Trust Domain Extension (TDX) Migration TD Design Guide

December 2023

Document Number: 348987-003

Document Number: 348987-003

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.

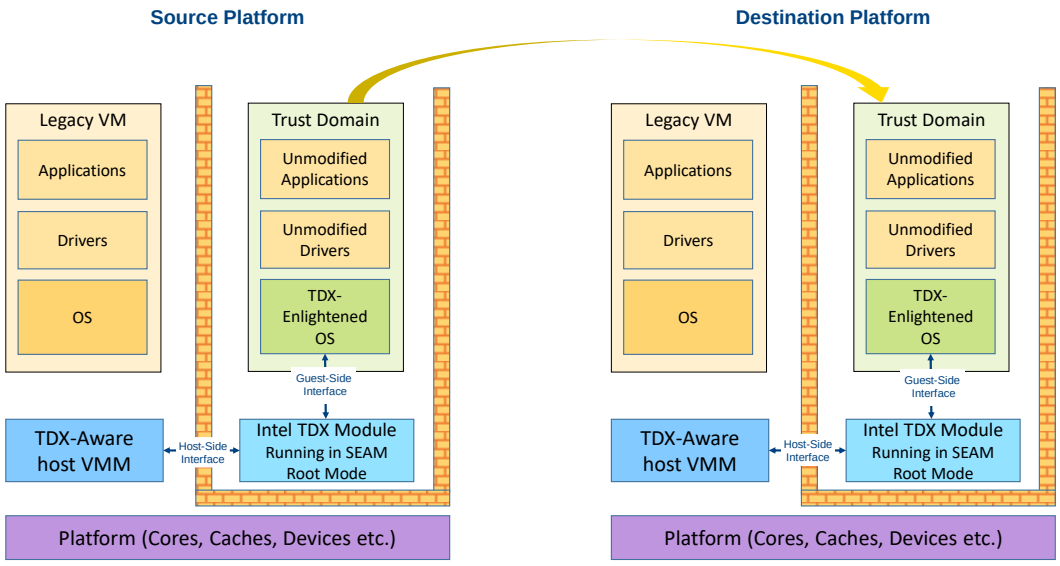

Figure 1-1: TD Migration

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 Keys (MSKs)** 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 MSKs has been successfully programmed for the target TD being migrated.

Figure 1-2 shows the TD Migration Architecture.

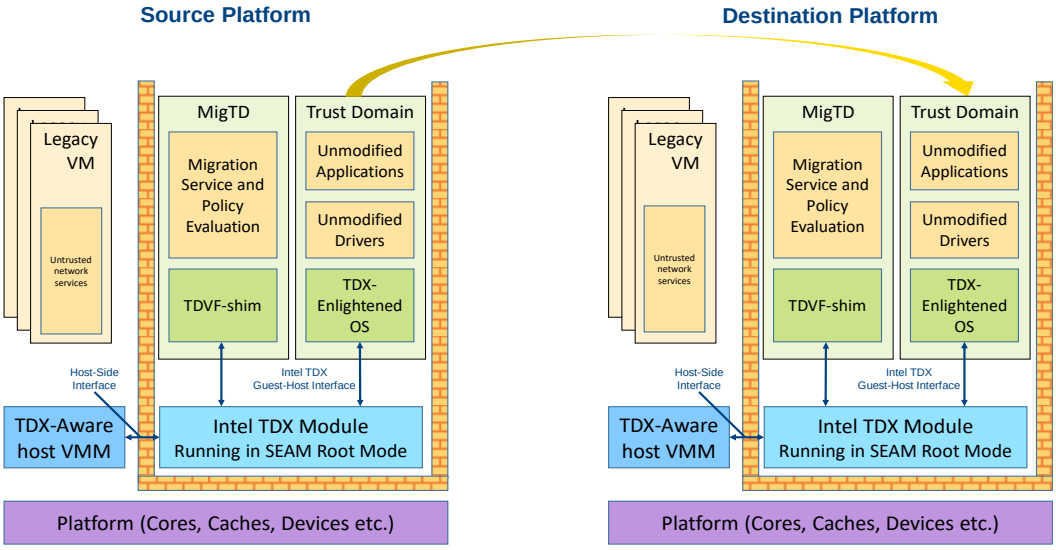

Figure 1-1: TD Migration Architecture

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.

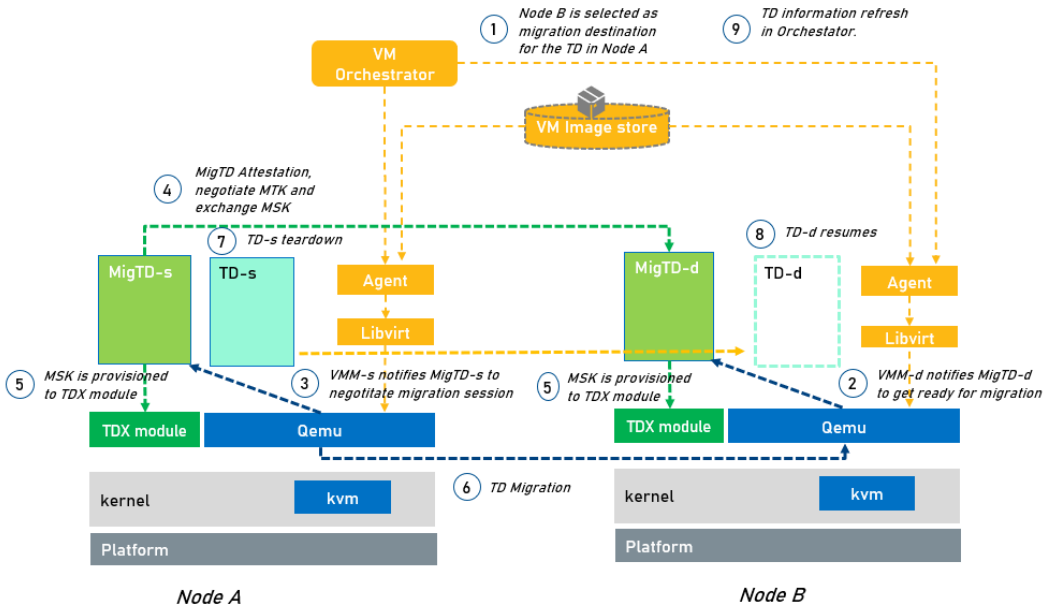

Figure 2-1: TD Migration Orchestration Flow

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 reads migration session forward key from TDX module as migration encryption key in source platform. MigTD-d reads migration session backward key from TDX module as migration encryption key in destination platform.

