIGRATE or CANCEL) based on each page state.
   6.1. All the pages detected by TDH.MEM.SCAN.COMP(DCHECK) as requiring re-export must be exported.
   6.2. If post-copy is supported (for a TD where TDX Connect is not enabled), pages identified by TDH.MEM.SCAN.COMP(DCHECK) as never having been exported may be exported later during the out-of-order phase.
   6.3. TDH.EXPORT.MEM may be called concurrently on multiple LPs, each exporting to a different migration stream.
7. End the in-order export phase by calling TDH.EXPORT.TRACK(DONE).

Concurrency and Order during Final Export

- TDH.MEM.SCAN.COMP(DCHECK) can run concurrently with TDH.EXPORT.MEM, as long as different pages are processed. E.g., it is possible to scan a certain GPA range while exporting memory of a different GPA range.
- Unlike TDH.MEM.SCAN.RANGE(DSCAN), if TDH.MEM.SCAN.COMP(DCHECK) encounters a busy SEPT entry due to a conflict with some other function, it returns to the host VMM indicating an interrupted operation. The host VMM is expected to resume TDH.MEM.SCAN.COMP(DCHECK) . For details, see the [ABI Spec].

8.5.1.6. *Interaction with Memory Management Operations*

8.5.1.6.1. Memory Management Restrictions on Pages that Have Been Exported

Exported Page Blocking and Removal

A page that has been exported can still be blocked (TDH.MEM.RANGE.BLOCK) and/or removed (TDH.MEM.PAGE.REMOVE) by the host VMM, subject to other restrictions (e.g., no attached TDIs for TDs configured for TDX Connect).

Synchronization with the destination is done by exporting a CANCEL operation for this page, so that the destination can remove it. A blocked or removed page is put in one of several special states (together called EXPORT_CANCEL_REQUIRED in the conceptual state diagram above). Pages in those states are detected by TDH.MEM.SCAN.RANGE(DSCAN) and TDH.MEM.SCAN.COMP(DCHECK) so that the host VMM can include them in the GPA list for TDH.EXPORT.MEM. Once TDH.EXPORT.MEM has exported a CANCEL operation for such a page, the page state is set to one of the NON_EXPORTED states.

Exported Page Promotion or Demotion

A page that has been exported can't be promoted (TDH.MEM.PAGE.PROMOTE). Since an exported page mapping size is 4KB, it can't be demoted (TDH.MEM.PAGE.DEMOTE) either.

Note: Even without migration, the host VMM should always be prepared for a Promote operation to fail. E.g., L1 may set the attributes of some of the small pages to be merged differently than the other small pages, preventing promotion.

8.5.1.6.2. Memory Management Restrictions During the Export Blackout Period

The following memory management restriction applies during the PAUSED_EXPORT state, which begins when the TD is paused by TDH.EXPORT.PAUSE and ends when a start token is generated by TDH.EXPORT.TRACK(DONE). These restrictions exist to facilitate export completeness tracking (for details, see 8.5.2.7 below).

No Blocked Pages

No page may be blocked while in the PAUSED_EXPORT state. This condition is checked by TDH.EXPORT.PAUSE and TDH.MEM.SCAN.COMP(DCHECK). TDH.MEM.RANGE.BLOCK and TDH.MEM.RANGE.UNBLOCK are not allowed.

Note: TDH.MEM.PAGE.DEMOTE and TDH.MEM.PAGE.RELOCATE are allowed without blocking in the PAUSED_EXPORT state.

Other Restrictions

- Page addition and removal by TDH.MEM.PAGE.AUG and TDH.MEM.PAGE.REMOVE is not allowed while in the PAUSED_EXPORT state. Note that TDH.MEM.PAGE.ADD is also not allowed since the TD build has been, by definition, finalized.
- SEPT tree changes by TDH.MEM.PAGE.PROMOTE, TDH.MEM.SEPT.ADD, and TDH.MEM.SEPT.REMOVE are not allowed while in the PAUSED_EXPORT state.

8.5.1.7. *Exporting Pending Pages*

~~increments~~The host VMM is not directly aware of whether a page is in a PENDING state or not:

- The guest TD may accept a PENDING page by TDG.MEM.PAGE.ACCEPT at any time, converting it to a MAPPED page.
- If supported, the guest TD may release a MAPPED page by TDG.MEM.PAGE.RELEASE, converting it to a PENDING page.

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.

8.5.1.8. *SEPT Cleanup after Export Abort*

After 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 that are not allowed for

migrated pages (such as TDH.MEM.PAGE.PROMOTE) are attempted on them, and/or before a new export session is attempted.

Cleanup is done either by TDH.EXPORT.RESTORE, which receives a list of up to 512 pages, or (if supported) by TDH.MEM.SCAN.RANGE(EXPORT_RESTORE), which scans a GPA range.

The TDX module holds a TDCS field called MIG_COUNT ~~for exported pages, and decrements~~, which counts the number of exported pages that require cleanup. This field is readable by the host VMM, using TDH.MNG.RD. The counter is initialized to 0. To start a new migration session, its value must be 0.

8.5.2. Details of Non-Blocking Export

This section provides a detailed view of non-blocking export. Details may be of interest to host VMM programmers who require a deeper understanding of TD Migration.

8.5.2.1. *Details: L1 SEPT Leaf Entry Partial State Diagram for Mapped Page Export*

To support non-blocking export of Mapped pages, the following SEPT states are added. Their main goal is to help keep the destination TD memory image in sync with the source TD memory image. The next section describes the states added for Pending pages.

Table 8.3: L1 SEPT Entry States Related to Mapped Page Export

State Name	Description
EXPORTED	The page has been exported. This state indicates that any change in the page content or attributes would require a re-export.
EXPORTED_MODIFIED	The page was exported and later either identified as dirty based on the Dirty bit (which was atomically cleared by the same operation that checked its value) or some memory management operation changed the page attributes or state. This state indicates that the page must be re-exported as a REMIGRATE operation.
EXPORTED_BLOCKED	The page was exported and later blocked by TDH.MEM.RANGE.BLOCK. This state indicates that the page must be re-exported as a CANCEL operation.
EXPORTED_REMOVED	The page was exported and later removed by TDH.MEM.PAGE.REMOVE. This state indicates that the page should be re-exported, as a CANCEL operation.
EXPORTED_REMOVE_IN_PROGRESS	The page was exported and later partially removed by TDH.MEM.PAGE.REMOVE. This state indicates that the page should be re-exported as a CANCEL operation. This state is only applicable for client platforms, which use ACT-based memory protection.

