OPTIGA™ Trust X1

Solution Reference Manual

About this document

Scope and purpose

The scope of this document is the OPTIGA™ Trust X1[1] solution spanning from the device with its external interface to the enabler components used for integrating the device with a bigger system.

Intended audience

This document addresses the audience: customers, solution providers, system integrators.

[1] All references regarding the OPTIGA™ Trust X are given generically without indicating the dedicated version (e.g. X1, ...).

Table of Contents

Table of Contents

Table of Contents

Figures

Figures

Tables

Tables

Tables

1 Definitions

This chapter Definitions provides abbreviations, naming conventions and references to maintain a common language throughout the document.

1.1 Abbreviations

Abbreviation	Term
AC	Access Condition
AES	Advanced Encryption Standard
APDU	Application Data Unit
API	Application Programming Interface
bdd	Block Definition Diagram
CA	Certification Authority
CCS	Infineon division Chip Card and Security
CRL	Certificate Revocation List
DD	Device Driver
DO	Data Object
DTLS	Datagram Transport Layer Security
EAL	Evaluation Assurance Level
EDH	Erase Disturb Handling
ESW	Embedded Software
GAD	Generic Authentication Device
GUI	Graphical User Interface
HFD/MCDS	Halogen-Free Declaration / Material Content Data Sheet
LC / LCM	Life Cycle / Life Cycle Management
NVM	Non-Volatile Memory
NW	Network
OID	Object Identifier
osd	Object Sequence Diagram
PMTU	Path Maximum Transmission Unit
RAM	Random-Access Memory
RoS	Restriction of Hazardous Substances
RS	Revocation Server
SEC	Security Event Counter
SecMC	Secure Microcontroller
SW	Software

Definitions

Abbreviation	Term
TLS	Transport Layer Security
UID	Unique Identifier
USB	Universal Serial Bus
VAR	Value-added Reseller
µC / MCU	Microcontroller

1.2 Naming Conventions

Throughout this document the naming of cryptographic material (e.g. keys) are constructed by concatenating abbreviations (in "camel notation") given in this section. (e.g. SmcPriAUT → OPTIGA™ Trust X Private Key for Authentication).

Abbreviation	Term
ECC	Elliptic Curve Crypto (Key)
EXT	Key Holder is External Entity
MAC	Message Authentication (Key, integrity)
PKI	Public Key Infrastructure
PRI	Private (Key)
PUB	Public (Key)
RND	Random Value
RSA	RSA (Key)
SEC	Secret (Key)
SES	Symmetric Session (Key)
SMC	Key Holder is Secure Micro Controller

1.3 References

This section provides references to information used in this document or information useful becoming familiar with in the context of the technology specified within this document.

[Data Sheet]	OPTIGA™ Trust X - Data Sheet
[DTLS]	https://tools.ietf.org/html/rfc6347 Datagram Transport Layer Security Version 1.2
[ESW]	Refer to chapter “OPTIGA™ Trust X External Interface” The external Interface of the OPTIGA™ Trust X
[Getting_Started]	OPTIGA™ Trust X - Getting Started Guide
[IFX_I2C]	Infineon Technologies AG; IFX I2C Protocol Specification

Definitions

[I²C]	http://www.nxp.com/documents/user_manual/UM10204.pdf www.nxp.com/documents/user_manual/UM10204.pdf NXP; UM10204 I²C-bus specification and user manual
[Keys_Certificates]	OPTIGA™ Trust X - Key and Certificates
[PC_AppNote]	OPTIGA™ Trust X - PC Application Note
[PKCS#1]	http://www.emc.com/emc-plus/rsa-labs/pkcs/files/h11300-wp-pkcs-1v2-2-rsa-cryptography-standard.pdf PKCS#1 v2.1: RSA cryptography standard. RSA Laboratories Technical Note, 2002.
[RFC4492]	https://tools.ietf.org/html/rfc4492 Elliptic Curve Cryptography (ECC) Cipher Suites for Transport Layer Security (TLS)
[RFC5116]	https://tools.ietf.org/html/rfc5116 An Interface and Algorithms for Authenticated Encryption
[RFC5280]	https://tools.ietf.org/html/rfc5280 Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile
[RFC6655]	https://tools.ietf.org/html/rfc6655 AES-CCM Cipher Suites for Transport Layer Security (TLS)
[RFC7925]	https://tools.ietf.org/html/rfc7925 Transport Layer Security (TLS)/ Datagram Transport Layer Security (DTLS) Profiles for the Internet of Thinks.
[TLS]	http://tools.ietf.org/html/rfc5246 The Transport Layer Security (TLS) Protocol, Version 1.2, August 2008
[UML]	http://www.omg.org/spec/UML/2.4.1 Object Management Group: “OMG Unified Modeling Language (OMG UML), ***Infrastructure*** *Version 2.4.1*”, August 2011, formal/2011-08-05 Object Management Group: “OMG Unified Modeling Language (OMG UML), ***Superstructure*** *Version 2.4.1*”, August 2011, formal/2011-08-06
[USB Auth]	<http://www.usb.org/developers/docs> Universal Serial Bus Type-C Authentication Specification
[X.509]	http://tools.ietf.org/html/rfc5280 Internet X.509 public key infrastructure certificate and certificate revocation list (CRL) profile. RFC5280, May 2008
[X.690]	ASN.1 encoding rules: Specification of Basic Encoding Rules (BER), Canonical Encoding Rules (CER) and Distinguished Encoding Rules (DER). X.690, 2002.
[XMC_AppNote]	OPTIGA™ Trust X - XMC Application Note

2 Supported Use Cases

In the Supported Use Cases chapter a collection of authentication use cases, which apply on the shown Architecture Decomposition, are expressed as UML sequence diagrams to show how to utilize the OPTIGA™ Trust X enabler components (OPTIGA™ Trust X device driver, Communication and Crypto APIs, ...) and the external OPTIGA™ Trust X interface to achieve the target functionality of the solution. This chapter is intended to maintain a well understanding of the OPTIGA™ Trust X eco system components particular for system integrators who like to integrate the OPTIGA™ Trust X with their solution.
The OPTIGA™ Trust X is produced within security production environment and meets the same high level security standards regarding access protection and secure work flows. The OPTIGA™ Trust X provides a number of arbitrary data objects which hold user/customer related information.
The subsequent document is structured in the chapters 'Supported Use Cases', 'Enabler APIs', 'OPTIGA™ Trust X External Interface' and 'Appendix'.
The 'Supported Use Cases' provides a number of sequence diagrams which are showing the detailed functionality of the OPTIGA™ Trust X.

2.1 Architecture Decomposition

Architecture Components

The architecture components contained in the various solution views (OPTIGA Trust X IP Protection View [bdd], OPTIGA Trust X Brand Protection View [bdd], OPTIGA Trust X Communication Protection View [bdd], OPTIGA Trust X Communication Protection View -toolbox- [bdd]) subsequently shown are listed and briefly described in the table below.

Name	Description
3rd Party Crypto Lib	Cryptographic functionalities are implemented in software and provided in 3rd Party Crypto Lib. The main cryptographic operations of interest for OPTIGA™ Trust X are certificate parsing, signature verification and certificate verification. 3rd Party Crypto Lib is supplied by 3rd party.
Accessory Application	The Accessory Application is the embedded application implementing the accessory functionality. For IP protection it uses the APIs exposed by the IntegrationLib and CommandLib.
CommandLib	CommandLib is the main interface to interact with OPTIGA™ Trust X. It is aware of the format of commands to be sent to OPTIGA™ Trust X. Commands to OPTIGA™ Trust X are sent and received in the form of APDUs. APDUs are described in chapter OPTIGA™ Trust X External Interface. CommandLib provides APIs for the cryptographic functionalities implemented in OPTIGA™ Trust X. Once the user calls the APIs with appropriate parameters the CommandLib prepares APDU and sends it to OPTIGA™ Trust X. The response from OPTIGA™ Trust X is received in the form of APDUs.

Supported Use Cases

Name	Description
	CommandLib extracts response from APDU and returns it to the application. CommandLib interacts with optiga_comms_ifx_i2c for reliable communication with OPTIGA™ Trust X.
CryptoLib	The CryptoLib wraps the Cryptographic Library specific API to a neutral API to enable multiple sourcing of a 3rd Party Crypto Lib maximizing the reuse of the cryptography consuming components of the OPTIGA™ Trust X solution. The CryptoLib exports only those API functions which are required by the OPTIGA™ Trust X solution.
Host Application	The *Host Application* is the embedded application implementing the host functionality.
IntegrationLib	The IntegrationLib mainly implements use cases which user can use for reference, for use cases which is made up of several commands implemented by command lib.
OCP	OCP (OPTIGA Crypto & Protected Comm Library) module is exposing the interface to the cryptographic functionalities. OCP fulfills the cryptographic functionalities by calling the implementation of the underlying components.
optiga_comms_ifx_i2c	optiga_comms_ifx_i2c is the interface used to communicate with OPTIGA™ Trust X. The CommandLib generates data in the form of APDUs to communicate with OPTIGA™ Trust X. Size of APDUs varies between few bytes to kilo bytes. The IFX I2C protocol handles this huge data transaction generated by different applications. The protocol implementation is done in multiple layers and seamlessly handles data transfer from Host to OPTIGA™ Trust X and OPTIGA™ Trust X to Host. More details of IFX I2C protocol can be found in [IFX_I2C].
optiga_comms_tc	optiga_comms_tc is used to communicate with the underlying HW interface like UDP/IP via the platform abstraction layer (pal). It takes care of seamless transfer of data over the transparent channel.
pal	The pal is a **P**latform **A**bstraction **L**ayer, abstracting HW and Operating System functionalities for the Infineon XMC family of µController or upon porting to any other µController. It abstracts away the low level device driver interface (socket_xmc, timer_xmc, i2c_xmc) to allow the modules calling it being platform agnostic. The pal is composed of hardware, software and an operating system abstraction part.
Server Application	The Server Application represents the entity to which the OPTIGA™ Trust X is used establishing a secure communication channel with.

2.1.1 OPTIGA Trust X IP Protection View [bdd]

Figure 'OPTIGA Trust X IP Protection View [bdd]' shows the block definition diagram of the IP Protection Solution Architecture containing its main functional blocks. This view is applied for IP protection solution kind of use cases, where the involved blocks are represented as dedicated lifelines.
The color coding provides information of whether the functional block gets entirely (yellow) or partly (green) provided by IFX or entirely (blue) provided by a 3rd party.