5) MigTD-s and MigTD-d negotiate with each other, including mutual attestation, Migration Policy evaluation, negotiate Migration Transport Keys (MTKs) and exchange the Migration Session Keys (MSKs) – the migration session forward key and migration session backward key.

6) MigTD-s writes migration session backward key to TDX module as migration decryption key in source platform. MigTD-d writes migration session forward key to TDX module as migration decryption key in destination platform.

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

8) TD-s is teardown by the VMM-s. 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 (MTKs)**: 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 Keys (MSKs)**: They are a set of AES-256-GCM keys generated by TDX-Module and exchanged by MigTD. The MSKs are protected by the MTKs during when they are exchanged. There are two MSKs used in the migration.

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

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

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

Figure 2-1 shows the Migration Session Keys used in migration setup.

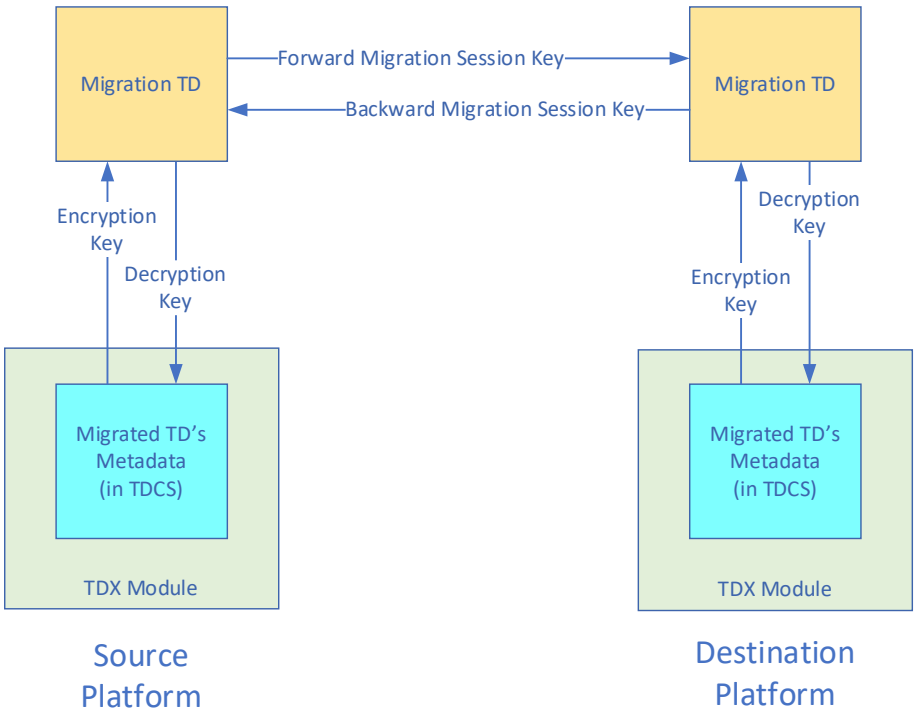

Figure 2-2: Migration Session Keys Setup

2.3 MigTD General Flow

A MigTD can be launched by VMM at any time, when VMM starts migration. Once the MigTD transfers the MSKs 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 vsock 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 (MTKs)**. 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] exchange **Migration Session Keys (MSKs)**
    - [MigTD-s] reads its own migration session forward key via **TDCALL<TDG.SERVTD.RD(MIG_ENC_KEY)>**.
    - [MigTD-d] reads its own migration session backward key via **TDCALL<TDG.SERVTD.RD(MIG_ENC_KEY)>**.
    - [MigTD-s/MigTD-d] transfer its own key via the secure channel protected by MTKs.
    - [MigTD-s] writes peer's migration session backward key via **TDCALL<TDG.SERVTD.WR(MIG_DEC_KEY)>**.
    - [MigTD-d] writes peer's migration session forward key via **TDCALL<TDG.SERVTD.WR(MIG_DEC_KEY)>**.

- [MigTD-s/MigTD-d] can be tear down.

Figure 2-2 shows the general Migration TD flow.

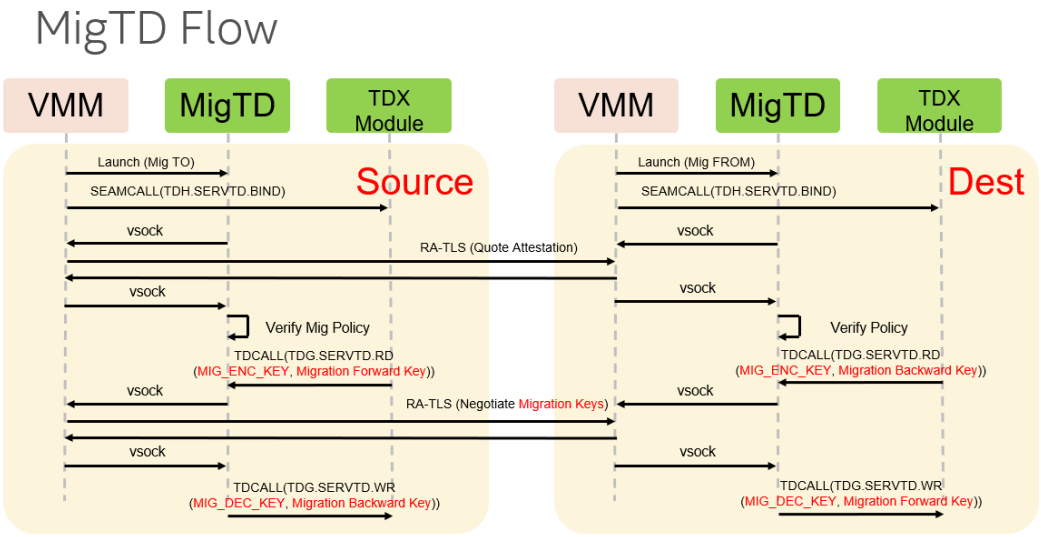

Figure 2-3: Migration TD flow

Care must be taken that a MigTD shall always use TDG.SERVTD.RD to read a fresh MIG_ENC_KEY from TDX-Module once, after the MigTD setup a secure session with a peer MigTD. This MIG_ENC_KEY shall be erased immediately after it is sent to the peer MigTD and shall never be used again in any migration session including current session or other sessions. This rule is for both source MigTD and destination MigTD.