A partial SEPT entry state diagram for exporting Mapped pages using the Dirty bit method is shown below. The following sections describe the details. The color-coding convention used in this diagram is described in 8.2.

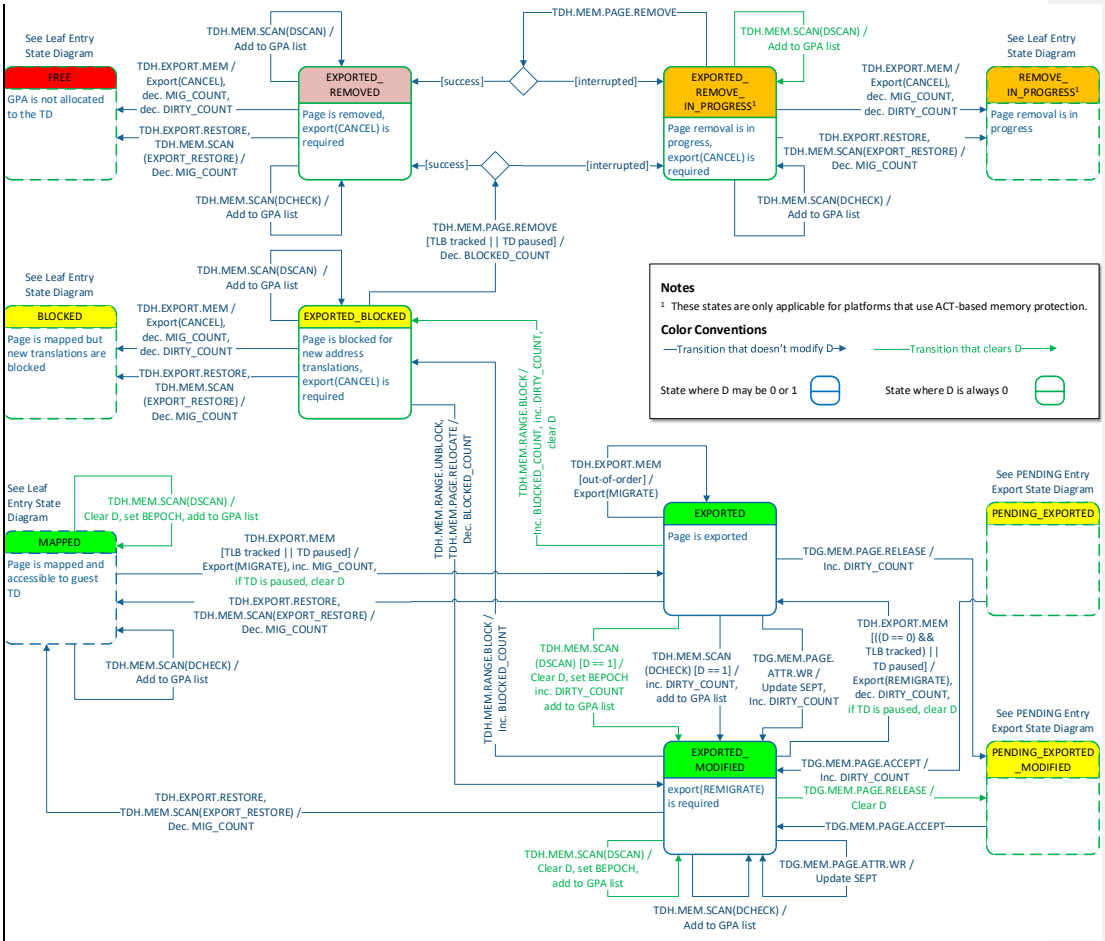


Figure 8.13: Partial L1 SEPT Leaf Entry State Diagram for MAPPED Page Non-Blocking Export

8.5.2.2. *Details: L1 SEPT Leaf Entry Partial State Diagram for Pending Page Export*

The state diagram for PENDING page export is very similar to the one above. Note that PENDING pages are considered non-present from the h/w perspective, so the CPU and DMA do not normally set the Dirty bit. An exception to this is pages converted to PENDING by TDG.MEM.PAGE.RELEASE where the h/w may hold a TLB translation.

Table 8.4: L1 SEPT Entry State Related to Pending Page Export

State Name	Description
PENDING_EXPORTED	The page has been exported. This state indicates that any change in the page attributes would require a re-export.
PENDING_EXPORTED_MODIFIED	The page was exported and later identified as dirty based on the Dirty bit (which was atomically cleared by the same operation that checked its value). This state indicates that the page must be re-exported as a REMIGRATE operation.

State Name	Description
PENDING_EXPORTED_BLOCKED	The page was exported and later blocked by TDH.MEM.RANGE.BLOCK. This state indicates that the page must be re-exported as a CANCEL operation.

A partial SEPT entry state diagram for exporting Pending pages using the Dirty bit method is shown below. The following sections describe the details. The color-coding convention used in this diagram is described in 8.2.

