# Intel® Trust Domain Extensions (Intel® TDX) Module Extension for Pre-Migration

## 869278-001US (DRAFT)

November 2025

Ref. # 869278-001US (DRAFT)

# Notices and Disclaimers

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

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

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

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

No product or component can be absolutely secure.

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

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

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

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



Intel® TDX Module Extension for Pre-Migration

# Table of Contents

**1. About this Document ..... 5**

*1.1. Scope of this Document ..... 5*

*1.2. Notation ..... 6*

1.2.1. Requirement and Definition Commitment Levels ..... 6

*1.3. References ..... 6*

**2. Overview of the NRX Framework ..... 7**

*2.1. Interfaces ..... 7*

2.1.1. Host VMM Interfaces ..... 7

*2.2. NRX Initialization ..... 7*

2.2.1. Intel TDX Module Lifecycle Update ..... 7

2.2.2. Host VMM Declaration of Features on Intel TDX Module Initialization ..... 8

2.2.3. Adding Pages to the Intel TDX Module's Memory Pool ..... 8

*2.3. NRX Sessions and Concurrent NRX requests ..... 9*

*2.4. NRX Updates ..... 10*

2.4.1. NRX session state management on SVN changes during TD preserving update ..... 10

*2.5. NRX Fatal Error Handling by VMM ..... 10*

*2.6. Security Perspective ..... 11*

**3. Intel TDX Module Extension for Migration (MIG NRX) ..... 12**

*3.1. Overview ..... 12*

*3.2. MIG NRX Requests and Sessions ..... 13*

*3.3. Migration Policy Binding to a TD ..... 13*

*3.4. MIG NRX Initialization and Runtime Interaction ..... 13*

3.4.1. Initialization of MIG NRX by VMM ..... 13

3.4.2. Runtime Preparation of the Migration Session ..... 13

3.4.3. Runtime Preparation of the Migration Session: Internal details ..... 14

*3.5. MIG NRX Updates ..... 15*

3.5.1. MIG NRX session state management on SVN changes during a TD preserving update ..... 15

*3.6. Fatal Error Handling in MIG NRX by VMM ..... 16*

**4. Migration related Intel TDX Module Specification Changes ..... 18**

*4.1. Intel TDX Module Enumeration & Configuration ..... 18*

*4.2. Data Types ..... 18*

4.2.1. New: Data Buffer List ..... 18

4.2.2. Update: SERVTD_EXT_STRUCT ..... 18

4.2.3. New: Constants ..... 19

*4.3. Host-Side (SEAMCALL) Functions ..... 20*

4.3.1. Update: TDH.MNG.RD/TDH.MNG.WR ..... 20

4.3.2. Update: TDH.SYS.RD ..... 20

4.3.3. Update: TDH.SYS.RD ..... 20

4.3.4. Update: TDH.SYS.RD/WR ..... 20

4.3.5. Update: TDH.SYS.RD/WR ..... 21

4.3.6. Update: TDH.EXT.MEM.ADD ..... 21

4.3.7. New: TDH.MIG.SETUP Leaf ..... 21

Input Operands ..... 21

Output operands ..... 22

Leaf Function Description ..... 22

Operands Information ..... 22

Completion Status Codes ..... 22

Intel® TDX Module Extension for Pre-Migration

# 1. About this Document

## 1.1. Scope of this Document

The purpose of this document is to demonstrate an alternative design for implementing the migration session establishment and migration policy enforcement steps of the Intel® Trust Domain Extensions (Intel® TDX) Migration architecture.

In the previous design of the Intel TDX Module, these steps were made by a specific type of **Service TD** called the **Migration TD (a.k.a. MigTD)**. The alternative design presented in this document eliminates the need for MigTD and instead moves the abovementioned functionality into an Intel TDX module extension, called MIG NRX.

The intention of this document is not to demonstrate the internal details of the MIG NRX architecture, but to provide enough information on how the functionality provided by this Intel TDX module extension can be used by an Operating System (OS) or Virtual Machine Manager (VMM).

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

**Table 1.1: Intel TDX Module Architecture Specification Set**

| Document Name | Reference | Description |
|---|---|---|
| **Intel TDX Module Base Architecture Specification** | [Intel TDX Module Base Spec] | Base Intel TDX Module architecture overview and specification, covering key management, TD lifecycle management, memory management, virtualization, measurement and attestation, service TDs, debug aspects etc. |
| **Intel TDX Module TD Migration Architecture Specification** | [TD Migration Spec] | Architecture overview and specification for TD migration |
| **Intel TDX Module TD Partitioning Architecture Specification** | [TD Partitioning Spec] | Architecture overview and specification for TD Partitioning |
| → **Intel TDX Module Extension for Pre-Migration** | [Intel TDX Pre-Migration module Extension] | Architecture overview and specification for Pre-Migration TDX Module Extension |
| **Intel TDX Module TDX Connect Specification** | [TDX Connect Spec] | Architecture overview and specification for Intel TDX Connect |
| **Intel TDX Module ABI Reference Specification** | [Intel TDX Module ABI Spec] | Detailed Intel TDX Module Application Binary Interface (ABI) reference specification, covering the entire Intel TDX Module architecture |
| **Intel TDX Module TDX Connect ABI Reference Specification** | [Intel TDX Connect ABI Spec] | Detailed Intel TDX Module Application Binary Interface (ABI) reference specification, covering the TDX connect architecture |
| **Intel TDX Module ABI Reference Tables** | [Intel TDX Module ABI Tables] | A set of files detailing Intel TDX Module Application Binary Interface (ABI) |
| **Intel TDX Module ABI Incompatibilities** | [Intel TDX Module ABI Incompatibilities] | Description of the incompatibilities between Intel TDX 1.0 and Intel TDX 1.4/1.5 that may impact the host VMM and/or guest TDs |

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 updating this document in real time when such changes occur.

## 1.2. Notation

This section describes the notation used in this document.

### 1.2.1. Requirement and Definition Commitment Levels

When specifying requirements or definitions, the level of commitment is specified following the convention of RFC 2119: Key words for use in RFCs to indicate Requirement Levels, as described in the following table:

**Table 1.2: Requirement and Definition Commitment Levels**

| Keyword | Description |
|---|---|
| **Must** | "**Must**", "**Required**" or "**Shall**" means that the definition is an absolute requirement of the specification. |
| **Must Not** | "**Must Not**" or "**Shall Not**" means that the definition is an absolute prohibition of the specification. |
| **Should** | "**Should**", or the adjective "**Recommended**", means that there may exist valid reasons in particular circumstances to ignore a particular item, but the full implications must be understood and carefully weighed before choosing a different course. |
| **Should Not** | "**Should Not**", or the phrase "**Not Recommended**" means that there may exist valid reasons in particular circumstances when the particular behavior is acceptable or even useful, but the full implications should be understood, and the case must be carefully weighed before implementing any behavior described with this label. |
| **May** | "**May**", or the adjective "**Optional**", means that an item is discretionary. An implementation may choose to include the item, while another may omit the same item, because of various reasons. |

## 1.3. References

See the [Intel TDX Module Base Spec].

# 2. Overview of the NRX Framework

NRX Framework is a framework for extending the Intel TDX Module's functionality, using NRXs (Non-Root Mode Extensions). The main objective of implementing some Intel TDX Module functionality as NRX is the support for long running stateful flows, including but not limited to cryptographic operations.

Figure 2.1 shows a conceptual view of the NRX Framework with three NRXs which use it, such as the TEE-IO provisioning agent, TPA (for Intel TDX Connect), Quoting (for Intel TDX Attestation) and MIG (for Intel TDX Pre-Migration). The TPA Extension is the first Intel TDX Module extension which is used for establishing and managing Security Protocols and Data Models (SPDM) session for Intel TDX Connect and its design and interfaces are covered elsewhere. This document introduces a new NRX used for Intel TDX Pre-Migration step.