§

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.
    - **VSock** 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.

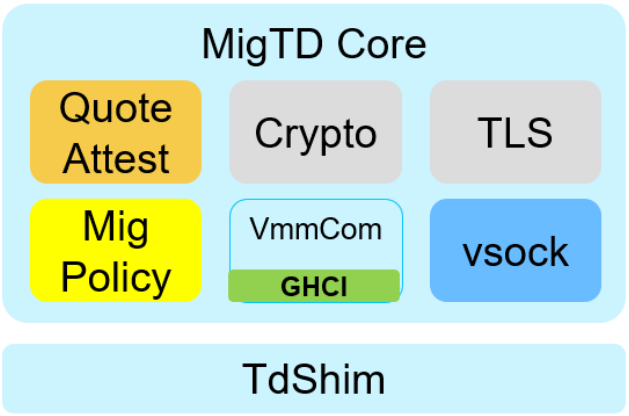

Figure 3-1: Migration TD Design

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 rest 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.

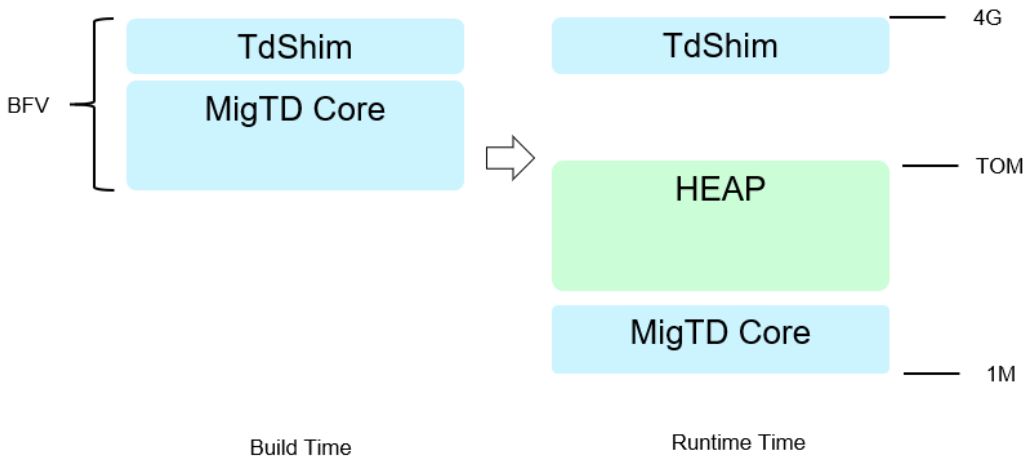

Figure 3-2: Migration TD Layout

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 vsock 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 vsock. 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.

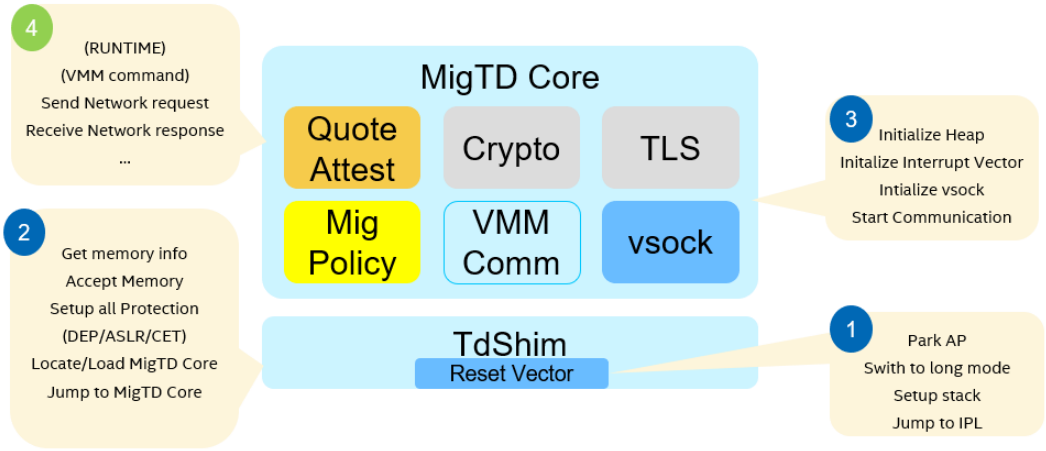

Figure 3-3: Migration TD Boot Flow

3.2 Reproducibility

A service TD binary should be reproducible. Please refer to "TDX Virtual Firmware Design Guide" for the reproducible support.

3.3 Security Consideration

MigTD should adopt the Security Consideration in the "TDX Virtual Firmware Design Guide".

3.3.1 Type safe language

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

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 an authenticated 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 not require private key or public certificate provision. The MigTD can generate an ephemeral 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.

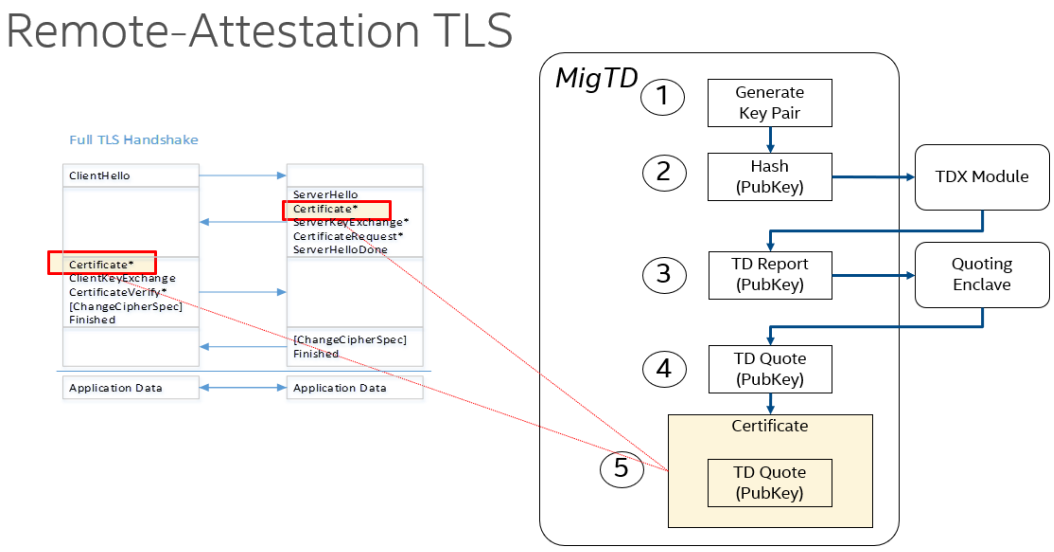

Figure 4-1: MigTD with RA-TLS

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.

