ETSI TS 103 457 V1.1.1 (2018-10)

TECHNICAL SPECIFICATION

CYBER;
Trusted Cross-Domain Interface:
Interface to offload sensitive functions to a trusted domain

Reference

DTS/CYBER-0019

Keywords

cybersecurity, interface

ETSI

650 Route des Lucioles
F-06921 Sophia Antipolis Cedex - FRANCE

Tel.: +33 4 92 94 42 00 Fax: +33 4 93 65 47 16

Siret N° 348 623 562 00017 - NAF 742 C
Association à but non lucratif enregistrée à la
Sous-Préfecture de Grasse (06) N° 7803/88

Important notice

The present document can be downloaded from:
http://www.etsi.org/standards-search

The present document may be made available in electronic versions and/or in print. The content of any electronic and/or print versions of the present document shall not be modified without the prior written authorization of ETSI. In case of any existing or perceived difference in contents between such versions and/or in print, the only prevailing document is the print of the Portable Document Format (PDF) version kept on a specific network drive within ETSI Secretariat.

Users of the present document should be aware that the document may be subject to revision or change of status. Information on the current status of this and other ETSI documents is available at
https://portal.etsi.org/TB/ETSIDeliverableStatus.aspx

If you find errors in the present document, please send your comment to one of the following services:
https://portal.etsi.org/People/CommiteeSupportStaff.aspx

Copyright Notification

Contents

Intellectual Property Rights

Essential patents

IPRs essential or potentially essential to normative deliverables may have been declared to ETSI. The information pertaining to these essential IPRs, if any, is publicly available for **ETSI members and non-members**, and can be found in ETSI SR 000 314: *"Intellectual Property Rights (IPRs); Essential, or potentially Essential, IPRs notified to ETSI in respect of ETSI standards"*, which is available from the ETSI Secretariat. Latest updates are available on the ETSI Web server (https://ipr.etsi.org/).

Pursuant to the ETSI IPR Policy, no investigation, including IPR searches, has been carried out by ETSI. No guarantee can be given as to the existence of other IPRs not referenced in ETSI SR 000 314 (or the updates on the ETSI Web server) which are, or may be, or may become, essential to the present document.

Trademarks

The present document may include trademarks and/or tradenames which are asserted and/or registered by their owners. ETSI claims no ownership of these except for any which are indicated as being the property of ETSI, and conveys no right to use or reproduce any trademark and/or tradename. Mention of those trademarks in the present document does not constitute an endorsement by ETSI of products, services or organizations associated with those trademarks.

Foreword

This Technical Specification (TS) has been produced by ETSI Technical Committee Cyber Security (CYBER).

Modal verbs terminology

In the present document "**shall**", "**shall not**", "**should**", "**should not**", "**may**", "**need not**", "**will**", "**will not**", "**can**" and "**cannot**" are to be interpreted as described in clause 3.2 of the ETSI Drafting Rules (Verbal forms for the expression of provisions).

"**must**" and "**must not**" are **NOT** allowed in ETSI deliverables except when used in direct citation.

Introduction

Deploying hosted sensitive functions in modern virtualized IT infrastructure is still a concern and a major issue.

The main threats are malicious administrators operating the IT infrastructure including: network, storage and host platform facilities and virtualization management. These threats and related issues are thoroughly discussed in ETSI TR 103 308 [i.1].

ETSI specification group NFV SEC is in charge of defining a secured standard architecture. Proprietary solutions providing trusted security for virtualized environments have started emerging.

These new envisioned architectures add security components at the hosting platform level and into centralized services in charge of security management. The key concept is Hardware Root of Trust to get strong guarantees on the integrity of the deployed elements. These architectures offer secured managed infrastructures that enable deployment, live migration of encrypted VMs.

In addition to these works, the present document proposes a new interoperable interface that should help building sensitive services with trust.

This interface applies in the setting where two trust domains (see ETSI GS NFV-SEC 013 [i.4] for details) are defined:

- The More Trusted Domain (MTD) contains resources (network, storage, processing) where sensitive functions can be offloaded.

- The Less Trusted Domain (LTD) contains resources that can be managed without the risk of compromising sensitive information, since these functionalities are offloaded to the MTD.

This Trusted Cross-Domain interface includes a set of basic functions called by the LTD entity but performed securely within the MTD. This set of basic functions enables the LTD entity to build more complex services.

1 Scope

The present document specifies a high-level service-oriented interface, as an application layer with a set of mandatory functions, to access secured services provided by, and executed in a More Trusted Domain. The transport layer is out of scope and left to the architecture implementation.

This interface is not considered as a replacement of the already existing technologies (such as PKCS#11, KMIP, etc.) but rather operating on top of these.

2 References

2.1 Normative references

References are either specific (identified by date of publication and/or edition number or version number) or non-specific. For specific references, only the cited version applies. For non-specific references, the latest version of the referenced document (including any amendments) applies.

Referenced documents which are not found to be publicly available in the expected location might be found at https://docbox.etsi.org/Reference.

NOTE: While any hyperlinks included in this clause were valid at the time of publication, ETSI cannot guarantee their long-term validity.

Not applicable.

2.2 Informative references

References are either specific (identified by date of publication and/or edition number or version number) or non-specific. For specific references, only the cited version applies. For non-specific references, the latest version of the referenced document (including any amendments) applies.

NOTE: While any hyperlinks included in this clause were valid at the time of publication, ETSI cannot guarantee their long-term validity.

The following referenced documents are not necessary for the application of the present document but they assist the user with regard to a particular subject area.

[i.1] ETSI TR 103 308: "CYBER; Security baseline regarding LI and RD for NFV and related platforms".

[i.2] ETSI GR NFV-SEC 011: "Network Functions Virtualisation (NFV); Security; Report on NFV LI Architecture".

[i.3] Wikipedia definition of Type-Length-Value.

[i.4] ETSI GS NFV-SEC 013: "Network Functions Virtualisation (NFV) Release 3; Security ; Security Management and Monitoring specification".

3 Definition of terms and abbreviations

3.1 Terms

For the purposes of the present document, the following terms apply:

domain: set of domain services

trusted cross domain interface: domain service with a set of dedicated domain interface functions for communication between domain services of different domains (inter-domain communication)

trusted cross domain interface function: function of a domain interface which is implemented by a domain service of another domain in order to realize inter-domain communication

trusted cross domain object: data generated by a domain service

trusted cross domain service: service with a set of dedicated domain service functions for communication with other domain services of the same domain (intra-domain communication)

trusted cross domain service function: function of a domain service which is implemented by the same or another domain service in order to realize intra-domain communication

trusted cross secured domain interface: domain interface offering access to secured domain services

3.2 Abbreviations

For the purposes of the present document, the following abbreviations apply:

AC	Access Control
AES	Advanced Encryption Standard
CN	Common Name
FW	FireWall
HSM	Hardware Security Module
IT	Information Technology
KMIP	Key Management Interoperability Protocol
LI	Lawful Interception
LTD	Less Trusted Domain
MF	Mediation Function
MTD	More Trusted Domain
NFV	Network Function Virtualization
PNRG	Pseudorandom Number Generator
RNG	Random Number Generator
RSA	Rivest-Shamir-Adleman
SEC	Security
TCDI	Trusted Cross-Domain Interface
TCF	Triggering Control Function
TCO	Trusted COntext
TLS	Transport Layer Security
TPM	Trusted Platform Module
TTLV	Tag-Type-Length-Value encoding
VM	Virtual Machine
VMM	Virtual Machine Manager
vPOI	virtual Point Of Interception

4 General

4.1 TCDI functional requirements

TCDI provides services to the application layer. MTD implements, exposes and delivers the required services.

TCDI provides the following high-level services:

- Key Management: TCDI allows symmetric and asymmetric keys to be requested and received.
- Cryptographic operations: TCDI allows basic cryptographic operation to be performed in the MTD.

EXAMPLE: Random number generator.

- File/Database/Storage access: TCDI provides services to append, push and store sensitive data in containers such as files or database.

TCDI shall allow the use of sensitive objects in several functions without regeneration and compromise.

TCDI shall allow the sharing of MTD domain objects by LTD entities.

A LTD entity shall provide attestations on demand to the MTD, and the MTD shall verify those attestations to ensure the trust relation between the domains.

TCDI shall allow cascading the execution of domain service functions of the LTD on domain objects of the MTD within a single session.

4.2 TCDI life cycle

4.2.1 Life cycle Diagram

The interface allows a LTD entity to establish a trusted connection to a server in the MTD to execute sensitive operations and compose results within a TCO guaranteed by the MTD server (illustrated in figure 1). Only one connection per LTD entity shall be accepted.

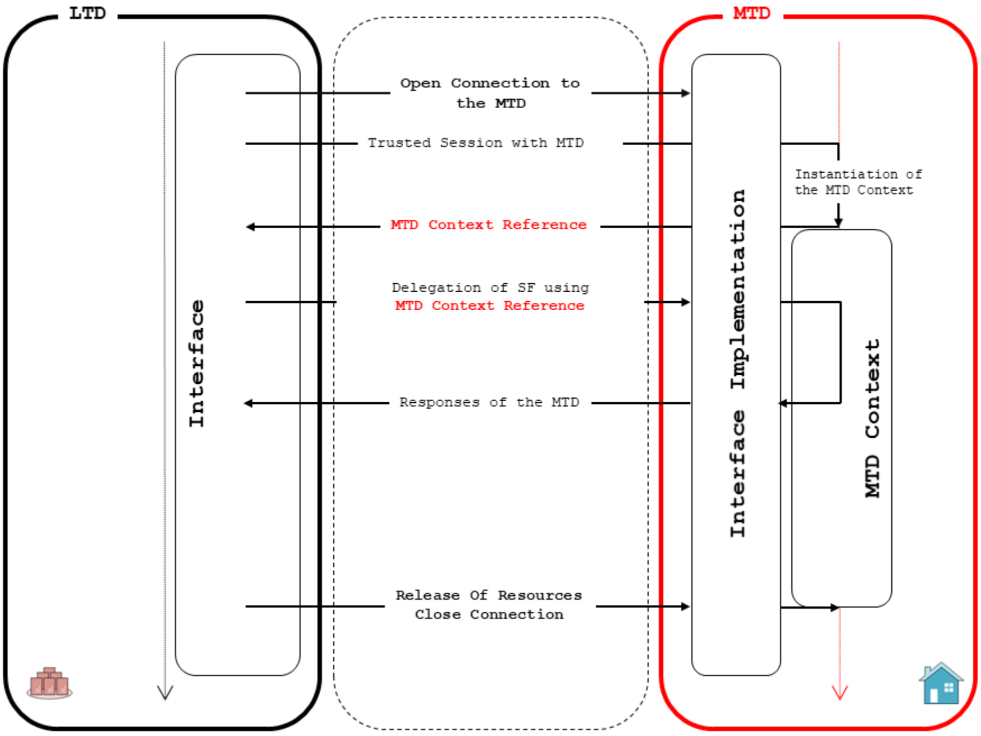
LTD
MTD
Interface
Open Connection to the MTD
Trusted Session with MTD
MTD Context Reference
Delegation of SF using MTD Context Reference
Responses of the MTD
Release Of Resources Close Connection
Interface Implementation
Instantiation of the MTD Context
MTD Context

Figure 1: Interface Life Cycle

4.2.2 Connection between LTD and MTD

The use of TCDI is initiated by one LTD with the establishment of a connection to one MTD service. The MTD may close the connection after some period of inactivity (see Good Practice section for recommendations).