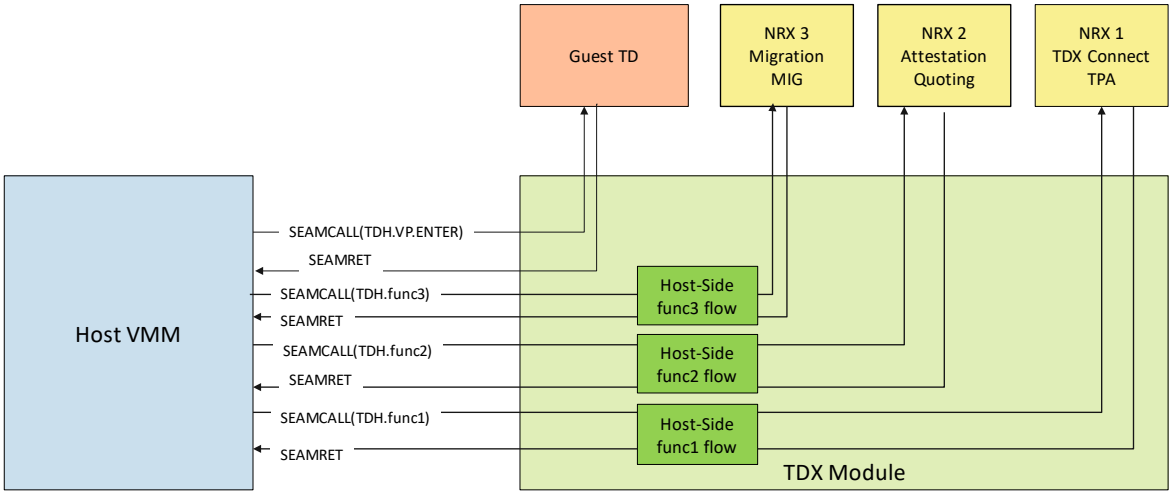


**Figure 2.1: NRX Framework Concept**

The internal design of NRXs, as well as the interaction between Intel TDX Module and NRXs is implementation specific and subject to change. VMM should not assume or rely on any behavior except the exposed host-side (SEAMCALL) interface functions.

## 2.1. Interfaces

A NRX is hidden from the outside world. Neither the host VMM nor guest TDs have direct interface with it.

### 2.1.1. Host VMM Interfaces

The host VMM is not directly aware of NRX; it only interfaces with the Intel TDX Module. The host-side interface functions of the Intel TDX Module have been extended to include resource allocation and additional initialization functions. From the host VMM's point of view it allocates resources to the Intel TDX Module.

NRX provides additional services to the Intel TDX Module. From the host VMM perspective, these interface functions behave just like any other host-side (SEAMCALL) interface function.

## 2.2. NRX Initialization

### 2.2.1. Intel TDX Module Lifecycle Update

As part of the Intel TDX Module initialization sequence, the following interface functions are called by the host VMM:

**TDH.SYS.CONFIG/UPDATE:** If the host VMM intends to use a certain feature, such as Intel TDX Connect or Intel TDX Pre-Migration Extension, it must declare so using an input parameter.

**TDH.SYS.RD:** The host VMM reads the missing number of Intel TDX Module's memory pool pages (see section 2.2.3 for memory pool definition).

The host VMM reads if NRX Framework initialization is required.

**TDH.EXT.MEM.ADD:** If required, the host VMM calls this interface function multiple times to add pages to the Intel TDX Module's memory pool.

**TDH.EXT.INIT:** If required, the host VMM calls this interface function to initialize the Intel TDX Module extensions.

To avoid long initialization latency, any Intel TDX Module interface function that does not leverage a NRX service, can run while the memory pool is being allocated and NRX is being initialized.

Calling interface functions that depend on the NRX Framework before the NRX Framework is initialized, will fail with a TDX_EXT_NOT_INITIALIZED status code.

The following sections provide more details about the NRX initialization.

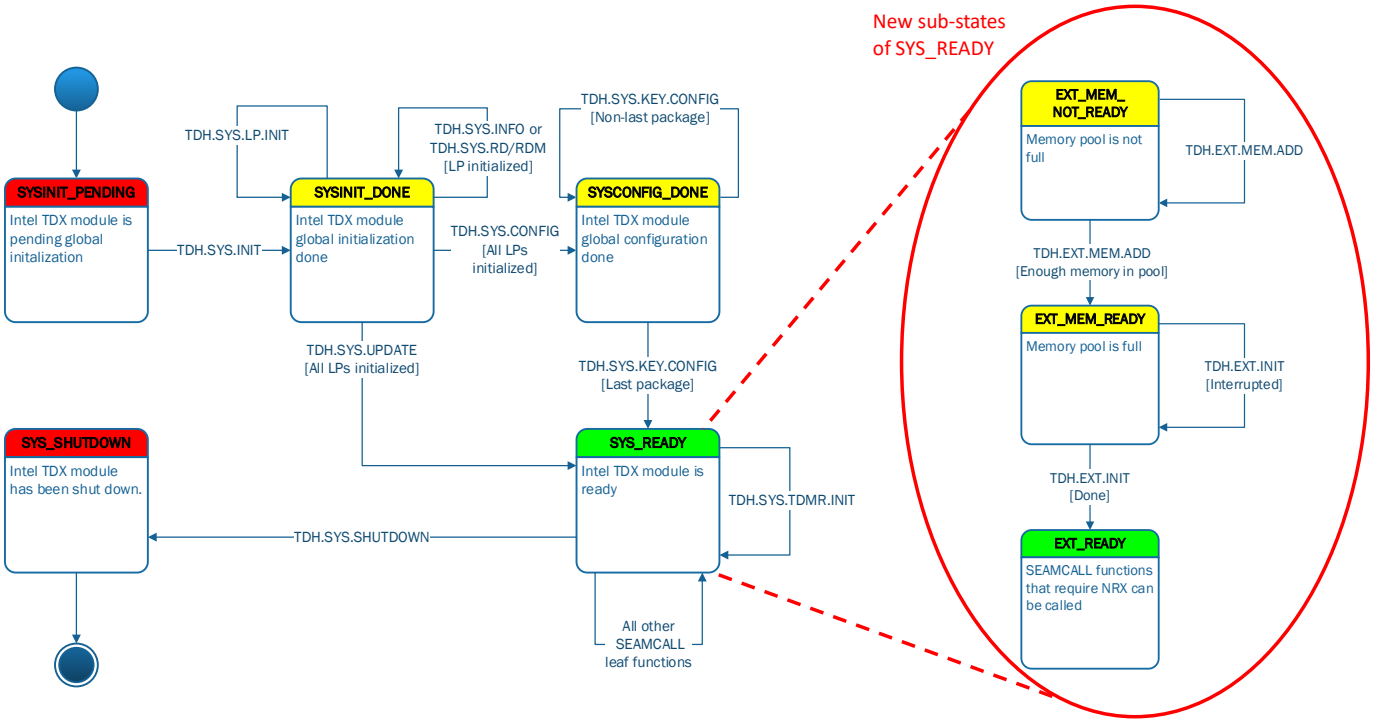


**Figure 2.2: Intel TDX Module Lifecycle Update**

### 2.2.2. Host VMM Declaration of Features on Intel TDX Module Initialization

TDH.SYS.CONFIG/UPDATE is extended with an input operand, allowing the host VMM to specify the Intel TDX feature it intends to use. This is done using a bitmap formatted the same as the TDX_FEATURES0/1 fields which enumerate Intel TDX features. All required features need to be specified by the host VMM.

For using TPA or MIG TDX Module extensions, the host VMM must set the applicable bit. The Intel TDX Module will use this information to calculate the following:

- If a memory pool is required
- The required number of memory pool pages
- If the NRX Framework initialization is required

### 2.2.3. Adding Pages to the Intel TDX Module's Memory Pool

To support NRX, the Intel TDX Module holds a pool of 4KB physical pages. The host VMM, as part of the Intel TDX Module initialization flow, allocates memory for the memory pool. The required size of the memory pool is calculated and provided to the host VMM by the Intel TDX Module.

All pages in the memory pool are considered equal. For example, the Intel TDX Module does not track their package affinity.

At any time, the host VMM can determine the necessity and the amount of memory that is still missing from the Intel TDX memory pool by using TDH.SYS.RD to read the MEMORY_POOL_REQUIRED_PAGES fields. The return values depend on the following:

- The set of Intel TDX features declared by the host VMM in TDH.SYS. CONFIG/UPDATE .
- The size of memory pool allocated so far.