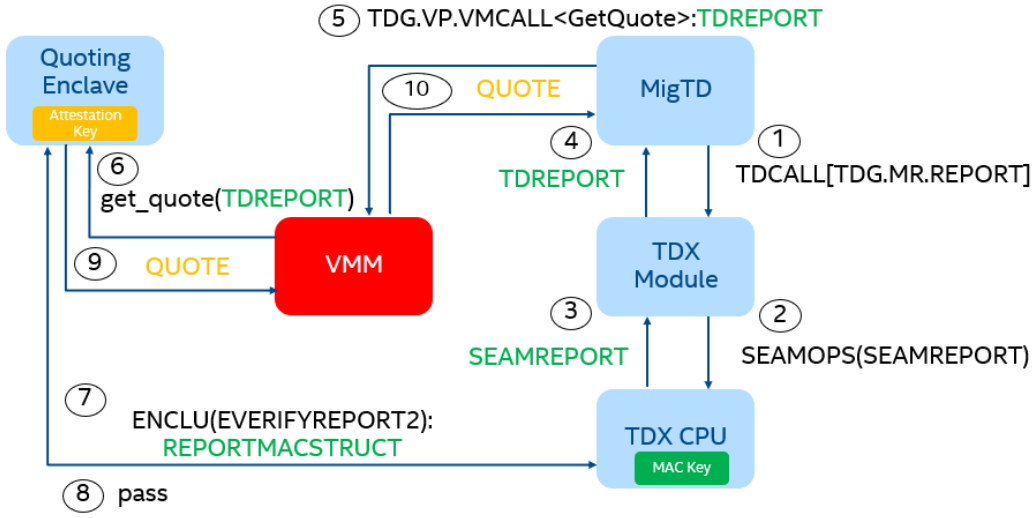

Figure 4-2: MigTD GetQuote Flow

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.

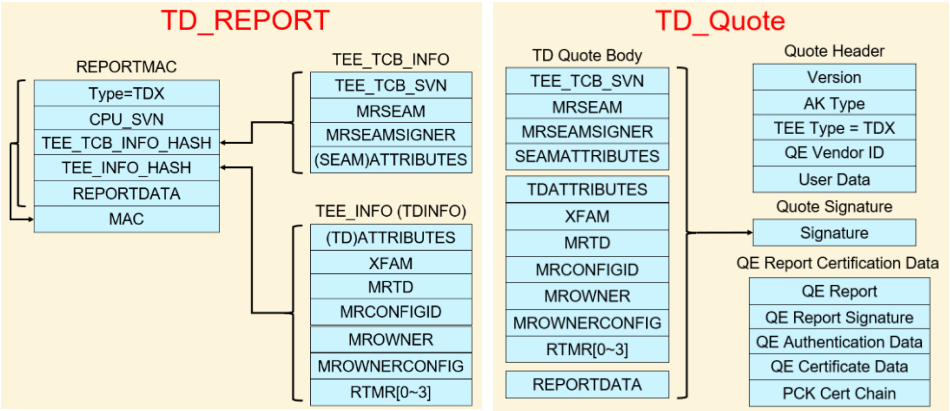

Figure 4-3: TD Report and TD Quote structure

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 4-3 shows the TD Report and Quote. Please refer to Intel TDX Module v1.5 ABI for detail of TD Report structure. Please refer to Intel TDX DCAP Quote Generation Library and Quote Verification Library for detail of TD Quote structure.

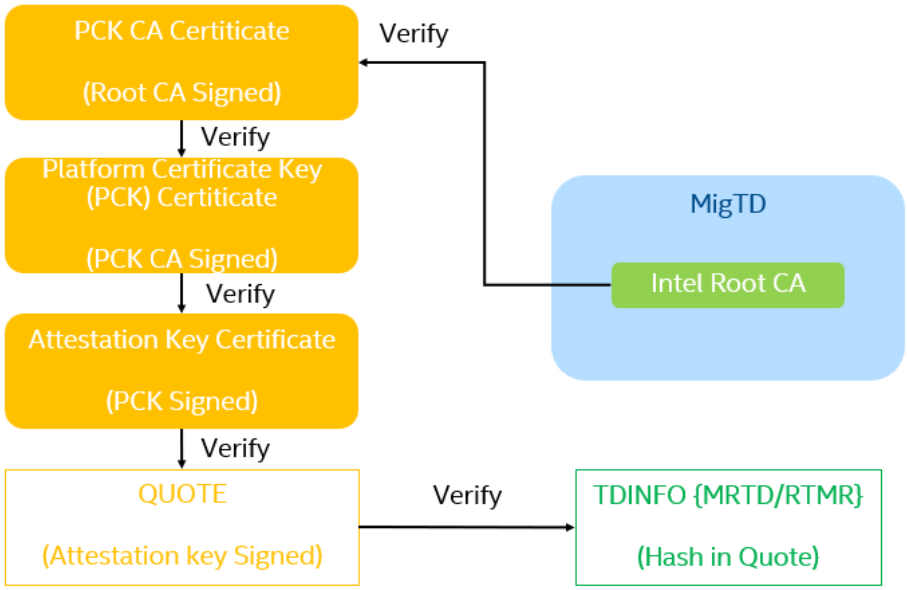

Figure 4-4: MigTD VerifyQuote Flow

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), the PCK certificate is signed with PCK CA Certificate and the PCK CA Certificate is signed with Intel root CA Certificate. See Figure 4-4. Please refer to Intel® SGX PCK Certificate and Certificate Revocation List Profile Specification for more detail.

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.