The MTD shall support two modes of operation depending on the LTD trust level:

- Trusted mode enables the MTD to verify that the LTD is running in an authorized environment (see Good Practice section for recommendations).
- Untrusted hardware mode enables the use of TCDI when the LTD does not have access to a TPM.

The MTD shall have a database of authorized RSA key pairs and the LTD shall be able to sign data as a TPM would.

MTD is responsible for granting the appropriate level of services available to a LTD connection depending on the trust level and the requested LTD-Role. MTD shall deny connections if the requested LTD-Role does not match the trust level.

MTD shall accept only one connection per LTD, and simultaneous connections from multiple LTD. New connections to the MTD shall get rejected if the supported limit of simultaneous connections is reached.

4.2.3 Session

A session describes a set of transactions between the LTD and MTD for which an ephemeral TCO is created to secure all the sensitive data generated or managed, either simple objects or containers.

The MTD generates and associates a unique identifier to sensitive data and guarantees their unicity:

- Session-Id to each session.
- Object-Id to each object.
- Container-Id to each container.

Session creation within an existing session returns an error.

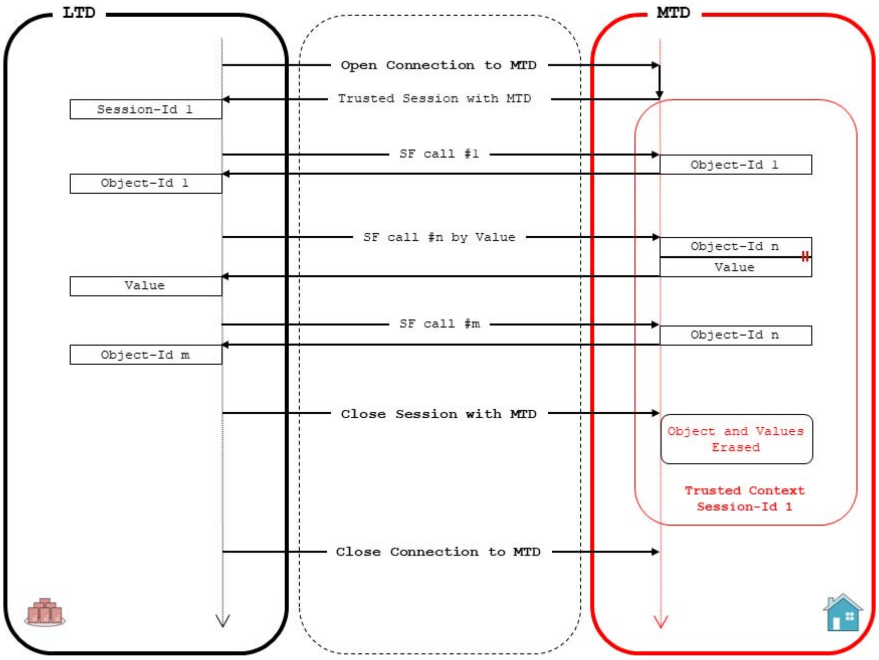

Figure 2: Session based calls to sensitive functions

4.2.4 Keep the trusted connection between the LTD and the MTD

Depending on the deployment scenario, the MTD may have different requirements for the acceptable interval between re-attestations. The MTD may take different actions depending on the attestation state of the LTD.

EXAMPLE: The MTD can give access to certain resources only when certain pre-conditions are met by an LTD attestation.

The LTD manages its connection's lifetime by renewing the trust connection.

Upon expired connection, the MTD terminates the current session with the LTD entity.

If the MTD ends a session and connection because of expired connection's lifetime, the LTD shall set up a new connection and a new session to re-start the delegation of sensitive functions.

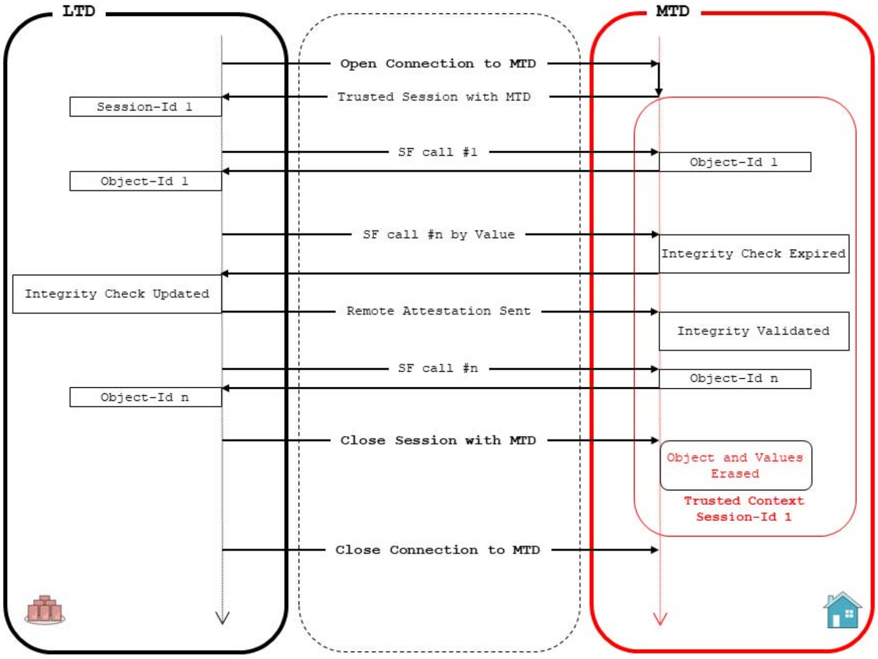

Figure 3: Attestation Check Success

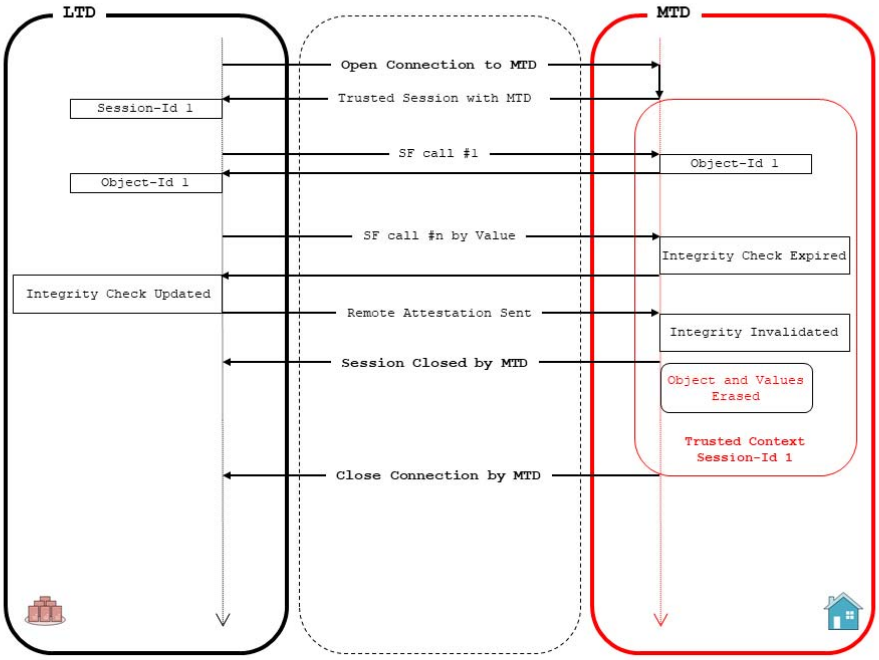

Figure 4: Attestation Check Failure

4.2.5 Releasing and erasing

When a LTD entity has finished offloading SFs or decides to request the erasing of the trusted context, the LTD entity may close the current session.

The MTD shall securely erase the trusted context of the session upon closure initiated by the LTD entity or when the connection's lifetime expires.

5 Interface Elementary Functions

5.1 General provisions

Elementary functions are achieved using simple communication command/response pattern where the MTD executes and returns response on the solicitation of the LTD entity.

Responses have the form of an optional result data value or a reference Object-id to that value and a status error code of the operation.

Every function runs inside a TCO initiated by a session.

Each function waits a synchronous response; therefore, functions shall be called sequentially.

Results may be void in the response message in case of error.

Protocol messages are composed of a one-byte message identifier, followed by a sequence of TTLV encoded parameters.

As described in [i.3], variable length typed element of information is binary encoded into 4 concatenated parts:

- Tag, 1 byte, used as a symbolic type for the element.
- Type, 2 bytes, used as the practical type of encoding.
- Length, 4 bytes.
- Value, variable sized information.

Several literal types, such as integer, Unicode or binary string, or symbolic constant are used for simple information. Some composed types such as DBKeyValue pairs are used for more structured information.

The complete list of Message identifiers is defined in clause 6.1. The list of Tags and Types for TTLV is defined in clauses 6.2 and 6.3.

For each message command and response defined in clauses 5.2 to 5.6, message parameters are defined in tables. In the column "status" the abbreviations have the following meaning:

M: Mandatory.
The parameter shall be present.

R: Recommended.
The parameter should be present.

O: Optional.
The parameter may be present.

C: Conditional.
The parameter shall be present when the defined conditions are met.

The description of and common provisions for message parameters are defined in clauses 6.3 and 6.4.

5.2 Connection and session management

5.2.1 General

The MTD is responsible of storing a configuration of database type for LTD entities based on their LTD-Role. A Container-Id reference to the configuration of the MTD is returned at connection opening. The configuration shall be read-only.

5.2.2 TD_OpenConnection

TD_OpenConnection opens a connection between the LTD and the MTD.

```
TD_OpenConnection (LTD-Id, LTD-Role, CN, Nonce, DATA)
```

Returns a Container-Id and status code.

Table 1: TD_OpenConnection command

Command parameter	Status	Description and Requirements
LTD-Id	M	Identifier of the requesting entity.
LTD-Role	M	Role of the requesting entity.
CN	M	The certificate common name CN associated to a RSA TPM-Key used by LTD. MTD shall verify that the LTD is running on an authorized hardware. The MTD shall have a database of authorized TPM RSA key pairs.
Nonce	M	Nonce value computed upon connection by the MTD.
DATA	M	See clause 6.3. MTD shall verify the signature to trust the LTD and accept the connection.

Table 2: TD_OpenConnection response

Command parameter	Status	Description and Requirements
Container-Id	M	Configuration container for the MTD.
Status code	M	It shall be one of these values: • TDSC_SUCCESS • TDSC_UNKNOWN_ROLE • TDSC_TRUST_REFUSED • TDSC_GENERAL_FAILURE

5.2.3 TD_CloseConnection

TD_CloseConnection closes the connection between the LTD and the MTD.

```
TD_CloseConnection ()
```

Returns a status code.