On a TD-preserving update the memory pool allocated during the previous Intel TDX Module runtime still exists. If there is a need to allocate more memory, the Intel TDX Module reports this via MEMORY_POOL_REQUIRED_PAGES.

To allocate memory for the pool, the host VMM calls TDH.EXT.MEM.ADD one or more times. With each invocation, the host VMM can provide up to 512 4K pages.

Additionally, for each Intel TDX Module Feature backed up by NRX, VMM has two parameters that it can read via TDH.SYS.RD: [NRX_NAME]_EXT_MAX_OVERALL_SESSIONS and [NRX_NAME]_EXT_MAX_CONCURRENT_THREADS. These define the absolute maximum number of NRX sessions that a given NRX can support, as well as the maximum number of concurrent queues that can be used to serve NRX requests for this NRX (see section 2.3 for definition of NRX session and NRX concurrent queue). These variables are calculated according to build time data provided by each NRX.

By default, each NRX will operate with the maximum values of these parameters unless explicitly reconfigured by VMM. The re-configuration can be done by VMM any time before calling TDH.SYS.CONFIG/UPDATE via TDH.SYS.WR SEAMCALL to specify the [NRX_NAME]_EXT_OVERALL_SESSIONS and [NRX_NAME]_EXT_CONCURRENT_THREADS variables. Setting these parameters can change the value of the MEMORY_POOL_REQUIRED_PAGES and therefore the final required value of MEMORY_POOL_REQUIRED_PAGES must be read after all NRXs has been configured to the desired state.

The overall flow diagram for initialization is outlined in Figure 2.3.

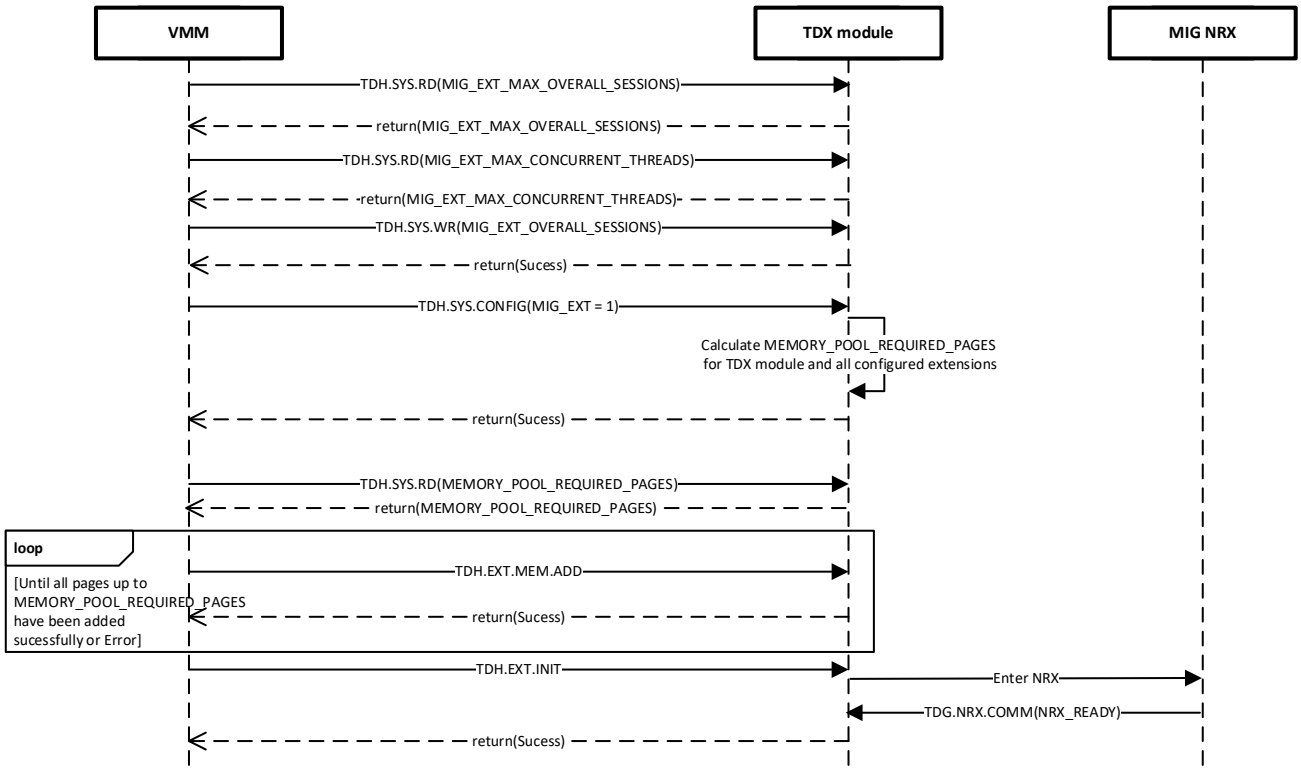


**Figure 2.3 Initialization process for MIG NRX**

## 2.3. NRX Sessions and Concurrent NRX requests

An ***NRX request*** is a single host-side (SEAMCALL) interface function served by an NRX. An ***NRX session*** is a sequence of NRX requests that has an associated stored context within the NRX Framework. The association between an NRX session and the stored context is 1:1. The NRX session is identified by a unique session ID that must be present in each NRX request associated with this NRX session. The NRX session starts from a first NRX request with the given session ID and ends when a Complete status is returned (success/error codes) for this NRX request. During this period an NRX session is considered active. For each NRX, a variable *EXT_MAX_OVERALL_SESSIONS (obtained via TDH.SYS.RD) defines the maximum amount of NRX sessions that can be active at the same time for a given Intel TDX Module feature backed up by NRX.

NRX requests are served sequentially by each NRX, i.e. only one ***concurrent NRX request*** to a given NRX can be served at a time, if only one ***concurrent processing thread (VCPU)*** is allocated for this NRX. The example for such case is shown in Figure 2.2.

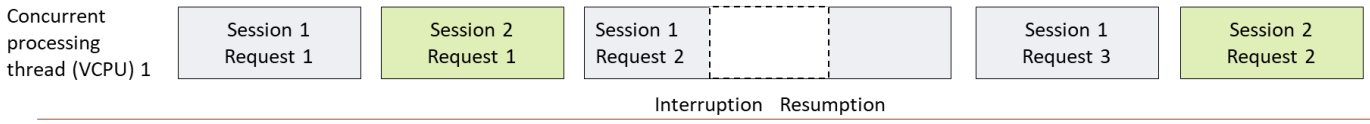


**Figure 2.4: NRX requests and sessions in a single concurrent processing thread (VCPU) case**

In case more than one concurrent processing thread (VCPU) is used, the NRX requests and sessions are distributed by the Intel TDX Module to any available VCPU from the VCPU pool at the time of servicing the request. Figure 2.5 shows the case where 2 concurrent processing threads (VCPUs) are present. The only restriction is that an NRX request that has been interrupted on a given concurrent processing thread (VCPU) must be always resumed on the same VCPU. Until this resumption happens, the corresponding VCPU has been reserved and cannot be used to process any other NRX requests.

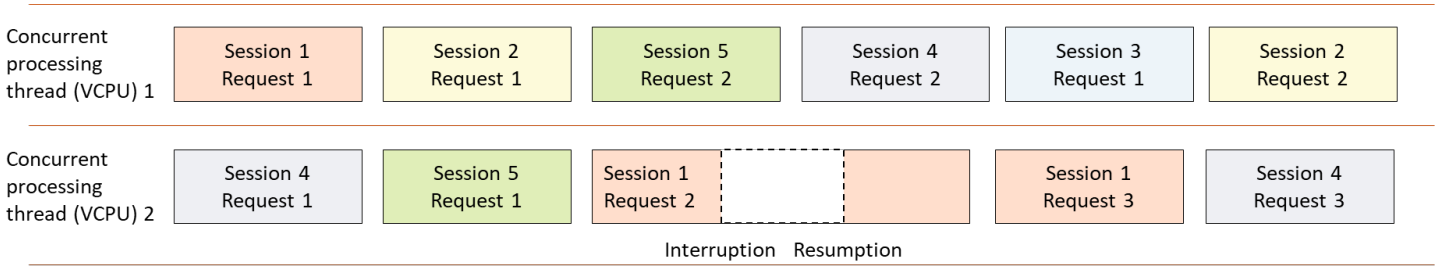


**Figure 2.5: NRX requests and sessions in two concurrent processing threads (VCPUs) case**

## 2.4. NRX Updates

In order to update an NRX, an Intel TDX Module TD preserving update is required. Depending on whether the TD preserving update changes the value of the MEMORY_POOL_REQUIRED_PAGES, the additional pages must be either added or not into the Intel TDX Module memory pool.

