# Trust Domain Extension (TDX) Migration TD Design Guide

***October 2021***

Document Number: 348987-001

Document Number: 348987-001

# Contents

## Figures

## Tables

# 1 Introduction

## 1.1 Background

Analogous to VM migration, a cloud-service provider may want to relocate/migrate an executing Trust Domain (TD) from a **source Trust Domain Extension (TDX) platform** to a **destination TDX platform** in the cloud environment. A cloud provider may use TD migration to meet customer Service Level Agreement (SLA), while balancing cloud platform upgradability, patching and other serviceability requirements. A TD runs in a CPU mode that protects 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 (the host VMM) to migrate TDs across compatible platforms. As the figure below shows, the TD typically will be assigned a different key (and will be always assigned a different ephemeral key) on the destination platform chosen to migrate the TD.

Figure 1-1 shows the TD Migration use case.

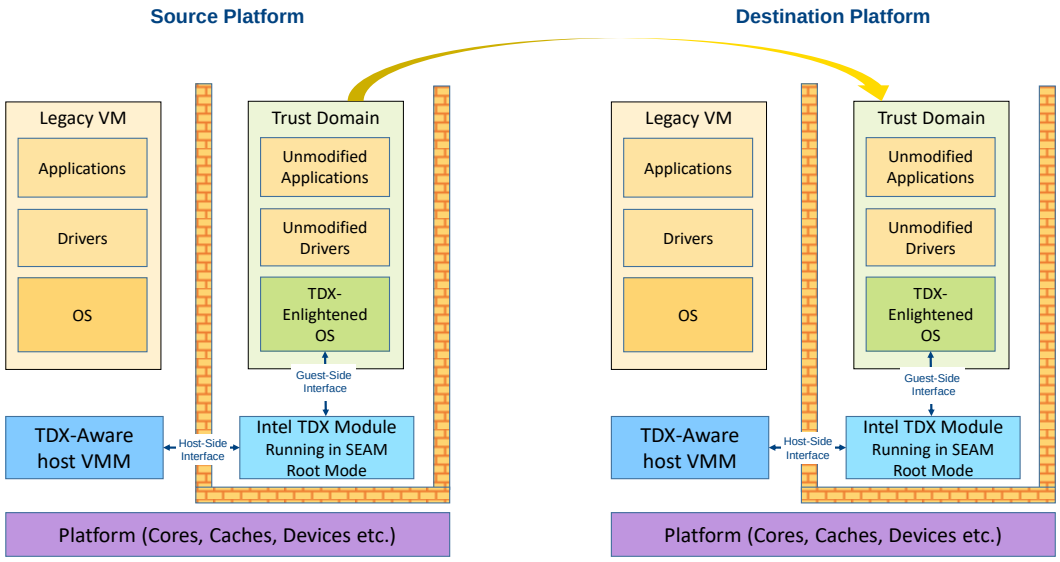


**Figure 1-1: TD Migration**

## 1.2 Overview

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 **Migration TD (**as known as **MigTD)** – which is used to

perform common functions for migrating TDs. TD Migration does NOT depend on any interaction with the TD guest software operating inside the TD being migrated.

TD contents is primarily protected and transferred by Intel TDX module using a **Migration Session Key (MSK)** used only for a unique Migration session for a TD.

A **Migration TD (MigTD)** is used to:
1. Evaluate potential migration sources and targets for adherence to the TD Migration Policy.
2. If approved, securely transfer a Migration Session Key from the source platform to the destination platform to migrate assets of a specific TD.

The host VMM need bind a MigTD to one (or more TDs) via a new **SEAMCALL [TDH.SERVTD.BIND]** or **SEAMCALL [TDH.SERVTD.PREBIND]**. The host VMM via the Intel TDX module is responsible for export/import of the TD content. The host VMM and existing untrusted SW stack responsible for migrating the encrypted TD content.

Since the MigTD is in the TCB of the TD being migrated, a MigTD must be pre-bound to the target TD being migrated before the target TD measurement is finalized. The MigTD lifecycle does not have to be coincidental with the target TD. The MigTD may be instantiated when required for Live Migration, and must be bound to the target TD before Live Migration can begin. MigTD must be operational until the MSK has been successfully programmed for the target TD being migrated.

Figure 1-2 shows the TD Migration Architecture.

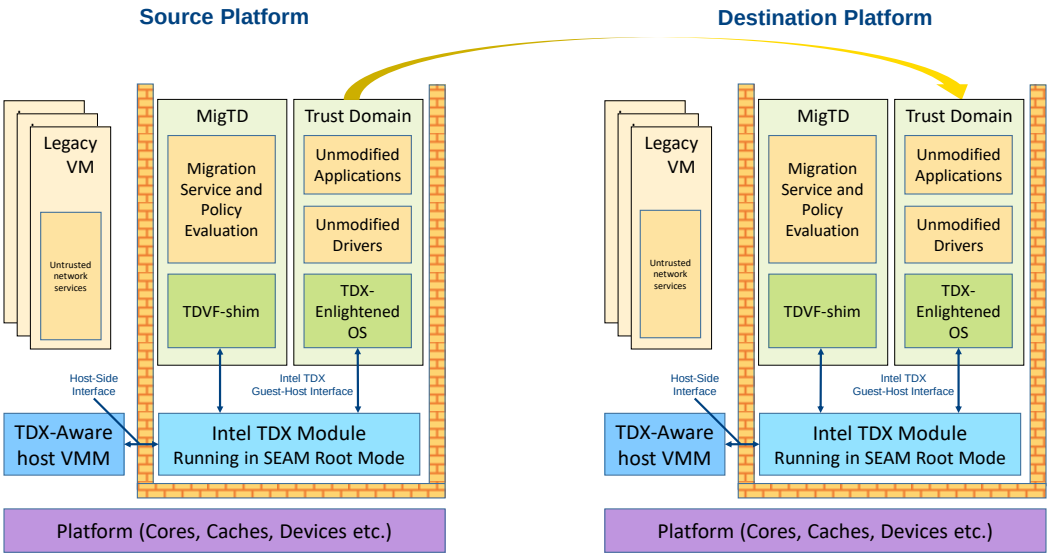


**Figure 1-1: TD Migration Architecture**

The full TD migration can be found at TDX TD Migration specification. This document only describes the software architecture for Migration TD.

## 1.3 Terminology

| Term | Description |
|---|---|
| GCM | Galois/Counter Mode |
| GKID | Guest Key ID |
| HKID | Host Key ID |
| HOB | Hand-off Block |
| MigTD | Migration TD |
| MigTD-s | Migration TD on source platform |
| MigTD-d | Migration TD on destination platform. |
| MKTME | Multi-Key Total Memory Encryption |
| MSK | Migration Session key |
| MTK | Migration Transport Key |
| PAMT | Physical Address Metadata Table |
| SEAM | Secure Arbitration Module |
| TCB | Trust Computing Base |
| TD | Trust Domain |
| TDVF | Trust Domain Virtual Firmware |
| TDX | Trust Domain Extension |
| TLS | Transport Layer Security |
| RA-TLS | Remote Attestation TLS |
| TME | Total Memory Encryption |
| VKMT | Virtual Key ID Mapping Table |

§

# 2 Migration Architectural Overview

## 2.1 TD Migration Orchestration

Figure 2-1 shows the TD Migration Orchestration Flow.

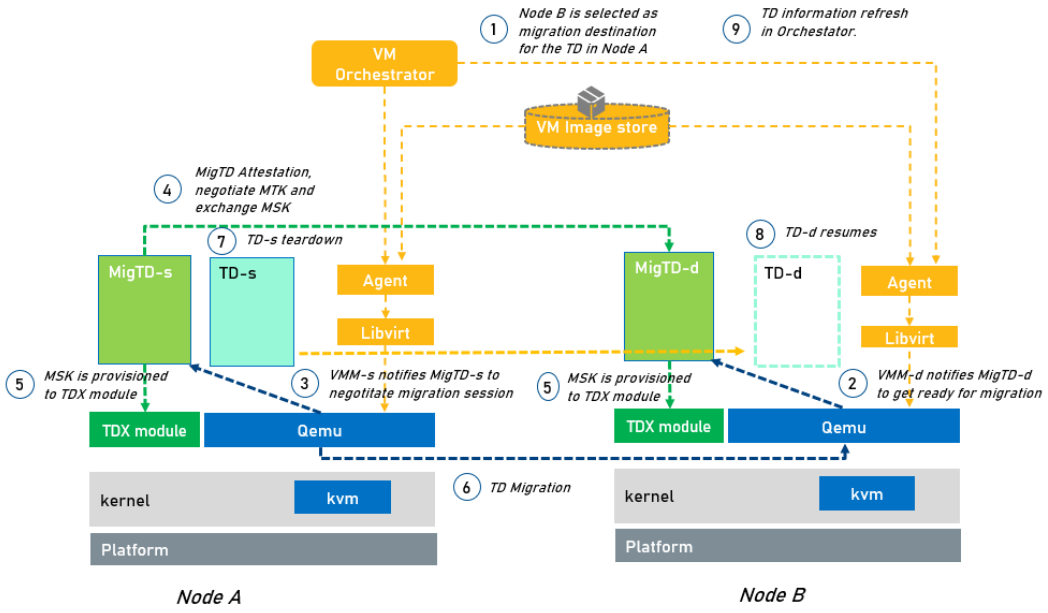


**Figure 2-1: TD Migration Orchestration Flow**

Assuming a VM orchestrator manages Node A and Node B. The VM orchestrator decides to migration the TD in Node A to Node B. The Node A is migration source, and the Node B is migration destination. The component in Node A is named as module-s, where s means source, and the component in Node B is named as module-d, where d means destination.

The high level TD migration flow is below:

1) The VM orchestrator selects node B as the migration destination.

2) The VMM-d in Node B notifies the MigTD-d to get ready for migration.

3) The VMM-s in Node A notifies the MigTD-s to negotiate the migration session.

4) MigTD-s and MigTD-d negotiate with each other, including mutual attestation, Migration Policy evaluation, negotiate Migration Transport Keys (MTK) and exchange the Migration Session Key (MSK).