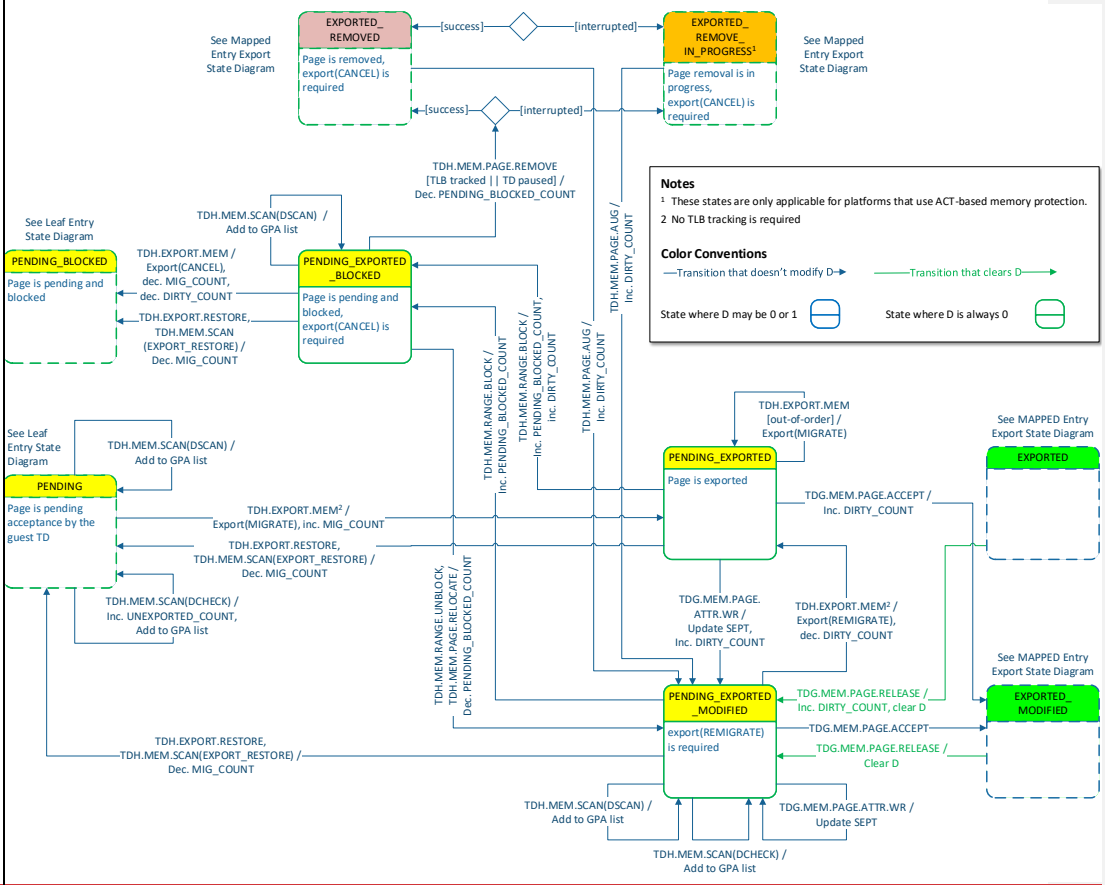


Figure 8.14: Partial L1 SEPT Leaf Entry State Diagram for PENDING Page Non-Blocking Export

8.5.2.3. *Details: TD Partitioning Considerations for Dirty Bit Operations*

With TD partitioning, in addition to the main SEPT tree used by L1, each L2 VM has its own L2 SEPT tree. TD private memory writes by an L2 VM, by devices associated with that L2 VM or by TDX module flows that run in the context of that L2 VM set the Dirty bits in L2 SEPT entries.

Thus, when the logical Dirty bit is discussed, the actual operation may involve the L1 SEPT entry's Dirty bit and/or one or more of the L2 SEPT entries' Dirty bits. The following table describes the various operations done on the Dirty bit and their actual meaning for partitioned TDs.

Table 8.5: Logical Dirty Bit vs. Actual L1 and L2 Dirty Bits

Logical Operation	Actual Operation
Is the Dirty bit 0?	Logical Dirty is 0 if the Dirty bit is 0 in the L1 SEPT entry and all L2 SEPT entries.
Is the Dirty bit 1?	Logical Dirty is 1 if the Dirty bit is 1 in the L1 SEPT entry or any of the L2 SEPT entries.
Atomically clear the Dirty bit and return its previous value.	Atomically test and clear the Dirty bit in L1 and each L2 SEPT entries. Return true if the Dirty bit was 1 in any of the SEPT entries.
Clear the Dirty bit.	Clear the Dirty bit in L1 and each L2 SEPT entries.
Set the Access and Dirty bits as a result of reading or writing to the TD private page by a TDX module flow operating in the context of the TD, e.g., TDG.MR.REPORT.	Set the A and D bits as part of the soft SEPT walk done by the TDX module.
Set the Access and Dirty bits as a result of translating GPAs to HPAs and caching the HPA values to be used by the CPU (as VMCS fields) and by the TDX module, e.g., as part of L1→L2 entry.	Set the A and D bits as part of the soft SEPT walk done by the TDX module.

8.5.2.4. *Details: Pending Pages Considerations*

As described in 8.5.1.7 above, the page state may change from PENDING to MAPPED (by TDG.MEM.PAGE.ACCEPT) and vice versa (by TDG.MEM.PAGE.RELEASE). Both operations are considered a change of page attributes and thus require re-export if the page has been exported.

8.5.2.5. *Details: Blocked Pages Considerations*

As described in 8.5.1.6.2 above, no blocked pages are allowed during the export blackout period (PAUSED_EXPORT phase). This applies to the following SEPT entry states: BLOCKED, EXPORTED_BLOCKED, PENDING_BLOCKED, PENDING_EXPORTED_BLOCKED.

There are two reasons for that:

- Blocked pages are not exported. The blocked state is always an interim state; the host VMM blocks a page when it intends to perform some other operation, such as page removal.
- Not allowing blocked pages during the PAUSED_EXPORT phase simplified the tracking of export completeness by TDH.MEM.SCAN.COMP(DCHECK) and TDH.EXPORT.MEM.

8.5.2.6. *Details: Memory Management Considerations*

8.5.2.6.1. Details: Host-Side Memory Management Considerations

The tables below summarize special consideration for the interaction of non-blocking export and TD private memory management operations done by the host VMM and guest TD.

Table 8.6: Host-Side Memory Management Considerations of Non-Blocking Export

Operation	Details
TDH.MEM.PAGE.AUG	A page that has been exported and then removed is put in an EXPORTED_REMOVED state (see below). While in this state, the host VMM can call TDH.MEM.PAGE.AUG to add a new page to the same GPA. The page state is set to EXPORTED_PENDING_MODIFIED.

Operation	Details
TDH.MEM.PAGE.DEMOTE	It is possible to demote a non-exported 2MB or 1GB page whose state is MAPPED, BLOCKED or PENDING or PENDING_BLOCKED. If the page state before the demote operation was MAPPED and a non-blocking export session is in the LIVE_EXPORT phase, the TDX module sets all the demoted pages' Dirty bits[1]. Otherwise, it clears them. A page that has been exported is mapped at 4KB, so it can't be demoted.
TDH.MEM.PAGE.PROMOTE	It is possible to promote a 2MB or 1GB GPA range if all the small pages within that range are non-exported whose state is MAPPED or PENDING. The TDX module sets the promoted page's Dirty bit[2]. It is not possible to promote a page that has been exported. Supporting that would complicate the SEPT state machine (e.g., handling the case where some 4KB pages have different states than other, add new states that indicate the need to export a CANCEL operation etc.). Note that the host VMM should already be able to handle situations where promotion is not possible, e.g., due to different L2 attributes set by L1.
TDH.MEM.PAGE.RELOCATE	A blocked page in the EXPORTED_BLOCKED or PENDING_EXPORTED_BLOCKED state can be relocated. This unblocks the page, and the page state is set to EXPORTED_MODIFIED or PENDING_EXPORTED_MODIFIED respectively.
TDH.MEM.PAGE.REMOVE	A page that has been exported can be removed. The page is put in an EXPORTED_REMOVED state to track the need for exporting a CANCEL operation for it, after which the page state is set to the regular FREE state.
TDH.MEM.SEPT.REMOVE	A page that has been exported and then removed is put in an EXPORTED_REMOVED state (see above). While in that state, the L1 SEPT entry mapping the page is not free. Thus, the L1 SEPT page where it resides cannot be removed.
TDH.MEM.RANGE.BLOCK	A page that has been exported can be blocked (and later removed or unblocked). Depending on whether the page is pending, the page is put in an EXPORTED_BLOCKED or PENDING_EXPORTED_BLOCKED state to track the need for exporting a CANCEL operation for it, after which the page state is set to the regular BLOCKED or PENDING_BLOCKED state respectively.
TDH.MEM.RANGE.UNBLOCK	A blocked page in the EXPORTED_BLOCKED or PENDING_EXPORTED_BLOCKED state can be unblocked. The page state is set to EXPORTED_MODIFIED or PENDING_EXPORTED_MODIFIED respectively.