VMM also has an option to calculate the required number of memory pages prior to the start of TD preserving update using the information provided with each release of Intel TDX Module. In this case VMM can proactively add additional memory pages to the Intel TDX Module using existing TDH.EXT.MEM.ADD interface function prior to the start of TD preserving update sequence. This ensures that after TD preserving update is completed, VMM can directly proceed to initialize required NRX extensions using TDH.EXT.INIT.

It is safe to perform a TD preserving update between each individual invocations of NRX requests since NRX Framework and NRXs ensure that the relevant state is preserved.

### 2.4.1. NRX session state management on SVN changes during TD preserving update

Currently the Intel TDX Module and all NRXs share the same Security Version Number (SVN). A TD preserving update might result in an increase in this SVN. Intel TDX Module will keep the SVN versioning of the NRX session state and this SVN version is returned to each NRX with each NRX Request. Each NRX can decide what is the appropriate action to take on such SVN increases. VMM must not make any assumptions about the internal state machine of NRX session management.

## 2.5. NRX Fatal Error Handling by VMM

If an NRX experiences a fatal error, the Intel TDX Module returns an TDX_EXT_FATAL_ERROR back to VMM as a status output of the ongoing host-side (SEAMCALL) interface function indicating that a given NRX must be restarted. The way to restart an NRX is to perform a TD preserving update or reload of the Intel TDX Module.

## 2.6. Security Perspective

The NRX Framework does not put the NRXs in the TCB of the Intel TDX Module. This means that NRX can't compromise the Intel TDX Module's security.

However, specific functionality implemented by NRX is in the TCB of TDs which consume that functionality. For example, the TPA NRX which handles Intel TDX Connect's device protocols is in the TCB of TDs which use Intel TDX Connect. Similarly, the MIG NRX is in TCB of all migratable TDs.

Intel® TDX Module Extension for Pre-Migration

# 3. Intel TDX Module Extension for Migration (MIG NRX)

## 3.1. Overview

Analogous to legacy VM migration, a cloud-service provider (CSP) may want to relocate/migrate an executing Trust Domain from a source Intel TDX platform to a destination Intel TDX platform in the cloud environment. For an overview of Intel TDX Migration, refer to the [TD Migration Spec].

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.

In the previous design of the Intel TDX Module, TD Migration policy was enforced in a scalable and extensible manner using a specific type of **Service TD** called the **Migration TD (a.k.a. MigTD)** which was used to provide services for migrating TDs. This specification introduces a new Intel TDX Module Feature for the Pre-Migration step, implemented using an NRX Module (MIG NRX), which is an alternative to MigTD design outlined in the [TD Migration Spec]. The components involved into Intel TDX Migration using MIG NRX are shown in Figure 3.1.

The goals and overall functionality of MIG NRX compared to MigTD remain the same. MIG NRX must exist on the source and destination platforms. Their main role is to evaluate migration sources and destinations for adherence to the **TD Migration Policy**. The migration policy may enumerate Intel TDX platform TCB requirements, platform features and acceptable destination Migration TD TCB requirements.

Similarly to MigTDs, MIG NRXs securely exchange unique per-session **Migration Session Key (MSK)** pair. The MSKs are used to migrate assets of a specific TD. The MIG NRX on each side reads an encryption key generated by the Intel TDX Module and securely transfers it to the MIG NRX on the other side, where it is written as the decryption key for its side. To securely exchange the MSK pair in the presence of untrusted VMM providing a communication channel, the source and destination MIG NRXs must establish an Authenticated Key Exchange (AKE) session. The choice of the AKE protocol and its implementation is specific to MIG NRX similarly as it is with the current MigTD.

Contrary to MigTD, MIG NRX does not maintain a TD Migration Policy by itself. Instead, it always refers to a TD Migration Policy associated with a source TD.

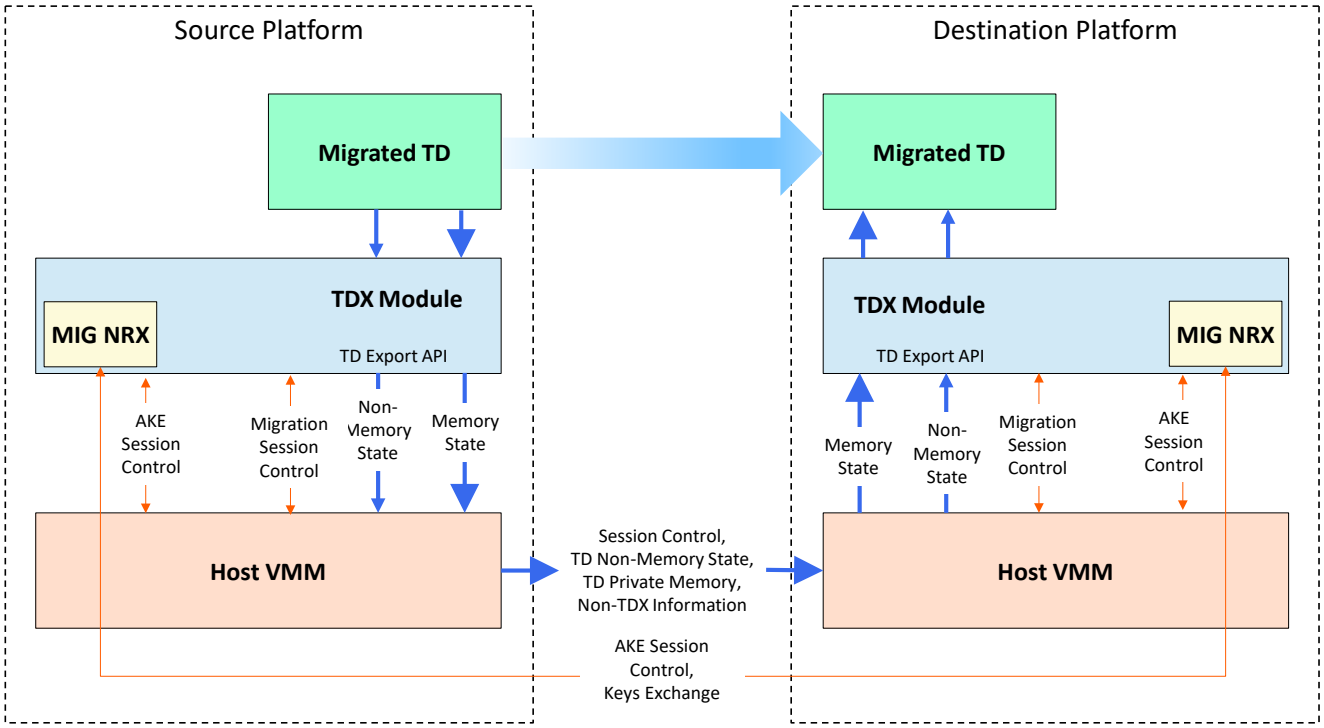


**Figure 3.1: Components Involved in TD Migration**

## 3.2. MIG NRX Requests and Sessions

For MIG NRX, each NRX request is a single TDH.MIG.SETUP host-side (SEAMCALL) interface function. The multiple-requests NRX session is a sequence of TDH.MIG.SETUP calls until TDX_SUCCESS or any error is returned. The exact number of the NRX requests within a single NRX session is implementation specific and depends on the internal state machine of a given NRX. For MIG NRX it among other things depends on the Authenticated Key Exchange (AKE) protocol being used to establish a secure session between a source and destination platform.