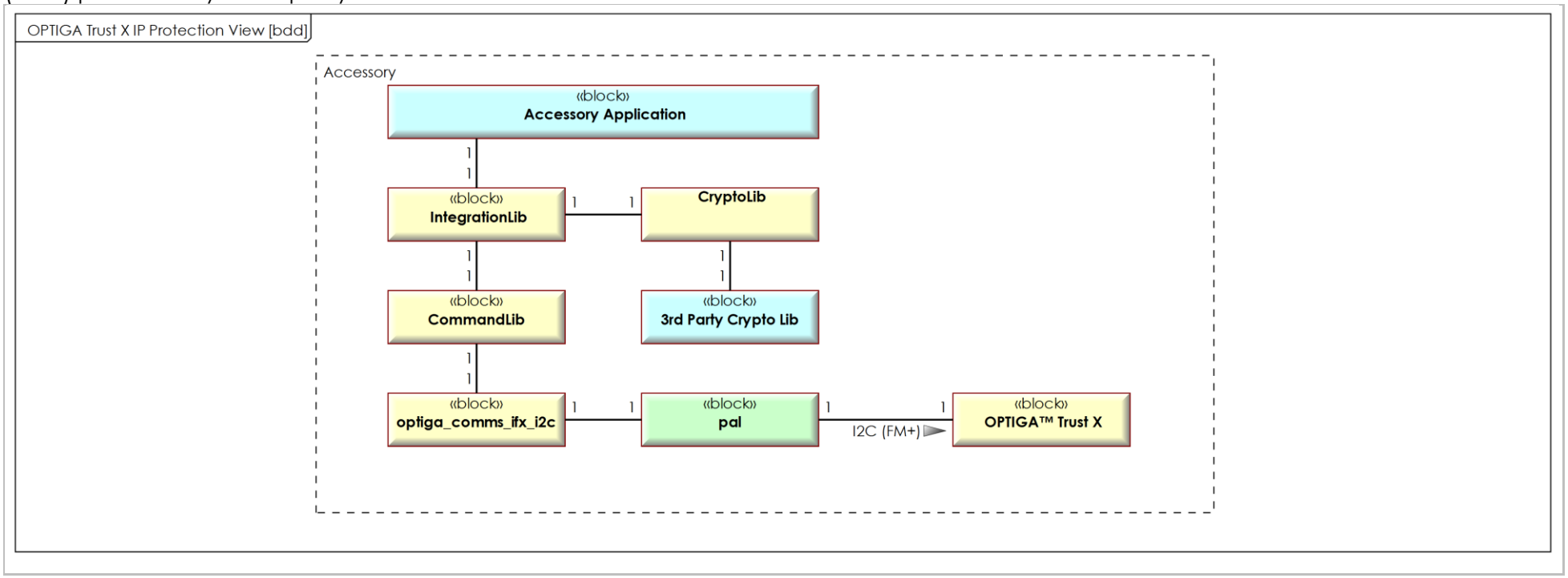

Figure 1 - OPTIGA Trust X IP Protection View [bdd]

2.1.2 OPTIGA Trust X Brand Protection View [bdd]

Figure 'OPTIGA Trust X Brand Protection View [bdd]' shows the block definition diagram of the Brand Protection Solution Architecture

containing its main functional blocks. The optiga_comms_tc and the Accessory Application are implementing a transparent channel between the CommandLib on the Host and the optiga_comms_ifx_i2c driver on the Accessory. This view is applied for Brand Protection solution kind of use cases, where the involved blocks are represented as dedicated lifelines.
The color coding provides information of whether the functional block gets entirely (yellow) or partly (green) provided by IFX or entirely (blue) provided by a 3rd party.

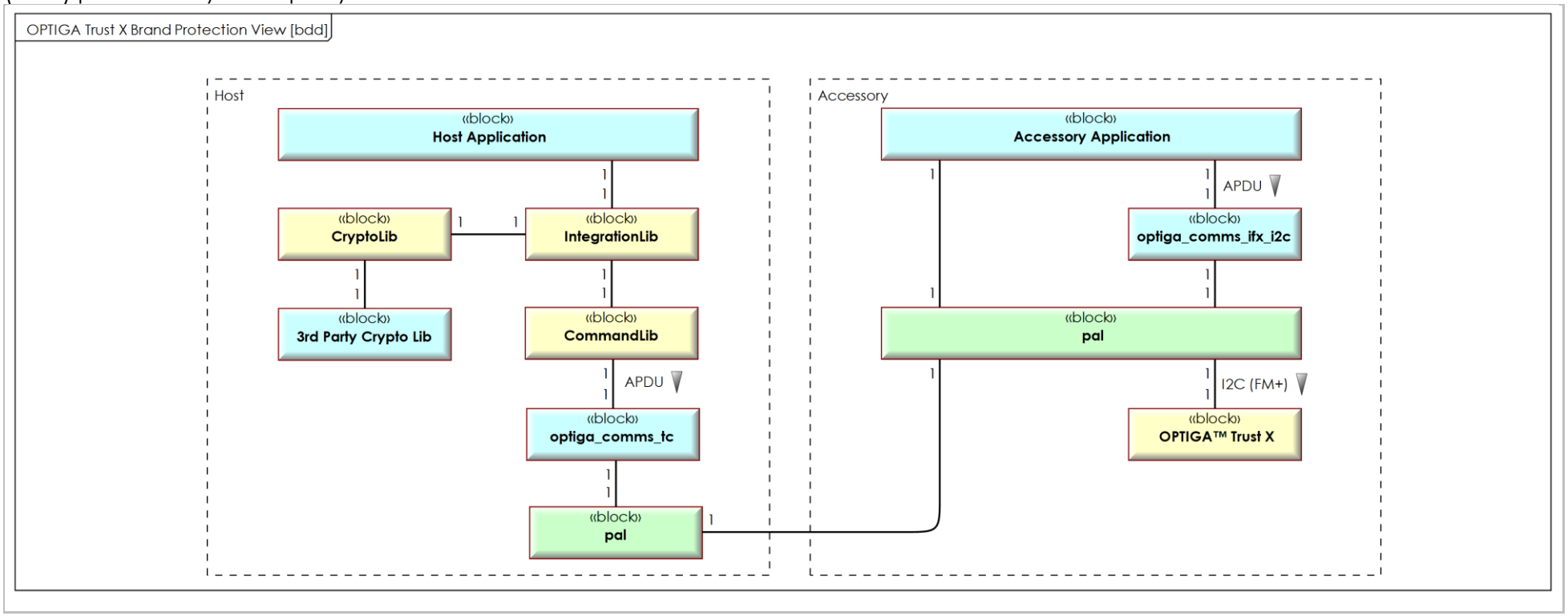

Figure 2 - OPTIGA Trust X Brand Protection View [bdd]

2.1.3 OPTIGA Trust X Communication Protection View [bdd]

Figure 'OPTIGA Trust X Communication Protection View [bdd]' shows the block definition diagram of the Communication Protection Solution Architecture containing its main functional blocks. The entities communicating across a protected channel are the Sever and the Client

(OPTIGA™ Trust X). This view is applied for Communication Protection Solution kind of use cases, where the involved blocks are represented as dedicated lifelines.
The color coding provides information of whether the functional block gets entirely (yellow) or partly (green) provided by IFX or entirely (blue) provided by a 3rd party.

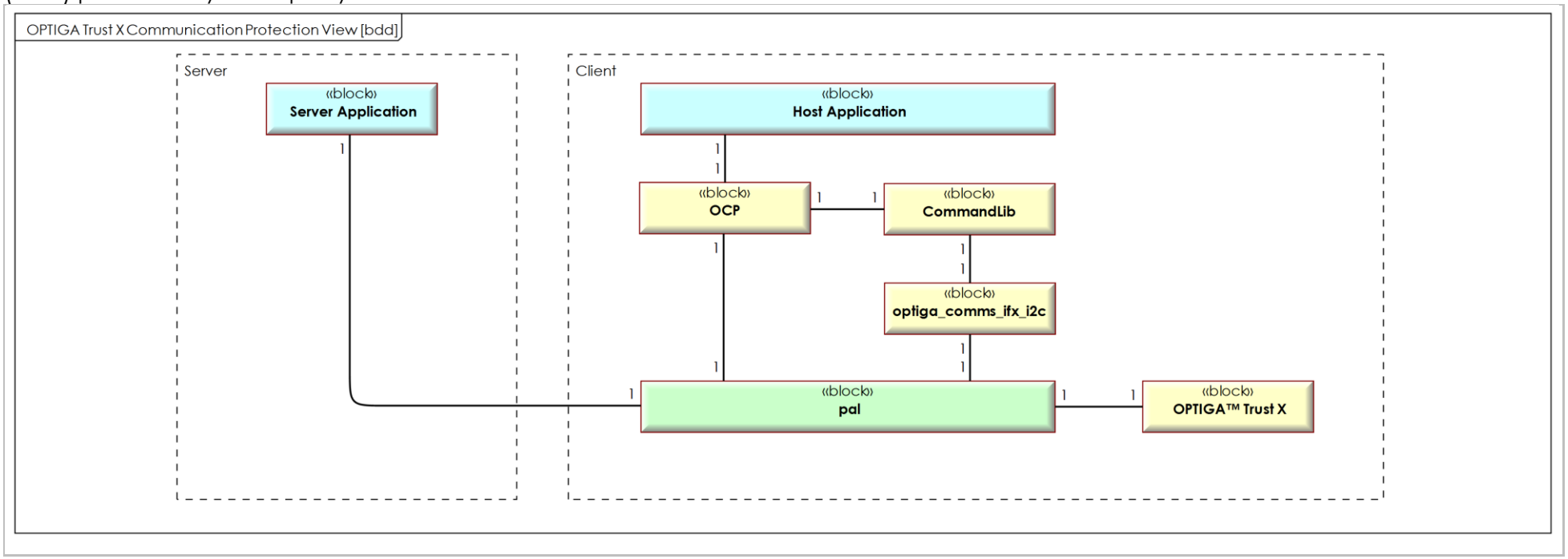

Figure 3 - OPTIGA Trust X Communication Protection View [bdd]

2.1.4 OPTIGA Trust X Communication Protection View -toolbox- [bdd]

Figure 'OPTIGA Trust X Communication Protection View -toolbox- [bdd]' shows the block definition diagram of the Toolbox based Communication Protection Solution Architecture, containing its main functional blocks. The entities communicating across a protected channel are the Sever and the Client (Host). This view is applied for Communication Protection Solution kind of use cases utilizing the toolbox functionality, where the involved blocks are represented as dedicated lifelines.
The color coding provides information of whether the functional block gets entirely (yellow) or partly (green) provided by IFX or entirely

(blue) provided by a 3rd party.