5) MigTD-s provision MSK to TDX module in source platform. MigTD-d provision MSK to TDX module in destination platform.

6) TD-s in Node A is migrated to TD-d in Node B, with help of VMM and QEMU.

7) TD-s is teardown by the VMM-s.

8) TD-d is resumed by the VMM-d.

9) The migration is done. The orchestrator refreshes the TD information.

## 2.2 Keys used in migration

Two types of keys are used in MigTD.

- **Migration Transport Keys (MTK)**: They are a set of keys generated during MigTD negotiation between source platform and destination platform. We call MigTD on source platform as **MigTD-s**, and MigTD on destination platform as **MigTD-d**. These keys are used to protect the communication message between MigTD-s and MigTD-d. If MigTD-s and MigTD-d use a network Transport Layer Security (TLS) protocol to set up a secure session, then the MTKs are the TLS session keys.
- **Migration Session Key (MSK)**: An AES-256-GCM key generated by MigTD-s and shared with MigTD-d. The GCM key is protected by the MTK when it is transported from MigTD-s to MigTD-d. The MSK is transferred to TDX-Module by **TDCALL<TDG.SERVTD.WR>**, and is used to protect the data migration between TDX-Module-s and TDX-Module-d. The MSK protects the TD private data and is used for export and import of the TD confidential assets.

Figure 2-1 shows the keys used in migration setup.

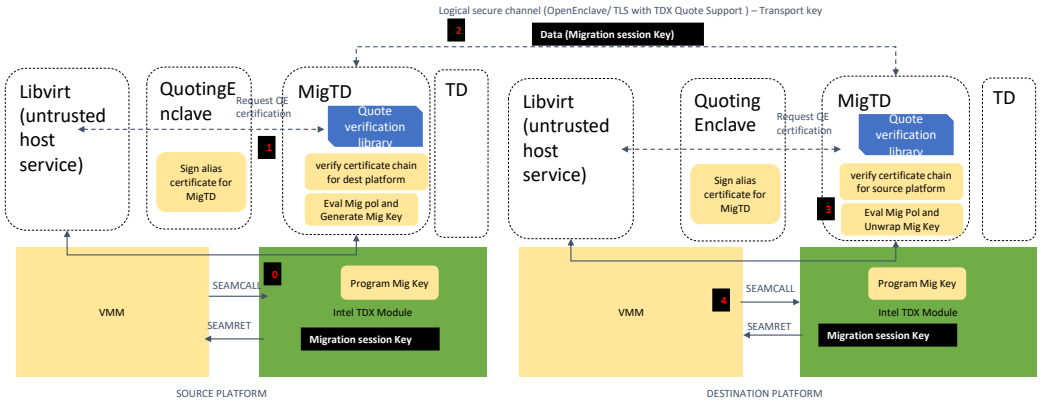


**Figure 2-2: Migration Key Setup**

## 2.3 MigTD General Flow

A MigTD can be launched by VMM at any time, when VMM starts migration. Once the MigTD transfers the MSK to TDX module, the MigTD can be tear down.

See below as a typical flow:

- [MigTD-s/MigTD-d] are launched by VMM, when VMM starts migration.
- The host VMM binds a [MigTD-s/MigTD-d] to one (or more TDs) via a new **SEAMCALL [TDH.SERVTD.BIND].** It gets back a binding handle. The binding handle will be part of input parameter for the migration information.
- [MigTD-s/MigTD-d] build secure channel to each other, e.g. via remote attestation TLS (RA-TLS) protocol.
    - [MigTD-s/MigTD-d] communicate with each other via network interface, such as a VirtIo-socket interface.
    - [MigTD-s/MigTD-d] generate local QUOTE via TDG.VP.VMCALL<GET_QUOTE> and pass the QUOTE to the peer in the TLS certificate.
    - [MigTD-s/MigTD-d] do mutual authentication, via peer QUOTE attestation.
    - Once RA-TLS authentication is done, [MigTD-s/MigTD-d] can derive a set of TLS session keys as the **Migration Transport Keys (MTK)**. Then the secure channel is set up.
- [MigTD-s/MigTD-d] check the Migration Policy.
    - Within the secure session, [MigTD-s/MigTD-d] can check the Migration Policy to see if the peer is allowed to migrate.
- [MigTD-s/MigTD-d] setup **Migration Session key (MSK)**
    - [MigTD-s] generates MSK and transfers to peer via the secure channel.
    - [MigTD-d] receives MSK from peer.
    - [MigTD-s/MigTD-d] set MSK via **TDCALL<TDG.SERVTD.WR>**
- [MigTD-s/MigTD-d] can be tear down.

Figure 2-2 shows the general Migration TD flow.

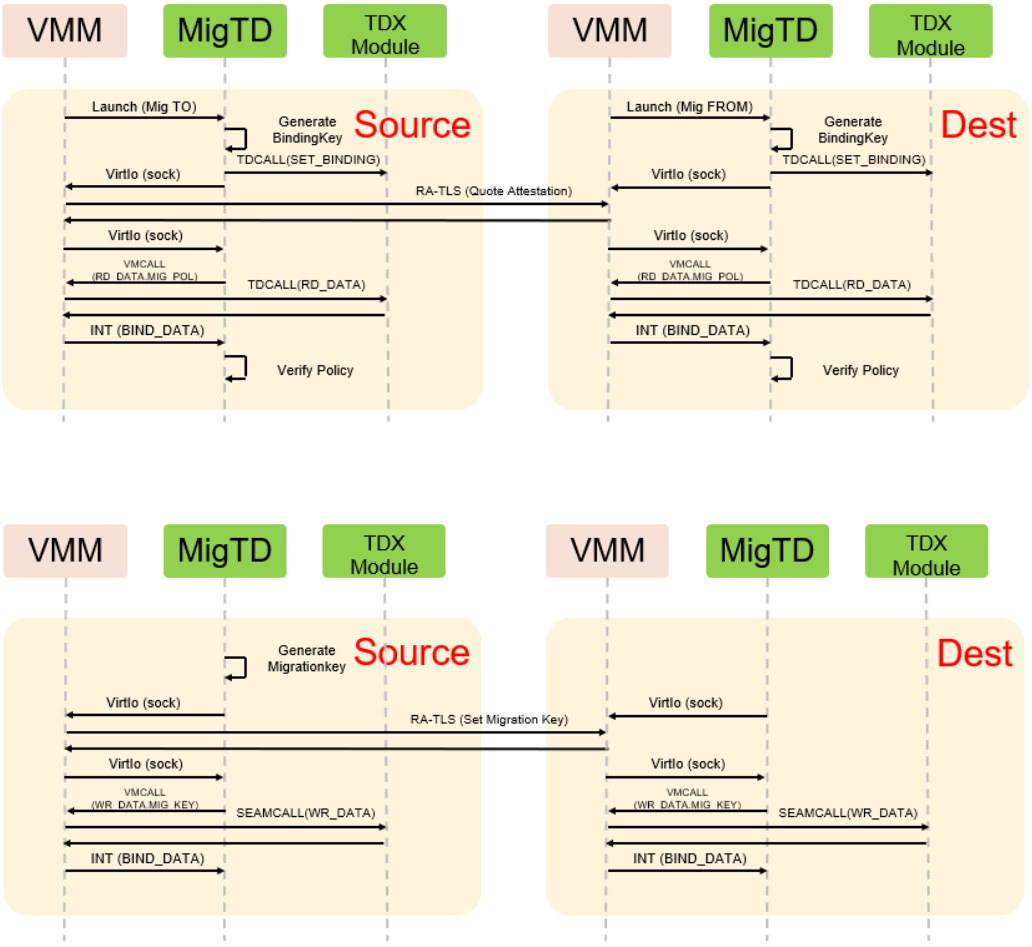


**Figure 2-3: Migration TD flow**

§

# 3 MigTD Design Overview

## 3.1 Design Overview

Migration TD is a lightweight bare-metal service TD. Because MigTD is in TCB, the code in MigTD should be as minimal as possible.

### 3.1.1 MigTD design

Figure 3-1 shows the design of MigTD. It includes a shim layer and a core.

- **TdShim** is a generic shim layer to boot any service TD, as a transient environment.
    - TdShim includes a **Reset Vector** that is the first instruction when a VMM launches a TD.
- **Core** is the Migration TD runtime execution environment.
    - **Crypto** and **TLS** are used to establish a secure communication session. It should reuse the existing known good crypto library and TLS configuration including version and cipher suite.
    - **VSocket** is a way to pass the TLS packet to the VMM and leverage the VMM network stack to communicate with peer MigTD.
    - **VMM communication** should follow the Guest-Host Communication Interface (GHCI) specification.
    - **Quote Attestation service** is used for the mutual authentication in Remote-Attestation TLS.
    - **Migration Policy** is used to evaluate the migration peer according to the predefined policy.

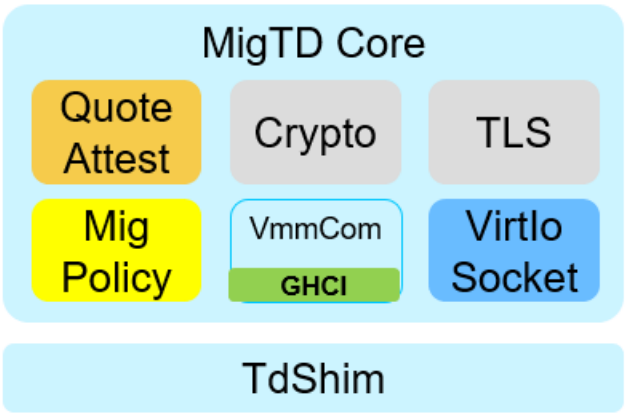

Figure 3-1: Migration TD Design

### 3.1.2 MigTD layout

Figure 3-2 shows the build time layout and runtime layout.

The left-hand side shows the build time layout. The Boot Firmware Volume (BFV) include the TdShim and migration core.