Table 3: TD_CloseConnection command

Command parameter	Status	Description and Requirements
none	-	-

Table 4: TD_CloseConnection response

Command parameter	Status	Description and Requirements
Status code	M	It shall be one of these values: • TDSC_SUCCESS • TDSC_GENERAL_FAILURE

5.2.4 TD_CreateSession

TD_CreateSession creates a session between the LTD and the MTD.

```
TD_CreateSession ()
```

Returns Session-Id and a status code.

Table 5: TD_CreateSession command

Command parameter	Status	Description and Requirements
none	-	-

Table 6: TD_CreateSession response

Command parameter	Status	Description and Requirements
Session-Id	C	See clause 6.3. The MTD shall return a Session-Id when the command is successful.
Status code	M	It shall be one of these values: • TDSC_SUCCESS • TDSC_SESSION_ID_ALREADY_OPENED • TDSC_TOO_MANY_EXISTING_SESSIONS • TDSC_TRUST_EXPIRED • TDSC_GENERAL_FAILURE

5.2.5 TD_CloseSession

TD_CloseSession closes a session between the LTD and the MTD.

```
TD_CloseSession (Session-Id)
```

Returns a status code.

Table 7: TD_CloseSession command

Command parameter	Status	Description and Requirements
Session-Id	M	See clause 6.3

Table 8: TD_CloseSession response

Command parameter	Status	Description and Requirements
Status code	M	It shall be one of these values: • TDSC_SUCCESS • TDSC_UNKNOWN_SESSION_ID • TDSC_ON_GOING_PROCESSES TDSC_GENERAL_FAILURE

5.2.6 TD_TrustRenewal

TD_TrustRenewal is used to verify that the LTD entity execution context has not been modified. The MTD shall have a database of authorized RSA Public keys indexed by CN reference measurements indexed to LTD-Role. The provisioning of this CN is out of scope, and it is assumed that it is trustworthy.

It is used to regain trust form MTD after some time.

```
TD_TrustRenewal (Session-Id, CN, Nonce, DATA)
```

Returns a status code.

The MTD shall verify the attestation to accept continuing delivering services. In case of failed verification, the MTD shall disconnect from the LTD.

Table 9: TD_TrustRenewal command

Command parameter	Status	Description and Requirements
Session-Id	M	See clause 6.3
CN	M	See clause 6.3 The MTD shall use CN to decrypt DATA
Nonce	M	Nonce value computed upon connection by the MTD
DATA	M	See clause 6.3 MTD shall verify the signature to accept to trust the LTD entity and the connection

Table 10: TD_TrustRenewal response

Command parameter	Status	Description and Requirements
Status code	M	It shall be one of these values: • TDSC_SUCCESS • TDSC_UNKNOWN_SESSION_ID • TDSC_ATTESTATION_FAILED • TDSC_TRUST_EXPIRED • TDSC_GENERAL_FAILURE

5.3 Data and value management

5.3.1 TD_CreateObject

TD_CreateObject returns the Object-Id that has been created.

```
TD_CreateObject (Session-Id)
```

Returns the Object-Id and a status code.

Table 11: TD_CreateObject command

Command parameter	Status	Description and Requirements
Session-Id	M	See clause 6.3

Table 12: TD_CreateObject response

Command parameter	Status	Description and Requirements
Object-Id	M	See clause 6.3
Status code	M	It shall be one of these values: • TDSC_SUCCESS • TDSC_UNKNOWN_SESSION_ID • TDSC_TRUST_EXPIRED • TDSC_OBJECT_CREATION_FAILED • TDSC_GENERAL_FAILURE

5.3.2 TD_GetObjectValue

TD_GetObjectValue returns the value associated with an Object-Id.

```
TD_GetObjectValue (Session-Id, Object-Id)
```

Returns the object content/value referenced by Object-Id and a status code.

Table 13: TD_GetObjectValue command

Command parameter	Status	Description and Requirements
Session-Id	M	See clause 6.3
Object-Id	M	See clause 6.3

Table 14: TD_GetObjectValue response

Command parameter	Status	Description and Requirements
DATA	C	Value corresponding to the Object-Id if TDSC_SUCCESS
Status code	M	It shall be one of these values: • TDSC_SUCCESS • TDSC_UNKNOWN_SESSION_ID • TDSC_UNKNOWN_OBJECT_ID • TDSC_TRUST_EXPIRED • TDSC_GENERAL_FAILURE

5.3.3 TD_PutObjectValue

TD_PutObjectValue modifies the value of the object referenced by Object-id in the MTD.

```
TD_PutObjectValue (Session-Id, Object-Id, DATA)
```

Returns a status code.

Table 15: TD_PutObjectValue command

Command parameter	Status	Description and Requirements
Session-Id	M	See clause 6.3
Object-Id	M	See clause 6.3
DATA	M	See clause 6.3

Table 16: TD_PutObjectValue response

Command parameter	Status	Description and Requirements
Status code	M	It shall be one of these values: • TDSC_SUCCESS • TDSC_UNKNOWN_SESSION_ID • TDSC_UNKNOWN_OBJECT_ID • TDSC_TRUST_EXPIRED • TDSC_GENERAL_FAILURE

5.4 Transferring cryptographic functionality

5.4.1 Entropy request

5.4.1.1 General

Accessing a trusted source of entropy and random number generation is fundamental in cryptographic environment. This interface delivers access to the MTD considered as a trusted and quality random source provider.

EXAMPLE: In cloud or NFV context, it enables the virtual machines to start with initialized RNG, avoiding collisions, or it is used to establish TLS tunnels.

5.4.1.2 TD_GetRandom

TD_GetRandom allows the LTD to request a random number to the MTD.

TD_GetRandom (Session-Id, SizeInBytes)

Returns an Object-Id and a status code.

Table 17: TD_GetRandom command

Command parameter	Status	Description and Requirements
Session-Id	M	See clause 6.3
SizeInBytes	M	See clause 6.3

Table 18: TD_GetRandom response

Command parameter	Status	Description and Requirements
Object-Id	C	The MTD shall return an Object-Id when the command is successful. The random content is a byte string stored in the MTD and addressable by the returned Object-Id.
Status code	M	It shall be one of these values: • TDSC_SUCCESS • TDSC_UNKNOWN_SESSION_ID • TDSC_TRUST_EXPIRED • TDSC_GENERAL_FAILURE

5.4.2 Encryption keys request

5.4.2.1 General

A MTD provides long term key management and storage.

EXAMPLE: Some providers encrypt their VNF, and keys are dynamically delivered by a server in the MTD. Thus, a VNF can be copied but not run without accessing the encryption key.

5.4.2.2 TD_GenerateEncryptionKey

TD_GenerateEncryptionKey requests the MTD to generate a key which will be used by the LTD.

TD_GenerateEncryptionKey (Session-Id, Key_Type)

Returns an Object-Id and a status code.

Table 19: TD_GenerateEncryptionKey command

Command parameter	Status	Description and Requirements
Session-Id	M	See clause 6.3
Key_Type	M	See clause 6.3

Table 20: TD_GenerateEncryptionKey response

Command parameter	Status	Description and Requirements
Object-Id	C	The MTD shall return an Object-Id when the command is successful.
Status code	M	It shall be one of these values: • TDSC_SUCCESS • TDSC_UNKNOWN_SESSION_ID • TDSC_UNKNOWN_KEY_TYPE • TDSC_KEY_SIZE_NOT_SUPPORTED • TDSC_TRUST_EXPIRED • TDSC_GENERAL_FAILURE

5.4.3 Trusted timestamping

5.4.3.1 General

Trusted time stamping enables all amenities to share the same trusted source of time.

5.4.3.2 TD_GetTrustedTimestamping

TD_GetTrustedTimestamping requests the MTD to timestamp some data.

```
TD_GetTrustedTimestamping (Session-Id, DATA)
```

Returns an Object-Id and a status code.

Table 21: TD_GetTrustedTimestamping command

Command parameter	Status	Description and Requirements
Session-Id	M	See clause 6.3
DATA	M	See clause 6.3

Table 22: TD_GetTrustedTimestamping response

Command parameter	Status	Description and Requirements
Object-Id	C	The MTD shall return the Object-Id associated with the trusted timestamping when the command is successful.
Status code	M	It shall be one of these values: • TDSC_SUCCESS • TDSC_UNKNOWN_SESSION_ID • TDSC_TRUST_EXPIRED • TDSC_GENERAL_FAILURE

5.4.4 Secure archive

5.4.4.1 General

This interface enables data to be archived in the MTD. This is a write only service. Predefined Container-Id in the MTD can be used as global containers across LTD. The MTD Container shall be identified by its Resource-Id.

EXAMPLE: In Cloud or NFV context, log files can be stored by default in the MTD.

5.4.4.2 TD_CreateArchive

TD_CreateArchive creates a container for archival purpose.

TD_CreateArchive shall only create a PERMANENT container. This container shall be preserved beyond the end of the session (TD_CloseSession). TD_CreateArchive is a write only function. The LTD shall not be able to destroy this archive. The LTD delegates the responsibility and the life cycle of the container to the MTD.

```
TD_CreateArchive (Session-Id, Container-Type)
```

Returns a Container-Id and a status code.

Table 23: TD_CreateArchive command

Command parameter	Status	Description and Requirements
Session-Id	M	See clause 6.3
Container-Type	M	See clause 6.3 Container-Type shall be PERMANENT_FILE or PERMANENT_DATABASE

Table 24: TD_CreateArchive response

Command parameter	Status	Description and Requirements
Container-Id	C	The MTD shall return a Container-Id when the command is successful.
Status code	M	It shall be one of these values: • TDSC_SUCCESS • TDSC_UNKNOWN_SESSION_ID • TDSC_STORAGE_FULL • TDSC_CONTAINER_TYPE_NOT_SUPPORTED • TDSC_TRUST_EXPIRED • TDSC_GENERAL_FAILURE

5.4.4.3 TD_Archive

TD_Archive writes data to the Container-Id. TD_Archive is a write-only function. TD_CreateArchive shall be executed prior to call this function.

```
TD_Archive (Session-Id, Container-Id, Data)
```

Returns a status code. The MTD shall append the data to the content of the container.

Table 25: TD_Archive command

Command parameter	Status	Description and Requirements
Session-Id	M	See clause 6.3
Container-Id	M	See clause 6.3
DATA	M	See clause 6.3

Table 26: TD_Archive response

Command parameter	Status	Description and Requirements
Status code	M	It shall be one of these values: • TDSC_SUCCESS • TDSC_UNKNOWN_SESSION_ID • TDSC_DATA_TYPE_NOT_SUPPORTED • TDSC_UNKNOWN_CONTAINER_ID • TDSC_STORAGE_FULL • TDSC_TRUST_EXPIRED • TDSC_GENERAL_FAILURE

5.4.4.4 TD_CloseArchive

TD_CloseArchive closes the archive, no further write access is allowed.

```
TD_CloseArchive (Session-Id, Container-Id)
```

Returns a status code.