8.5.2.6.2. Details: Guest-Side Memory Management Considerations

The guest TD is not directly aware of memory export; memory management operations are still available to it.

Table 8.7: Guest-Side Memory Management Considerations of Non-Blocking Export

Operation	Details
TDG.MEM.PAGE.ACCEPT	The guest TD may accept a PENDING_EXPORTED or a PENDING_EXPORTED_MODIFIED page. The page state becomes EXPORTED_MODIFIED so that TDH.MEM.SCAN.RANGE(DSCAN) or TDH.MEM.SCAN.COMP(DCHECK) will detect the need to re-export the page.

[1] With block-less demote, the h/w may still set the Dirty bit of the large page's SEPT entry while TDH.MEM.PAGE.DEMOTE is running and before it updated that SEPT entry to be a non-leaf entry. Thus, setting the Dirty bity of the small pages SEPT entries is necessary.

[2] This is done for simplification of implementation, since the page will have to be demoted in any case for migration.

Operation	Details
TDG.MEM.PAGE.ATTR.WR	The guest TD may update the attributes of an EXPORTED or EXPORTED_MODIFIED. The page state becomes EXPORTED_MODIFIED so that TDH.MEM.SCAN.RANGE(DSCAN) or TDH.MEM.SCAN.COMP(DCHECK) will detect the need to re-export the page. Similarly, the guest TD may update the attributes of a PENDING_EXPORTED, or PENDING_EXPORTED_MODIFIED page. The page state becomes PENDING_EXPORTED_MODIFIED.
TDG.MEM.PAGE.RELEASE	If supported, the guest TD may release an EXPORTED or an EXPORTED_DIRTY page. The page state becomes PENDING_EXPORTED_MODIFIED so that TDH.MEM.SCAN.RANGE(DSCAN) or TDH.MEM.SCAN.COMP(DCHECK) will detect the need to re-export the page.

8.5.2.7. *Details: Export Completeness Tracking*

8.5.2.7.1. Overview

After the TD is paused by TDH.EXPORT.PAUSE, the TDX module needs to enforce memory export completeness before a start token can be exported to enable the TD to run on the destination platform. This means that the exported memory image reflects the up-to-date state of the source TD memory image. Note that the migration protocol ensures that the imported memory image is in sync with the exported one.

The host VMM is expected to do the following:

1. Run a comprehensive scan of the TD's GPA space, by calling TDH.MEM.SCAN.COMP(DCHECK).
2. Export all memory pages reported by TDH.MEM.SCAN.COMP(DCHECK) as export candidates, using TDH.EXPORT.MEM.

The above operations can be done concurrently.

8.5.2.7.2. Memory Counters and Checking by TDH.EXPORT.TRACK(DONE)

The TDX module holds the following counters in TDCS. They are reset when a migration session starts (TDH.EXPORT.STATE.IMMUTABLE).

MEM_COUNT Counts the number of TD private memory pages, in multiples of 4KB. MEM_COUNT is incremented when a new page is added to the TD, and decremented when a page is removed from the TD.

MIG_COUNT ~~for export cancels.~~ Counts of the number of 4KB pages that have been exported at least once in the current migration session.

~~TD Private~~ **DIRTY_COUNT** Counts the number of 4KB pages that have been exported but either their content or their attributes have changed since, so a re-export is required. Counting is in units of 4KB. DIRTY_COUNT is incremented by any function which transitions the page state out of EXPORTED or PENDING_EXPORTED; it is decremented by TDH.EXPORT.MEM.

TDH.EXPORT.TRACK(DONE) checks the counter values as follows:

- TDH.MEM.SCAN.COMP(DCHECK) was called and completed a comprehensive scan of the TD's GPA space.
- DIRTY_COUNT is 0 – i.e., all memory pages that required a re-export were indeed re-exported.
- If the TDX module does not support post-copy for non-blocking export, or if the TD is enabled for TDX Connect (TD_PARAMS.CONFIG_FLAGS.TDX_CONNECT), MIG_COUNT is equal to MEM_COUNT – i.e., all memory pages have been exported.

Note: If the TDX module supports post-copy for non-blocking export and the TD is not enabled for TDX Connect, the host VMM is allowed not to export some of the memory image at this point. Typically, unexported pages are exported on demand during the post-copy phase. Failure to export (but not re-export) memory is considered a denial of service, which is not prevented by TDX.

The host VMM can read the above counters, if required, using TDH.MNG.RD.

Tracking of export completeness is based on the following properties, which are true during the PAUSED_EXPORT state, once the TD is paused by TDH.EXPORT.PAUSE and before a start token is generated by TDH.EXPORT.TRACK(DONE):

- TD private memory is immutable.
  - The TD is paused.
  - No TDIs are bound.
  - Host-side memory management operations are restricted, as described in 8.5.1.6.2. Specifically, no page may be blocked. Note that BLOCKED and PENDING_BLOCKED are intermediate states, as preparations for some memory management operation. Not allowing blocked pages simplifies tracking for correctness.
- TDH.MEM.SCAN.COMP(DCHECK) scans each TD private page exactly once.
- TDH.EXPORT.MEM can only export a page once.

8.5.2.7.3. Backward Compatibility

The migrated TD may have been created by an older TDX module that didn't support non-blocking export, and later the TDX module was updated using TD-preserving update. In this case, TDH.MEM.SCAN may be less efficient; some metadata maintained by a TDX module that supports non-blocking export during the TD lifecycle for optimizing the scan may be missing.