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. For example: **Serial Number: 1**	Mandatory
Signature Algorithm	Signature algorithm shall be present. For example: **sha384WithRSAEncryption**	Mandatory
Issuer	Issuer distinguished name shall be specified. 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. 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	Mandatory

	notAfter field should be assigned the generalized **99991231235959Z** time value.	
Subject Public Key Info	Device public key and the algorithm shall be present. For example: **Public Key Algorithm: rsaEncryption** **Modulus: ...** **Exponent: 65537 (0x10001)**	Mandatory
X509v3 Extension: Basic Constraints	**CA: FALSE**.	Optional
X509v3 Extension: Subject Key Identifier	Subject Key Identifier	Optional
X509v3 Extension: Authority Key Identifier	It should be same as Subject Key Identifier.	Optional
X509v3 Extension: Extended Key Usage	MigTD Quote Certificate indicator "**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 "**1.2.840.113741**.1.5.5.1.2" - Intel	Mandatory

OID: MigTD_event_log	EventLog Report “**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: ias_response_body_oid	Attestation Report. ias_report “**1.2.840.113741**.1337.2” - Intel	Mandatory
OID: ias_root_cert_oid	Attestation Report. ias_sign_ca_cert “**1.2.840.113741**.1337.3” - Intel	Mandatory
OID: ias_leaf_cert_oid	Attestation Report. ias_sign_cert “**1.2.840.113741**.1337.4” - Intel	Mandatory
OID: ias_report_signature_oid	Attestation Report. ias_report_signature “**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 "**1.2.840.113556**.10.1.1" - Microsoft	Mandatory
OID: oe_evidence	Attestation Report "**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: On export: **MIG_VERSION** is between [**MIN_EXPORT_VERSION, MAX_EXPORT_VERSION**] On import: **MIG_VERSION** is between [**MIN_IMPORT_VERSION, MAX_IMPORT_VERSION**] *Reference*: TDX Module Migration Specification.	MigTD on each side SHALL read the **MIN_EXPORT_VERSION, MAX_EXPORT_VERSION, MIN_IMPORT_VERSION, MAX_IMPORT_VERSION** using **TDG.SYS.RD**. MigTD on each side SHALL agree and set the **MIG_VERSION** to the migrated TD using **TDG.SERVD.WR**. MigTD SHOULD choose the highest common supported version. Code is extended to MRTD or RTMR[1] (See below).
TDX Module Policy (Required)	Intel	TDX Module Specification	**TDX_Module's MAJOR_VERSION/ MINOR_VERSION** is compatibility. **TD's ATTRIBUTES/XFAM/CPUID** is compatible. *Reference*: TDX Module Migration Specification.	MigTD optional checks it to detect error earlier. Code is extended to MRTD or RTMR[1] (See below).
Platform Policy	CSP	MigTD Design Guide	Platform is secure. The platform means the **TCB components**, such as **Microcode, SINIT ACM,**	MigTD checks the migration policy.

			SEAMLDR ACM, Quoting Enclave, TDX Module. There are two possible policies: • If source platform and destination platform are same, then destination should be equal or securer than the source. • If source platform and destination platform are different, then the destination platform should be up to date.	Policy Data is extended to RTMR[2]. (See below).
MigTD Default Policy (Recommended)	CSP	MigTD Design Guide	**MigTD** is compatible. (**SVN, MRTD**, etc.) **Certificate** is valid and not expired. (**Root CA certificate**, **certificate revocation list**)	MigTD checks the migration policy. 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
Platform	TcbInfo	tdxtcbcomponents	Integer Array	TDX TCB Component Security Level (16 bytes array)
		pcesvn	Integer	PCE SVN (UINT16)
		sgxtcbcomponents	Integer Array	SGX TCB Component Security Level (16 bytes array)
QE	QE_Identity	MISCSELECT	String	QE's MiscSelect. (UINT32)
		ATTRIBUTES	String	QE's Attributes (UINT128)
		MRENCLAVE	String	Measurement of the QE. (48 Bytes Hash)
		MRSIGNER	String	Measurement of the QE Signer. (48 Bytes Hash)
		ISVPRODID	Integer	Product ID of QE (UINT16)
		ISVSVN	Integer	SVN of QE (UINT16)

TDXModule	TDXModule_Identity *(The integrity is verified with TD Quote)*	TDXModuleMajorVersion	Integer	Intel TDX module Major Version. (UINT8)
		TDXModuleSVN	Integer	Intel TDX module SVN based on the major version. (UINT8)
		MRSEAM	String	Measurement of the Intel TDX module. (48 Bytes Hash)
		MRSIGNERSEAM	String	Measurement of TDX module signer if valid. (48 Bytes Hash)
		ATTRIBUTES	String	Additional configuration ATTRIBUTES if valid. (UINT64)
MigTD	TDINFO *(The integrity is verified with TD Quote)*	ATTRIBUTES	String	TD's Attributes. (UINT64)
		XFAM	String	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. (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.

The TcbInfo can be retrieved according to https://api.portal.trustedservices.intel.com/content/documentation.html#pcs-tcb-info-tdx-v4.

```
curl -v -X GET "https://api.trustedservices.intel.com/tdx/certification

/v4/tcb?fmspc={}"
```

For example:

```
============
{
    "tcbInfo": {
        "id": "TDX",
        "version": 3,
        "issueDate": "2023-12-13T10:46:28Z",
        "nextUpdate": "2024-01-12T10:46:28Z",
        "fmspc": "50806F000000",
        "pceId": "0000",
        "tcbType": 0,
        "tcbEvaluationDataNumber": 16,
        "tdxModule": {
            "mrsigner":
"000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000",
            "attributes": "0000000000000000",
            "attributesMask": "FFFFFFFFFFFFFFFF"
        },
        "tdxModuleIdentities": [
            {
                "id": "TDX_01",
                "mrsigner":
"000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000",
                "attributes": "0000000000000000",
                "attributesMask": "FFFFFFFFFFFFFFFF",
                "tcbLevels": [
                    {
```

```
                    "tcb": {
                        "isvsvn": 2
                    },
                    "tcbDate": "2023-08-09T00:00:00Z",
                    "tcbStatus": "UpToDate"
                }
            ]
        }
    ],
    "tcbLevels": [
        {
            "tcb": {
                "sgxtcbcomponents": [
                    {
                        "svn": 6,
                        "category": "BIOS",
                        "type": "Early Microcode Update"
                    },
                    {
                        "svn": 6,
                        "category": "OS/VMM",
                        "type": "SGX Late Microcode Update"
                    },
                    {
                        "svn": 2,
                        "category": "OS/VMM",
                        "type": "TXT SINIT"
                    },
                    {
                        "svn": 2,
                        "category": "BIOS"
                    },
                    {
                        "svn": 3,
                        "category": "BIOS"
                    },
                    {
                        "svn": 1,
                        "category": "BIOS"
                    },
                    {
                        "svn": 0
                    },
                    {
                        "svn": 3,
                        "category": "OS/VMM",
                        "type": "SEAMLDR ACM"
                    },
                    {
                        "svn": 0
                    },
```


```
        {
            "svn": 0
        },
        {
            "svn": 0
        },
        {
            "svn": 0
        },
        {
            "svn": 0
        },
        {
            "svn": 0
        },
        {
            "svn": 0
        },
        {
            "svn": 0
        }
    ],
    "pcesvn": 11,
    "tdxtcbcomponents": [
        {
            "svn": 3,
            "category": "OS/VMM",
            "type": "TDX Module"
        },
        {
            "svn": 0,
            "category": "OS/VMM",
            "type": "TDX Module"
        },
        {
            "svn": 6,
            "category": "OS/VMM",
            "type": "TDX Late Microcode Update"
        },
        {
            "svn": 0
        },
        {
            "svn": 0
        },
        {
            "svn": 0
        },
        {
            "svn": 0
        },
```

```
                        {
                            "svn": 0
                        },
                        {
                            "svn": 0
                        },
                        {
                            "svn": 0
                        },
                        {
                            "svn": 0
                        },
                        {
                            "svn": 0
                        },
                        {
                            "svn": 0
                        },
                        {
                            "svn": 0
                        },
                        {
                            "svn": 0
                        },
                        {
                            "svn": 0
                        },
                        {
                            "svn": 0
                        }
                    ]
                },
                "tcbDate": "2023-08-09T00:00:00Z",
                "tcbStatus": "UpToDate"
            }
            ...

        ]
    },
    "signature":
"ad76a5aad61adab356dfa23b102230af16e29922c8cbcf9fd7aadf6d2234ef297fe3d0
fa974b039023aa71251ab7765a929e5d01d473379a2c3070f03fe8305e"
}
============
```