An example of such a state machine for MIG NRX is shown in Figure 3.2.

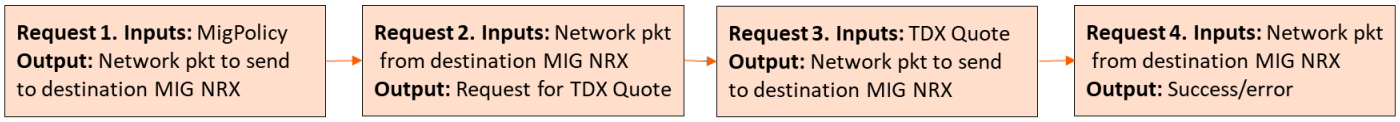


**Figure 3.2 Example state machine for MIG NRX processing of TDH.MIG.SETUP**

Depending on the number of the concurrent processing threads (VCPUs) allocated for MIG NRX, the NRX Requests for ongoing MIG NRX sessions are going to be distributed by the Intel TDX Module to any available VCPU from the VCPU pool at the time of servicing the request. Figure 3.3 shows an example of how multiple-requests NRX session for TDH.MIG.SETUP call is processed in case two VCPUs are used.

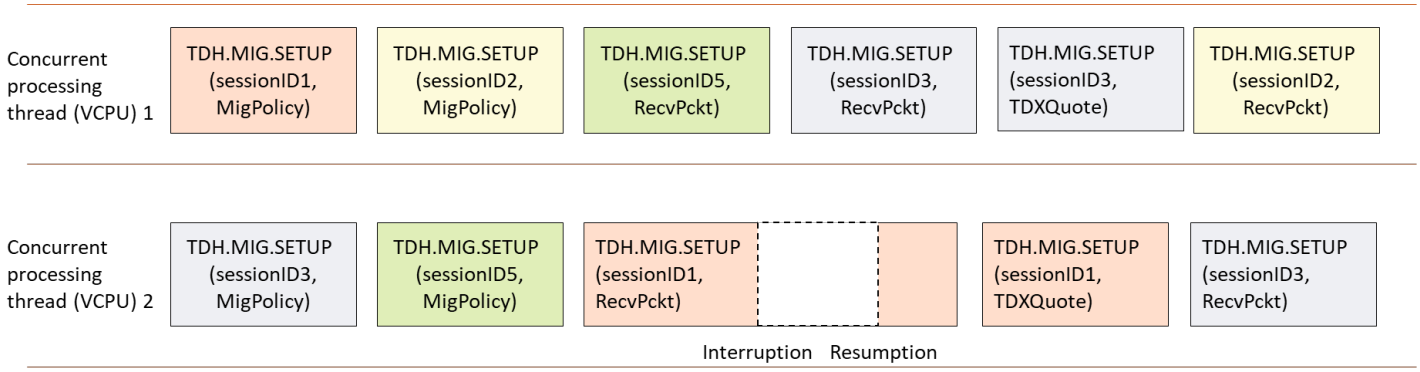


**Figure 3.3 Example for TDH.MIG.SETUP handling in two concurrent processing threads (VCPUs) case**

## 3.3. Migration Policy Binding to a TD

The migration policy is linked to a TD via the setting of a new constant **MIG_POLICY_HASH** via TDH.MNG.WR (see 4.3.1) prior to calling TDH.MR.FINALIZE. This field stays the same during the whole lifecycle of a TD, i.e. there is no way to update the initial migration policy that was bound to a TD during its creation.

For the Intel TDX security it is crucial that this value is included in the TD's attestation report and that is done by including it into the INIT_SERVTD_HASH field of the Intel TDX attestation reports as part of the Extended Service TD Info (SERVTD_EXT_STRUCT) structure. See Table 4.3 for the details of SERVTD_EXT_STRUCT structure in case when MIG NRX is used.

## 3.4. MIG NRX Initialization and Runtime Interaction

### 3.4.1. Initialization of MIG NRX by VMM

The initialization of MIG NRX happens in the same way as any other NRX as outlined in Section 2.2.

### 3.4.2. Runtime Preparation of the Migration Session

For Intel TDX migration to start, the following steps must be done:

1. The source and destination need to establish a secure session via an AKE protocol of their choice. This includes exchanging Intel TDX Quotes authenticating the source and destination platforms.
2. The source and destination need to verify that their respected migration policies allow the migration
3. The source and destination need to program the migration keys
4. The source and destination need to end the secure session

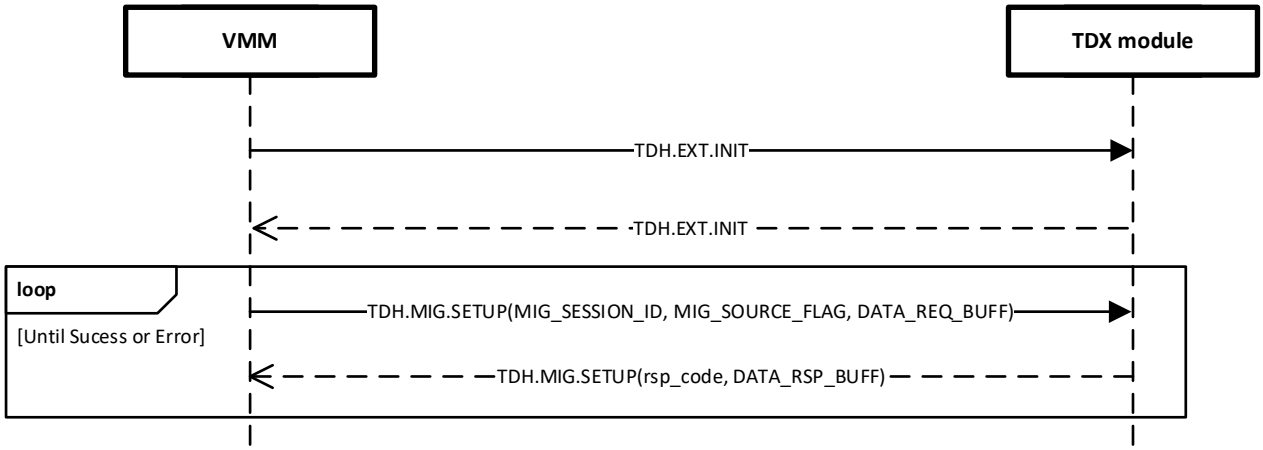


**Figure 3.4 TDH.MIG.SETUP runtime operation: VMM view**

All these steps are expected to happen via a single new **TDH.MIG.SETUP** host-side (SEAMCALL) interface function (see section 4.3.7 for detailed definition) and the overall process is shown in Figure 3.4Figure 3.4 TDH.MIG.SETUP runtime operation: VMM view. The VMM at both the source and destination invokes the TDH.MIG.SETUP, designating itself as the initiator and the remote end as the responder for the AKE protocol.

The initial content of the host input buffer contains the migration policy associated with the TD to be migrated. The migration policy is passed as it is and the TDH.MIG.SETUP host-side (SEAMCALL) interface function will verify that the SHA-384 hash of the migration policy matches the CUR_SERVTD_HASH of the TD to be migrated (see Table 4.3 for details of the field). In case of a mismatch, a TDX_MIG_POLICY_HASH_MISMATCH error code is returned.

At some point during execution of TDH.MIG.SETUP, both source and destination would need to request an Intel TDX Quote from the VMM. This is done by returning a TDX_MIG_GET_QUOTE completion status code and supplying the respected Intel TDX Report into the host output buffer identified by DATA_RSP_BUFF_LIST. Upon receiving this exit code, VMM is expected to extract the Intel TDX report from the response buffer and use existing ABIs to convert this Intel TDX Report into Intel TDX Quote and resume execution of TDH.MIG.SETUP providing the obtained Intel TDX Quote in the host input buffer identified by DATA_REQ_BUFF_LIST.

At the end of all above steps, the TDH.MIG.SETUP host-side (SEAMCALL) interface function terminates the secure session connection with the remote end and returns a TDX_SUCCESS completion status code. The Intel TDX Module also releases all associated resources for this TDH.MIG.SETUP NRX session.