~~8.5.~~8.6. Memory Import

8.6.1. Host VMM Perspective

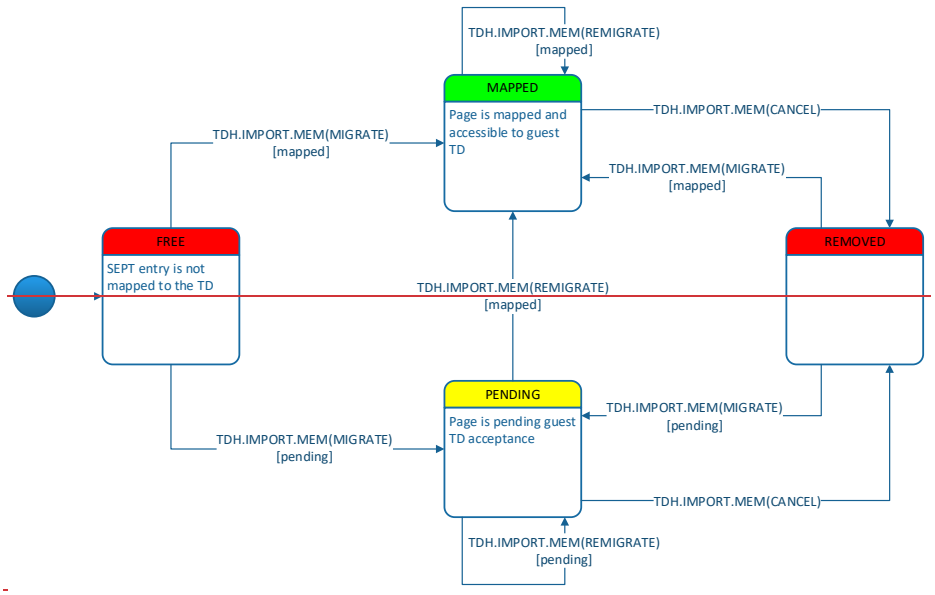
This section describes memory import from the host VMM perspective. This is a simplified view; a more detailed view is discussed later.

~~8.5.1.~~8.6.1.1. *In-Order Import Phase*

~~8.5.1.1~~8.6.1.1.1. 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~~8.6.1.1.2. 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. 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.

~~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. These include:

- 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~~1.1.1.1.1. ~~Enforcing a Single Import Operation per Migration Epoch~~

~~When a page is imported during the in-order phase, the current migration epoch is recorded. Page re-import and import cancel operations check the recorded migration epoch. For the import to succeed, it should be older than the current migration epoch.~~

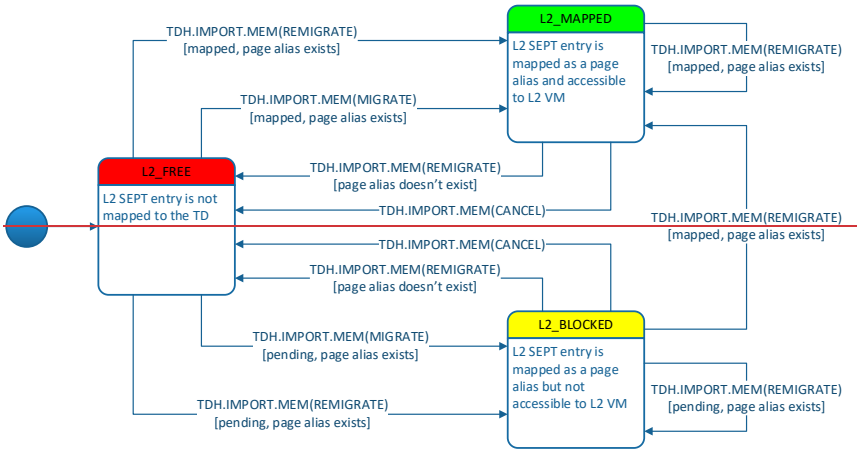
~~When a page import is cancelled during the in-order phase, the physical page is removed but its SEPT entry is put into a REMOVED state, and the current migration epoch is recorded. For partitioned TDs, any L2 SEPT entries that map the page become L2_FREE. Page first-time import operation check the recorded migration epoch. For the import to succeed, it should be older than the current migration epoch.~~

~~Section 2: TD Migration Architecture Specification~~

~~8.5.1.4~~1.1.1.1.1. ~~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.~~



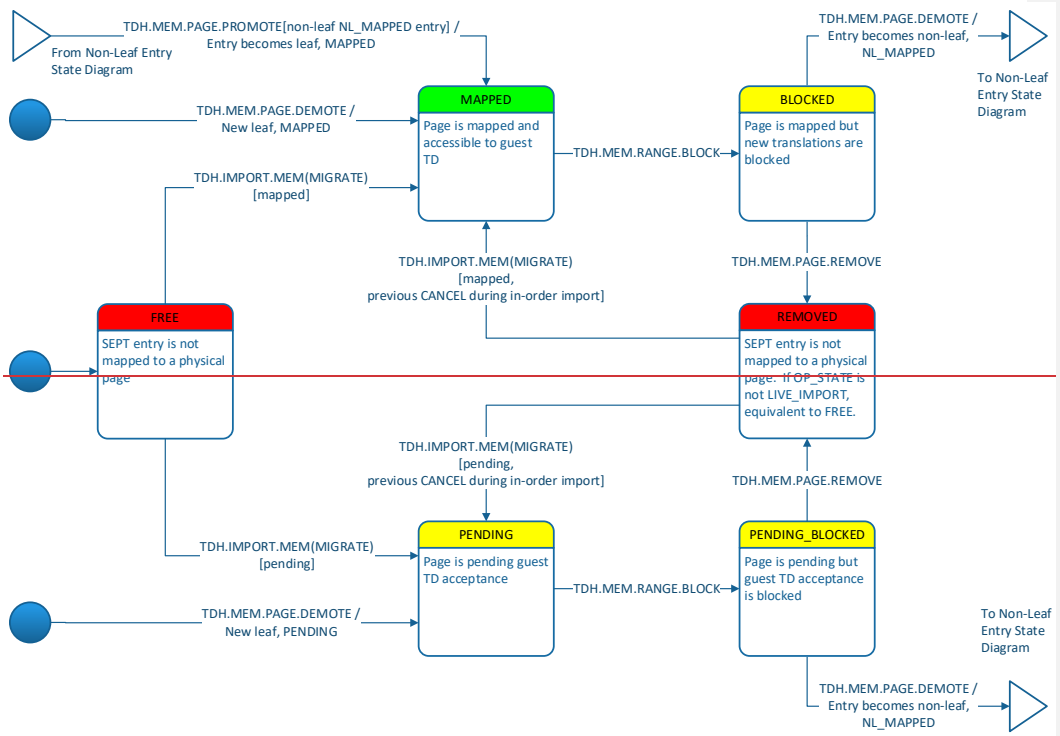
~~Figure 8.12: Partial L2 SEPT Leaf Entry State Diagram for L2 Page Mapping Import During the In-Order Phase~~