The right-handle side shows the runtime layout. The TdShim should be loaded to the top of 4GB memory where the reset vector sits. Then the TdShim will relocate the MigTD core to low memory space and setup x64 long mode environment. This approach is highly recommended because the top of 4GB is usually mapped to a flash device. Executing code in flash device may bring some special restriction even in virtualization environment. All reset additional memory below the top of memory (TOM) can be used as heap, and the stack can be allocated from the heap. Avoid using the memory below 1MB, because the top of 1MB is also mapped to legacy flash ROM.

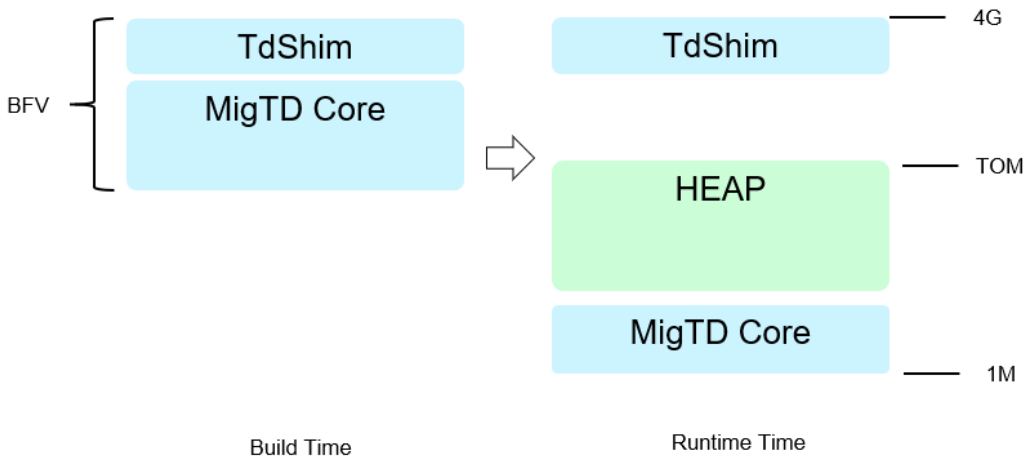


Figure 3-2: Migration TD Layout

### 3.1.3 MigTD boot flow

Figure 3-3 shows the Migration TD boot flow.

1. **ResetVector in TdShim**. The reset vector code should park all application processors (APs) to a known place and only let bootstrap processor (BSP) to initialize the environment. The BSP should switch to long mode, setup stack and jump to the TdShim initial program loader (IPL).

2. **IPL in TdShim**. When TdShim is launched, it should get the memory information internally, such as metadata area. Then it accepted the TD memory and set up all required protection such as data execution prevention (DEP). The final step is to find the MigTD core, load it to low memory and jump to the MigTD core.

3. **Entrypoint of MigTD Core**. The MigTD should initialize the final execution environment including heap and interrupt vector etc. It should also initialize the virtio-socket driver to prepare the communication with VMM.

4. **MigTD Core runtime**. The MigTD should set up secure communication session with peer MigTD. MigTD should send and receive the TLS packet via network socket interface, such as VirtIo-socket. After the secure session is established, the MigTD-s will generate the Migration Session key and pass to MigTD-d. Then both MigTD can pass the key to the TDX module. After that, the MigTD can tear down.

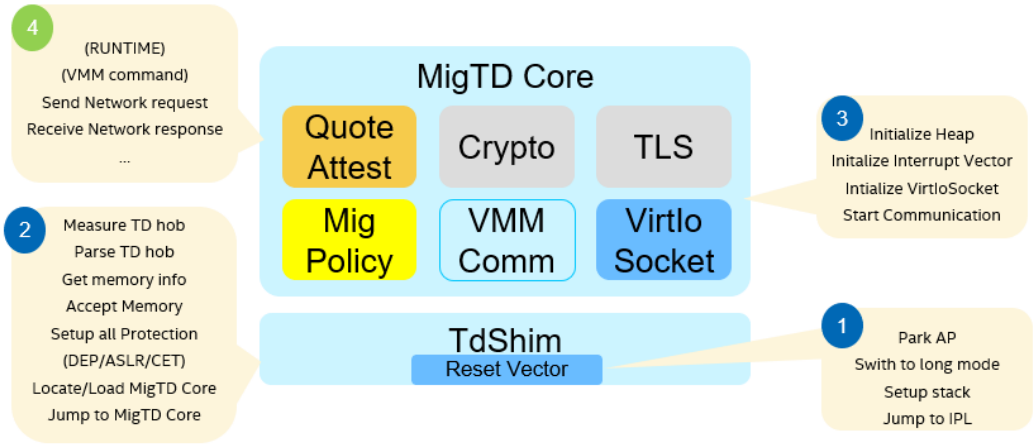


**Figure 3-3: Migration TD Boot Flow**

## 3.2 Reproducibility

A service TD binary should be reproducible, meaning you can recreate the same **service-TD release binary**, with the same service-TD source code version and same compiler version. This condition should be true under any circumstances, anytime, on any machine, and any source path location.

### 3.2.1 Challenge

We are aware of challenges today. For example, some code may use __FILE__, __LINE__, __DATE__, __TIME__ as the build-in information. The __FILE__ may change based upon the file location. __DATA__ and __TIME__ may change based upon the build time.

The compiler may generate debug entry, including file path. The compiler may generate time stamp on when the binary is created. In addition, the compiler may include unique GUID to provide detailed debugging information.

Refer to Microsoft PE/COFF specification.

### 3.2.2 Solution

To resolve above challenge, we have below recommendation:

#### 3.2.2.1 Code

The code should use __FILE__, __LINE__, __DATE__, __TIME__ only for debug purpose.

If present, the info should be included in DEBUG macro and removed in release build.

#### 3.2.2.2 Compiler Option

Don't generate any debug info in release build.

#### 3.2.2.3 Additional tools

The additional tool should be created to zero the debug data fields in the PE image, and zero the time stamp and unique GUID.

For example, EDKII project includes a `GenFW` tool that has -z, --zero option to zero such information.

## 3.3 Security Consideration

### 3.3.1 Type safe language

The MigTD should consider use type safe language, such as Rust programming language to eliminate memory safety issue.

### 3.3.2 Data Execution Protection (DEP)

The MigTD should set up DEP to mark data region (including stack and heap) to be non-executable, and code region be read-only, to prevent code injection attack.

### 3.3.3 Address Space Layout Randomization (ASLR)

The MigTD should use ASLR for the heap / stack allocation. The effectiveness of ASLR is based upon random bit entropy. To be realistic, if the MigTD only has limited memory resource, then the entropy bit cannot be too large. But we can at least shift some bits, which is better than nothing.

### 3.3.4 Control Flow Enforcement Technology (CET)

#### 3.3.4.1 Backward Control Flow

The MigTD should set up CET Shadow Stack (SS) protection to prevent Return Oriented Programming (ROP) attack.

#### 3.3.4.2 Forward Control Flow

The MigTD should set up CET Indirect Branch Tracking (IBT) to prevent a Jump Oriented Programming (JOP) attack, depending upon the compiler's capability. See [rust CET].

### 3.3.5 Side Channel

The MigTD should also consider side-channel attack to steal the MSK or MTK.

#### 3.3.5.1 Bound Check Bypass

The MigTD should use LFENCE before parsing any VMM input data, because the negotiated key is secret.

#### 3.3.5.2 Branch Target Injection

The MigTD should use Indirect Branch Predictor Barrier (IBPB) by WRMSR(IA32_PRED_CMD.IBPD), which is needed in an RA-TLS context switch if a migration TD supports multiple TLS connections.

#### 3.3.5.3 Rogue Data Cache load

N/A. MigTD is a bare metal execution environment. All code runs in supervisor mode.

#### 3.3.5.4 Rogue System Register load

N/A.

#### 3.3.5.5 Speculative Store Bypass

N/A.

# 4 MigTD Mutual Authentication

Two MigTDs use mutual authentication to verify each other when they build the secure communication channel.

## 4.1 Remote Attestation TLS

The network TLS protocol is a standard to allow two entities create a secure session. A typical mutual authentication in TLS requires X.509 certificate provision. However, it is hard to provision a private key to MigTD. A way to resolve this problem is to use remote attestation TLS (RA-TLS).

RA-TLS does note require private key or public certificate provision. The MigTD can generate a keypair at runtime and include the public key in the TD report data and TD quote data. Then the MigTD generates an X.509 certificate at runtime, includes the TD Quote in the X.509 certificate, and send this TD certificate to peer as the TLS certificate. When the peer MigTD receives the certificate, it gets the TD Quote, verifies the Quote, then it can trust the public key. Finally, the peer MigTD can use the public key to verify rest TLS messages.

Figure 4-1 shows the MigTD with RA-TLS.

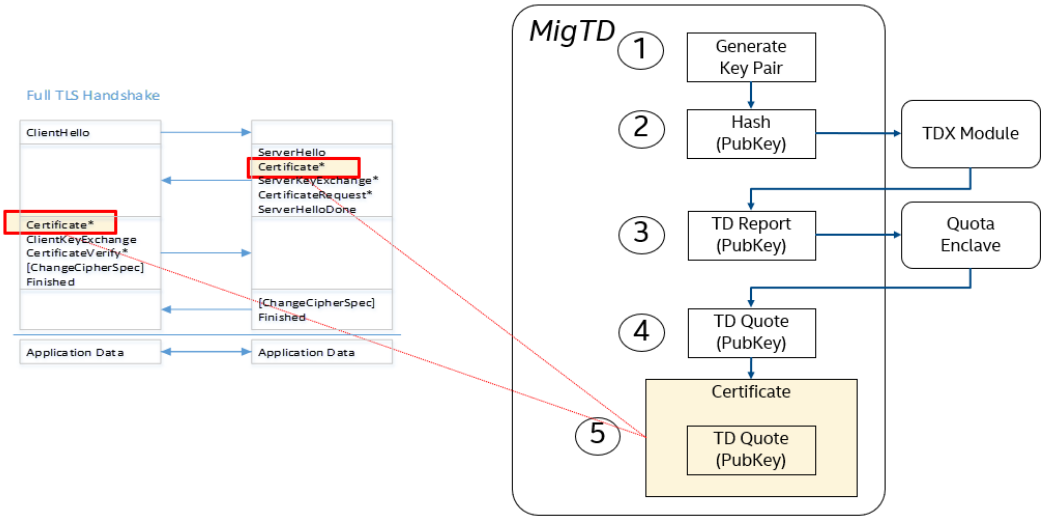


**Figure 4-1: MigTD with RA-TLS**

For more detail of RA-TLS, refer to **[Intel RA-TLS]** or **[Open Enclave RA-TLS].**

## 4.2 Attestation Flow

The MigTD attestation flow is similar to the normal TD attestation flow. Figure 4-2 shows the flow to get TD Quote in MigTD.