The list of FMSPC can be found at
api.trustedservices.intel.com/sgx/certification/v4/fmspcs.

The QE_Identity can be retrieved according to
https://api.portal.trustedservices.intel.com/content/documentation.html#pcs-enclave-identity-v4.

```
curl -v -X GET " https://api.trustedservices.intel.com/sgx/certificatio

n/v4/qe/identity"
```

For example:

```
============
{
    "enclaveIdentity": {
        "id": "QE",
        "version": 2,
        "issueDate": "2023-12-13T10:42:00Z",
        "nextUpdate": "2024-01-12T10:42:00Z",
        "tcbEvaluationDataNumber": 16,
        "miscselect": "00000000",
        "miscselectMask": "FFFFFFFF",
        "attributes": "11000000000000000000000000000000",
        "attributesMask": "FBFFFFFFFFFFFFFF0000000000000000",
        "mrsigner":
"8C4F5775D796503E96137F77C68A829A0056AC8DED70140B081B094490C57BFF",
        "isvprodid": 1,
        "tcbLevels": [
            {
                "tcb": {
                    "isvsvn": 8
                },
                "tcbDate": "2023-08-09T00:00:00Z",
                "tcbStatus": "UpToDate"
            },
            ...
        ]
    },
    "signature":
"58acf3e37502ecca36d516e07e4a75686b5cac7502e196832c28ec31389c3ebecb27c3
1e9211873287d67d8494cdbf4422913644e2b5db125e9245f3d14fc197"
}
============
```

5.3.2 MigTD Policy Format

MigTD should use the industry standard for the policy data format – **JSON**.
The fmspc is 6 bytes strings to identify the platform. If fmspc is absent, the source platform and destination platform shall be same.

```
============
{
    "id": <GUID>,
```

```
    "policy": [
        {
            "fmspc": <FMSPC-value>,
            <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”
Integer	“equal”	“self”	Used to ensure two MigTDs have same data.
Integer Array	“array-equal”, “array-greater-or-equal”	An array of Integer value	Each integer value in the array shall be equal or greater-or-equal than the reference integer value in the array.
Integer Array	“equal”	“self”	Used to ensure two MigTDs have same data.
String	“equal”	String value for a hex byte array	
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. (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",
    "policy": [
        {
            "Platform": {
                "fmspc": "self",
                "TcbInfo": {
                    "sgxtcbcomponents": {
                        "operation": "array-equal",
                        "reference": "self"
                    },
                    "pcesvn": {
                        "operation": "equal",
                        "reference": "self"
                    },
                    "tdxtcbcomponents": {
                        "operation": "array-equal",
                        "reference": "self"
                    }
                }
            }
        },
        {
            "QE": {
                "QeIdentity": {
```

```
                "MISCSELECT": {
                    "operation": "equal",
                    "reference": "self"
                },
                "ATTRIBUTES": {
                    "operation": "equal",
                    "reference": "self"
                },
                "MRSIGNER": {
                    "operation": "equal",
                    "reference": "self"
                },
                "ISVPRODID": {
                    "operation": "equal",
                    "reference": "self"
                },
                "ISVSVN": {
                    "operation": "equal",
                    "reference": "self"
                }
            }
        }
    },
    {
        "TDXModule": {
            "TDXModule_Identity": {
                "TDXModuleMajorVersion": {
                    "operation": "equal",
                    "reference": "self"
                },
                "TDXModuleSVN": {
                    "operation": "equal",
                    "reference": "self"
                },
                "MRSEAM": {
                    "operation": "equal",
                    "reference": "self"
                },
                "MRSIGNERSEAM": {
                    "operation": "equal",
                    "reference": "self"
                },
                "ATTRIBUTES": {
                    "operation": "equal",
                    "reference": "self"
                }
            }
        }
    },
    {
        "MigTD": {
```

```
                "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"
                    },
                    "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",
    "policy": [
        {
            "Platform": {
                "fmspc": "self",
                "TcbInfo": {
                    "sgxtcbcomponents": {
                        "operation": "array-equal",
                        "reference": "self"
                    },
                    "pcesvn": {
                        "operation": "equal",
                        "reference": "self"
                    },
                    "tdxtcbcomponents": {
                        "operation": "array-equal",
                        "reference": "self"
                    }
                }
            }
        },
        {
            "QE": {
                "QeIdentity": {
                    "MISCSELECT": {
                        "operation": "equal",
                        "reference": "self"
                    },
                    "ATTRIBUTES": {
                        "operation": "equal",
                        "reference": "self"
                    },
                    "MRSIGNER": {
                        "operation": "equal",
                        "reference": "self"
                    },
                    "ISVPRODID": {
                        "operation": "equal",
                        "reference": "self"
                    },
                    "ISVSVN": {
                        "operation": "equal",
                        "reference": "self"
```

```
                }
            }
        }
    },
    {
        "TDXModule": {
            "TDXModule_Identity": {
                "TDXModuleMajorVersion": {
                    "operation": "equal",
                    "reference": "self"
                },
                "TDXModuleSVN": {
                    "operation": "equal",
                    "reference": "self"
                },
                "MRSEAM": {
                    "operation": "equal",
                    "reference": "self"
                },
                "MRSIGNERSEAM": {
                    "operation": "equal",
                    "reference": "self"
                },
                "ATTRIBUTES": {
                    "operation": "equal",
                    "reference": "self"
                }
            }
        }
    },
    {
        "MigTD": {
            "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": {
                    "Digest.TdShim": {
                        "operation": "equal",
                        "reference": "self"
                    },
                    "Digest.SecureBootKey": {
                        "operation": "equal",
                        "reference": "self"
                    },
                    "Digest.MigTdPolicy": {
                        "operation": "equal",
                        "reference": "self"
                    },
                    "Digest.MigTdCoreSvn": {
                        "operation": "greater-or-equal",
                        "reference": 13
                    }
                }
            }
        }
    ]
}
============
```