~~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.~~8.6.1.2. *Out-of-Order import Phase*

~~8.5.2.1~~8.6.1.2.1. 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~~to import 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~~8.6.1.2.2. 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~~ (since the TD had been finalized before the migration session began). TDH.MEM.PAGE.AUG is allowed ~~in the LIVE~~ after TDH.IMPORT~~.OP_STATE~~.COMMIT, when the TD is allowed to run on the destination platform while the import session continues.

If a page was not imported but 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~~exists 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~~after TDH.IMPORT.COMMIT, when the TD is allowed to run on the destination platform while the import session continues, until TDH.IMPORT.END:

- 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.~~

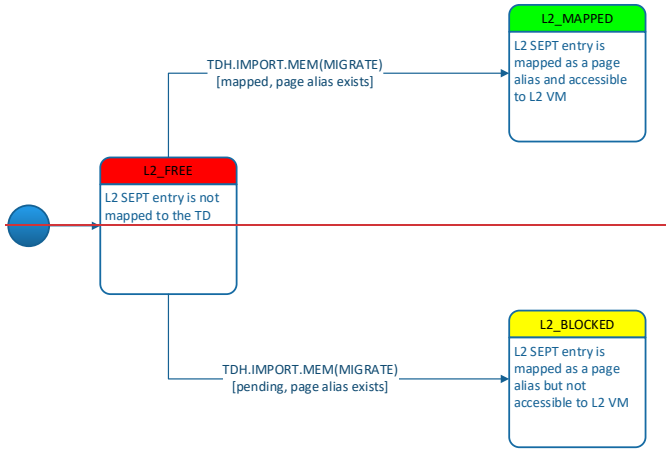
~~If the host VMM succeeded in re-importing the page, it would have rolled back the page content. Remember we do not enforce order of import during the out-of-order phase. But setting the SEPT entry state to REMOVED when the page was removed prevents this attack.~~

~~8.5.2.2.6.~~ Page Relocation ~~During Out-of-Order Import~~

Page relocation is supported at any stage without any changes.

~~8.5.2.3.1.1.1.1.1.~~ ~~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.~~8.6.1.3. *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.

8.6.2. Details of Memory Import

8.6.2.1. *Details: In-Order Import Phase*

8.6.2.1.1. Details: Overview of In-Order Import

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

The state diagram below shows the L1 SEPT entry state during in-order import. The color-coding convention used in this diagram is described in 8.2.

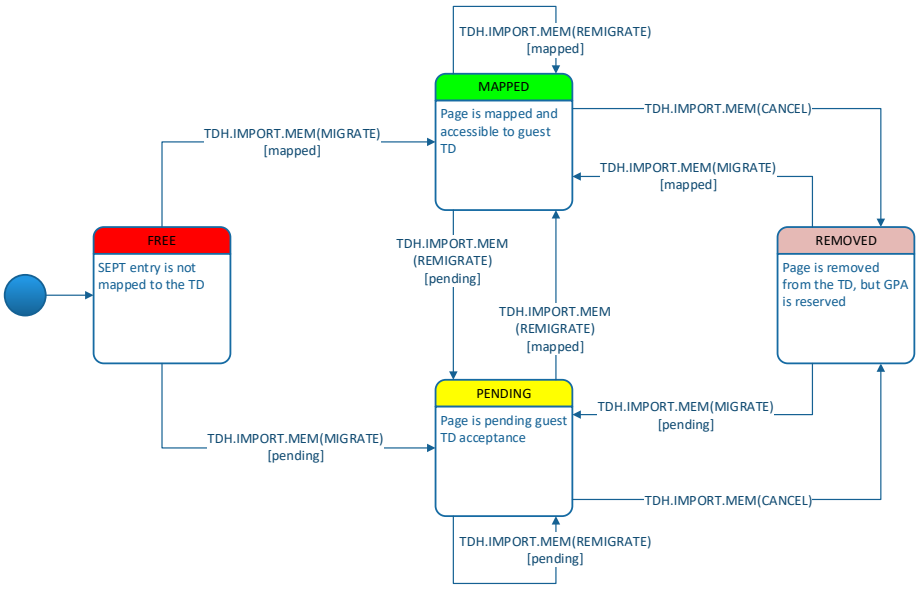


Figure 8.15: Partial L1 SEPT Entry State Diagram for Page In-Order Import Phase

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.

8.6.2.1.2. Details: Enforcing a Single Import Operation per Migration Epoch

When a page is imported during the in-order phase, the current migration epoch is recorded. Page re-import and import cancel operations check the recorded migration epoch. For the import to succeed, it should be older than the current migration epoch.

When a page import is cancelled during the in-order phase, the physical page is removed but its SEPT entry is put into a REMOVED state, and the current migration epoch is recorded. For partitioned TDs, any L2 SEPT entries that map the page become L2_FREE. A page first-time import operation checks the recorded migration epoch. For the import to succeed, it should be older than the current migration epoch.

8.6.2.1.3. 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.

The state diagram below shows the L2 SEPT entry state during in-order import. The color-coding convention used in this diagram is described in 8.2.