Table 27: TD_CloseArchive command

Command parameter	Status	Description and Requirements
Session-Id	M	See clause 6.3
Container-Id	M	See clause 6.3

Table 28: TD_CloseArchive response

Command parameter	Status	Description and Requirements
Status code	M	It shall be one of these values: • TDSC_SUCCESS • TDSC_UNKNOWN_SESSION_ID • TDSC_UNKNOWN_CONTAINER_ID • TDSC_STORAGE_BUSY • TDSC_TRUST_EXPIRED • TDSC_GENERAL_FAILURE

5.4.5 Secure storage

5.4.5.1 General

This interface enables data to be stored in the MTD and retrieve from the MTD. This is a read/write service. The secure storage in the MTD Container shall be identified by its Container-Id and permanent containers shall enable concurrent access by all the LTD entities.

5.4.5.2 TD_CreateStorage

TD_CreateStorage creates a new storage container in the MTD.

```
TD_CreateStorage (Session-Id, Container-Name, Container-Type)
```

Returns a Container-Id and a status code.

Table 29: TD_CreateStorage command

Command parameter	Status	Description and Requirements
Session-Id	M	See clause 6.3
Container-Name	M	See clause 6.3
Container-Type	M	See clause 6.3

Table 30: TD_CreateStorage response

Command parameter	Status	Description and Requirements
Container-Id	C	The MTD shall return a Container-Id when the command is successful
Status code	M	It shall be one of these values: • TDSC_SUCCESS • TDSC_UNKNOWN_SESSION_ID • TDSC_CONTAINER_TYPE_NOT_SUPPORTED • TDSC_STORAGE_FULL • TDSC_TRUST_EXPIRED • TDSC_CONTAINER_NAME_ALREADY_EXISTS • TDSC_GENERAL_FAILURE

5.4.5.3 TD_DeleteStorage

The function TD_DeleteStorage enables the LTD to request the deletion of a container created by the function CreateStorage and referenced by its Container-Id. The MTD should delete container, the Container-Id and the Object-Id of the data stored in that container.

```
TD_DeleteStorage (Session-Id, Container-Id)
```

Returns a status code.

Table 31: TD_DeleteStorage command

Command parameter	Status	Description and Requirements
Session-Id	M	See clause 6.3
Container-Id	M	See clause 6.3

Table 32: TD_DeleteStorage response

Command parameter	Status	Description and Requirements
Status code	M	It shall be one of these values: • TDSC_SUCCESS • TDSC_UNKNOWN_SESSION_ID • TDSC_UNKNOWN_CONTAINER_ID • TDSC_STORAGE_BUSY • TDSC_TRUST_EXPIRED • TDSC_GENERAL_FAILURE

5.4.5.4 TD_StoreData

The function TD_StoreData enables to store data in a container. The data shall remain accessible thanks to the returned Object-Id. This Object-Id is not deleted at the end of the session but will be deleted along with the function DeleteStorage. The container is viewed as a collection of objects addressable by their respective Object-Ids.

```
TD_StoreData (Session-Id, Container-Id, DATA|, DB_KeyValue)
```

Returns an Object-Id and a status code.

Table 33: TD_StoreData command

Command parameter	Status	Description and Requirements
Session-Id	M	See clause 6.3
Container-Id	M	See clause 6.3
DATA	C	It shall be present if the related Container-Type is FILE or PERMANENT_FILE
DB_KeyValue	C	It shall be present if the related Container-Type is DATABASE or PERMANENT_DATABASE

Table 34: TD_StoreData response

Command parameter	Status	Description and Requirements
Object-Id	C	The MTD shall return an Object-Id when the command is successful
Status code	M	It shall be one of these values: • TDSC_SUCCESS • TDSC_UNKNOWN_SESSION_ID • TDSC_UNKNOWN_CONTAINER_ID • TDSC_DATA_TYPE_NOT_SUPPORTED • TDSC_STORAGE_FULL • TDSC_TRUST_EXPIRED • TDSC_GENERAL_FAILURE

5.4.5.5 TD_GetStorageValue

TD_GetStorageValue allows returning an Object-Id value fetch from the Container-Id. This value can either be a DATA or a DB_KeyValue

```
TD_GetStorageValue (Session-Id, Container-Id, Object-Id)
```

Returns a value and a status code.

Table 35: TD_GetStorageValue command

Command parameter	Status	Description and Requirements
Session-Id	M	See clause 6.3
Container-Id	M	See clause 6.3 Container shall be of type DATABASE
Object-Id	M	See clause 6.3

Table 36: TD_GetStorageValue response

Command parameter	Status	Description and Requirements
DB_KeyValue	C	The MTD shall return a DB_KeyValue when the command is successful
DATA	C	The MTD shall return a DATA when the command is successful
Status code	M	It shall be one of these values: • TDSC_SUCCESS • TDSC_UNKNOWN_SESSION_ID • TDSC_UNKNOWN_CONTAINER_ID • TDSC_CONTAINER_TYPE_NOT_SUPPORTED • TDSC_UNKNOWN_KEY • TDSC_CONTAINER_WRITE_ONLY • TDSC_TRUST_EXPIRED • TDSC_GENERAL_FAILURE

5.6 Search capabilities

5.6.1 Container search

5.6.1.1 General

This interface enables searching in a container, hosted in the MTD.

5.6.1.2 TD_GetStorage

TD_GetStorage requests the MTD to find the Container-Id of a named container.

```
TD_GetStorage (Session-Id, Container-Name)
```

Returns a Container-Id and a status code.

Table 37: TD_GetStorage command

Command parameter	Status	Description and Requirements
Session-Id	M	See clause 6.3
Container-Name	M	See clause 6.3

Table 38: TD_GetStorage response

Command parameter	Status	Description and Requirements
Container-Id	C	The MTD shall return a Container-ID when the command is successful
Status code	M	It shall be one of these values: • TDSC_SUCCESS • TDSC_UNKNOWN_SESSION_ID • TDSC_CONTAINER_WRITE_ONLY • TDSC_CONTAINER_NAME_NOT_FOUND • TDSC_TRUST_EXPIRED • TDSC_GENERAL_FAILURE

5.6.1.3 TD_Search

TD_Search searches for a value or an event in a container referenced by its Container-Id. An event is a pair (subject, context) where the search is performed on the Subject enriched with added Context information.

```
TD_Search (Session-Id, Container-Id, DB_KeyValue|Event)
```

Returns an Object-Id and a status code.

Table 39: TD_Search command

Command parameter	Status	Description and Requirements
Session-Id	M	See clause 6.3
Container-Name	M	See clause 6.3
DB_KeyValue	C	DB_KeyValue or Event shall be present
Event	C	See clause 6.3 DB_KeyValue or Event shall be present

Table 40: TD_Search response

Command parameter	Status	Description and Requirements
Object-Id	C	The MTD shall return an Object-Id when the command is successful
Status code	M	It shall be one of these values: • TDSC_SUCCESS • TDSC_UNKNOWN_SESSION_ID • TDSC_UNKNOWN_CONTAINER_ID • TDSC_VALUE_NOT_FOUND • TDSC_CONTAINER_WRITE_ONLY • TDSC_TRUST_EXPIRED • TDSC_GENERAL_FAILURE

6 Encoding

6.1 Message identifiers

These codes shall be used to identify the messages in the protocol.

Table 41

Message	Type	Code	Description
TD_GetObjectValue	Command	100	Requests an object content by its Object-Id
	Response	101	
TD_OpenConnection	Command	2	Requests a connection with the MTD
	Response	3	
TD_PutObjectValue	Command	4	Requests the storage of an object value
	Response	5	
TD_CreateSession	Command	6	Requests the initialization of a session
	Response	7	
TD_CloseSession	Command	8	Requests the termination of a session
	Response	9	
TD_GetRandom	Command	10	Requests the generation of random bytes
	Response	11	
TD_GenerateEncryptionKey	Command	12	Requests the generation of key material
	Response	13	
TD_CreateArchive	Command	14	Requests the creation of a write-only container
	Response	15	
TD_Archive	Command	16	Requests the addition of data to archive
	Response	17	
TD_CloseArchive	Command	18	Requests the closing of archive
	Response	19	
TD_CreateStorage	Command	20	Requests the creation of a container
	Response	21	
TD_DeleteStorage	Command	22	Requests the deletion of a container
	Response	23	
TD_StoreData	Command	24	Requests the addition of data to the container
	Response	25	
TD_GetValue	Command	26	Requests a stored object
	Response	27	
TD_GetStorageValue	Command	28	Requests a stored object from database container
	Response	29	
TD_GetStorage	Command	30	Requests access to a container
	Response	31	
TD_Search	Command	32	Requests a search for a value or an event in a container
	Response	33	
TD_GetTrustedTimestamping	Command	34	Requests the timestamping of some data
	Response	35	
TD_TrustRenewal	Command	36	Requests renewing of trust with some attestation
	Response	37	

6.2 Type (TTLV) codes

Table 42

Type	Code	Length
Symbol	1	1 byte
ByteString	2	Variable
Unicode String	3	Variable
Integer	4	8 bytes
Short Integer	5	2 bytes
Pair	6	Variable

6.3 Tag (TTLV) codes

Table 43

TAG	Type	TAG Code	Description and requirements
Container-Type	Symbol	1	Type of the container. CONTAINER-TYPE shall be one of these values: • FILE • DATABASE • PERMANENT_FILE • PERMANENT_DATABASE
LTD-Id	Unicode String	2	LTD identifier generated by the MTD
LTD-Role	Unicode String	4	Information about the LTD entity in the untrusted domain to the MTD enabling the MTD to behave accordingly at application level.
CN	Unicode String	5	Certificate common Name.
Object-Id	Integer	6	Object identifier unique within a session. The Object-Id between 0 and 65536 shall be reserved for predefined constant objects in the MTD. Object-Id 65536 corresponds to read-only predefined data. All Object-Id within a session refer to a corresponding Value stored in the MTD.
Session-Id	ByteString	7	Session identified. Fixed length 16 bytes It shall be a cryptographic strong random value generated in the MTD.