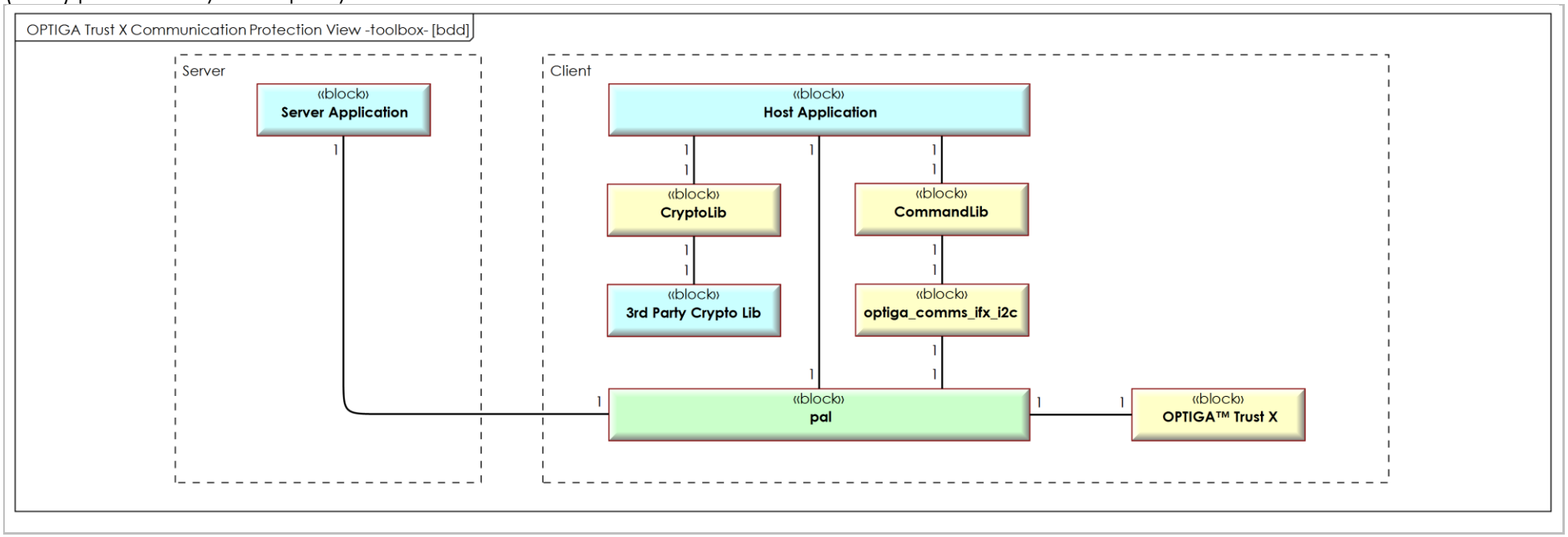

Figure 4 - OPTIGA Trust X Communication Protection View -toolbox- [bdd]

2.1.4.1 Host Code Size

The Table Host Code Size shows the footprint of the various host side configurations. The "Note" column names the component(s) contained in the footprint calculation. All other components even shown by the architecture diagram are project specific and provided by the system integrator. The values specified in the table Host Code Size are based on Keil ARM MDK v5.14, targeting Cortex M (32 bit) controller. These values are subjected to vary based on the target controller architecture (8/16/32 bit), compiler and optimization level chosen.

Configuration	Footprint (RAM / CODE)	Note
The IP Protection Architecture is shown by OPTIGA Trust X IP Protection View [bdd].	11 / 50 KByte	The components IntegrationLib, CryptoLib, 3rd Party Crypto Lib, CommandLib, optiga_comms_ifx_i2c are covered.
The Brand Protection Architecture (Accessory) is shown by OPTIGA Trust X Brand Protection View [bdd].	2 / 6 KByte	The component optiga_comms_ifx_i2c is covered.
The Brand Protection Architecture (Host) is shown by OPTIGA Trust X Brand Protection View [bdd].	11 / 50 KByte	The components IntegrationLib, CryptoLib, 3rd Party Crypto Lib, CommandLib are covered.
The Communication Protection Architecture (Server <=> OPTIGA Trust X) is shown by OPTIGA Trust X Communication Protection View [bdd].	8 / 27 KByte	The components OCP, CommandLib, optiga_comms_ifx_i2c are covered.

Table 1 - Host Code Size

2.2 Sequence Diagrams

2.2.1 Use Case: One-way Authentication - IP Protection [osd]

The Accessory Application likes to verify the authenticity of the OPTIGA™ Trust X. In case the authenticity verification succeeds, the Accessory considers itself not being cloned.
The CryptoLib provides a challenge (random value) and the OPTIGA™ Trust X simply applies a signature on it and returns the result as response by which it proofs its authenticity.
This sequence diagram is provided to show the functions involved in performing a one-way authentication based on a public key signature scheme.
Note: This use case applies on the OPTIGA Trust X IP Protection View [bdd].

Pre-condition:

- The OPTIGA™ Trust X application is already launched

Post-condition:

- The Accessory Application considers the OPTIGA™ Trust X as an authentic member of the regarded PKI domain (either IFX or Customer)

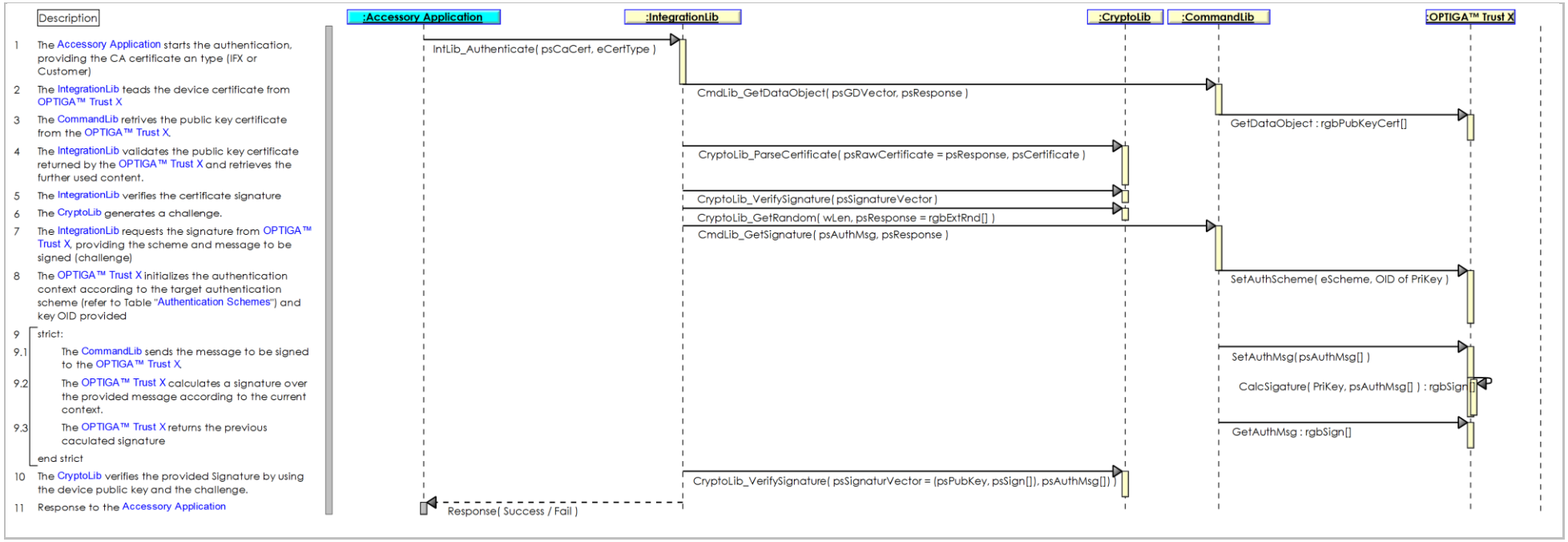

Figure 5 - Use Case: One-way Authentication - IP Protection [osd]

2.2.2 Use Case: One-way Authentication - Brand Protection [osd]

The Host Application likes to verify the authenticity of the OPTIGA™ Trust X. In case the authenticity verification succeeds, the Host considers the Accessory not being cloned and thus a legitimate member of the branded ecosystem.
The Host Application provides a challenge (random value) and the OPTIGA™ Trust X simply applies a signature on it and returns the result as response by which it proofs its authenticity.
This sequence diagram is provided to show the functions involved in performing a one-way authentication based on a public key signature scheme.
Note 1: This use case applies on the OPTIGA Trust X Brand Protection View [bdd]*.*
Note 2: The transparent channel (TC) is implemented by optiga_comms_tc *and* Accessory Application*, but not shown here, since the TC is not adding any semantics. However, the TC is adding a communication technology specific delay!*

Pre-condition:

- The OPTIGA™ Trust X application is already launched

Post-condition:

- The Host Application considers the OPTIGA™ Trust X as an authentic member of the regarded PKI domain (either IFX or Customer)