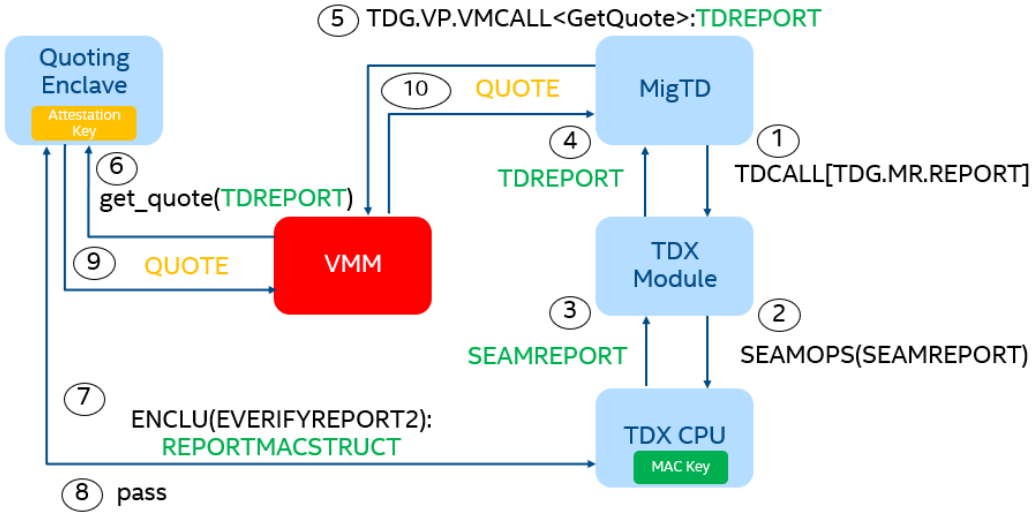


**Figure 4-2: MigTD GetQuote Flow**

With step 1~4, the MigTD gets a TDREPORT for itself. The integrity of TDREPORT is guaranteed by the REPORTMACSTRUCT, which is MACed with TDX CPU with a unique MAC key. Figure 4-3 shows the TDREPORT structure.

In step 5~6, the MigTD uses the TDG.VP.VMCALL<GetQuote> and try to get the Quote from the Quoting Enclave (QE).

In step 7~8, the QE verifies the MAC of the TD report with the TDX CPU. This verification is needed because Quoting is requested from the VMM and is not trusted.

In step 9~10, if the verification passed, then the QE signs the TDREPORT with its attestation key to generate the TD QUOTE and returns it to the MigTD via the VMM.

Now the MigTD can include the TD Quote in the TLS certificate and send to peer.

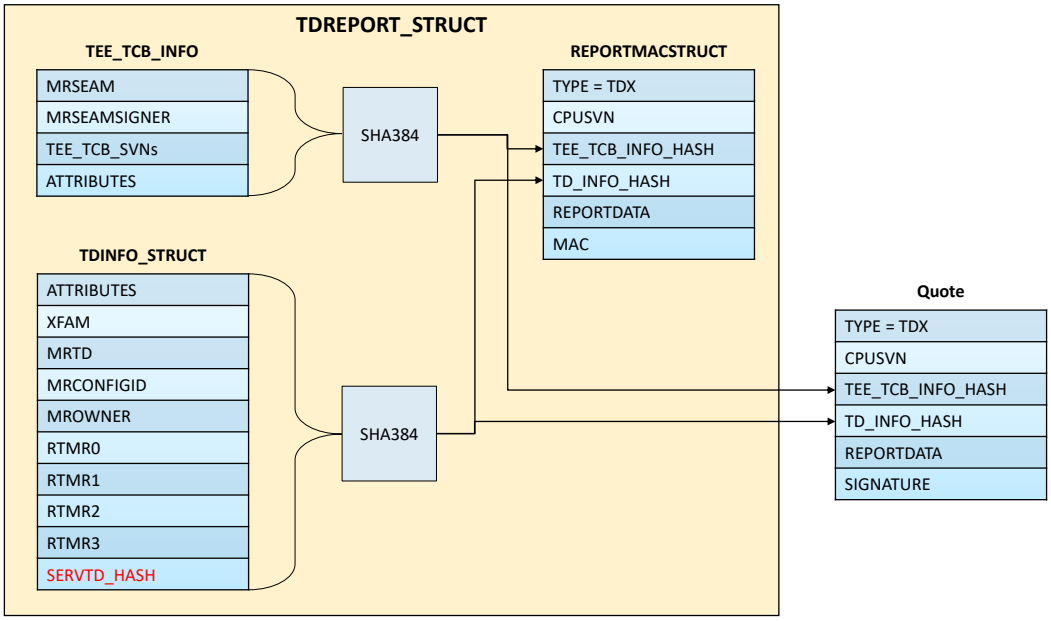


**Figure 4-3: TD Report structure**

Once the MigTD receives the TLS certificate from the peer, it extracts the TD Quote from the TLS certificate, and verifies integrity of the TD Quote.

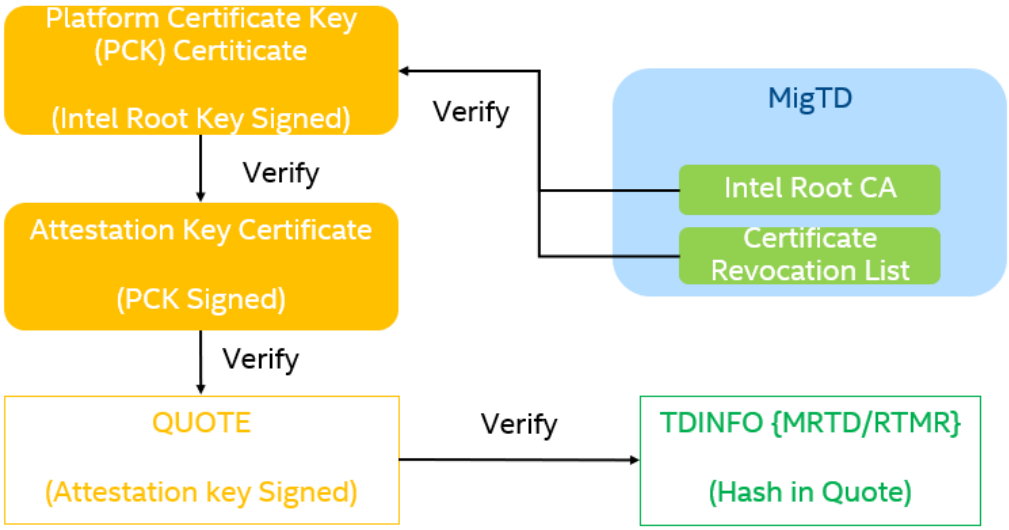


**Figure 4-4: MigTD VerifyQuote Flow**

## 4.3 Trust Anchor

The Quote is signed by the Attestation Key by the Quoting Enclave (QE). The Attestation Key Certificate is signed with the Provision Certification Key (PCK) by Provision Certification Enclave (PCE), and the PCK certificate is signed with Intel root CA Certificate. See Figure 4-4.

MigTD shall enroll the Intel Root CA certificate. And the Intel Root CA certificate shall be measured as part of migration policy.

## 4.4 Certificate Revocation List

Besides Root CA certificate, the verification process shall also include the check for the revoked certificate. See Figure 4-4.

MigTD shall enroll the Certificate Revocation List (CRL). And the Certificate Revocation List (CRL) shall be measured as part of migration policy.

## 4.5 Certificate Structure

The TD Quote Certificate shall be generated as an DER-encoded X.509 v3 format certificate. See table 4-1.

**Table 4-1: MigTD Certificate Structure**

**Certificate Field**

| Field | Description | Required |
|---|---|---|
| Version | Version of the encoded certificate shall be present and shall be version 3 (**value 0x2**) | Mandatory |
| Serial Number | Serial number shall be present with a positive integer value.<br><br>For example: **Serial Number: 1** | Mandatory |
| Signature Algorithm | Signature algorithm shall be present.<br><br>For example: **sha384WithRSAEncryption** | Mandatory |
| Issuer | Issuer distinguished name shall be specified.<br><br>For example: **"CN=MigTD SDK, O=mbed TLS, C=US"** | Mandatory |
| Subject Name | Subject name shall be present and shall represent the distinguished name associated with the certificate.<br><br>It shall be same as Issuer. | Mandatory |

| | | |
|---|---|---|
| Validity | Certificate may include this attribute. If the validity attribute is present, the value for **notBefore** field should be assigned the generalized **19700101000000Z** time value and **notAfter** field should be assigned the generalized **99991231235959Z** time value. | Mandatory |
| Subject Public Key Info | Device public key and the algorithm shall be present.<br><br>For example:<br><br>**Public Key Algorithm: rsaEncryption**<br><br>**Modulus: ...**<br><br>**Exponent: 65537 (0x10001)** | Mandatory |
| X509v3 Extension:<br><br>Basic Constraints | **CA: FALSE**. | Optional |
| X509v3 Extension:<br><br>Subject Key Identifier | Subject Key Identifier | Optional |
| X509v3 Extension:<br><br>Authority Key Identifier | It should be same as Subject Key Identifier. | Optional |
| X509v3 Extension:<br><br>Extended Key Usage | MigTD Quote Certificate indicator<br><br>"**1.2.840.113741**.1.5.5.1.1" - Intel | Mandatory |

## 4.6 TD Report Data

During MigTD attestation, a pair of asymmetric keys will be generated. The public key hash (**SHA384**) is treated nonce for TD report data.