In case an error happens, the TDH.MIG.SETUP host-side (SEAMCALL) interface function also terminates the secure session connection with the remote end, and corresponding error code is returned to VMM as indicated by Table 0. The Intel TDX Module also releases all associated resources for this TDH.MIG.SETUP NRX session.

### 3.4.3. Runtime Preparation of the Migration Session: Internal details

For informational purpose, Figure 3.5 TDH.MIG.SETUP runtime operation: Internal details shows the same flow as in Figure 3.4 TDH.MIG.SETUP runtime operation: VMM view, but with additional interactions between the Intel TDX Module and MIG NRX.

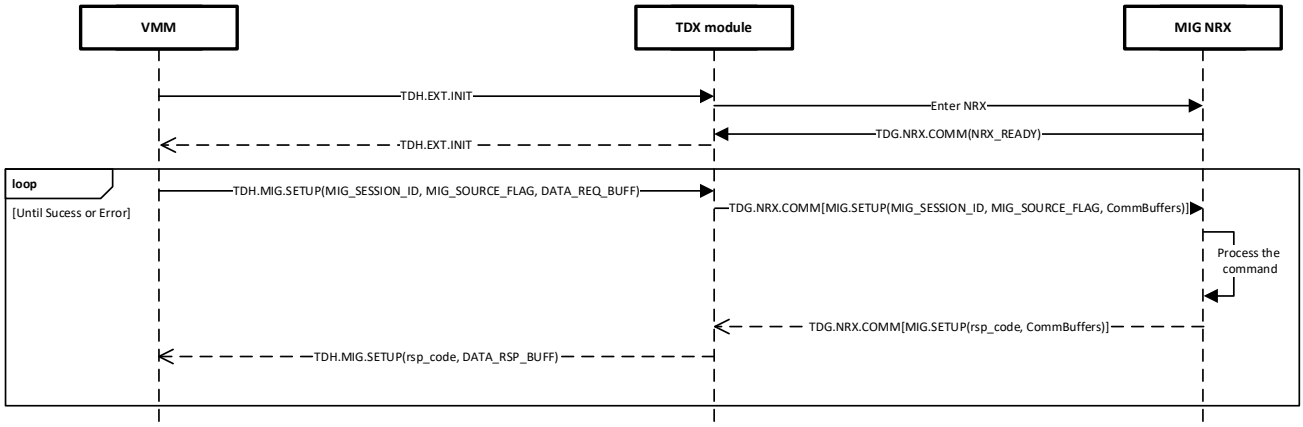


**Figure 3.5 TDH.MIG.SETUP runtime operation: Internal details**

## *3.5. MIG NRX Updates*

As was described in Section 2.3, in order to update any NRX, an Intel TDX Module TD preserving update is required. The flow is depicted in Figure 3.6 and it is safe to perform between each individual invocations of TDH.MIG.SETUP host-side (SEAMCALL) interface function since it is ensured that the relevant session state is preserved over a TD-preserving update.

### 3.5.1. MIG NRX session state management on SVN changes during a TD preserving update

Internally MIG NRX resets the NRX session state back to the first NRX request, resulting in NRX session to be started from the first request. This behavior is transparent to VMM, meaning no error indication is returned, it only results in more NRX requests that VMM has to make in the loop to complete an NRX session.

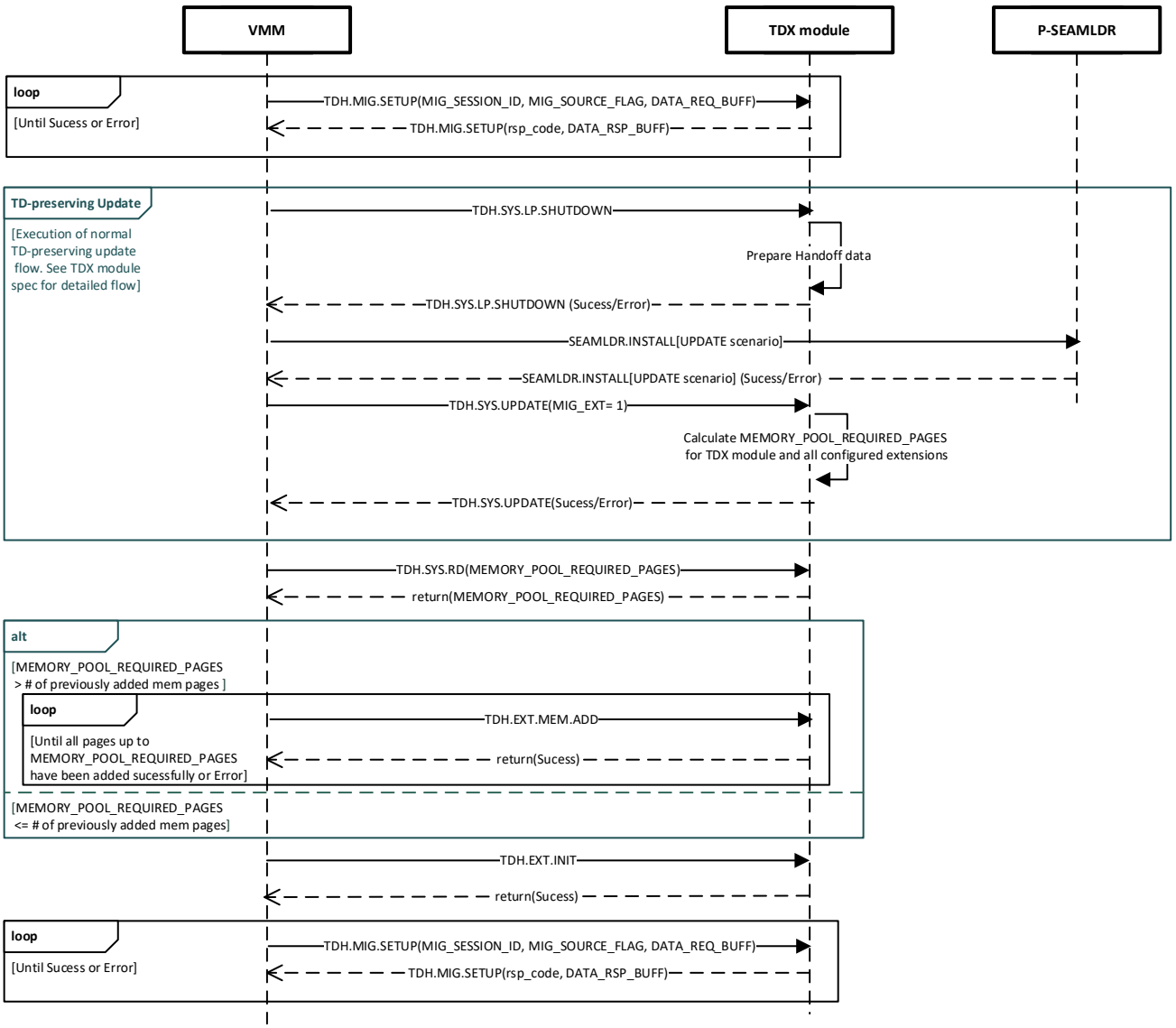


**Figure 3.6. Update of MIG NRX or Intel TDX Module via TD Preserving update: VMM view**

## *3.6. Fatal Error Handling in MIG NRX by VMM*

If MIG NRX experience a fatal error, the Intel TDX Module returns an TDX_EXT_FATAL_ERROR back to VMM as a result of ongoing host-side (SEAMCALL) interface function indicating that MIG NRX must be restarted via a TD preserving update. The TD preserving update flow in case MIG NRX experiences a fatal error is shown in Figure 3.7.