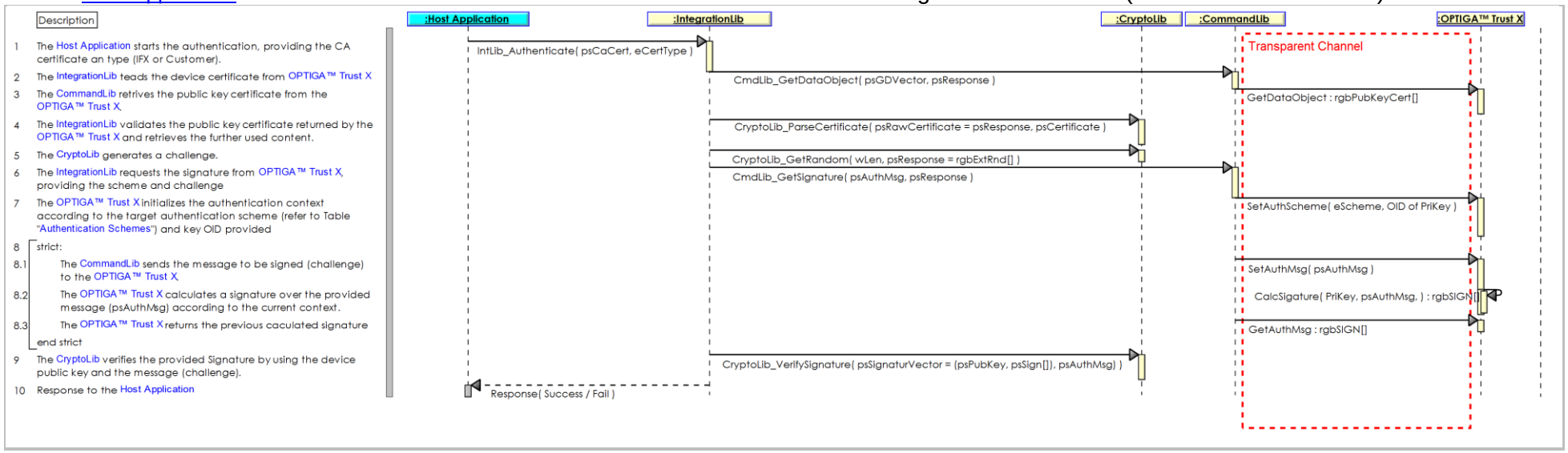

Figure 6 - Use Case: One-way Authentication - Brand Protection [osd]

2.2.3 Use Case: Mutual Authentication (DTLS-Client-Overview) [osd]

The Server Application and the Host Application, like to prove that they are mutually authentic. Both the Server Application and the Host Application, with the help of OPTIGA™ Trust X, executing the DTLS Handshake and ChangeCipherSpec protocol by which, upon successful execution, both parties prove their authenticity and negotiating session key material for further use with application records.

Sequence diagram 'Use Case: Mutual Authentication (DTLS-Client-Overview) [osd]' shows the sequence of operations for DTLS handshake. Messages are grouped to measure the retransmission timeout and upon elapsing retransmitting the uplink message. For more details about DTLS handshake refer [DTLS].

Note: This use case applies on the OPTIGA Trust X Communication Protection View [bdd].

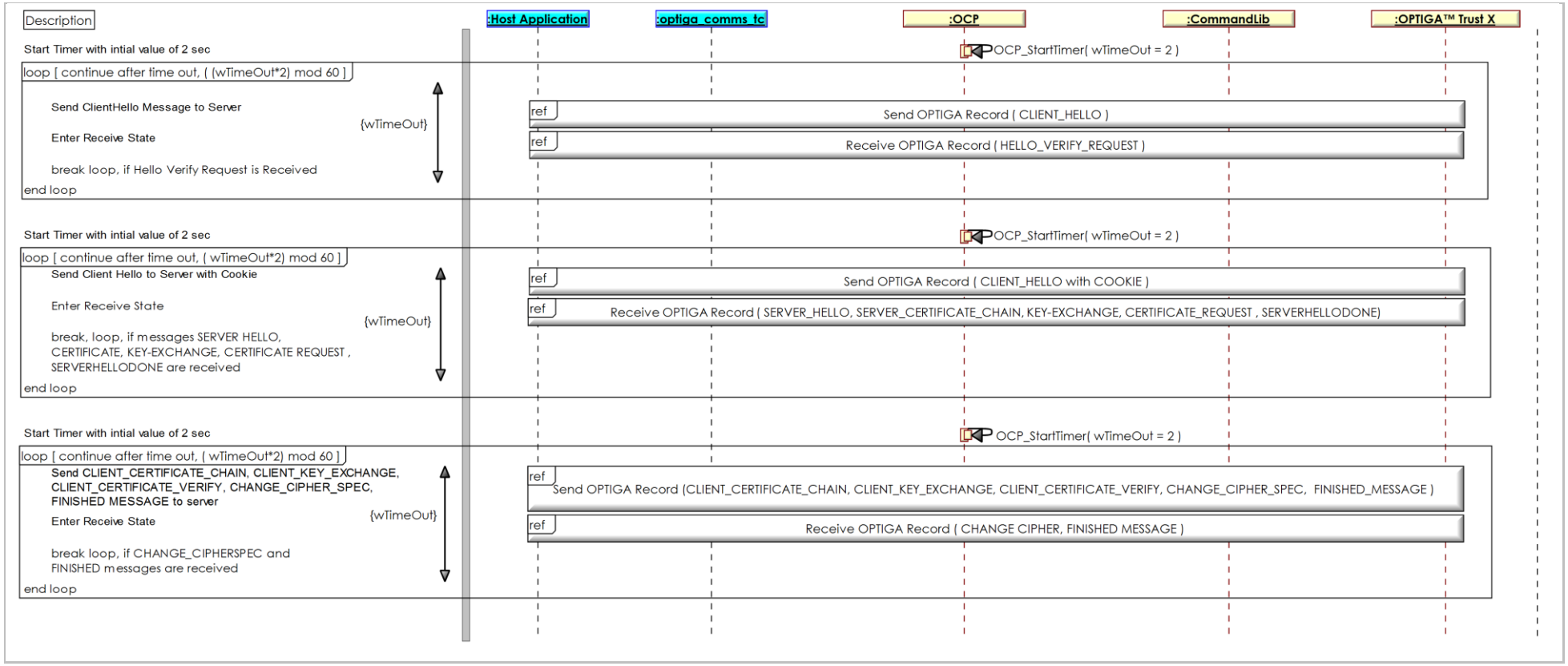

Figure 7 - Use Case: Mutual Authentication (DTLS-Client-Overview) [osd]

2.2.4 Use Case: Mutual Authentication (DTLS-Client-Detailed) [osd]

The External World and the OPTIGA™ Trust X, like to prove that they are mutually authentic. Both the External World and the OPTIGA™ Trust X executing the DTLS Handshake and ChangeCipherSpec protocol by which upon successful execution both parties prove their authenticity and negotiating session key material for further use with application records. The scope of this diagram is on the OPTIGA™ Trust X external interface invoked by the CommandLib.

Figure "Use Case: Mutual Authentication (DTLS-Client-Detailed) [osd]" shows the sequence of functions to be executed in case the OPTIGA™ Trust X

Supported Use Cases

performs the client side part of the protocol.
Note: This use case applies on the OPTIGA Trust X Communication Protection View [bdd].

Pre-condition:

- The OPTIGA™ Trust X application is already launched
- Unique identities, expressed by a Private key and the corresponding public key certificate, are available to both parties.
- Public key certificates provided to the OPTIGA™ Trust X roots back (maybe multiple certificates in a chain) to the trust anchor residing in the OPTIGA™ Trust X.

Post-condition:

- The External World considers the OPTIGA™ Trust X as an authentic member of the regarded PKI domain (either IFX or Customer) and vice versa.
- Session key material is negotiated and known by both parties.

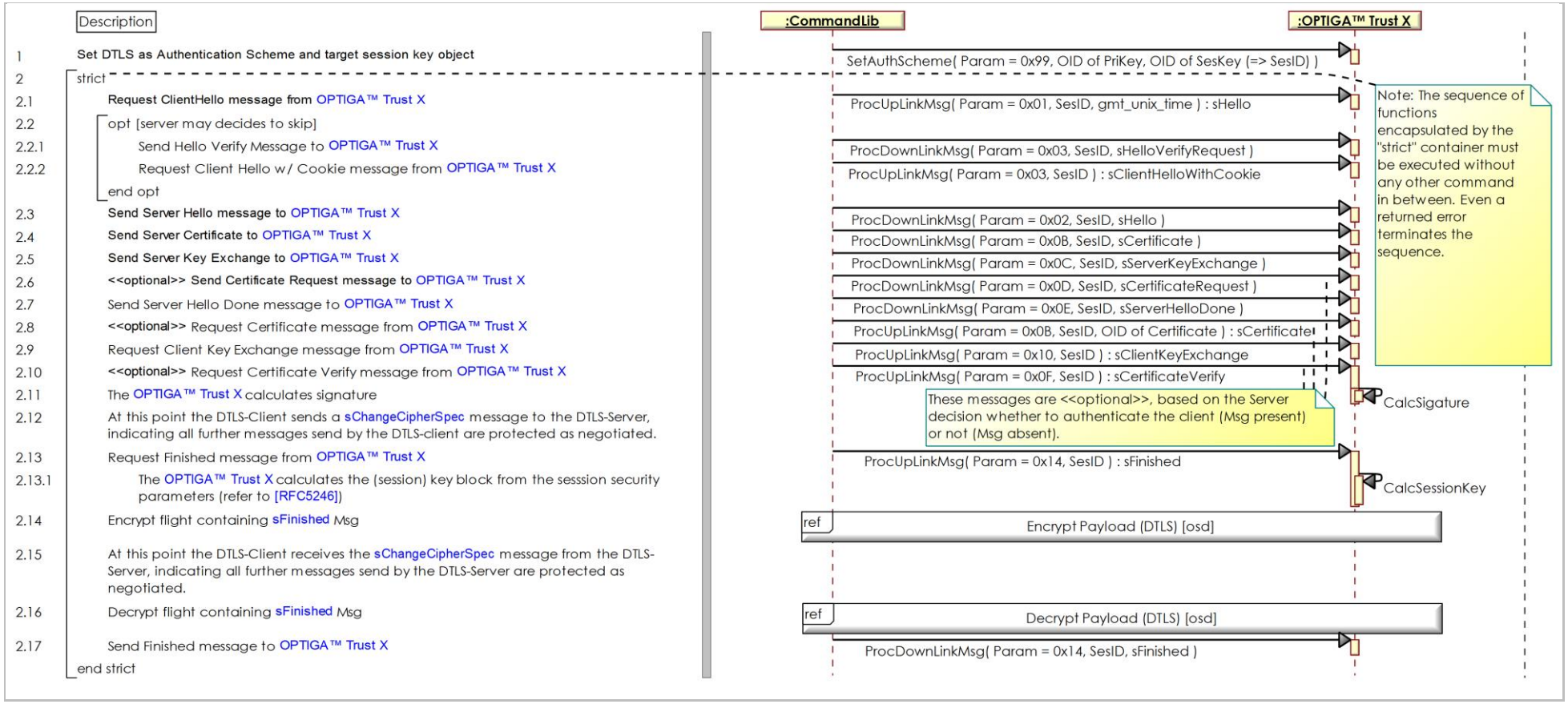

Figure 8 - Use Case: Mutual Authentication (DTLS-Client-Detailed) [osd]

2.2.5 Use Case: Protect communication data with OPTIGA™ Trust X [osd]

Once the Use Case: Mutual Authentication (DTLS-Client-Detailed) [osd] is successful executed the user likes to send / receive integrity and confidentiality protected application data to / from authenticated server. Sequence diagram 'Use Case: Protect communication data with OPTIGA™ Trust X [osd]' shows the regarded sequence of operations. The data is either consumed (uplink)or provided (downlink) by the External World (Host Application).
Note: This use case applies on the OPTIGA Trust X Communication Protection View [bdd].