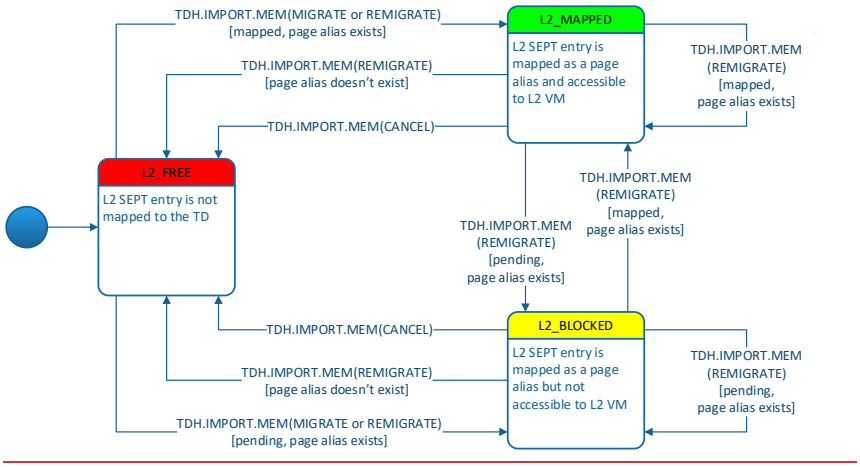


Figure 8.16: Partial L2 SEPT Leaf Entry State Diagram for L2 Page Mapping Import During the In-Order Phase

8.6.2.2. *Details: Out-of-Order import Phase*

8.6.2.2.1. Details: Overview of Out-of-Order Import

The state diagram below shows the L1 SEPT entry state during out-of-order import. The color-coding convention used in this diagram is described in 8.2.

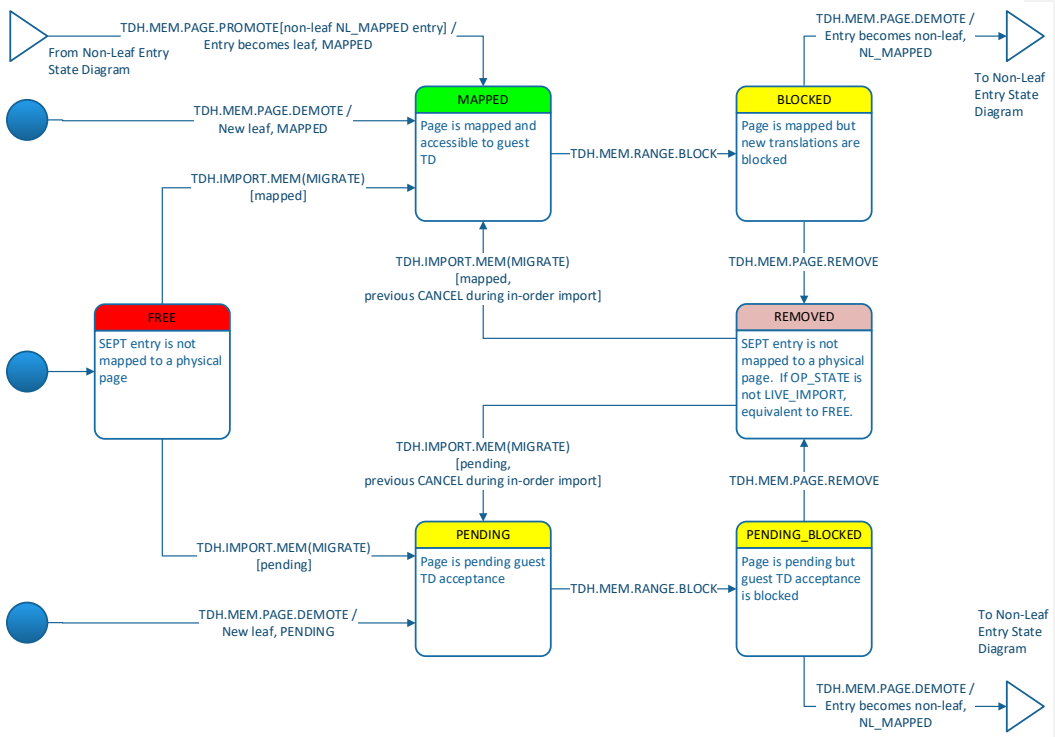


Figure 8.17: Partial L1 SEPT Entry State Diagram for Page Out-of-Order Import Phase

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.

8.6.2.2.2. Details: 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. The state diagram below shows the L2 SEPT entry state during out-of-order import. The color-coding convention used in this diagram is described in 8.2.

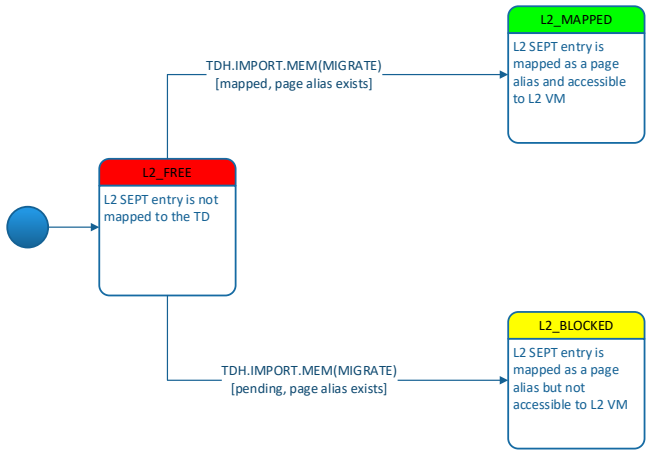


Figure 8.18: Partial L2 SEPT Leaf Entry State Diagram for Page Out-of-Order Import Phase

~~8.6.~~8.7. Secure EPT Concurrency Considerations

~~**Note:** OP_STATE related concurrency considerations are described in 6.2.6.~~

8.7.1. Overview

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.

8.7.2. GPA List Processing Implications

On the source side, export functions that process a GPA list (TDH.EXPORT.BLOCKW, TDH.EXPORT.MEM, TDH.EXPORT.RESTORE) that encounter a busy SEPT entry skip the processing of that GPA. In this case, they set the STATUS field of the GPA list entry to SEPT_ENTRY_BUSY_HOST_PRIORITY and the OPERATION field to NOP. The host VMM can scan the GPA list to detect skipped entries and retry the operation later.

On the destination side, if TDH.IMPORT.MEM encounters a busy SEPT entry, its behavior depends on the import phase. During the in-order phase, any SEPT entry error causes the import session to fail. In the out-of-order phase, TDH.IMPORT.MEM skips the busy entry, similar to TDH.EXPORT.MEM.

8.8. Security Analysis: Achieving Memory Migration Security Objectives

8.8.1. 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).
- When write-blocking based export is used 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.
- When non-blocking export is used on the source platform, a private page is detected as requiring export based on an atomic test-and-clear of the SEPT entry's Dirty bit set by the h/w. For partitioned TDs, this includes applicable L2 SEPT entries. Following from previous security requirements, this scan also requires TLB tracking prior to export to help ensure that no active writable cached GPA address translations prevent further updates of the SEPT entry's Dirty bit.
- 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 importing TD memory and CPU state is distinct from keys used for other operations such as Paging.