TAG	Type	TAG Code	Description and requirements
Container-Id	Integer	8	Unique identifier for a database or a file in the MTD. The Container-Id between 0 and 65536 shall be reserved for predefined constant container in the MTD.
Container-Name	Unicode String	9	Identifier of a secure storage container. It allows the retrieval of a secure storage container independently from the session.
Signed-Data	ByteString	10	Attestation of state integrity of the LTD entity computed as Data = SIGN-RSA (hash) with hash being the cryptographic hash of a reference measurements appended by the Nonce value to attest.
DB_KeyValue	Pair	11	Pair (DB_Key, DB_Value) structured type.
DB_Key	ByteString	12	
DB_Value	ByteString	13	
Status Code	Short Integer	14	
PERMANENT_FILE	Symbol	15	
PERMANENT_DATABASE	Symbol	16	
FILE	Symbol	17	
DATABASE	Symbol	18	
Key_Type	Symbol	29	Type of cryptographic key requested by the LTD. The following symbolic constant shall be supported: • SYMMETRIC_KEY_128 • SYMMETRIC_KEY_256 • RSA_KEY_1024 • RSA_KEY_2048 • RSA_KEY_4096
RSA_KEY_1024	Symbol	30	RSA key pair (public and private key) to be used for RSA encryption and signature. It should be 1024-bit length.
RSA_KEY_2048	Symbol	31	RSA key pair (public and private key) to be used for RSA encryption and signature. It should be 2048-bit length.
RSA_KEY_4096	Symbol	32	RSA key pair (public and private key) to be used for RSA encryption and signature. It should be 4096-bit length.
SYMMETRIC_KEY_128	Symbol	33	Key to be used for symmetric encryption. The key should be randomly generated using a standard cryptographic PNRG. Key size shall be 128-bit long. AES should be used.
SYMMETRIC_KEY_256	Symbol	34	Key to be used for symmetric encryption. The key should be randomly generated using a standard cryptographic PNRG. Key size shall be 256-bit long. AES should be used.
Event	Pair	22	Pair (subject,context) structured type.
Subject	ByteString	23	
Context	ByteString	24	
SizeInBytes	Integer	25	
DATA	ByteString	26	
Nonce	ByteString	27	Used to convey MTD generated nonce values.

6.4 Status Codes

Status Codes shall reflect the success of the function execution or the reason why the function failed. In order to do so, the status code is defined as a constant and specified in table 44.

Table 44

Constant	Value	Reason
TDSC_SUCCESS	0	Function succeeded For TD_ functions, returns a non-nil
TDSC_GENERAL_FAILURE	1	Generic status code.
TDSC_SESSION_ID_ALREADY_OPENED	2	Session has already been created for the current Connection
TDSC_TOO_MANY_EXISTING_SESSIONS	3	The maximum concurrent sessions supported by the MTD is reached
TDSC_ON_GOING_PROCESSES	4	Processes are still running on the MTD
TDSC_TOO_MANY_OPENED_CONNECTIONS	5	The connection is refused by the MTD
TDSC_TRUST_REFUSED	6	The connection is refused by the MTD because the trust of the LTD cannot be established
TDSC_TRUST_EXPIRED	10	Trust needs to be renewed between LTD and MTD
TDSC_UNKNOWN_ROLE	11	The LTD role is not known by the MTD
TDSC_UNKNOWN_SESSION_ID	100	Session-Id is not known by the MTD
TDSC_UNKNOWN_OBJECT_ID	101	Object-Id is not known by the MTD
TDSC_OBJECT_CREATION_FAILED	110	Unable to create new object
TDSC_UNKNOWN_CONTAINER_ID	202	Container-Id is not known by the MTD
TDSC_CONTAINER_TYPE_NOT_SUPPORTED	203	Container-Type is not supported
TDSC_CONTAINER_WRITE_ONLY	204	Container-Id references an Archive Container
TDSC_CONTAINER_NAME_ALREADY_EXISTS	205	Container-Name already in use
TDSC_CONTAINER_NAME_NOT_FOUND	206	No container named Container-Name found
TDSC_DATA_TYPE_NOT_SUPPORTED	207	Data provided by the LTD mismatch MTD Container's data type
TDSC_STORAGE_FULL	208	MTD allocated storage is full
TDSC_STORAGE_BUSY	209	Storage/Archive process is still busy on the MTD
TDSC_UNKNOWN_KEY	300	Key is unknown in the MTD Container
TDSC_UNKNOWN_KEY_ID	301	Key-Id is unknown in the MTD
TDSC_UNKNOWN_KEY_TYPE	302	Requested key type is not supported by the MTD
TDSC_KEY_SIZE_NOT_SUPPORTED	303	Requested key size is not supported by the MTD
TDSC_VALUE_NOT_FOUND	400	Searched value not found in the MTD
TDSC_NOT_ENOUGH_ENTROPY	500	No enough entropy to fulfill an entropy request
TDSC_ATTESTATION_FAILED	600	Remote attestation failed

Annex A (informative): Use Cases

A.1 Entropy request scenario

A.1.1 Description

Entropy quality is the key element to generating quality cipher keys. As cloud is extensively used, it is vital that machines generating those keys have a good source of entropy, and cannot rely on their own entropy source, as they can be virtual machines that would have the very same state when started.

Therefore, the entropy is delivered by a trusted source of entropy that belongs to the MTD Infrastructure.

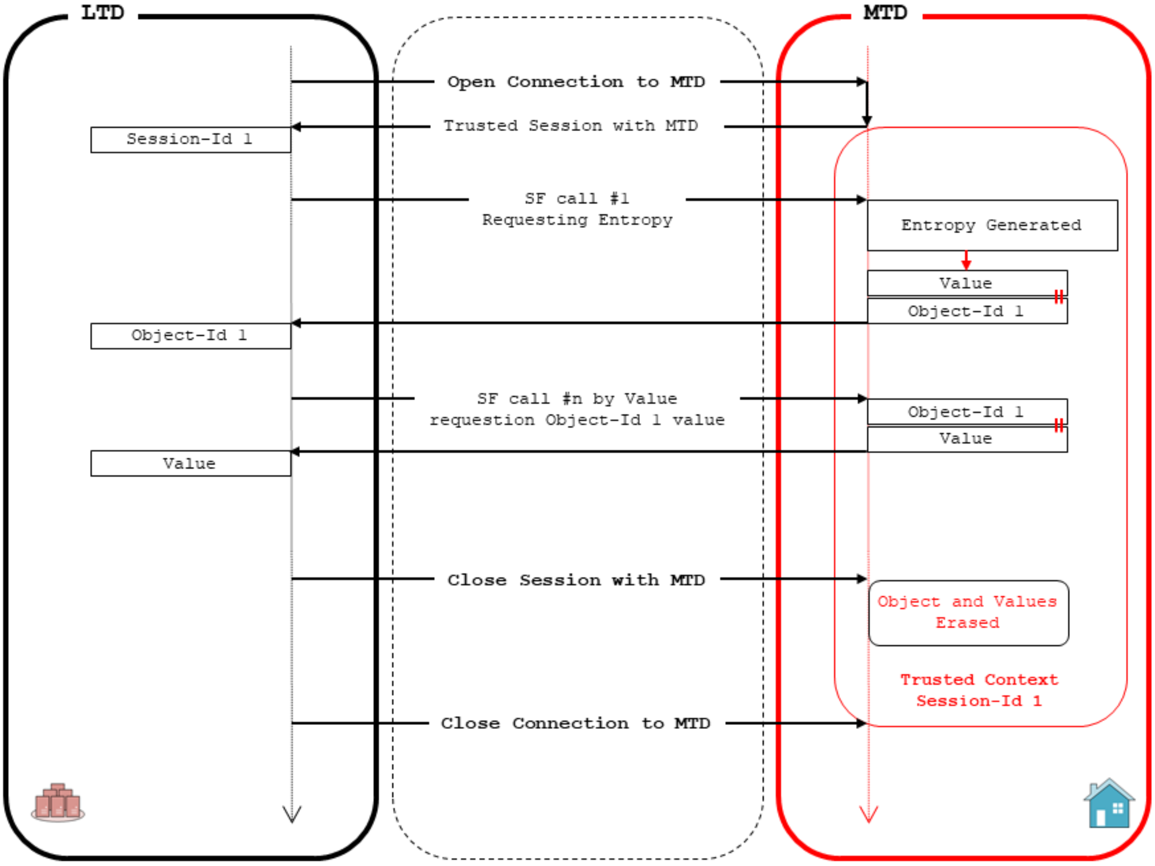

Figure A.1: Entropy Request scenario

A.1.2 Example

In that example, the LTD entity needs 64 bits of random and requested the MTD through the interface. It can be used to seed its own PRNG with some quality assurance.

The LTD entity generates a nonce and a TPM signed attestation.

TD_OpenConnection → Command `6501234, "LTD-VM-FW", CN, nonce, signed_data`
TD_OpenConnection ← Response `8541, TDSC_SUCCESS`

Successful connection with the configuration Container-id 8560 returned.

TD_CreateSession	→ Command	
TD_CreateSession	← Response	`6501234, TDSC_SUCCESS`
TD_GetRandom	→ Command	`6501234, 8`
TD_GetRandom	← Response	`22500, TDSC_SUCCESS`
TD_GetObjectValue	→ Command	`6501234, 22500`
TD_GetObjectValue	← Response	`0x3B45E45FF570ACCB, TDSC_SUCCESS`
TD_CloseSession	→ Command	`6541543`
TD_CloseSession	← Response	`TDSC_SUCCESS`
TD_CloseConnection	→ Command	
TD_CloseConnection	← Response	`TDSC_SUCCESS`

A.2 Encrypted Virtual Machine use case (including LTD execution environment check)

A.2.1 Description

Virtual machines can be ciphered to protect against malicious infrastructure operators. An MTD agent operating at VMM level could request the VM key in order to load and run the VM.

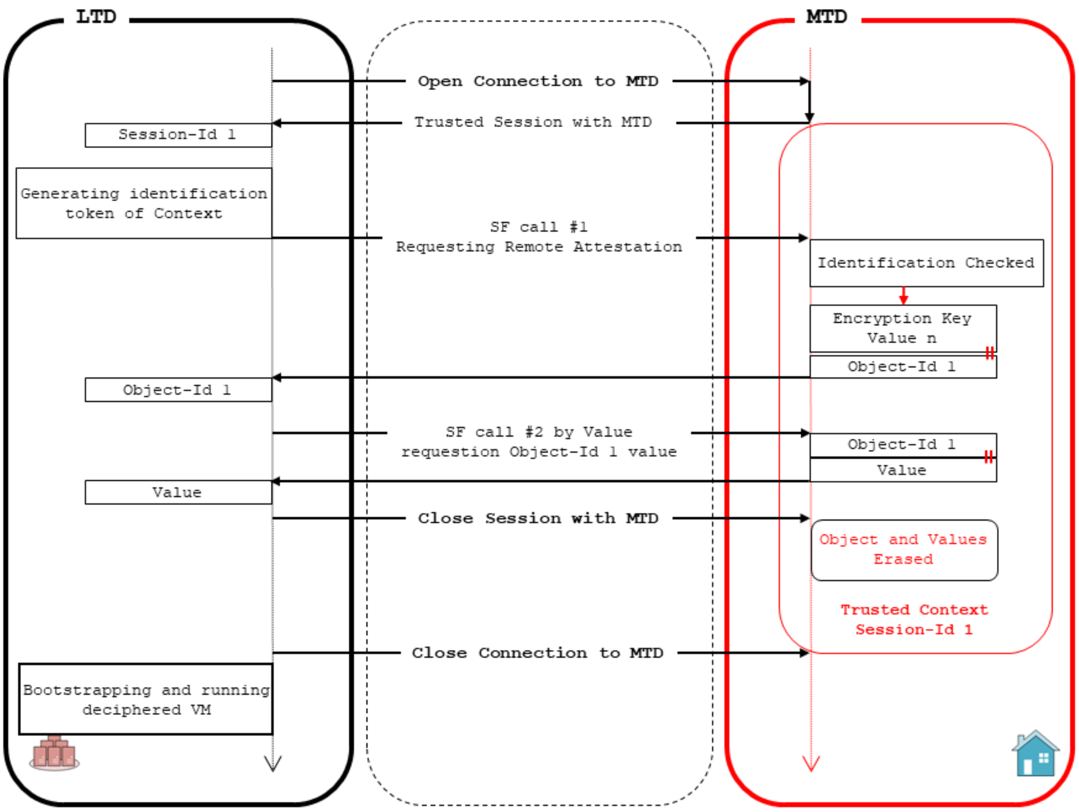

Figure A.2: Virtual Machine use case

A.2.2 Example

A.2.2.1 Introduction

In that example, the LTD with Id 6501234 and role "LTD-VM-BOOT" uses the interface to obtain the key to boot an encrypted VM. The MTD will check if hardware is trusted enough before giving the boot key.