Supported Use Cases

Pre-condition:

- The Use Case: Mutual Authentication (DTLS-Client-Detailed) [osd] is successful executed and the session key material is negotiated and initialized.

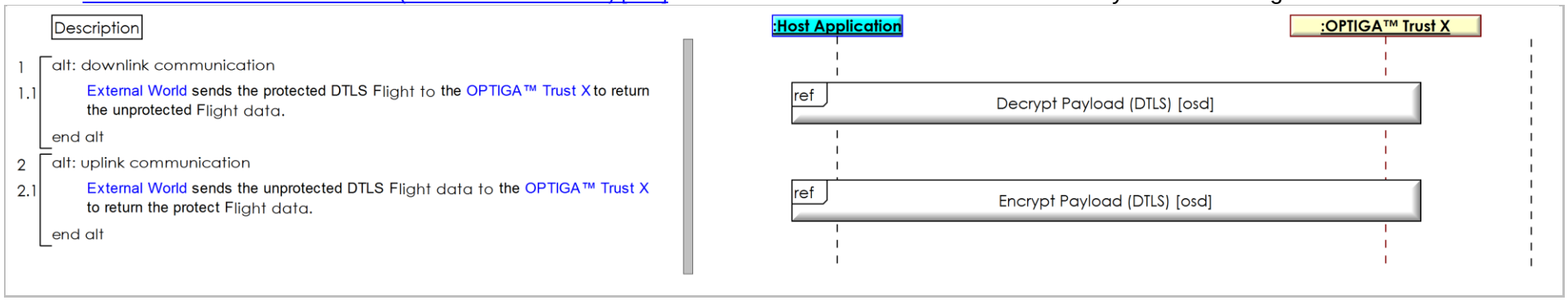

Figure 9 - Use Case: Protect communication data with OPTIGA™ Trust X [osd]

2.2.6 Use Case: Write General Purpose Data - data object [osd]

The External World (Host Application or Accessory Application) likes to update a data object maintained by the OPTIGA™ Trust X.
The sequence diagram Use Case: Write General Purpose Data - data object [osd] is provided to show the functions involved in performing an update of data object.
Note: This use case applies on all architecture views.

Pre-condition:

- The *OPTIGA™ Trust X* application is already launched
- The necessary access condition for writing the target data object are satisfied

Post-condition:

- The target data object is updated

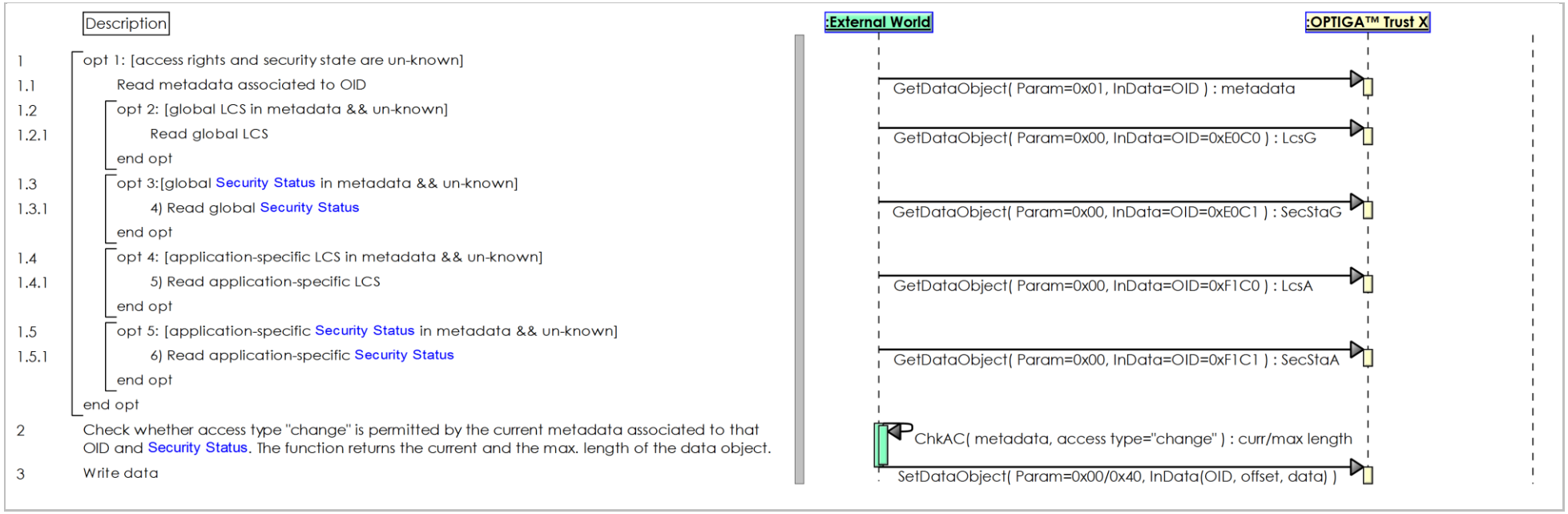

Figure 10 - Use Case: Write General Purpose Data - data object [osd]

2.2.7 Use Case: Write General Purpose Data - metadata [osd]

The External World (Host Application or Accessory Application) likes to update the metadata associated to a data object which is maintained by the OPTIGA™ Trust X.

The sequence diagram Use Case: Write General Purpose Data - metadata [osd] is provided to show the functions involved in updating metadata associated to a data object.

Note: This use case applies on all architecture views.

Pre-condition:

- The *OPTIGA™ Trust X* application is already launched
- The necessary access condition for writing the metadata associated to a data object are satisfied.

Post-condition:

- The metadata associated to the target data object are updated

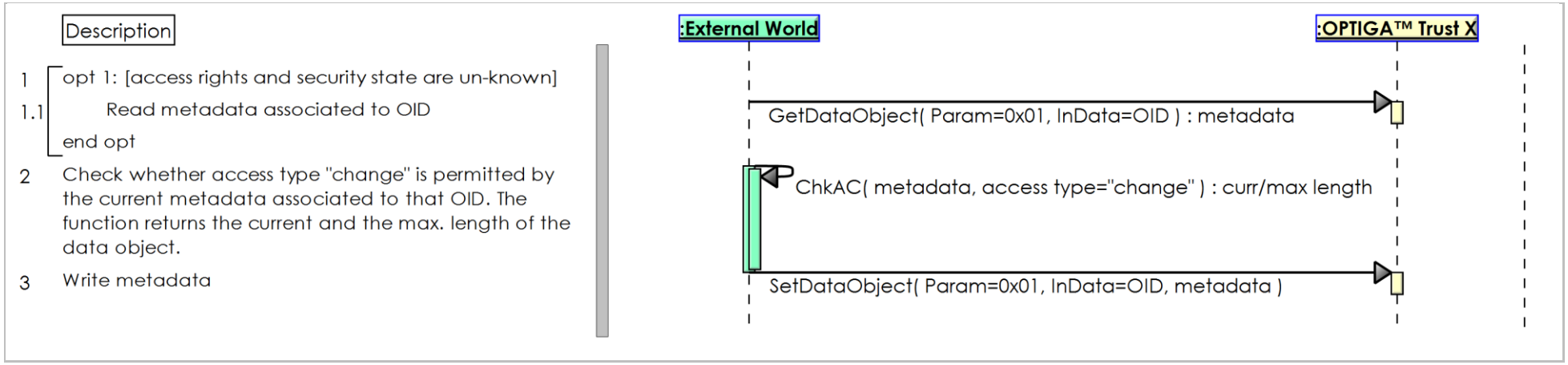

Figure 11 - Use Case: Write General Purpose Data - metadata [osd]

2.2.8 Use Case: Read General Purpose Data - data object [osd]

The External World (Host Application or Accessory Application) likes to read the content of a data object maintained by the OPTIGA™ Trust X.
The sequence diagram Use Case: Read General Purpose Data - data object [osd] is provided to show the functions involved in reading a data object.
Note: This use case applies on all architecture views.