## 4.7 OID based TD Quote

The MigTD Quote report is X509v3 Extension OID based data.

**Table 4-2: MigTD OID Field**

**MigTD OID Field**

| Field | Description | Required |
|---|---|---|
| OID: MigTD_quote_report | Quote Report<br>"**1.2.840.113741**.1.5.5.1.2" - Intel | Mandatory |
| OID:<br>MigTD_event_log | EventLog Report<br>"**1.2.840.113741**.1.5.5.1.3" - Intel | Mandatory |

*NOTE: Current **RA-TLS and OpenEnclave are using below fields.***

**Table 4-3: RA-TLS OID Field**

**RA-TLS OID Field**

| Field | Description | Required |
|---|---|---|
| OID:<br>ias_response_body_oid | Attestation Report. ias_report<br>"**1.2.840.113741**.1337.2" - Intel | Mandatory |
| OID: ias_root_cert_oid | Attestation Report. ias_sign_ca_cert<br>"**1.2.840.113741**.1337.3" - Intel | Mandatory |
| OID: ias_leaf_cert_oid | Attestation Report. ias_sign_cert<br>"**1.2.840.113741**.1337.4" - Intel | Mandatory |
| OID:<br>ias_report_signature_oid | Attestation Report. ias_report_signature<br>"**1.2.840.113741**.1337.5" - Intel | Mandatory |

**Table 4-4: Open Enclave TLS OID Field**

**Open Enclave OID Field**

| Field | Description | Required |
|---|---|---|
| OID: oe_report | Quote Report<br>"**1.2.840.113556**.10.1.1" - Microsoft | Mandatory |
| OID: oe_evidence | Attestation Report<br>"**1.2.840.113556**.10.1.2" - Microsoft | Mandatory |

## 4.8 Sample Certificate

A sample structure is shown here assuming a minimum certificate structure. Field names and structure types match the ASN.1 names in PKCS #1 [RFC8017] and PKCS #7 [RFC2315].

```
Certificate:
  Data:
    Version: 3 (0x2)
    Serial Number: 1 (0x1)
    Signature Algorithm: sha256WithRSAEncryption
    Issuer: CN = 127.0.0.1, O = mbed TLS, C = UK
    Validity
      Not Before: Jan  1 00:00:00 2001 GMT
      Not After : Dec 31 23:59:59 2030 GMT
    Subject: CN = 127.0.0.1, O = mbed TLS, C = UK
    Subject Public Key Info:
      Public Key Algorithm: rsaEncryption
        RSA Public-Key: (3072 bit)
        Modulus:
          00:a7:f5:93:01:2d:c9:d6:c6:11:ca:1c:0e:ab:31:
          cf:51:6d:81:f6:8c:49:bc:49:ee:c2:6d:da:6c:01:
          1a:2d:d0:fb:23:f2:c9:c9:9b:4b:47:47:5b:96:f2:
          fa:00:bb:f9:a8:af:7f:73:1f:9d:c9:c7:73:88:3e:
          c1:98:a4:87:54:db:8b:43:bb:c1:57:76:9f:a5:d0:
          1c:d6:25:bc:a2:5b:fa:cd:20:ef:bd:bd:c0:13:ac:
          74:ef:a8:58:14:33:49:0f:9b:b5:f2:6f:3f:8c:ac:
          42:b3:8e:a2:03:6d:7a:dc:77:b6:fc:02:87:d6:cb:
          f2:43:99:c4:75:d9:a6:98:3b:96:2b:aa:c5:3d:3d:
          36:92:c5:e5:30:84:78:db:b4:a6:83:6a:20:ca:08:
          2b:00:c8:49:50:b4:ce:ad:e6:66:79:44:c4:79:40:
          51:62:b5:79:60:e8:56:53:88:a3:47:71:5a:49:c3:
          1d:de:31:c6:2b:96:f8:19:c3:4b:d1:c8:fa:c5:0c:
          e4:ce:c3:68:89:4b:b3:64:e6:3e:70:e8:1c:3f:8e:
          b2:08:7d:ab:50:9d:8b:f6:1f:85:85:eb:06:80:2d:
          f8:74:0f:0e:c2:93:9b:f4:6d:34:3f:54:1b:93:27:
          be:48:c1:0f:de:49:70:44:2e:ad:05:26:ca:cf:23:
          6d:62:7f:72:c6:e3:51:27:be:d5:ee:b5:8d:72:ce:
          b6:a7:50:ea:0f:38:99:4e:db:f8:3f:ee:81:e7:11:
          5a:b0:70:f4:31:f4:47:88:53:1d:9b:b9:6d:87:68:
          71:e0:79:34:bc:da:79:04:f9:fc:9c:66:a2:7b:09:
          e4:d4:19:90:cd:2b:0a:f8:64:f1:92:c0:da:0f:d0:
          e9:9e:1f:71:2d:fb:56:5c:53:ab:13:99:e7:36:ff:
          a6:47:d4:c8:3a:22:75:a6:b7:c8:a8:da:b9:0f:64:
          21:d4:e1:0e:c2:76:ce:84:e6:48:5d:9b:3d:7d:08:
          d0:4b:b1:fe:60:c4:db:c3:2b:4f
        Exponent: 65537 (0x10001)
    X509v3 extensions:
      X509v3 Basic Constraints:
        CA:FALSE
      X509v3 Subject Key Identifier:
```

```
            E0:F1:C8:11:40:8D:3A:24:C1:9C:FA:B9:31:17:08:60:2E:E3:2A:1A
          X509v3 Authority Key Identifier:
            keyid:E0:F1:C8:11:40:8D:3A:24:C1:9C:FA:B9:31:17:08:60:2E:E3:2A:1A

          1.2.840.113741.1337.2:
            report
          1.2.840.113741.1337.3:
            cacert
          1.2.840.113741.1337.4:
            signed
          1.2.840.113741.1337.5:
            ..............................
    Signature Algorithm: sha256WithRSAEncryption
       33:8d:62:27:78:0c:f8:4c:24:6f:11:37:5f:54:fd:3e:8d:c4:
       93:a9:31:ff:0e:f8:f9:88:6c:92:e2:b2:e8:d3:53:26:bd:b8:
       03:fc:15:e6:8f:55:e4:7a:06:77:18:87:5d:57:dc:69:dd:e8:
       45:5a:db:22:5b:73:ab:e8:b1:0c:b3:4c:00:d5:07:22:28:cb:
       7b:7c:87:19:5f:bc:9d:c5:28:b6:ac:e9:9e:c1:21:54:51:bc:
       86:d0:b2:1a:41:9c:af:52:67:cc:4a:d7:3a:c1:c0:74:b8:68:
       bf:22:bf:32:2e:72:46:97:ba:ad:f8:77:25:a4:3e:f9:f9:49:
       79:70:2d:a4:16:20:b1:f2:34:d9:90:0e:7b:e2:84:40:b2:99:
       38:ce:40:4e:fc:94:1d:70:f5:93:93:a5:fa:26:42:7b:df:08:
       cf:de:3e:ca:b8:8f:ba:61:58:b0:57:8f:60:20:96:29:32:35:
       b3:19:e6:6b:e3:40:fc:02:04:97:6c:64:6c:7c:b6:95:48:24:
       a2:5a:70:0d:1f:a9:ba:d9:97:92:b3:9b:e7:46:a3:39:bf:76:
       d4:e6:85:62:cd:f2:86:49:b1:c0:04:db:a5:de:9d:da:68:59:
       32:33:17:ff:94:78:77:ea:dd:f0:3f:45:d2:97:e1:cf:5b:58:
       23:1e:6a:25:fc:ce:0e:89:d0:31:1b:f9:59:1a:34:4e:8a:bb:
       57:2f:99:ec:f3:b2:41:ec:a2:58:60:63:0b:64:06:4b:51:5b:
       4c:2f:fe:47:92:70:2d:4c:b2:d8:31:31:9d:5b:33:52:15:96:
       6f:fb:4e:6e:f5:b0:39:07:f1:90:88:fd:d5:7a:71:a3:fc:cf:
       15:ff:18:0c:15:0f:46:c6:4c:9a:dc:31:ce:7c:5c:81:19:9f:
       b5:53:22:d1:b9:5d:d1:f2:9b:46:9a:0f:b7:7d:cb:a0:6c:fc:
       e9:75:72:1d:df:35:f5:f7:2e:64:3c:75:5d:8c:ce:1d:18:98:
       95:f7:30:96:57:c1
```

## 4.9 TD Quote API

### 4.9.1 Get TD Quote

In MigTD mutual authentication, the RA-TLS need get the TD Quote and put it OID:MigTD_quote_report and create a RA-TLS certificate around it.

### 4.9.2 Verify TD Quote

In MigTD mutual authentication, the peer will get the RA-TLS certificate and verify the integrity of the TD Quote. If the integrity verification fails, the TLS will be terminated.

NOTE: The MigTD policy verification is irrelevant in TD Quote attestation. MigTD policy verification will be the next step after MigTD mutual authentication. The migration session key (MSK) is passed after the MigTD policy verification.

§

# 5 MigTD Migration Policy

The MigTD need a set of migration policy to determine if it is allowed to migrate a TD from a source platform to a destination platform.