The prerequisite for this use case is that:

- The MTD knows a list of TPM keys indexed to LTD-Role of the hardware used in the LTD;
- The LTD has access to the TPM-Id and it knows the following predefined Container-Id 8560 which is associated with the boot key for LTD-Role database.

A.2.2.2 Successful case

The LTD entity generates a nonce and a TPM signed attestation.

TD_OpenConnection	→ Command	6501234, "LTD-VM-BOOT", CN, nonce, signed_data
TD_OpenConnection	← Response	8560, TDSC_SUCCESS

Successful connection with the configuration Container-id 8560 returned.

TD_CreateSession	→ Command	
TD_CreateSession	← Response	6541543, TDSC_SUCCESS

A new session is opened with Session-Id 6541543.

TD_GetStorageValue	→ Command	6541543, 8560, "BOOT-KEY"
TD_GetStorageValue	← Response	0xABC456820FFFBCE3D5D3257, TDSC_SUCCESS
TD_CloseSession	→ Command	6541543
TD_CloseSession	← Response	TDSC_SUCCESS
TD_CloseConnection	→ Command	
TD_CloseConnection	← Response	TDSC_SUCCESS

Then the hypervisor deciphers and boots the VM.

A.2.2.3 Failure case

TD_OpenConnection	→ Command	6501234, "LTD-VM-BOOT", CN, nonce, signed_data
TD_OpenConnection	← Response	8560, TDSC_TRUST_REFUSED

A.3 Secure archive use case

A.3.1 Description

Archiving sensitive data can be addressed by delegating the Archive function to the MTD. The LTD requests an Archiving token - embedded in an Object-Id, and when data are ready to be archived, push them to the MTD with the Object-Id. Archive are not meant to be retrieved by the LTD, therefore the Object-Id is volatile.

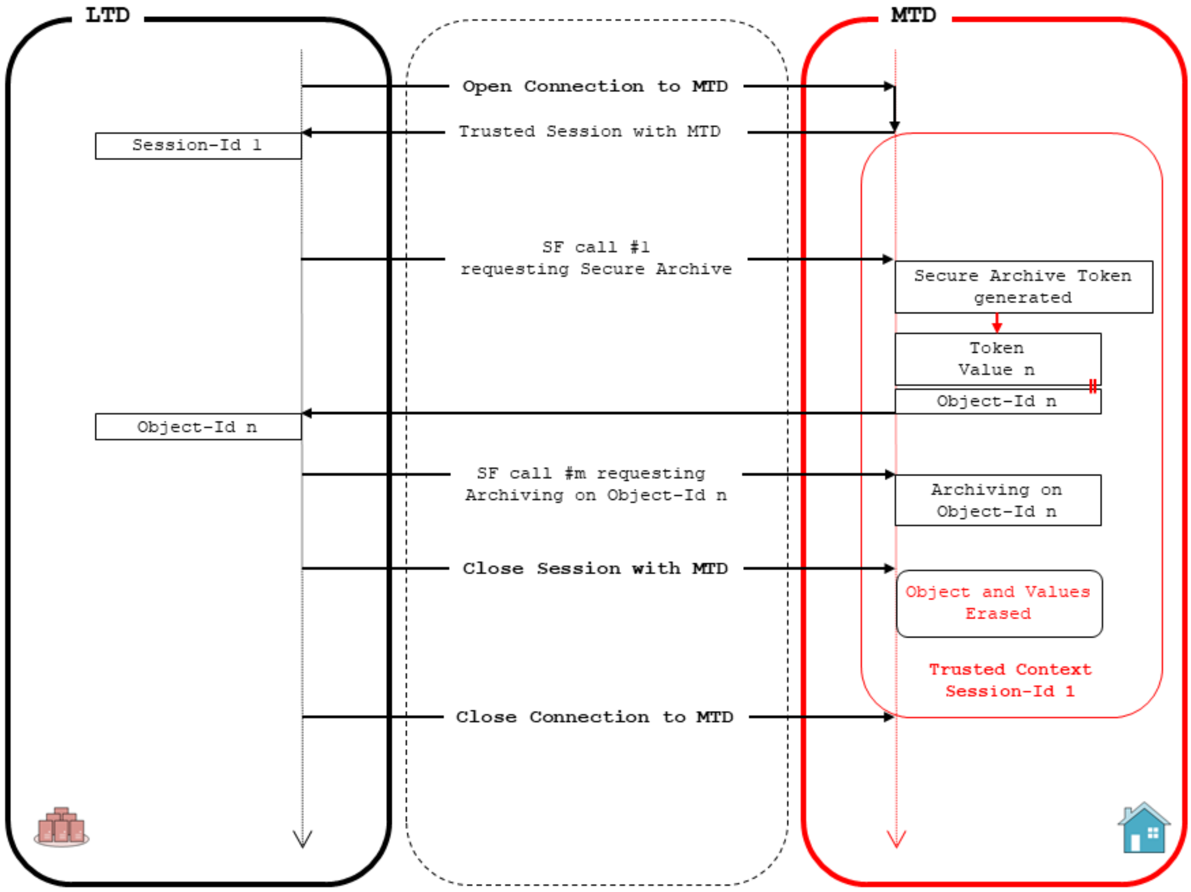

Figure A.3: Secure Archive use case

A.3.2 Example

In that example, the LTD with Id 6501234 and named "LTD-VM-FW" uses the interface to archive in a MTD log files coming from the firewall of the LTD and the system log in two different archive files named "FW-Log" and "Sys-Log". The configuration reference 8541 gives access to the container-id relative to the LTD log archives.

A nonce is generated by the LTD entity.

TD_OpenConnection	→ Command	`6501234, "LTD-VM-FW", CN, nonce, signed_data`
TD_OpenConnection	← Response	8541, TDSC_SUCCESS
TD_CreateSession	→ Command	
TD_CreateSession	← Response	`6554341, TDSC_SUCCESS`

A session with id `6554341` *is accepted by the MTD.*

TD_GetStorageValue	→ Command	`6541541, 8541, "FW-Log"`
TD_GetStorageValue	← Response	`8004341, TDSC_SUCCESS`
TD_Archive	→ Command	`6554341, 8004341, Log_data_1`
TD_Archive	← Response	`TDSC_SUCCESS`
TD_GetStorageValue	→ Command	`6541541, 8541, "Sys-Log"`
TD_GetStorageValue	← Response	`8004355, TDSC_SUCCESS`
TD_Archive	→ Command	`6554341, 8004341, Log_data_2`
TD_Archive	← Response	`TDSC_SUCCESS`

TD_Archive	→ Command	`6554341, 8004355, Log_data_3`
TD_Archive	← Response	`TDSC_SUCCESS`

The lifetime of the connection is going to expire. LTD entity needs to renew the trust.

TD_TrustRenewal	→ Command	`6554341, CN, nonce, signed_data`
TD_TrustRenewal	← Response	`TDSC_SUCCESS`
TD_Archive	→ Command	`6554341, 8004355, Log_data_4`
TD_Archive	← Response	`TDSC_SUCCESS`
TD_CloseSession	→ Command	`6554341`
TD_CloseSession	← Response	`TDSC_SUCCESS`
TD_CloseConnection	→ Command	
TD_CloseConnection	← Response	`TDSC_SUCCESS`

A.4 Secure query use case

A.4.1 Description

This use case is a progression from the previous use case where a LTD can query the MTD to check for the existence of a value within the MTD.

The LTD may or may not have permission to store values in the MTD (i.e. the values that the LTD can query may have been externally provisioned).

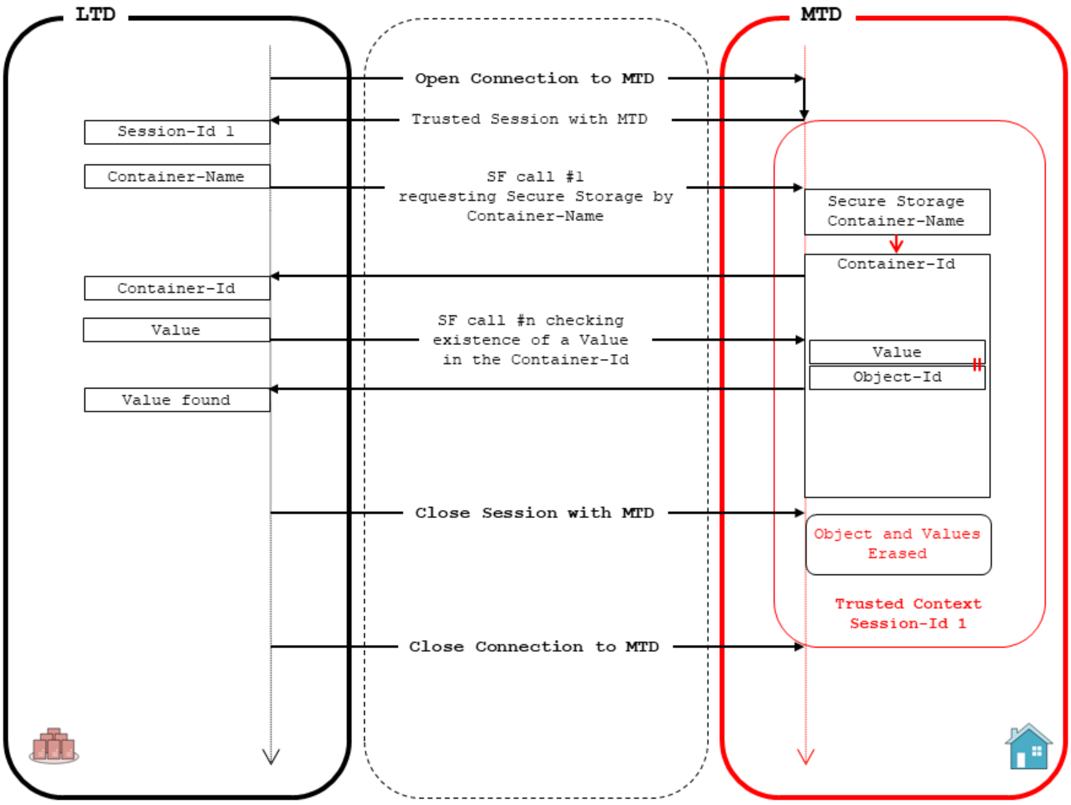

Figure A.4: Secure Query use case

A.4.2 Example

The LTD entity retrieves a container named "LTD-Q" that may have been created by another entity and check the presence of some stored value.