Pre-condition:

- The *OPTIGA™ Trust X* application is already launched
- The necessary access condition for reading the target data object are satisfied

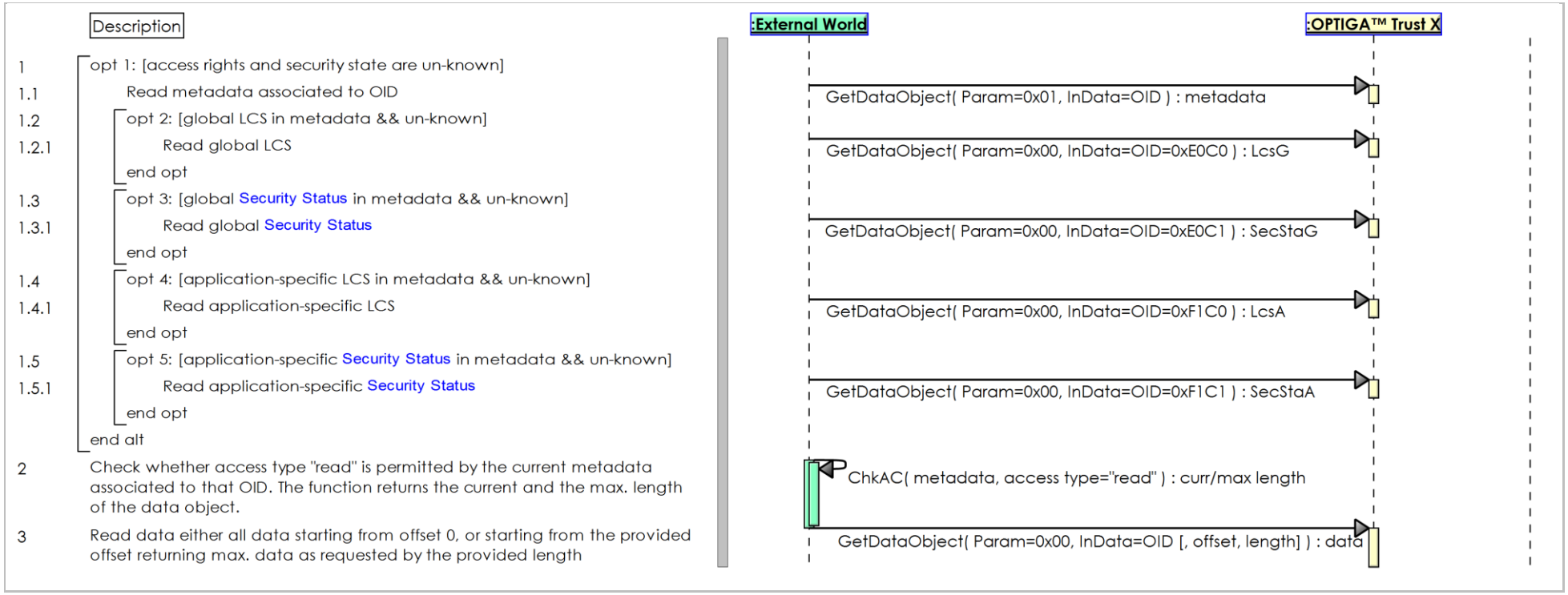

Figure 12 - Use Case: Read General Purpose Data - data object [osd]

2.3 Toolbox based Sequence Diagrams

All use case shown in this section apply on OPTIGA Trust X Communication Protection View -toolbox- [bdd].

2.3.1 Use Case: Mutual Auth establish session -toolbox- (TLS-Client) [osd]

The Server and the Client (on behalf of the User), which incorporates the OPTIGA™ Trust X, like to proof the authenticity of each other. Both the Server

Supported Use Cases

and OPTIGA™ Trust X providing challenges (random value) and both entities return one or multiple cryptograms (depending on the applied authentication protocol) as response by which both parties proof their authenticity. The Server and Client executing ECDHE for key agreement and ECDSA FIPS 186-3 sign SHA256 hash for authentication.

Note: the hashing of the handshake messages by the Client is not shown. This could be performed by SW at the Client or via CalcHash command by the OPTIGA™ Trust X. In the latter case the intermediate results shall be returned by OPTIGA™ Trust X and provided for continuing the hashing with further commands.

Pre-conditions:

- The OPTIGA™ Trust X application is already launched
- The public key pairs for authentication purpose and public key certificates are properly installed at the OPTIGA™ Trust X.
- The Trust Anchor for verifying the Public Key Certificates of the authentication partner (Server) is properly installed.

Post-condition:

- The Client knows the session keys (write_key) to run the application protocol without the help of the OPTIGA™ Trust X.

Supported Use Cases

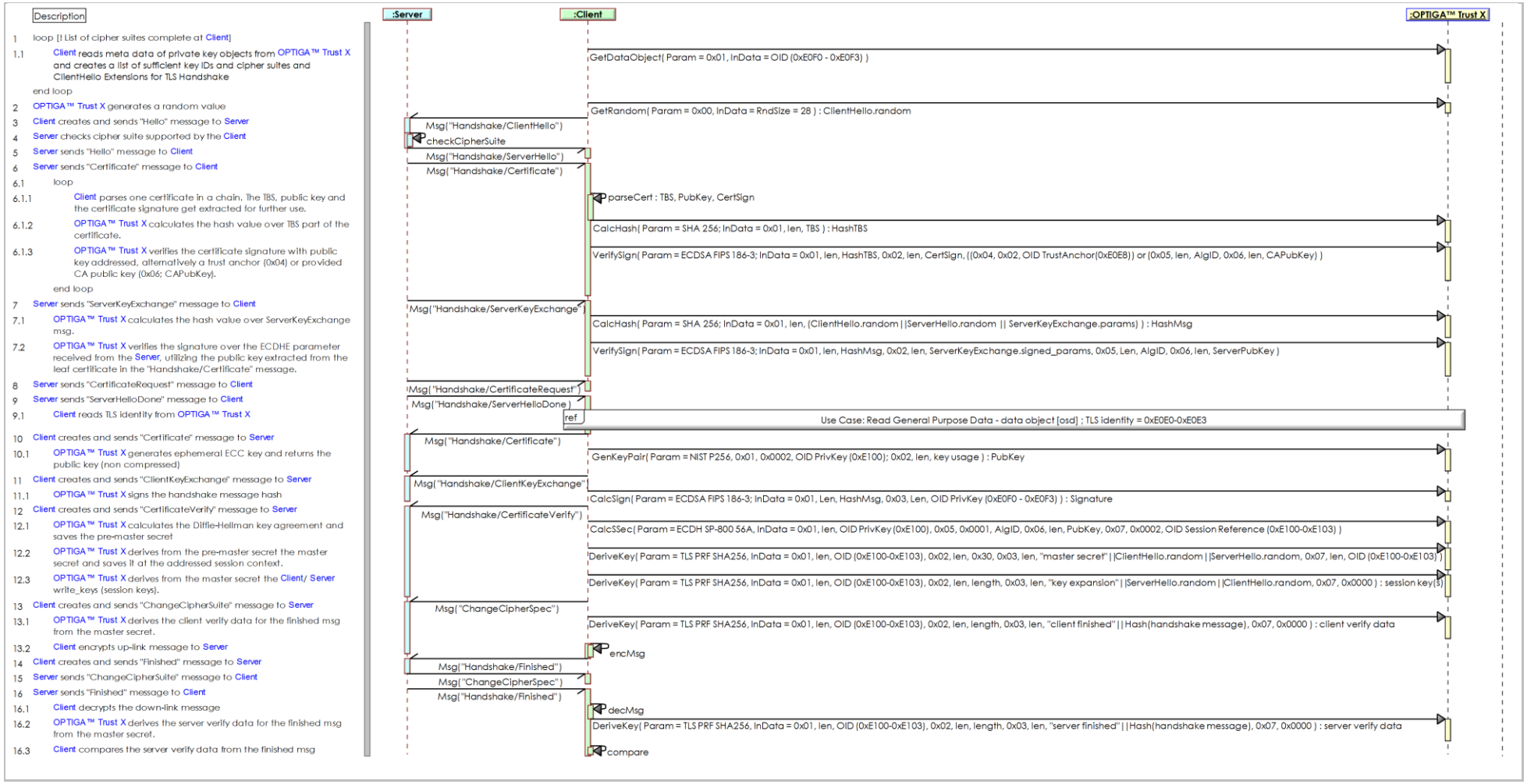

Figure 13 - Use Case: Mutual Auth establish session -toolbox- (TLS-Client) [osd]

2.3.2 Use Case: Abbreviated Handshake -toolbox- (TLS-Client) [osd]

The Server and the Client (on behalf of the User), which incorporates the OPTIGA™ Trust X, like to resume an established session. Both the Server and OPTIGA™ Trust X providing challenges (random value via "Hello" msg) and both entities providing verification data to prove the possession of the cryptographic parameters (master secret) previously negotiated.

Note: the hashing of the handshake messages by the Client is not shown. This could be performed by SW at the Client or via CalcHash command by the OPTIGA™ Trust X. In the latter case the intermediate results shall be returned by OPTIGA™ Trust X and provided for continuing the hashing with further commands.

Pre-conditions:

- The OPTIGA™ Trust X session master secret, which was calculated by the previous handshake - is available at the regarded session context and gets used as input by DeriveKey for the new session keys.
- The Client is able to hash all handshake messages without the help of OPTIGA™ Trust X.

Post-condition:

- The Client knows the session keys (write_key) to run the application protocol without the help of the OPTIGA™ Trust X.

Supported Use Cases

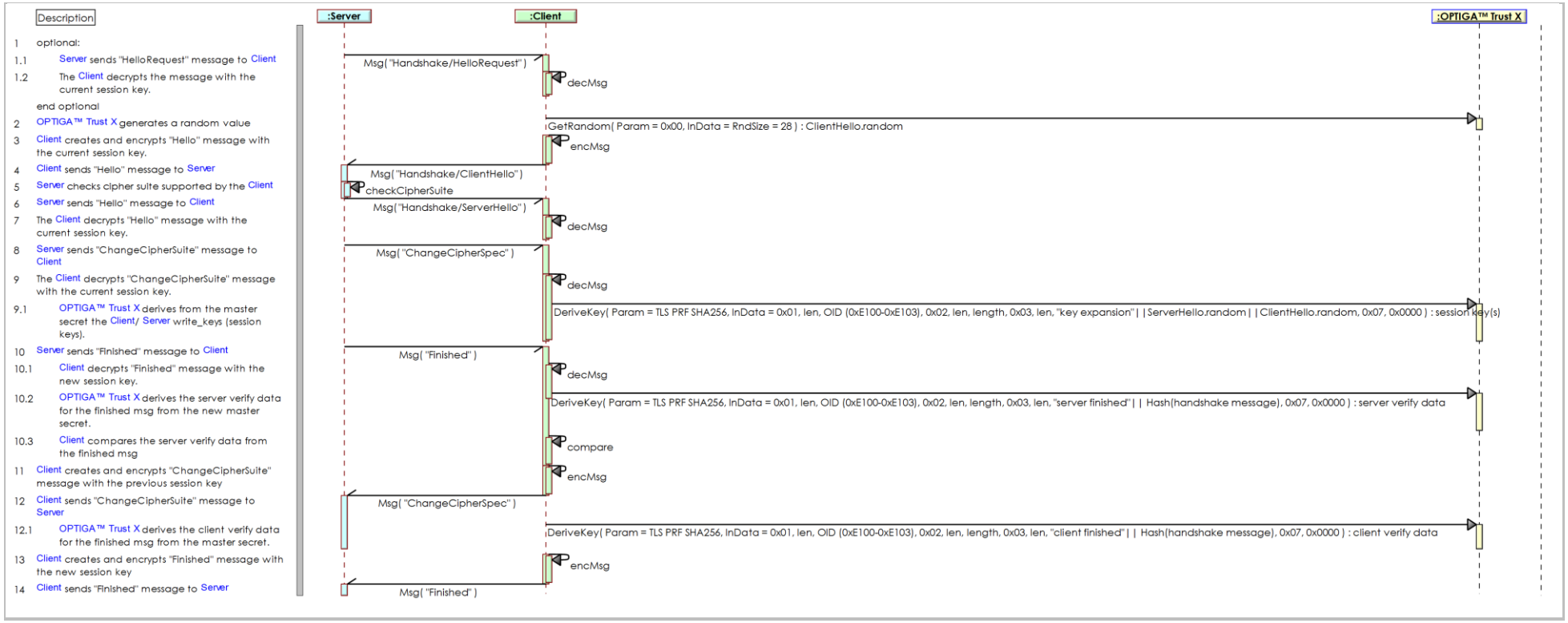

Figure 14 - Use Case: Abbreviated Handshake -toolbox- (TLS-Client) [osd]

2.3.3 Use Case: Host FW Update -toolbox-

The Host likes to update its FW in a protected way, which prevents from installation and execution of unauthorized code. This sequence diagram is provided to show the OPTIGA™ Trust X functions (CalcHash, VerifySign, DeriveKey) involved in performing the use case.