## 5.1 MigTD Policy Type

In general, there might be multiple types of policy:

**Table 5-1: MigTD Policy Type**

| Type | Policy Owner | Definition | Example | MigTD Action |
|---|---|---|---|---|
| TDX Module Policy (Required) | Intel | TDX Module Specification | TD Global Metadata Field:<br><br>On export: **MIG_VERSION** is between [**MIN_EXPORT_VERSION, MAX_EXPORT_VERSION**]<br><br>On import: **MIG_VERSION** is between [**MIN_IMPORT_VERSION, MAX_IMPORT_VERSION**]<br><br>**TDX_Module's MAJOR_VERSION/ MINOR_VERSION** is compatibility.<br><br>**TD's ATTRIBUTES/XFAM/CPUID** is compatible.<br><br>*Reference*: TDX Module Migration Specification. | MigTD optional checks it to detect error earlier.<br><br>Code is extended to MRTD or RTMR[1] (See below). |
| MigTD Default Policy (Recommended) | CSP | MigTD Design Guide | **MigTD** is compatible. (**SVN, MRTD**, etc.)<br><br>**Certificate** is valid and not expired. (**Root CA certificate**, **certificate revocation list**) | MigTD checks the migration policy.<br><br>Policy Data is extended to RTMR[2]. (See below). |
| MigTD Extension Policy (Optional) | CSP | CSP Extension. Out of scope of this document. | CSP Extension. Out of scope of this document. | CSP Extension. Out of scope of this document. |

## 5.2 MigTD Policy Measurement

A MigTD should design the policy in an extensible way that the CSP/Solution vendor may add their own policy. The policy could be:

- Code – executed by a common policy engine to evaluate
    - The code (a statically linked library or a dynamic loaded module) could be part of MigTD Core, which is extended to MRTD in non-SecureBoot mode or RTMR[1] in Secure Boot Mode. See more details about these two modes in Chapter 10.
    - If the MigTD checks the "TDX Module Policy" to detect the error earlier, then this check can be done in the code, and then there is no need to define it as policy data.
- Data - consumed by a common policy module.
    - The data could be provisioned later in Configuration Firmware Volume (CFV) of the MigTD, which is extended to RTMR[2].
    - The MigTD includes the "MigTD Default Policy" in CFV, so that it can be easily updated. For example, the TCB SVN or QE certificate expired date. The trust anchor "Intel Root CA Cert" and Certificate Revocation List (CRL) shall also be included in CFV as part of migration policy.

## 5.3 MigTD Policy Definition

The typical MigTD policy is that the TCBs and hardware are newer than the baseline date, which can be evaluated with TDX Quote attestation. Figure 4-3 shows the TDX Quote and TD Report structure.

### 5.3.1 MigTD Policy Property

This document defines below default policy properties.

**Table 5-2: MigTD Policy Property**

| Family Name | Group Name | Property Name | Type | Comment |
|---|---|---|---|---|
| MigTD | TEE_TCB_INFO *(The integrity is verified with TD Quote)* | TEE_TCB_SVN.SEAM | Integer | Intel TDX module SVN. (UINT16) |
| | | MRSEAM | String | Measurement of the Intel TDX module. (48 Bytes Hash) |
| | | MRSIGNERSEAM | String | Measurement of TDX module signer if valid. (48 Bytes Hash) |

| | | | | |
|---|---|---|---|---|
| | | ATTRIBUTES | Integer | Additional configuration ATTRIBUTES if valid. (UINT64) |
| | TDINFO *(The integrity is verified with TD Quote)* | ATTRIBUTES | Integer | TD's Attributes. (UINT64) |
| | | XFAM | Integer | TD's XFAM. (UINT64) |
| | | MRTD | String | Measurement of the initial contents of the TD. (48 Bytes Hash) |
| | | MRCONFIGID | String | Software-defined ID for non-owner-defined configuration of the guest TD. (48 Bytes Hash) |
| | | MROWNER | String | Software-defined ID for the guest TD's owner. (48 Bytes Hash) |
| | | MROWNERCONFIG | String | Software-defined ID for owner-defined configuration of the guest TD. (48 Bytes Hash) |
| | | RTMR0 | String | Runtime extendable measurement register 0. (48 Bytes Hash) |
| | | RTMR1 | String | Runtime extendable measurement register 1. (48 Bytes Hash) |
| | | RTMR2 | String | Runtime extendable measurement register 2. (48 Bytes Hash) |
| | | RTMR3 | String | Runtime extendable measurement register 3. (48 Bytes Hash) |

| | | | | |
|---|---|---|---|---|
| | EventLog *(The integrity of "Digest" is verified with RTMR. The integrity of "Data" is verified with "Digest")* | Digest.TdShim | String | Digest for TDShim. (48 Bytes Hash) |
| | | Digest.MigTdCore | String | Digest for MigTD Core. (48 Bytes Hash) |
| | | Digest.SecureBootKey | String | Digest for SecureBootKey. (48 Bytes Hash) |
| | | Digest.MigTdPolicy | String | Digest for MigTdPolicy.<br><br>(This field is only valid with "self" string to indicate that the MigTdPolicy from source and destination must be same.) |
| | | Data.MigTdCoreSvn | Integer | MigTD Core's SVN. (UINT64) |
| QE | Quote *(The integrity is verified with TD Quote)* | PckCert.ExpiredTime | String | Expired time for the PCK certificate. |
| | | AttestationKeyCert. ExpiredTime | String | Expired time for the Attestation Key certificate. |

### 5.3.2 MigTD Policy Format

MigTD should use the industry standard for the policy data format - **JSON**.

```
============
{
  "id": <GUID>,
  <Family-Name>: {
    <Group-Name>: {
      <Property-Name>: {
        "operation": <Operation-Name>,
        "reference": <Reference-Value>
      },
      ...
    },
```

```
      ...
    },
    ...
}
============
```

| Property Type | Operation-Name | Reference-Value type | Comment |
|---|---|---|---|
| Integer | “equal”, “greater-or-equal”, “subset” | Integer value | Not sure if we want to support “greater”, “less”, “less-or-equal”, or “superset” |
| String | “equal” | String value | |
| String | “equal” | “self” | Used to ensure two MigTDs have same data. |
| String | “in-range” | “<integer>..<integer>” | The property shall be greater-or-equal than the begin value, and less than the end value. |
| String | “in-time-range” | “<integer>..<integer>” | The property time shall be within the time range. The value in the time range is UNIX time.<br><br>(Unit time is the number of seconds that have elapsed since the UNIX epoch, minus leap seconds; the UNIX epoch is 00:00:00 UTC on 1 January 1970.) |

### 5.3.3 MigTD Policy Example

#### 5.3.3.1 Sample Policy in non-SecureBoot Mode

In this mode, we cannot know the SVN of the MigTDCore. We use policy to indicate the MigTD must be same.

```
{
```

```
"id": "xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx",
"MigTD": {
 "TEE_TCB_INFO": {
    "TEE_TCB_SVN.SEAM": {
       "operation": “greater-or-equal”,
       "reference": “self”
    },
    "MRSEAM": {
       "operation": “equal”,
       "reference": “self”
    },
    "MRSIGNERSEAM": {
       "operation": “equal”,
       "reference": “self”
    },
    "ATTRIBUTES": {
       "operation": “equal”,
       "reference": “self”
    }
 },

 "TDINFO": {
    "ATTRIBUTES": {
       "operation": “equal”,
       "reference": “self”
    },
    "XFAM": {
       "operation": “equal”,
       "reference": “self”
    },

    "MRTD": {
       "operation": “equal”,
       "reference": “self”
    },
    "MRCONFIGID": {
       "operation": “equal”,
       "reference": “self”
    },
    "MROWNER": {
       "operation": “equal”,
       "reference": “self”
    },
    "MROWNERCONFIG": {
       "operation": “equal”,
       "reference": “self”
```

```
    },


    // use RTMR
    "RTMR0": {
       "operation": “equal”,
       "reference": “self”
    },
    "RTMR1": {
       "operation": “equal”,
       "reference": “self”
    },
    "RTMR2": {
       "operation": “equal”,
       "reference": “self”
    },
    "RTMR3": {
       "operation": “equal”,
       "reference": “self”
    }
  }
 }
}
```

#### 5.3.3.2 Sample Policy in Secure Boot Mode

In this mode, we can know the SVN of the MigTDCore. We use policy to indicate a set of MigTD that the SVN in a fixed range.

```
{
 "id": "xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx",
 "MigTd": {
  "TEE_TCB_INFO": {
    "TEE_TCB_SVN.SEAM": {
       "operation": “greater-or-equal”,
       "reference": “self”
    },
    "MRSEAM": {
       "operation": “equal”,
       "reference": “self”
    },
    "MRSIGNERSEAM": {
       "operation": “equal”,
       "reference": “self”
    },
    "ATTRIBUTES": {
```

```
        "operation": “equal”,
        "reference": “self”
    },
},

"TDINFO": {
    "ATTRIBUTES": {
        "operation": “equal”,
        "reference": “self”
    },
    "XFAM": {
        "operation": “equal”,
        "reference": “self”
    },

    "MRTD": {
        "operation": “equal”,
        "reference": “self”
    },
    "MRCONFIGID": {
        "operation": “equal”,
        "reference": “self”
    },
    "MROWNER": {
        "operation": “equal”,
        "reference": “self”
    },
    "MROWNERCONFIG": {
        "operation": “equal”,
        "reference": “self”
    },

},

"EventLog”: {
    // Do not use RTMR, but SVN in event log
    // Below policy means and SVN in [13, 18).
    "Digest.TdShim": {
        "operation": “equal”,
        "reference": “self”
    },
    "Digest.SecureBootKey": {
        "operation": “equal”,
        "reference": “self”
    },
    "Digest.MigTdPolicy": {
```

```
            "operation": “equal”,
            "reference": “self”
        },
        "Digest.MigTdCoreSvn": {
            "operation": “in-range”,
            "reference": “13..18”
        },
    }
  }
}
```

§

# 6 MigTD Network Communication

Two MigTD need communicate with each other to establish a secure session. Because the MigTD is in TCB, it is not recommended to include a full TCP/IP network stack in a MigTD. Instead, the MigTD should use the network stack in the untrusted VMM host environment.

## 6.1 TDVMCALL

The MigTD-s may use **TDG.VP.VMCALL<Service.MigTD.Send>** and **TDG.VP.VMCALL<Service.MigTD.Receive>** to communicate the VMM host environment and pass the MigTD-to-MigTD communication packet (such as TLS packet). The VMM on the source platform need pass the communication packet to the VMM on destination platform. Then the VMM on the destination platform can return the packet to the MigTD-d. When MigTD-d finishes processing the input packet, MigTD-d can return the response back to MigTD-s by using the same mechanism. Figure 6-1 shows the flow.