A nonce is generated by the LTD entity.

TD_OpenConnection	→ Command	6501234, "LTD-VM-FW", CN, nonce, signed_data
TD_OpenConnection	← Response	8542, TDSC_SUCCESS
TD_CreateSession	→ Command	
TD_CreateSession	← Response	6554341, TDSC_SUCCESS
TD_GetStorage	→ Command	6554341, "LTD-Q"
TD_GetStorage	← Response	65500, TDSC_SUCCESS

The reference container id 65500 is returned

TD_Search	→ Command	6554341, 65500, value
TD_Search	← Response	TDSC_SUCCESS
TD_CloseSession	→ Command	6554341
TD_CloseSession	← Response	TDSC_SUCCESS
TD_CloseConnection	→ Command	
TD_CloseConnection	← Response	TDSC_SUCCESS

A.5 Secure Storage use case

A.5.1 Description

This scenario is close to the previous one, except that data stored in the MTD can be retrieved, therefore the token is non-volatile.

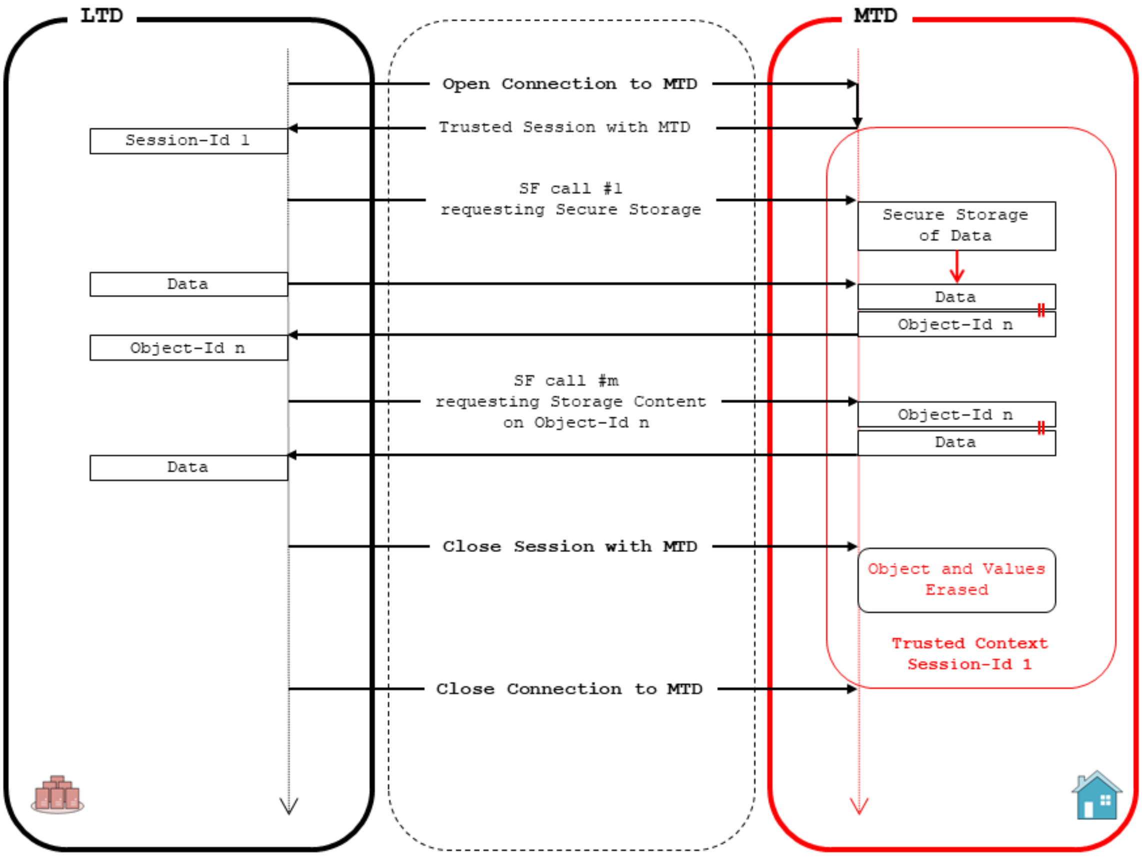

Figure A.5: Secure Storage use case

A.5.2 Example

In that example, a LTD named LTD-VM-FW with Id 6501234 stores a configuration file called "FW data" which could be used by other instances in the LTD with Id 6501235 in the example.

A nonce is generated by the LTD entity.

TD_OpenConnection	→ Command	6501234, "LTD-VM-FW", CN, nonce, signed_data
TD_OpenConnection	← Response	8544, TDSC_SUCCESS
TD_CreateSession	→ Command	
TD_CreateSession	← Response	6554342, TDSC_SUCCESS
TD_CreateStorage	→ Command	6554342, "FW data", PERMANENT_FILE
TD_CreateStorage	← Response	100100, TDSC_SUCCESS
TD_StoreData	→ Command	6554342, 100100, "data"
TD_StoreData	← Response	300452, TDSC_SUCCESS
TD_CloseSession	→ Command	6554342
TD_CloseSession	← Response	TDSC_OK
TD_CloseConnection	→ Command	
TD_CloseConnection	← Response	TDSC_SUCCESS

Another entity with id 6501235 connects.

TD_OpenConnection	→ Command	6501235, "LTD-VM-FW", CN, nonce, signed_data
TD_OpenConnection	← Response	8544, TDSC_SUCCESS

```
TD_CreateSession      → Command
TD_CreateSession      ← Response    6554353, TDSC_SUCCESS

TD_GetStorage         → Command     6541553, "FW data"
TD_GetStorage         ← Response    300452, TDSC_SUCCESS
```

A reference to the named container is returned.

```
TD_GetObjectValue     → Command     6541553, 300452
TD_GetObjectValue     ← Response    "data", TDSC_SUCCESS
```

The stored data is retrieved.

```
TD_CloseSession       → Command     6554353
TD_CloseSession       ← Response    TDSC_SUCCESS

TD_CloseConnection    → Command
TD_CloseConnection    ← Response    TDSC_SUCCESS
```

A.6 Authentication use case

A.6.1 Description

Prerequisites: The predefined Container-Id 100 in the MTD has been associated with the database (login, salt) and Container-Id 101 for (hash, login). This a generic "salted" authentication scheme backed by the secure storage of hashed password by the MTD. Cryptographic computations keep done by the LTD at application level.

A.6.2 Example

In this example, the LTD entity tries to check a password given "pwd" for an "admin" login.

A nonce is generated by the LTD entity.

```
TD_OpenConnection     → Command     6501234, "LTD-VM-FW", CN, nonce, signed data
TD_OpenConnection     ← Response    8545, TDSC_SUCCESS

TD_CreateSession      → Command
TD_CreateSession      ← Response    6554442, TDSC_SUCCESS

TD_GetStorageValue    → Command     6554442, 100, "admin"
TD_GetStorageValue    ← Response    100456, TDSC_SUCCESS

TD_GetObjectValue     → Command     6554442, 100456
TD_GetObjectValue     ← Response    0x1234567890, TDSC_SUCCESS
```

Salt 0x1234567890 is returned for login "admin"

The LTD computes hash=SHA256(login || PBKDF2(salt,pwd)).

```
TD_Search             → Command     6554442, 101, hash
TD_Search             ← Response    100458, TDSC_SUCCESS
```

As hash value exists as a key in the MTD database, the authentication can be accepted.

A.7 Lawful intercept use case

A.7.1 Description

This is a demonstration of how an LTD can implement Lawful Intercept by extending use case A.4.

In this example the LI selector list exists within the MTD and is queried by the LTD so that the LTD understands whether or not traffic needs to be duplicated to a Mediation Function (MF).

Communications will typically be point-to-point, and therefore the LTD will send two queries, one for each end of the communication.

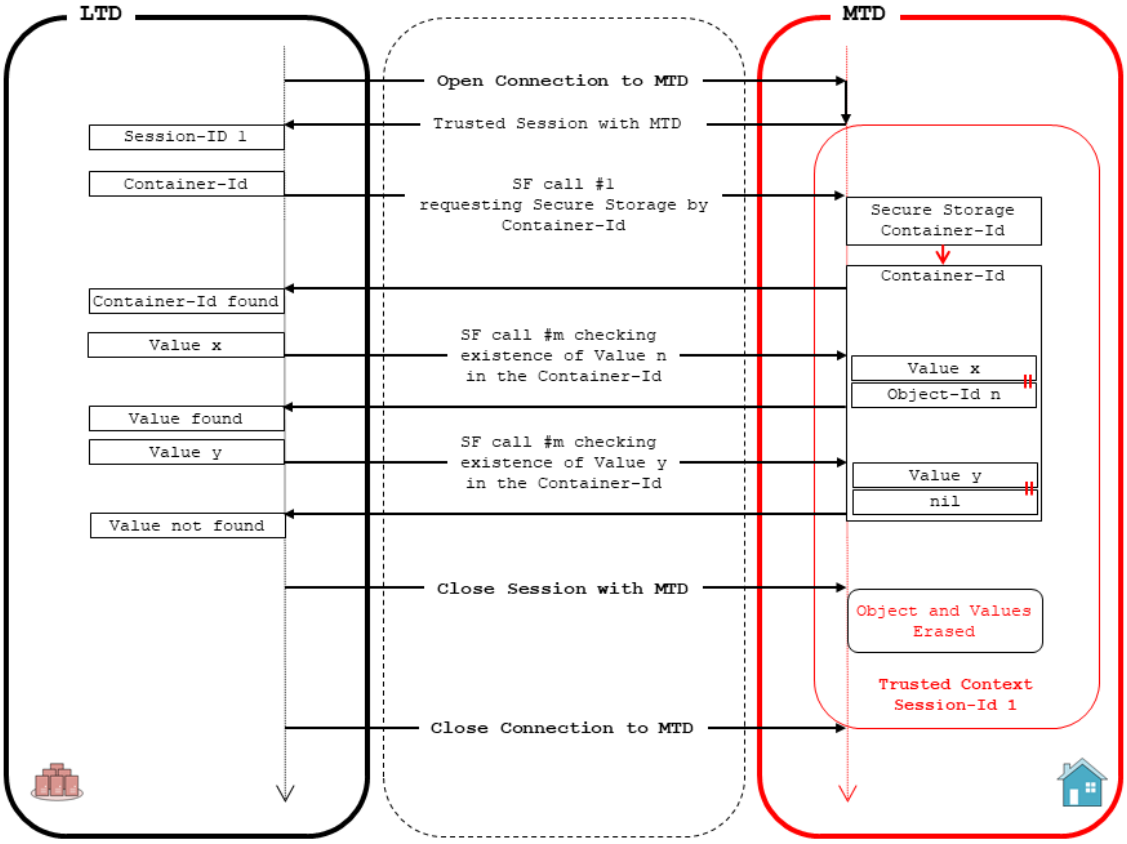

Figure A.6: Lawful Intercept use case

A.7.2 Example

The MTD stores LI selectors in a container named "LI_SELECTOR_LIST" that is requested by the LTD entity LI probe to determine if some event (phone number) needs triggering interception.