Pre-condition:

- The FW-image shared secret is loaded to an arbitrary data object (e.g. 0xF1D0-0xF1DF), which should be locked for read = NEV and in operational mode at least.
- The Platform Integrity Trust Anchor (0xE0EF) is loaded.

Post-condition:

- The metadata signature is verified
- The FW-image decryption secret is returned to the *Host*

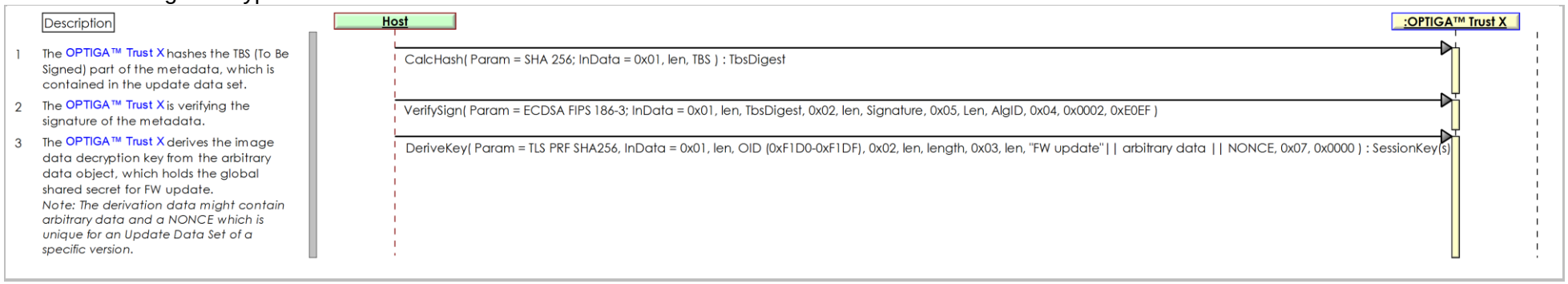

Figure 15 - Use Case: Host FW Update -toolbox-

2.4 Referenced Sequence Diagrams

2.4.1 Encrypt Payload w/o chaining (DTLS) [osd]

The External World likes to protect (e.g. encrypt, integrity value) fragments of a DTLS payload. The OPTIGA™ Trust X returns the protected data as long as encryption is involved. In case of integrity protection the regarded integrity value is returned as response to the command providing the last part of the fragment.

Pre-condition(s):

- The OPTIGA™ Trust X knows the session key.

Post-condition(s):

- The un-protected payload provided is protected, which means
  in case of encryption "available encrypted" and
  in case of integrity protection the integrity value is generated and returned by OPTIGA™ Trust X.

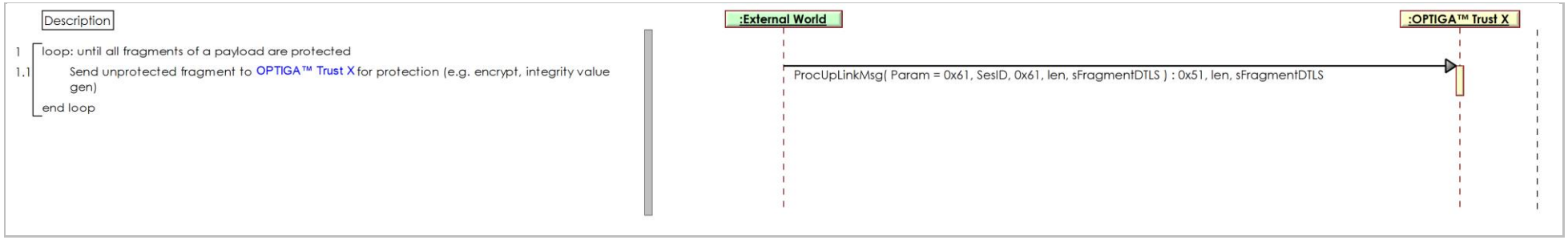

Figure 16 - Encrypt Payload w/o chaining (DTLS) [osd]

2.4.2 Decrypt Payload w/o chaining (DTLS) [osd]

The External World likes to unprotect (e.g. decrypt, integrity check) Fragments of a DTLS payload. The OPTIGA™ Trust X returns the plain data as long as decryption is involved. In case of integrity protection the regarded state gets returned as response to the command providing the last part of the fragment.

Pre-condition(s):

- The OPTIGA™ Trust X knows the session key.

Post-condition(s):

- The protected payload provided is unprotected, which means
  in case of decryption "available in plain" and
  in case of integrity protection the integrity is verified and the regarded status is available.

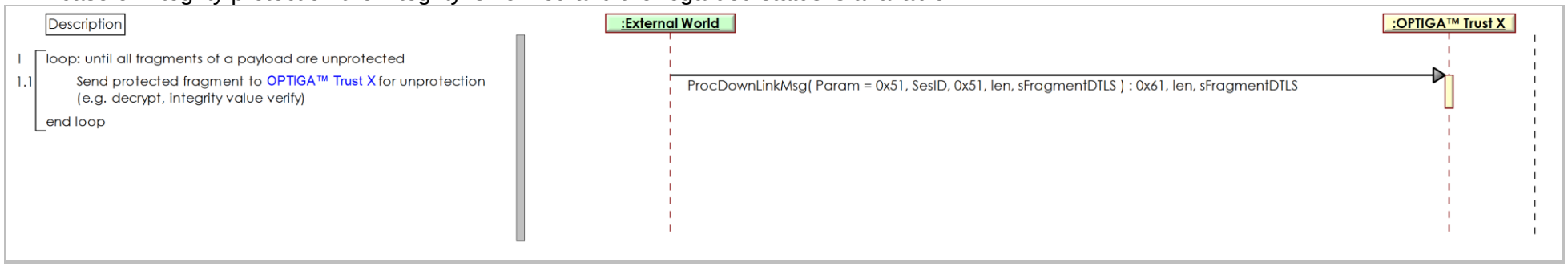

Figure 17 - Decrypt Payload w/o chaining (DTLS) [osd]

3 Enabler APIs

The Enabler APIs chapter provides the specification of the host side APIs of the enabler components which get provided by the OPTIGA™ Trust X solution. The target platforms for those enabler components are embedded systems, Linux and Windows.

3.1 CommandLib

CommandLib is the main interface to interact with OPTIGA™ Trust X. It is aware of the format of commands to be sent to OPTIGA™ Trust X. Commands to OPTIGA™ Trust X are sent and received in the form of APDUs. APDUs are described in [ESW] .

CommandLib provides APIs for the cryptographic functionalities implemented in OPTIGA™ Trust X. Once the user calls the APIs with appropriate parameters CommandLib prepares APDU and sends it to OPTIGA™ Trust X. The response from OPTIGA™ Trust X is received in the form of APDUs. CommandLib extracts response from APDU and returns it to the application.

CommandLib interacts with optiga_comms_ifx_i2c for reliable communication with OPTIGA™ Trust X.

3.1.1 CmdLib_CloseSession

CmdLib_CloseSession	
Description	CmdLib_CloseSession closes a OPTIGA™ Trust X Security Chip session as indicated by the Session Reference (OID)
Note	Return values: CMD_LIB_OK, CMD_LIB_ERROR, CMD_LIB_INVALID_SESSIONID, CMD_LIB_INSUFFICIENT_MEMORY
Signature	CmdLib_CloseSession (in PwSessionRefId : uint16_t) : int32_t
Parameters	`in PwSessionRefId : uint16_t` OID of session to be closed

3.1.2 CmdLib_Decrypt

CmdLib_Decrypt	
Description	CmdLib_Decrypt decrypts data by issuing ProcDownLink command to OPTIGA™ Trust X Security Chip. Notes: • Application on security chip must be opened using CmdLib_OpenApplication before using this operation. • Input and Output buffers must be provided by the caller. Buffer deallocation is the responsibility of the user. • The input data in sbBlob_d, sInData should contain sufficient memory to accommodate APDU header, data formatting, Ciphertext. • The Ciphertext and any specific data for decryption should start after an offset of size OVERHEAD_UPDOWNLINK. • wInDataLength in sProcCryptoData_d should be greater than zero. • Plaintext is returned in sCmdResponse_d* sOutData from zero offset. • In addition to the Plaintext, the length of buffer in sOutData should be sufficient to accommodate Response APDU header and data formatting. This is defined as OVERHEAD_ENCDEC_RESPONSE • The total length of the Plaintext is returned in wRespLength of sCmdResponse_d. • The current implementation of Security chip does not support command chaining. The maximum value of wInDataLength depends on the value

Enabler APIs

CmdLib_Decrypt	
	supported by the security chip. • Currently, the security chip supports only 0xE100 as session key OID.
Note	Return values: CMD_LIB_OK, CMD_LIB_ERROR, CMD_LIB_INSUFFICIENT_MEMORY, CMD_LIB_INVALID_SESSIONID, CMD_LIB_INVALID_LEN, CMD_DEV_ERROR, CMD_LIB_NULL_PARAM
Signature	CmdLib_Decrypt (inout PpsDecVector : sProcCryptoData_d) : int32_t
Parameters	`inout PpsDecVector : sProcCryptoData_d` Pointer to structure containing Ciphertext and Plaintext