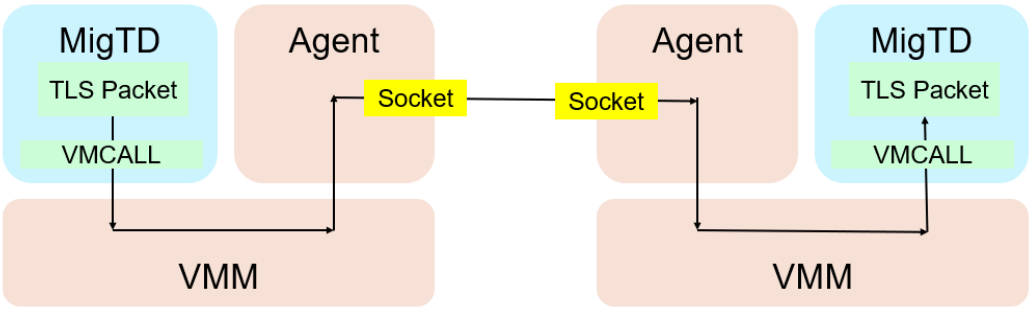


Figure 6-1: VMCALL based communication in MigTD

### 6.1.1 VSock information

During MigTD launch, the VMM can pass the **MIGTD_STREAM_SOCKET_INFO** HOB to the MigTD to allow MigTD to communicate with the agent.

The context ID (CID) of the VMM is 2 - Well-known CID for the host. The context ID (CID) of the MigTD can be retrieved from **MIGTD_STREAM_SOCKET_INFO** HOB. The VMM or the MigTD can be client or server. The server listening port can be found in the **MIGTD_STREAM_SOCKET_INFO** HOB.

### 6.1.2 VSock data payload

The MigTD should put the TLS packet to the VMCALL-vsock data payload.

§

# 7 MigTD Cryptography Capability

MigTD needs to have cryptography for key negotiation.

## 7.1 Cryptography

### 7.1.1 Cryptography Algorithm

**Table 7-1: MigTD Cryptography Algorithm**

| Cryptography | Options | Recommendation |
|---|---|---|
| Digital Signature | RSA, ECDSA | ECDSA-NIST_P384 |
| Key Exchange | ECDHE | ECDHE-secp384r1 |
| AEAD | AES-GCM, Chacha20-Poly1305 | AES-256-GCM |
| Hash | SHA-2 | SHA384 |

### 7.1.2 TLS Configuration

**Figure 7-1: MigTD TLS Configuration**

| TLS Configuration | Options | Recommendation |
|---|---|---|
| Version | 1.2, 1.3 | 1.3 |
| Cipher Suite | **V1.3**<br>TLS_AES_256_GCM_SHA384<br>TLS_AES_128_GCM_SHA256<br>TLS_CHACHA20_POLY1305_SHA256<br><br>**V1.2**<br>TLS_ECDHE_RSA_WITH_AES_256_GCM_SHA384 | **V1.3**<br>TLS_AES_256_GCM_SHA384 |

| | | |
|---|---|---|
| | TLS_ECDHE_ECDSA_WITH_AES_256_GCM_SHA384<br><br>TLS_ECDHE_RSA_WITH_AES_128_GCM_SHA256<br><br>TLS_ECDHE_ECDSA_WITH_AES_128_GCM_SHA256 | |

## 7.2 Random Number Generation

The MigTD should use Intel instruction **RDSEED/RDRAND** to get random number.

§

# 8 MigTD Binary Image

MigTD is a customized tiny TDVF.

## 8.1 Boot Firmware Volume (BFV)

The MigTD includes one Firmware Volumes (FV) - Boot Firmware Volume. The format of FV is defined in PI specification.

The Boot Firmware Volume includes all component required during boot.

The file system GUID must be **EFI_FIRMWARE_FILE_SYSTEM2_GUID** or **EFI_FIRMWARE_FILE_SYSTEM3_GUID**, which is defined in PI specification.

1) **TD Shim**

   a) **ResetVector** – this component provides the entrypoint for MigTD, switch to long mode, and jumps to the ShimIpl. The FFS GUID must be **EFI_FFS_VOLUME_TOP_FILE_GUID**, which is defined in PI specification.

   b) **ShimIpl** – This component prepares the required parameter for MigTDCore and jump to the MigTDCore. Module type is **EFI_FV_FILETYPE_SECURITY_CORE**, which is defined in PI specification.

2) **MigTDCore** – The main MigTD module, which finishes all its work described in chapter 2 then tears down itself. MigTD core includes key exchange cryptography capability and networking capability via VMM interface. File type is **EFI_FV_FILETYPE_DXE_CORE**, which is defined in PI specification.

The BFV may include an initial static page table to assist the ResetVector switch from 32-bit mode to 64-bit mode.

## 8.2 Configuration Firmware Volume (CFV)

MigTD includes a Configuration Firmware Volume (CFV) to hold the migration policy data.

**Table 8-1: MigTD Policy Data**

| Item | Variable GUID | Variable Name | Comment |
|---|---|---|---|
| MigTD Policy | MIGTD_GUID | “MigTD_Policy” | JSON file |
| Root CA Cert (Trust Anchor) | MIGTD_GUID | “Quote_db” | One EFI_SIGNATURE_LIST, one EFI_SIGNATURE_DATA |

| | | | |
|---|---|---|---|
| Certificate Revocation List | MIGTD_GUID | “Quote_dbx” | One EFI_SIGNATURE_LIST, multiple EFI_SIGNATURE_DATA |
| Secure Boot Key | MIGTD_GUID | “MigTdCore_db” | One EFI_SIGNATURE_LIST, one EFI_SIGNATURE_DATA<br><br>(only applicable for SecureBootMode) |

```
#define MIGTD_GUID { \
0x2ff7a3, 0x5fc, 0x453c, {0x91, 0xc8, 0x76, 0x66, 0xfe, 0x5, 0x43, 0x74} \
}
```

§

# 9 MigTD Launch

MigTD launch is similar to TDVF launch.

## 9.1 MigTD initialization

The MigTD initialization is the same as TDVF initialization flow, and starts from 32-bit protected mode with paging disabled, then switch to 64-bit long mode.

## 9.2 MigTD Hand-Off Block (HOB)

The TD HOB list is used to pass the information from VMM to TDVF. The HOB format is defined in PI specification. If the TD HOB is used, it must be extended to RTMR register.

MigTD should ignore the TD HOB passed from VMM. The TD memory information should be indicated by MigTD in the metadata area.

After launch, the MigTD should call **TDG.VP.VMCALL <Service.MigTD.WaitForRequest>** to get the new migration information.

The MigTD could be transient or persistent, the MigTD should use **TDG.VP.VMCALL <Service.MigTP.ReportStatus>** to report the migration status, after it programs the MSK to the TDX module.

## 9.3 MigTD teardown

MigTD can teardown itself after the migration is done with a special **TDG.VP.VMCALL <Service.MigTD.Shutdown>**.

MigTD is only launched when the VMM starts to migrate.

## 9.4 MigTD AP handling

For simplicity, a MigTD may choose to only support one processor.

As an alternative, a MigTD may support multiple processors if multiple TD migrations are supported. See chapter 12 for details.

§

# 10 MigTD Measurement

MigTD measurement is same as a normal TDVF.

## 10.1 Non-Secure Boot Mode

Usually, TD-Shim is treated as firmware code and extended to MRTD. The MigTD Core is treated as OS code and extended to RTMR[1]. The MigTD Policy is treated as application code/config and extended to RTMR[2]. See table 10-1.

However, if we choose this approach, there is no way to know a secure version number (SVN) of a MigTD. If we want to allow a MigTD to support a different peer MigTD identified with SVN, then we need a different way – Secure Boot Mode.

**Table 10-1: TD Measurement Registers for MigTD (Non-Secure Boot)**

| Typical Usage | Register | Event Log | Extended by | Content |
|---|---|---|---|---|
| Firmware Code | MRTD | NO | VMM: SEAMCALL [TDH.MR.EXTEND] | TD-Shim |
| Firmware Config | RTMR [0] | YES | TD-Shim: TDCALL [TDG.MR.RTMR.EXTEND] | N/A |
| OS code / Config | RTMR [1] | YES | TD-Shim: TDCALL [TDG.MR.RTMR.EXTEND] | MigTD Core |
| APP code/ Config | RTMR [2] | YES | MigTD Core: TDCALL [TDG.MR.RTMR.EXTEND] | MigTD Policy (Including JSON style policy, Root CA Cert and CRL) |

## 10.2 Secure Boot Mode

In Secure Boot Mode, only TD-Shim is extended to MRTD. TD-Shim can follow secure boot policy to verify the signature of MigTD and its SVN. The secure boot key is treated as firmware config and extended to RTMR[0]. The MigTD Core and its SVN are treated as OS code and config. They are extended to RTMR[1]. The MigTD policy is still treated as APP config and extended to RTMR[2]. See table 10-2.

With this approach, the verifier can get the MigTD core SVN. If TD-Shim and secure boot key are same and MigTD Core and SVN are updated, the MigTD can verify peer by using the SVN.

Table 10-2: TD Measurement Registers for MigTD (Secure Boot)

| Typical Usage | Register | Event Log | Extended by | Content |
|---|---|---|---|---|
| Firmware Code | MRTD | NO | VMM: SEAMCALL [TDH.MR.EXTEND] | TD-Shim |
| Firmware Config | RTMR [0] | YES | TD-Shim: TDCALL [TDG.MR.RTMR.EXTEND] | Secure Boot Key |
| OS code / Config | RTMR [1] | YES | TD-Shim: TDCALL [TDG.MR.RTMR.EXTEND] | MigTD Core, MigTdCore SVN |
| APP code/ Config | RTMR [2] | YES | MigTD Core: TDCALL [TDG.MR.RTMR.EXTEND] | MigTD Policy (Including JSON style policy, Root CA Cert and CRL) |

## 10.3 MigTD Binding

MigTD should be bound with the target TD to be migrated. If VMM chooses to keep MigTD alive, the VMM can launch the MigTD at first then use of **SEAMCALL[TDH.SERVTD.BIND]** to bind the target TD. Then the VMM gets the **TargetTD_UUID** and **BindingHandle** and passes them to MigTD in **MIGTD_MIGRATION_INFORMATION**.