5.3.3.3 Sample Policy for cross platform migration

In this mode, we must indicate the TcbInfo and QEIdentity explicitly in the policy.

```
============
{
    "id": "xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx",
    "policy": [
        {
            "fmspc": "20c06f000000",
            "Platform": {
                "TcbInfo": {
                    "sgxtcbcomponents": {
                        "operation": "array-equal",
                        "reference":
[33,33,0,0,0,0,0,0,0,0,0,0,0,0,0,0]
                    },
                    "pcesvn": {
```


```
                            "operation": "equal",
                            "reference": 0
                        },
                        "tdxtcbcomponents": {
                            "operation": "array-equal",
                            "reference": [0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0]
                        }
                    }
                }
            },
            {
                "fmspc": "30806f000000",
                "Platform": {
                    "TcbInfo": {
                        "sgxtcbcomponents": {
                            "operation": "array-equal",
                            "reference": [1,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0]
                        },
                        "pcesvn": {
                            "operation": "equal",
                            "reference": 11
                        },
                        "tdxtcbcomponents": {
                            "operation": "array-equal",
                            "reference": [0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0]
                        }
                    }
                }
            },
            {
                "QE": {
                    "QeIdentity": {
                        "MISCSELECT": {
                            "operation": "equal",
                            "reference": "00000000"
                        },
                        "ATTRIBUTES": {
                            "operation": "equal",
                            "reference": "11000000000000000000000000000000"
                        },
                        "MRSIGNER": {
                            "operation": "equal",
                            "reference":
"8C4F5775D796503E96137F77C68A829A0056AC8DED70140B081B094490C57BFF"
                        },
                        "ISVPRODID": {
                            "operation": "equal",
                            "reference": 1
                        },
                        "ISVSVN": {
                            "operation": "equal",
```

```
                        "reference": 8
                    }
                }
            }
        },
        {
            "TDXModule": {
                "TDXModule_Identity": {
                    "TDXModuleMajorVersion": {
                        "operation": "equal",
                        "reference": 1
                    },
                    "TDXModuleSVN": {
                        "operation": "equal",
                        "reference": 0
                    },
                    "MRSIGNERSEAM": {
                        "operation": "equal",
                        "reference":
"000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000"
                    },
                    "ATTRIBUTES": {
                        "operation": "equal",
                        "reference": "0000000000000000"
                    }
                }
            }
        },
        {
            "MigTD": {
                "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"
                    },
                    "RTMR0": {
                        "operation": "equal",
                        "reference": "self"
                    },
                    "RTMR1": {
                        "operation": "equal",
                        "reference": "self"
                    },
                    "RTMR2": {
                        "operation": "equal",
                        "reference": "self"
                    },
                    "RTMR3": {
                        "operation": "equal",
                        "reference": "self"
                    }
                }
            }
        }
    ]
}
============
```

§

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.

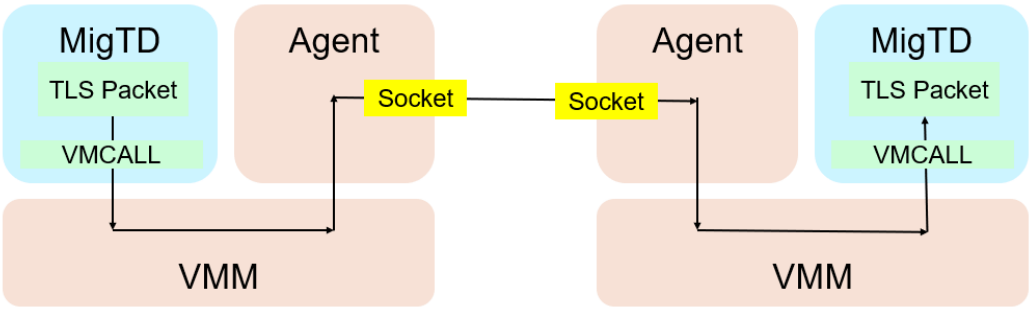

Figure 6-1: VMCALL based communication in MigTD

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

Table 7-2: MigTD TLS Configuration

TLS Configuration	Options	Recommendation
Version	1.2, 1.3	1.3
Cipher Suite	**V1.3** TLS_AES_256_GCM_SHA384 TLS_AES_128_GCM_SHA256 TLS_CHACHA20_POLY1305_SHA256 **V1.2** TLS_ECDHE_RSA_WITH_AES_256_GCM_SHA384	**V1.3** TLS_AES_256_GCM_SHA384

	TLS_ECDHE_ECDSA_WITH_AES_256_GCM_SHA384 TLS_ECDHE_RSA_WITH_AES_128_GCM_SHA256 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.

The file system GUID shall be **EFI_FIRMWARE_FILE_SYSTEM2_GUID** or **EFI_FIRMWARE_FILE_SYSTEM3_GUID**, which is defined in PI specification, or **EFI_SYSTEM_NV_DATA_FV_GUID**, which is defined in https://github.com/tianocore/edk2/blob/master/MdeModulePkg/Include/Guid/SystemNvDataGuid.h.

Table 8-1: MigTD Policy Data

Item	Description	Measurement
MigTD Policy	JSON file	RTMR[2]
Root CA Cert (Trust Anchor)	One X.509 certificate.	RTMR[2]
Certificate Revocation List	One Certificate Revocation List.	RTMR[2]
Secure Boot Key	One secure boot public key. (only applicable for SecureBootMode)	RTMR[0]

§

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.