A nonce is generated by the LTD entity and an attestation is signed by the TPM.

TD_OpenConnection	→ Command	`6501234, "LTD-VM-LI", CN, nonce, signed data`
TD_OpenConnection	← Response	`8546, TDSC_SUCCESS`
TD_CreateSession	→ Command	
TD_CreateSession	← Response	`6554341, TDSC_SUCCESS`
TD_GetStorage	→ Command	`6554341, "LI_SELECTOR_LIST"`
TD_GetStorage	← Response	`65501, TDSC_SUCCESS`

Requesting the selector list on phone number 00441234567890.

TD_Search	→ Command	`6554341,65501, ("00441234567890", "MO Call")`
TD_Search	← Response	`TDSC_VALUE_NOT_FOUND`

No selection to be done.

TD_Search	→ Command	`6554341, 65501, ("00440987654321", "MO Call")`
TD_Search	← Response	`TDSC_SUCCESS`

Selection confirmed.

The LTD keeps requesting the MTP on each event.

TD_CloseSession	→ Command	`6554341`
TD_CloseSession	← Response	`TDSC_SUCCESS`

TD_CloseConnection	→ Command	
TD_CloseConnection	← Response	`TDSC_SUCCESS`

A.8 NFV sec use case

A.8.1 Description

This scenario is an advanced case of LI conforming to ETSI GR NFV-SEC 011 [i.2] in low trust scenario where the vPOI (virtual Point of Interception) and TCF (Triggering Control Function) are virtualized. The TCF is expected to keep its selector list secured and securely logs the triggering events it receives from its vPOI entities. The triggering event can have multiple forms as it can be generated by several kind of vPOI and relate to different telecom identifiers.

There are two different ways to use the interface in this NFV low trust scenario:

- If the TCF is secured enough, the TCF is considered as being in the MTD domain and then the vPOI are in the LTD domain; or

- If the TCF is not secured enough, the TCF is considered as being in the LTD domain. In that case, the selectors list could be stored in a more secure place within a MTD domain. Clause A.8.2 illustrates this case.

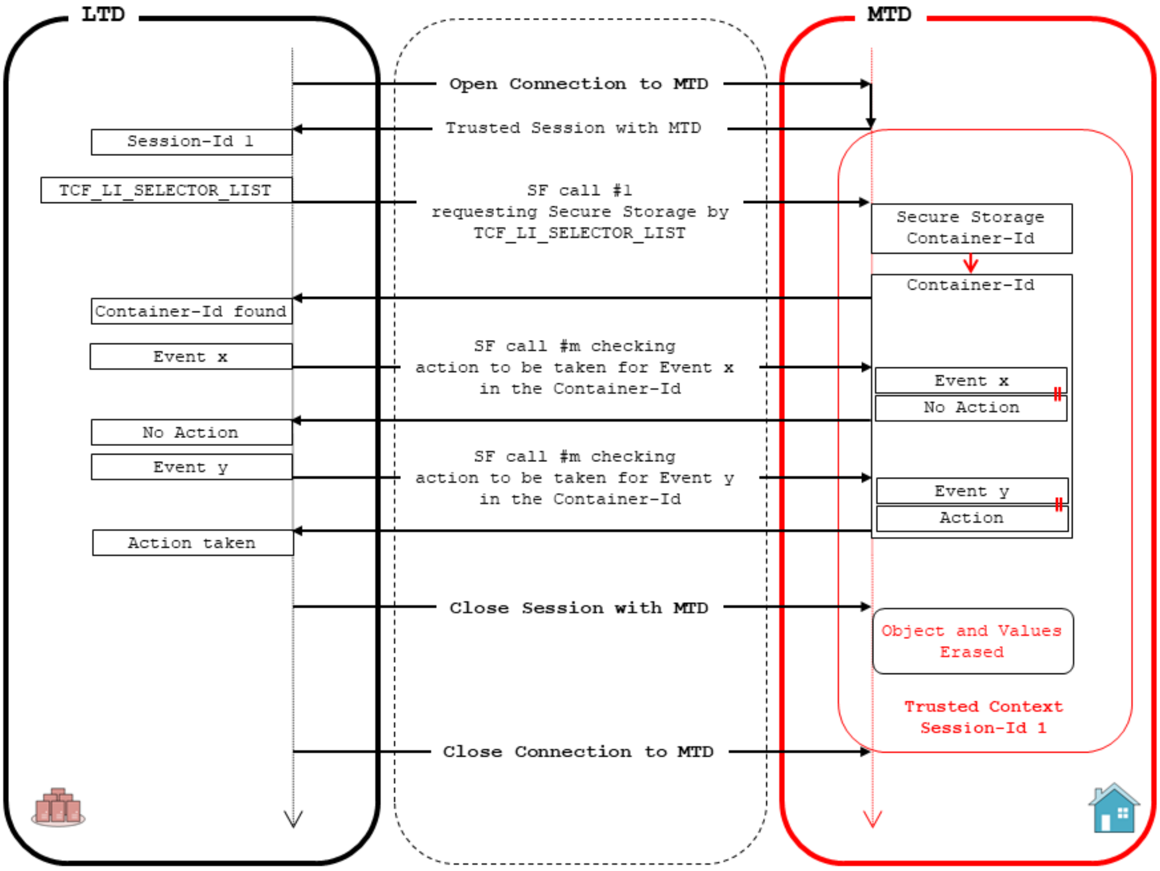

Figure A.7: NFV Sec use case

A.8.2 Example

TD_OpenConnection	→ Command	`6501234, "LTD-VM-TCF", CN, nonce, signed_data`
TD_OpenConnection	← Response	`8547, TDSC_SUCCESS`
TD_CreateSession	→ Command	
TD_CreateSession	← Response	`6554341, TDSC_SUCCESS`
TD_GetStorage	→ Command	`6554341, TCF_LI_SELECTOR_LIST`
TD_GetStorage	← Response	`65501, TDSC_SUCCESS`
TD_Search	→ Command	`6554341, 65501, Event1`
TD_Search	← Response	`TDSC_VALUE_NOT_FOUND`

Where *Event1=<Subject=00441234567890, Context=CALLER=00441234567890, SOURCE:vPOI1,CALLEE:00449876543210, KIND:voice-call>*

TD_Search	→ Command	`6554341, 65501, Event2`
TD_Search	← Response	`TDSC_SUCCESS`

Where Event2=<Subject=4100412345678,Context=4100412345678, IMSI=41004123456789,SOURCE:vPOI2>

TD_CloseSession	→ Command	`6554341`
TD_CloseSession	← Response	`TDSC_SUCCESS`
TD_CloseConnection	→ Command	
TD_CloseConnection	← Response	`TDSC_SUCCESS`

Annex B (informative): Guidelines

B.1 Implementation guidelines

B.1.1 Global architecture

TCDI is intended to be used in a centralized way with one MTD serving many LTD "agents".

A LTD entity might get support from multiple MTD through individual connection.

Sensitive functions (random generation, key storage, state integrity check, data signature) should be protected in hardware. Ideally with an HSM in MTD side and a TPM or other kind of secure element in the LTD side.

This interface should be secured by TLS with pre-deployed AC and MTD certificate installed in LTD.

B.1.2 Connection management

Fall back mechanisms should be implemented in order to prevent core business discontinuity in case of problems such as domains disconnection. Fall backs mechanisms are the responsibility of implementers. Notifications are needed for admin to solve rapidly.

In case of a disconnection, from either side of the link, a call to sensitive function or service should return a specific status code (TDSC_TRUST_EXPIRED) to inform the LTD that the session is not secured anymore. It is the responsibility of the LTD to request a new connection and start the process again.

The interface should also handle the situation whereby a link is abnormally terminated (i.e. this function would be called by the MTD upon itself in response to a timeout or other abnormal termination of the link).

B.1.3 Predefined Container-id and Object-id

By default, Container-ids are created by the MTD upon LTD request within a session and so are erased at the end of session. Thus, to ease the use of permanent container (e.g. key database) a set of Container-Id is reserved. These containers cannot be created by one of the LTD. To avoid deny of service coming from a rogue LTD, these containers should be permanent (read-only).

The MTD-LTD administrator can also rely on predefined Object-Id to store permanent values.

B.2 Good practice

When a container is named with the CreateStorage, the LTD-Role and LTD-Id should be used to avoid collisions.

EXAMPLE: CreateStorage 6554355, PERMANENT_FILE, "Firewall-rules.cfg" rather than;
CreateStorage 6554355, PERMANENT_FILE, "rules.cfg".

In order to enable the trusted operation mode when instantiating the connection, the MTD should have a database of authorized RSA key pairs and CN, and the LTD should have access to a TPM identified by its common name.

In order to maintain the connection alive between the LTD and the MTD, new session should be requested by the LTD periodically.

B.3 TTLV encoding examples

B.3.1 GetRandom

GetRandom command encoding:

	TAG	TYPE	LENGTH	Value
Session-Id	7	2	16	0x00112233445566778899aabbccddeeff
Length	24	4	8	8

GetRandom response encoding:

	TAG	TYPE	LENGTH	Value
Object-Id	6	4	8	468558
TDSC_SUCCESS	13	5	2	0

B.3.2 StoreData

The StoreData function is used to store whether unstructured data or a (Key,Value) database pair.

StoreData command encoding:

	TAG	TYPE	LENGTH	Value
Session-Id	7	2	16	0x00112233445566778899aabbccddeeff
Container-Id	8	4	8	100100
Data	26	2	756	config_data

StoreData command encoding:

	TAG	TYPE	LENGTH	Value
Session-Id	7	2	16	0x00112233445566778899aabbccddeeff
Container-Id	8	4	8	100100
DB_KeyValue	11	6	34	DB_Key, DB_Value
DB_Key	12	2	8	LTD-Role
DB_Value	13	2	16	0xABC456820FFFBCE3D5D3257

Annex C (informative): Bibliography

- ETSI GS NFV-SEC 001: "Network Functions Virtualisation (NFV); NFV Security; Problem Statement".
- ETSI GS NFV-SEC 003: "Network Functions Virtualisation (NFV); NFV Security; Security and Trust Guidance".
- ETSI GS NFV-SEC 012: "Network Functions Virtualisation (NFV) Release 3; Security; System architecture specification for execution of sensitive NFV components".
- ETSI TS 103 487: "CYBER; Baseline security requirements regarding sensitive functions for NFV and related platforms".
- OASIS PKCS #11: "Cryptographic Token Interface Base Specification".
- OASIS: "Key Management Interoperability Protocol Specification".
- TTLV OASIS KMIP: "Key Management Interoperability Protocol Specification Version 1.4".
- FIPS 180-4: "Secure Hash Standard".
- FIPS 197: "Advanced Encryption Standard".
- PKCS#1 v2: "RSA Cryptography Standard".
- TCG TPM: "Trusted Platform Module (TPM) Specifications".

History

Document history		
V1.1.1	October 2018	Publication