If VMM chooses to launch MigTD when it is required, then the VMM can use **SEAMCALL[TDH.SERVTD.PREBIND]** to prebind the target TD. The VMM shall calculate the MigTD's TDINFO_STRUCT then SERVTD_INFO_HASH. The SERVTD_INFO_HASH shall be input as parameter during prebind.

§

# 11 MigTD Metadata

MigTD uses the same metadata defined for TDVF. BFV is required. CFV might be required, if CFV based policy is supported. The temp memory (stack / heap) is required, and it is used to indicate the memory that will be used by MigTD.

TD Hob is NOT required in the MigTD.

§

# 12 Multiple TDs Migration

## 12.1 Multi TDs Migration

A VMM might need migration multiple TDs at one time. If one source MigTD and one destination MigTD set up the RA-TLS connection with mutual authentication, this connection can be used to migrate multiple TDs. See figure 12-1.

This model should be supported by default, because there is no need to set up multiple RA-TLS sessions if source MigTD and destination MigTD are same.

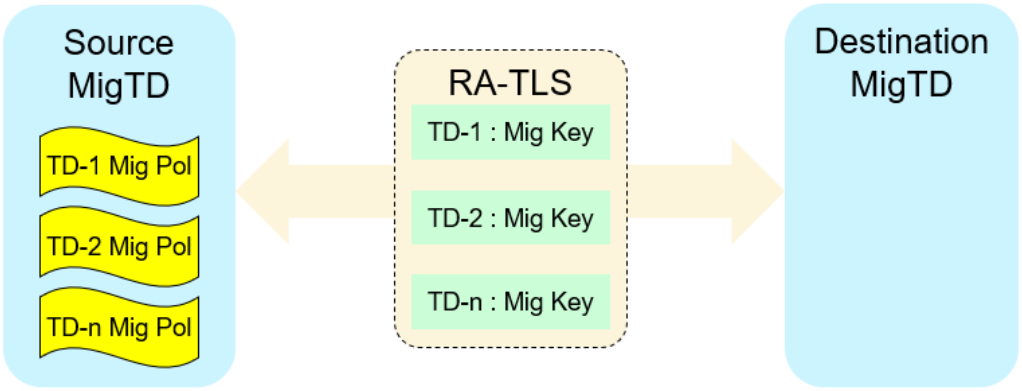


Figure 12-1: Multiple TDs Migration

## 12.2 Multi RA-TLS connections

One migration TD might support multiple RA-TLS connection sessions to or from different migration TDs.

### 12.2.1 Multi destination MigTDs

In some case, a VMM might need migration multiple TDs to different platforms. As such, one source MigTD need setup connection to multiple destination MigTDs. See figure 12-2.

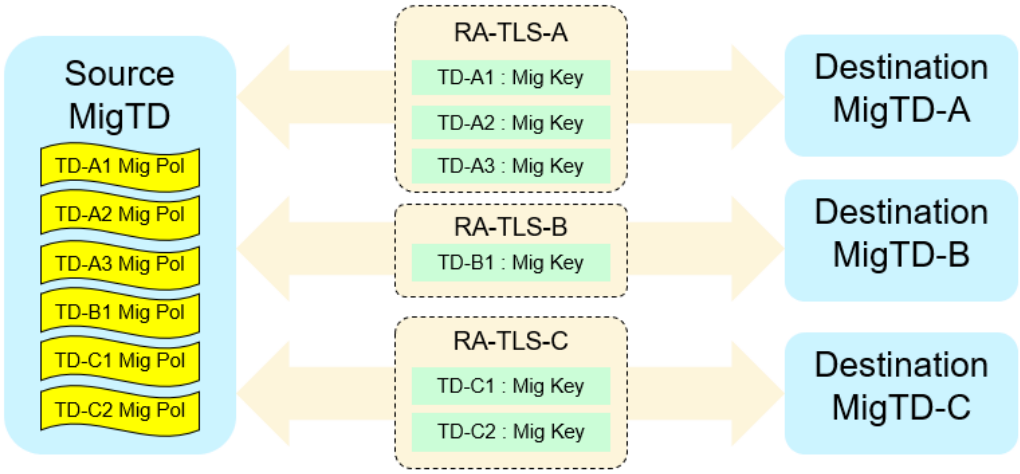


**Figure 12-2: Multiple Destination Migration TDs**

## 12.2.2 Multi source MigTDs

Similar to above case, a VMM might launch a destination and migration TD from different platforms. As such, one destination MigTD need wait for the connection from multiple source MigTDs. See figure 12-3.

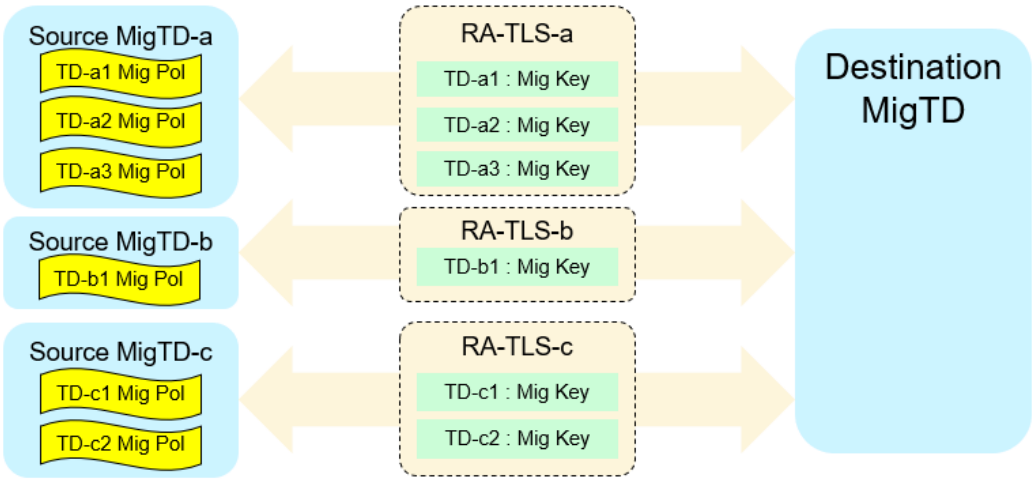


**Figure 12-3: Multiple Source Migration TDs**

## 12.3 Multi-Processor

If a MigTD supports multiple connections to or from different MigTDs, the VMM can enable multiple processors. MigTD may allocate different stack and heap for different processor and launch the MigTD Core. Then each CPU can take responsibility to set up a session with a subset of destination MigTD. See figure 12-4.

For example, if there are 20 different MigTD connections need to be set up and 4 CPUs, then each CPU can handle 5 MigTD connections.

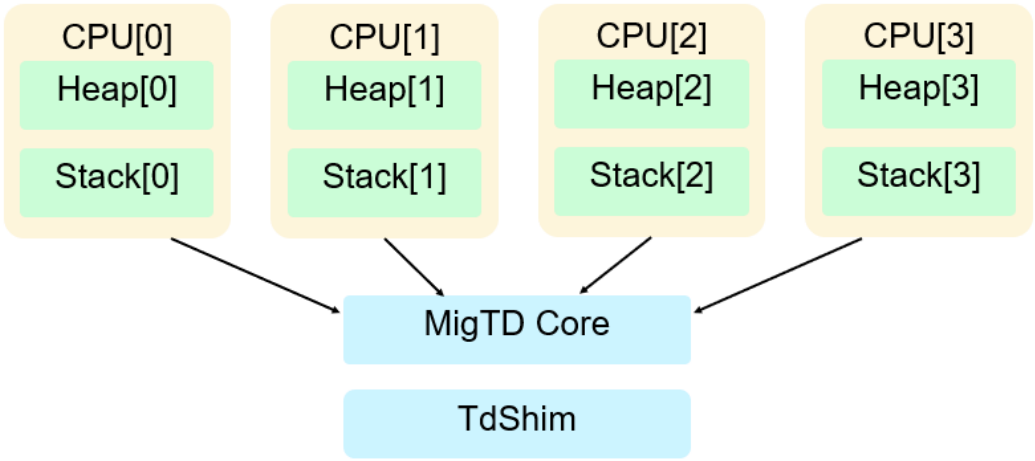


Figure 12-4: Multiple Processor Support in Migration TD

§

# *Appendix A Reference*

**[TDVF]** Intel TDX Virtual Firmware Design Guide, https://software.intel.com/content/www/us/en/develop/articles/intel-trust-domain-extensions.html

**[GHCI]** Guest Hypervisor Communication Interface v1.5, https://software.intel.com/content/www/us/en/develop/articles/intel-trust-domain-extensions.html

**[TDX Module EAS]** Intel TDX Module v1.5 Base Architecture, Intel TDX Module v1.5 ABI, https://software.intel.com/content/www/us/en/develop/articles/intel-trust-domain-extensions.html

**[TDX Migration]** Intel TDX Module v1.5 TD Migration Architecture, https://software.intel.com/content/www/us/en/develop/articles/intel-trust-domain-extensions.html

**[JSON]** JSON data model, https://www.json.org/json-en.html

**[CBOR]** Concise Binary Object Representation, http://cbor.io/

**[Intel RA-TLS]** Intel Remote Attestation TLS, https://github.com/cloud-security-research/sgx-ra-tls

**[Open Enclave RA-TLS]** open enclave Remote Attestation TLS, https://github.com/openenclave/openenclave/tree/master/samples/attested_tls, https://github.com/openenclave/openenclave/blob/master/include/openenclave/attestation/attester.h, https://github.com/openenclave/openenclave/blob/master/include/openenclave/attestation/verifier.h

§