§

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 as required for TDVF. BFV and payload are required to indicate the MigTD image. CFV might be required, if CFV based policy is supported. The temp memory (stack / heap) and permanent memory are required to indicate the memory to be used by MigTD. TD HOB is NOT required in the MigTD. VMM shall follow the entries in TDVF descriptor to create the MigTD memory one by one.

§

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.

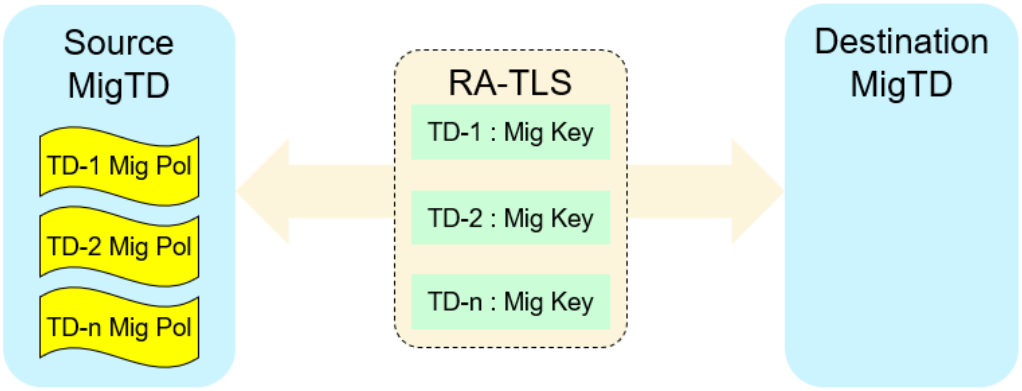

Figure 12-1: Multiple TDs Migration

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.

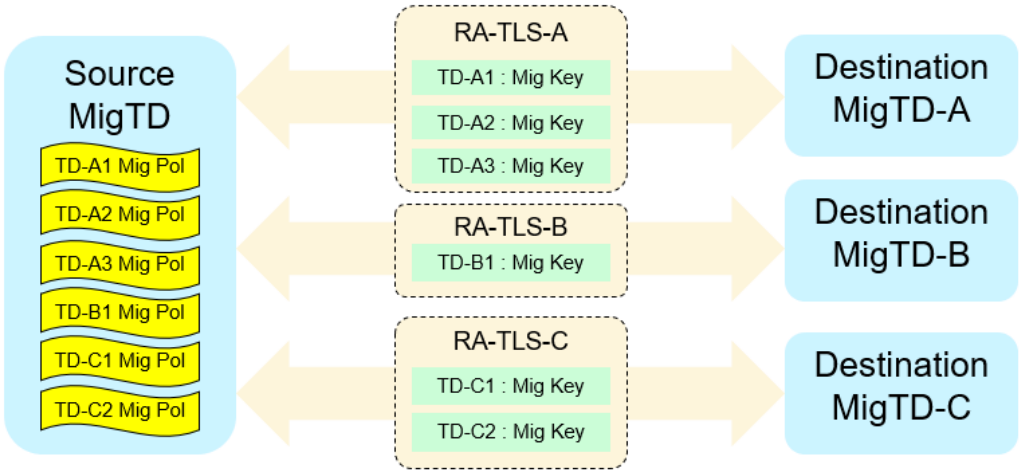

Figure 12-2: Multiple Destination Migration TDs

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.

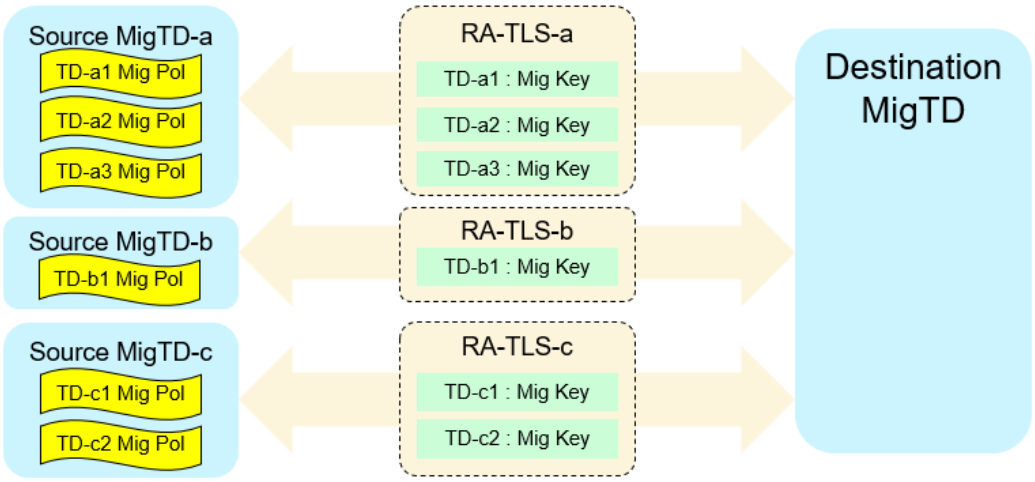

Figure 12-3: Multiple Source Migration TDs

§

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

[Intel Attestation] Intel Attestation Service, https://api.portal.trustedservices.intel.com/documentation

[Intel PCK Service] Intel Provisioning Certificate Service, https://api.portal.trustedservices.intel.com/provisioning-certification

[Intel PCK CRL] Intel® SGX PCK Certificate and Certificate Revocation List Profile Specification, https://api.trustedservices.intel.com/documents/Intel_SGX_PCK_Certificate_CRL_Spec-1.5.pdf

[Intel SGX DCAP Quote Lib] Intel® SGX Data Center Attestation Primitive: ECDSA Quote Library API, https://download.01.org/intel-sgx/latest/dcap-latest/linux/docs/Intel_SGX_ECDSA_QuoteLibReference_DCAP_API.pdf

[Intel TDX DCAP Quote Lib] Intel® Trust Domain Extensions Data Center Attestation Primitives (Intel® TDX DCAP): Quote Generation Library and Quote Verification Library, https://download.01.org/intel-sgx/latest/dcap-latest/linux/docs/TDX_Quoting_Library_API.pdf

[Intel SGX DCAP Grace Periods] Quote Verification Grace Periods with Intel® Software Guard Extensions Data Center Attestation Primitive, https://www.intel.com/content/www/us/en/developer/articles/technical/grace-periods-for-intel-sgx-dcap.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

§