8.8.2. Preventing Usage of Stale Memory Copies due to Mis-Ordering

Multiple versions of a page that are exported in a certain order are imported in the same order. This is achieved using the following mechanisms:

- Migration streams help ensure ordering within each stream.
- Migration epochs, limiting migration of a specific page to no more than one time per epoch, help ensure total ordering per page.

8.8.3. Enforcing Export of the Entire Memory Image

TDX Connect

When the TD is enabled for TDX Connect, while TDIs are bound and have DMA access to the TD memory, DMA must not fail. Therefore, TDX Connect requires that all the TD mapped pages remain pinned as long as TDI can access them. The entire TD private memory image must be migrated intact. This is enforced by the TDX module as described below.

Only non-blocking export is supported when TDX Connect is enabled for a TD. A counter of unexported pages (TDCS.UNEXPORTED_COUNT) is maintained by the TDX module at the source platform based on a comprehensive scan of the TD's GPA space by TDH.MEM.SCAN.COMP(DCHECK) and on TDH.EXPORT.MEM. If the value of that counter is not 0, then TDH.EXPORT.TRACK fails. See 8.5.2.7 for details.

Non TDX Connect

If the TD is not configured for TDX Connect, the TDX module does not enforce exporting all memory. If a page is not migrated, an EPT violation happens when the guest TD attempts to access it on the destination platform. This is normal behavior for the post-copy method of migration, and in the worst case is considered a denial of service.

8.8.4. Non-Blocking Export: Detecting Memory State Change

Memory state change is detected by scanning the SEPT tree and detecting pages where the Dirty bit is set.

1. TDH.MEM.SCAN.RANGE(DSCAN) does the following with the SEPT entry exclusively locked (but Accessed/Dirty bits can be set by h/w):
   1.1. Detect that an EXPORTED page is dirty (either L1 or any L2 SEPT entries' Dirty bit is 1).
   1.2. Clear the Dirty bit in each of L1 and L2 SEPT entries.
   1.3. Update the page state to EXPORTED_MODIFIED.
   1.4. Records the TD epoch in the page's PAMT.
2. TDH.EXPORT.MEM does the following with the SEPT entry exclusively locked (but Accessed/Dirty bits can be set by h/w):
   2.1. Find that the page state is MODIFIED.
   2.2. Check the TLB tracking conditions.

2.3. Export the page.
2.4. Update the page state to EXPORTED.

The above sequence helps ensure the following:

- Setting the EXPORTED_MODIFIED state in step 1.3 means that this page will be detected later by DSCAN or DCHECK and will be re-exported as a precondition to start token generation (see below).
- The order of TLB tracking check (in step 2.2) before the page is exported (in step 2.3) means that any write access by either the TD s/w, by the TDX module working on behalf of the TD s/w or by a TDI will result in an SEPT walk and setting of the Dirty bit. The page will be detected later by DSCAN or DCHECK and will be re-exported again as a precondition to start token generation (see below).

8.8.5. Preventing Usage of Stale Memory Copies due to Failure to Re-export

Running the destination TD with a stale copy of a memory page, because the source VMM failed to re-export a newer copy of a page, is prevented as described below.

Assume for example that the source VMM exported an older version (v1) of page but never re-exported a newer version (v2) of that page. In this case, generating a start token by TDH.EXPORT.TRACK is prevented.

- With write-blocking based export, a counter of dirty pages (TDCS.DIRTY_COUNT) is maintained by the TDX module at the source platform, based on tracking the TDH.EXPORT.MEM and TDX.EXPORT.UNBLOCKW operations. If the value of that counter is not 0, then TDH.EXPORT.TRACK fails. See 8.4.2.38.4.2.3 for details.
- With non-blocking export, the same counter is maintained by the TDX module based on a comprehensive scan of the TD's GPA space by TDH.MEM.SCAN.COMP(DCHECK) and on TDH.EXPORT.MEM. If the value of that counter is not 0, then TDH.EXPORT.TRACK(DONE) fails. See 8.5.2.7 for details.

8.8.6. Preventing Usage of Missing or Stale Memory Copies due to Failure to Import

Running the destination TD with a missing or a stale copy of a memory page, because the destination VMM failed to import a copy of a page, is prevented as described below.

The in-order phase commitment protocol is designed to ensure that the export will fail, and the destination TD will not run. As discussed above, TDH.EXPORT.TRACK(DONE) generates a start token that is dependent of the exact export sequence; it checks that no unexported newer versions of previously exported pages remain. If the TD is enabled for TDX Connect, TDH.EXPORT.TRACK(DONE) checks that no unexported 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.

8.8.7. Preventing Usage of Stale Memory GPA Mapping and Attributes

The destination TD is prevented 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.

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

In the in-order import phase the host VMM can import pages 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 us to use 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.

~~8.7.~~8.9. *Memory Migration Interface Functions Summary*

This section provides a short ~~overview~~summary 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],**~~**Table 8.8: Memory Migration Interface Functions**

Name	Export Mode	Description	Preconditions
TDH.EXPORT.BLOCKW	Write Blocking	Block a list of 4KB pages for writing, as preparation for export.	Write-blocking export session is in the in-order phase and the TD has not been paused yet
TDH.EXPORT.MEM	Any	Export, re-export or sends an export cancellation request for a list of 4KB pages.	Export session is in progress
TDH.EXPORT.RESTORE	Any	Restore a list of 4KB pages after an export session abort.	Export session is not in progress
TDH.EXPORT.UNBLOCKW	Write Blocking	Unblock a single 4KB page that has been blocked for writing.	The TD is configured for write-blocking based export, and either an export session is in progress, but the committed export phase has not begun, or the TD is allowed to run
TDH.IMPORT.MEM	Any	Import, re-import or cancel a previous import for a list of 4KB pages.	Import session is in progress
TDH.MEM.SCAN.COMP(DCHECK)	Non-Blocking	Do a comprehensive scan of the TD private memory and detect pages that must be exported for the export session to complete.	The non-blocking export session is in the EXPORT_PAUSED state.
TDH.MEM.SCAN.CONFIG	Non-Blocking	Configure comprehensive scan parameters	
TDH.MEM.SCAN.RANGE(DSCAN)	Non-Blocking	Scan a TD private memory GPA range and detect pages that need to be exported.	

Name	Export Mode	Description	Preconditions
TDH.MEM.SCAN.RANGE (EXPORT_RESTORE)	Any	Scan the TD private memory and restore SEPT after an export session abort.	Export session is not in progress
TDH.MEM.SCAN.RESET	Non-Blocking	Reset comprehensive scan state	