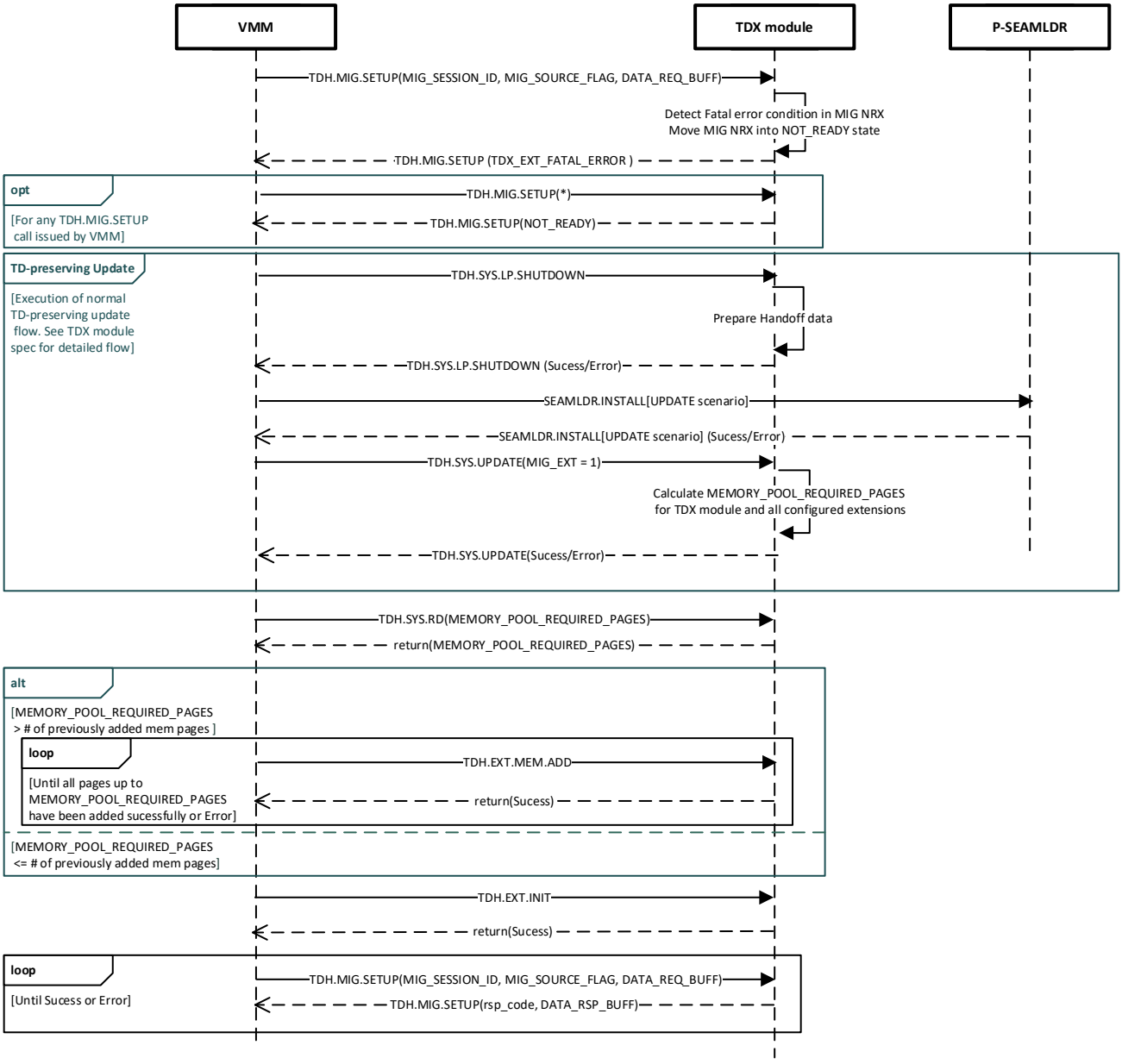


**Figure 3.7. Recovering of MIG NRX in case of the fatal error: VMM view**

# 4. Migration related Intel TDX Module Specification Changes

## 4.1. Intel TDX Module Enumeration & Configuration

A new bit will be added to TDX_FEATURES0: MIG_EXT (bit TBD), to enumerate Intel TDX Module support for the Intel TDX Module a Pre-Migration Feature.

**Table 4.1: TDX_FEATURES0 Definition**

| Bit(s) | Name | Description |
|---|---|---|
| … | … | … |
| TBD | MIG_EXT | The Intel TDX Module supports a Pre-Migration Feature |
| … | … | … |

## 4.2. Data Types

### 4.2.1. New: Data Buffer List

A data buffer list specifies a list of HPAs of 4KB pages in shared memory, to be used as output or input by TDH.MIG.SETUP. The list may have up to 512 64-bit entries, each containing a 4KB-aligned HPA (including HKID bits) of a page in shared memory. The list is contained in a single 4KB page and must be aligned on 4KB. The page list may contain null entries, indicated by the INVALID bit.

**Table 4.2: Data Buffers List Entry**

| Bits | Name | Description |
|---|---|---|
| 11:0 | **RESERVED** | Reserved: must be 0 (unless bit 63 indicates an invalid entry) |
| 51:12 | **HPA** | Bits 51:12 of the host physical address (including HKID) of the buffer page, which must be a shared HPA |
| 62:52 | **RESERVED** | Reserved: must be 0 (unless bit 63 indicates an invalid entry) |
| 63 | **INVALID** | A value of 1 indicates that this entry is invalid |

### 4.2.2. Update: SERVTD_EXT_STRUCT

**Table 4.3 SERVTD_EXT_STRUCT Definition in case MIG NRX is used**

| Name | Offset (Bytes) | Size (Bytes) | Type | Description |
|---|---|---|---|---|
| **INIT_SERVTD_HASH** | 0 | 48 | SHA384 | If TDX_FEATURES0. MIG_EXT (bit TBD) = 1, readable using TDH.SYS.RD*<br>MIG_POLICY_HASH of initial migration policy provided via TDH.MG.WR<br>else<br>Initial SERVTD_HASH |
| **INIT_SERVTD_ATTR** | 48 | 8 | MASK | If TDX_FEATURES0. MIG_EXT (bit TBD) = 1, readable using TDH.SYS.RD*<br>0 |

| Name | Offset (Bytes) | Size (Bytes) | Type | Description |
|---|---|---|---|---|
| | | | | else<br>value for INIT_SERVTD_ATTR |
| **RESERVED** | 56 | 8 | | Must Be Zero |
| **INIT_CPUSVN** | | 16 | | TD's initial CPUSVN from creation. |
| **INIT_TEE_TCB_SVN** | | 16 | | TD's initial TEE_TCB_SVN from creation. |
| **INIT_TEE_MODEL** | | 12 | | Model information corresponding to the model that INIT_TEE_TCB_SVN was captured on.<br>All zeros is reserved to reflect inter-model migration is not supported, the INIT_TEE_TCB_SVN was captured on the same model as the Quote that it's contained within. |
| **RESERVED** | | 4 | | |
| **CUR_SERVTD_HASH** | | 48 | SHA384 | If TDX_FEATURES0. MIG_EXT (bit TBD) = 1, readable using TDH.SYS.RD*<br>MIG_POLICY_HASH of currently enforced migration policy. The initial value of this field must be equal to the value in the INIT_SERVTD_HASH field<br>else<br>Current SERVTD_HASH |
| **CUR_SERVTD_ATTR** | | 8 | MASK | If TDX_FEATURES0. MIG_EXT (bit TBD) = 1, readable using TDH.SYS.RD*<br>0<br>else<br>value for CUR_SERVTD_ATTR |
| **RESERVED** | | 8 | | Must Be Zero |
| **RESERVED** | | 48 | SHA384 | Reserved for Cascading hash of Audit Log |
| **RESERVED** | | 48 | | Must be Zero |

#### 4.2.3. New: Constants

| Constant Name | Description |
|---|---|
| MIG_POLICY_HASH | This is a per TD scope field (see TDH.MNG.RD/TDH.MNG.WR) VMM can read or set that reflects the SHA-384 hash of the migration policy associated with given TD. Can be only set until TDH.MR.FINALIZE is called. |
| MIG_EXT_MAX_OVERALL_SESSIONS | This is a Global Field (see TDH.SYS.RD) VMM shall read in order to determine the maximum number of sessions that MIG NRX can support |

| Constant Name | Description |
|---|---|
| MIG_EXT_MAX_CONCURRENT_THREADS | This is a Global Field (see TDH.SYS.RD) VMM shall read in order to determine the maximum number of concurrent threads that MIG NRX can support |
| MIG_EXT_OVERALL_SESSIONS | This is a Global Field (see TDH.SYS.RD) VMM shall read in order to determine the currently configured number of sessions that MIG NRX is using. VMM can also write the selected value into this field using TDH.SYS.WR. The value must be less or equal to MIG_EXT_MAX_OVERALL_SESSIONS |
| MIG_EXT_CONCURRENT_THREADS | This is a Global Field (see TDH.SYS.RD) VMM shall read in order to determine the currently configured number of concurrent threads that MIG NRX is using. VMM can also write the selected value into this field using TDH.SYS.WR. The value must be less or equal to MIG_EXT_MAX_CONCURRENT_THREADS |

## 4.3. Host-Side (SEAMCALL) Functions

### 4.3.1. Update: TDH.MNG.RD/TDH.MNG.WR

A new per TD field MIG_POLICY_HASH is added, which is 48 bytes that can be set by VMM for each TD before calling TDH.MR.FINALIZE. The field contains SHA-384 hash of migration policy associated with a given TD. The field is initialized by Intel TDX Module to zero and for non-migratable TDs the value of this field remains zero and invocation of TDH.MNG.WR is not required. After TDH.MR.FINALIZE has been called, the invocation of TDH.MNG.WR on this field must return an error as this field is read only.

The field can be read using TDH.MNG.RD at all times.

### 4.3.2. Update: TDH.SYS.RD

A new global field MIG_EXT_MAX_OVERALL_SESSIONS is added, which returns the maximum number of sessions that a MIG NRX can support. Calculated according to build time data provided by MIG NRX.

*Note: the returned value always has the correct values as soon as Intel TDX Module has been built with MIG NRX TD. The returned value doesn't depend on whenever the MIG NRX has been successfully initialized or not.*

### 4.3.3. Update: TDH.SYS.RD

A new global field MIG_EXT_MAX_CONCURRENT_THREADS is added, which returns the maximum number of concurrent threads that MIG NRX can support. Calculated according to build time data provided by MIG NRX.

*Note: the returned value always has the correct values as soon as Intel TDX Module has been built with MIG NRX TD. The returned value doesn't depend on whenever the MIG NRX has been successfully initialized or not.*

### 4.3.4. Update: TDH.SYS.RD/WR

A new global field MIG_EXT_OVERALL_SESSIONS is added, which returns currently configured number of sessions that MIG NRX is using. The initial value for this field equals to MIG_EXT_MAX_OVERALL_SESSIONS. VMM can use TDH.SYS.WR to change this value to less or equal to MIG_EXT_MAX_OVERALL_SESSIONS.

*Note: the returned value always has the correct values as soon as Intel TDX Module has been built with MIG NRX TD. The returned value doesn't depend on whenever the MIG NRX has been successfully initialized or not.*

### 4.3.5. Update: TDH.SYS.RD/WR

A new global field MIG_EXT_CONCURRENT_THREADS is added, which returns currently configured number of concurrent threads that MIG NRX is using. The initial value for this field equals to MIG_EXT_MAX_CONCURRENT_THREADS. VMM can use TDH.SYS.WR to change this value to less or equal to MIG_EXT_MAX_CONCURRENT_THREADS.

*Note: the returned value always has the correct values as soon as Intel TDX Module has been built with MIG NRX TD. The returned value doesn't depend on whenever the MIG NRX has been successfully initialized or not.*

### 4.3.6. Update: TDH.EXT.MEM.ADD

The leaf internal behavior is changed to allow host VMM to invoke this leaf even after the TDH.EXT.INIT has been called. This is required to support memory pre-allocation before TD-preserving update. For details see Section 3.5.

### 4.3.7. New: TDH.MIG.SETUP Leaf

Prepare for the migration session.

***Input Operands***

**Table 5.1: TDH.MIG.SETUP Input Operands Definition**

| Operand | Name | Description | | |
|---|---|---|---|---|
| RAX | Leaf and Number | SEAMCALL instruction leaf number and version | | |
| | | **Bits** | **Field** | **Description** |
| | | 15:0 | Leaf Number | Selects the SEAMCALL interface function |
| | | 23:16 | Version Number | Selects the SEAMCALL interface function version |
| | | 63:24 | Reserved | Must be 0 |
| RCX | | The physical address of the target TD's TDR page (HKID bits must be 0) | | |
| RDX | Control parameters | Migration session ID and direction of AKE | | |
| | | **Bits** | **Field** | **Description** |
| | | 31:0 | MIG_SESSION_ID | Session ID for the migration establishment session |
| | | 32:32 | MIG_SOURCE_FLAG | Direction of AKE:<br>REQUESTOR – 0<br>RESPONDER – 1 |
| | | 63:33 | Reserved | Must be 0 |
| R8 | DATA_RSP_BUFF_LIST | HPA (including HKID bits) of a buffer list in shared memory, containing host input buffer. See section 4.2.1 New: Data Buffer List. | | |
| R9 | DATA_RSP_BUFF_SIZE | The size of host input buffer in bytes | | |
| R10 | DATA_REQ_BUFF_LIST | HPA (including HKID bits) of a buffer list in shared memory, containing host output buffer. See section 4.2.1 New: Data Buffer List. | | |

### *Output operands*

**Table 5.2: TDH.MIG.SETUP Output Operands Definition**

| Operand | Description |
|---|---|
| RAX | SEAMCALL instruction return code |
| RCX | If RAX returns TDX_MIG_AKE_REQUEST – Generated MIG output message length in bytes.<br>If RAX returns TDX_MIG_GET_QUOTE – The size of provided Intel TDX Report in bytes.<br>Zero otherwise. |
| Other | Unmodified |

### *Leaf Function Description*

**Note:** The description below is provided at a high level. Actual details, order of checks, returned status codes, etc. may vary.

#### Overview

TDH.MIG.SETUP prepares for the migration session, including securely programming the migration keys between the source and destination.

#### Enumeration

Support of TDH.MIG.SETUP is enumerated by TDX_FEATURES0.MIG_EXT (bit TBD), readable by TDH.SYS.RD*.

#### Interruptibility

TDH.MIG.SETUP is interruptible. If a pending interrupt is detected during operation, TDH.MIG.SETUP returns with a TDX_INTERRUPTED_RESUMABLE status in RAX. Rest of details are TBD

### *Operands Information*

TBD

### *Completion Status Codes*

**Table 5.4: TDH.MIG.SETUP Completion Status Codes (Returned in RAX) Definition**

| Completion Status Code | Description |
|---|---|
| TDX_OPERAND_BUSY | Operation encountered a busy operand, indicated by the lower 32 bits of the status. In many cases, this can be resolved by retrying the operation. |
| TDX_OPERAND_INVALID | |
| TDX_SUCCESS | Operation is successful |
| TDX_LIMIT_CPUID_MAXVAL_SET | IA32_MISC_ENABLES MSR bit 22 (Limit CPUID Maxval) is set. |
| TDX_INCONSISTENT_MSR | IA32_TSC_ADJUST MSR value is different than the value captured during the TDH.SYS.INIT interface function. |
| TDX_TSC_ROLLBACK | Time Stamp Counter value is lower than on last TD exit. |
| TDX_INTERRUPTED_RESUMABLE | Interrupt is pending, the operation can be resumed |
| TDX_EXT_NOT_READY | |
| TDX_EXT_NOT_INITIALIZED | |

| Completion Status Code | Description |
|---|---|
| TDX_EXT_FATAL_ERROR | |
| TDX_MIG_POLICY_HASH_MISMATCH | The provided policy SHA-384 hash does not match the MIG_POLICY_HASH associated with TD to be migrated |
| TDX_MIG_AKE_REQUEST | Intel TDX Module requires VMM to send the AKE request to the destination platform in RSP_PA with length |
| TDX_MIG_GET_QUOTE | Intel TDX Module requires VMM to obtain the Intel TDX Quote from the Intel TDX report in RSP_PA with length |
| TDX_MIG_AKE_FAILURE | The AKE exchange failed |
| TDX_MIG_POLICY_FAILURE | The migration policy check failed, migration is not allowed |
| TDX_MIG_QUOTE_INVALID | The Intel TDX Quote provided by VMM is invalid |