3.1.3 CmdLib_Encrypt

CmdLib_Encrypt	
Description	CmdLib_Encrypt encrypts data by issuing ProcUpLink command to OPTIGA™ Trust X Security Chip. Notes: • Application on security chip must be opened using CmdLib_OpenApplication before using this operation. • Input and Output buffers must be provided by the user. Buffer deallocation is the responsibility of the user. • The input data in sbBlob_d sInData should contain sufficient memory to accommodate APDU header, data formatting,Plaintext. • The Plaintext and any specific data for encryption should start after an offset of size OVERHEAD_UPDOWNLINK. • wInDataLength in sProcCryptoData_d should be greater than zero. • Ciphertext is returned in sCmdResponse_d* sOutData from zero offset. • In addition to the Ciphertext, the length of buffer in sOutData should be sufficient to accommodate Response APDU header and data formatting. This is defined as OVERHEAD_ENCDEC_RESPONSE • The total length of the Ciphertext is returned in wRespLength of sCmdResponse_d. • The current implementation of Security chip does not support command chaining. The maximum value of wInDataLength depends on the value supported by the security chip. • Currently, the security chip supports only 0xE100 as session key OID.
Note	Return values: CMD_LIB_OK, CMD_LIB_ERROR, CMD_LIB_INSUFFICIENT_MEMORY, CMD_LIB_INVALID_SESSIONID, CMD_LIB_INVALID_LEN, CMD_DEV_ERROR, CMD_LIB_NULL_PARAM
Signature	CmdLib_Encrypt (inout PpsEncVector : sProcCryptoData_d) : int32_t
Parameters	`inout PpsEncVector : sProcCryptoData_d` Pointer to structure containing Plaintext and Ciphertext

3.1.4 CmdLib_GetMaxCommsBufferSize

CmdLib_GetMaxCommsBufferSize	
Description	CmdLib_GetMaxCommsBufferSize returns the maximum communication buffer size supported by the security chip. Notes:

Enabler APIs

CmdLib_GetMaxCommsBufferSize	
	• Application on security chip must be opened using CmdLib_OpenApplication before using this operation. • The operation does not verify if the read access is permitted for the data object.
Note	Return values: CMD_LIB_OK, CMD_LIB_ERROR
Signature	CmdLib_GetMaxCommsBufferSize (inout PpsGMsgVector : sProcMsgData_d)
Parameters	`inout PpsGMsgVector : sProcMsgData_d`

3.1.5 CmdLib_GetMessage

CmdLib_GetMessage	
Description	CmdLib_GetMessage generates uplink message by issuing ProcUpLink command to OPTIGA™ Trust X Security Chip. Notes: • Application on security chip must be opened using CmdLib_OpenApplication before using this operation. • Caller should provide a callback through sCallBack_d. • This callback allows the caller to allocate memory for the message and keep copying data into the memory in case of lengthy messages. • Allocated buffer is returned to caller in sCBGetMsg_d. • The callback should return CMD_LIB_OK for successful allocation of memory else CMD_LIB_ERROR in case of error. • Any Message specific data must be provided by the caller in the union puMsgParams. The union is defined as uMsgParams_d. • The caller must provide correct data in puMsgParams. This function does not validate the content of the message specific data. E.g For sending gmt_unix_time for Client Hello message, uMsgParams_d.sMsgParamCH_d.dwUnixTime must be set. If puMsgParams is set to NULL, then random dwUnixTime will be considered for Client Hello message and certificate will not be send for Client Certificate message. • The psBlobInBuffer pointer which is member of sProcMsgData_d should be set to NULL.
Note	Return values: CMD_LIB_OK, CMD_LIB_ERROR, CMD_LIB_INVALID_PARAM, CMD_LIB_INSUFFICIENT_MEMORY, CMD_DEV_ERROR, CMD_LIB_NULL_PARAM
Signature	CmdLib_GetMessage (inout PpsGMsgVector : sProcMsgData_d) : int32_t
Parameters	`inout PpsGMsgVector : sProcMsgData_d` Pointer to (D)TLS Handshake Message parameters

3.1.6 CmdLib_PutMessage

CmdLib_PutMessage	
Description	CmdLib_PutMessage processes downlink message by issuing ProcDownLink command to OPTIGA™ Trust X Security Chip Notes: • Application on security chip must be opened using CmdLib_OpenApplication before using this operation.

Enabler APIs

CmdLib_PutMessage	
	• Input buffer must be provided by the caller. • Clearing of the buffers is the responsibility of the user. • The input pointer should contain sufficient memory to accommodate APDU header and data formatting. • The operation will not recopy the Authentication message data but add the header and data formatting information before it, in the same input buffer. The puMsgParams and psCallBack pointer which is member of sProcMsgData_d should be set to NULL
Note	Return values: CMD_LIB_OK, CMD_LIB_ERROR, CMD_LIB_INVALID_PARAM, CMD_LIB_INSUFFICIENT_MEMORY, CMD_DEV_ERROR, CMD_LIB_NULL_PARAM
Signature	CmdLib_PutMessage (in PpsPMsgVector : sProcMsgData_d) : int32_t
Parameters	`in PpsPMsgVector : sProcMsgData_d` Pointer to (D)TLS Handshake Message parameters

3.1.7 CmdLib_CalcHash

CmdLib_CalcHash	
Description	CmdLib_CalcHash calculates a hash of the input data using the OPTIGA™ Trust X. Input: • Provide the required type of input data for hashing. Use sCalcHash_d::eHashDataType with the following options, eDataStream : Indicates, sDataStream is considered as hash input. eOIDData : Indicates, sOIDData is considered for hash input. • Provide the input to import/export the hash context. Use sContextInfo_d::eContextAction with the following options, eImport : Import hash context to perform the hash. eExport : Export current active hash context. eImportExport : Import hash context and Export back the context after hashing. eUnused : Context data import/export feature is not used. This option is also recommended for eHashSequence_d as eStartFinalizeHash or eTerminateHash. Output: • Successful API execution, Hash is returned in sOutHash only if eHashSequence_d is eStartFinalizeHash,eIntermediateHash or eFinalizeHash. Hash context data is returned only if sContextInfo_d::eContextAction is eExport or eImportExport. Notes: • Application on security chip must be opened using CmdLib_OpenApplication before using this operation. • eTerminateHash in eHashSequence_d is used to terminate any existing hash session. Any input data or hash context options supplied with this sequence is ignored. • Sequences for generating a hash successfully can be as follows: eStartHash, eFinalizeHash eStartHash, eContinueHash (single or multiple), eFinalizeHash eStartFinalizeHash eStartHash, eIntermediateHash, eContinueHash, eFinalizeHash

Enabler APIs

CmdLib_CalcHash	
	• If the memory buffer is not sufficient to store output hash/hash context or the data to be sent to security chip is more than communication buffer, CMD_LIB_INSUFFICIENT_MEMORY error is retured. • This operation does not maintain any state of hashing operations. • There is no support for chaining while sending data therefore in order to avoid communication buffer overflow, the caller must take care of fragmenting the data for hashing. • Use the operation CmdLib_GetMaxCommsBufferSize to check the maximum communication buffer size supported by the security chip. In addition, the overhead for command APDU header and TLV encoding must be considered as explained below. • Read the maximum communication buffer size using CmdLib_GetMaxCommsBufferSize and store in a variable "wMaxCommsBuffer" • Substract the header overheads and hash context size(depends on applicable Hash algorithm) respectively from **wMaxCommsBuffer**. The result gives the Available_Size to frame the hash data input. *Only hash calculation :* Available_Size = (wMaxCommsBuffer - CALC_HASH_FIXED_OVERHEAD_SIZE) *Import context to security chip and calculate hash :* Available_Size = (wMaxCommsBuffer - CALC_HASH_FIXED_OVERHEAD_SIZE - CALC_HASH_IMPORT_OR_EXPORT_OVERHEAD_SIZE - CALC_HASH_SHA256_CONTEXT_SIZE) *Calulate hash and export context out of security chip :* Available_Size = (wMaxCommsBuffer - CALC_HASH_FIXED_OVERHEAD_SIZE - CALC_HASH_IMPORT_OR_EXPORT_OVERHEAD_SIZE *Import context to security chip, calculate hash and export context out of security chip :* Available_Size = (wMaxCommsBuffer - CALC_HASH_FIXED_OVERHEAD_SIZE - CALC_HASH_IMPORT_AND_EXPORT_OVERHEAD_SIZE - CALC_HASH_SHA256_CONTEXT_SIZE)
Note	**Return values:** CMD_LIB_OK, CMD_LIB_ERROR, CMD_LIB_INVALID_PARAM, CMD_LIB_NULL_PARAM, CMD_LIB_INSUFFICIENT_MEMORY, CMD_DEV_ERROR, CMD_DEV_EXEC_ERROR
Signature	CmdLib_CalcHash (inout PpsCalcHash : sCalcHash_d*) : int32_t
Parameters	`inout PpsCalcHash : sCalcHash_d*` Pointer to the message from which calculating the hash.

3.1.8 CmdLib_OpenApplication

CmdLib_OpenApplication	
Description	CmdLib_OpenApplication opens the OPTIGA™ Trust X Application. The Unique Application Identifier is used internally by the function while forming the command APDU.
Note	Return values: CMD_LIB_OK, CMD_LIB_ERROR
Signature	CmdLib_OpenApplication (in PpsOpenApp : const sOpenApp_d*) : int32_t

Enabler APIs

CmdLib_OpenApplication	
Parameters	`in PpsOpenApp : const sOpenApp_d*` Pointer to the open application structure sOpenApp containing inputs for opening application on security chip.

3.1.9 CmdLib_GetDataObject

CmdLib_GetDataObject	
Description	CmdLib_GetDataObject reads data or metadata of the specified data object by issuing GetDataObject command based on input parameters. Notes: • Application on security chip must be opened using CmdLib_OpenApplication before using this operation. • The function does not verify if the read access is permitted for the data object.
Note	Return values: CMD_LIB_OK, CMD_LIB_ERROR, CMD_LIB_INSUFFICIENT_MEMORY, CMD_DEV_ERROR, CMD_LIB_NULL_PARAM
Signature	CmdLib_GetDataObject (in PpsGDVector : sGetData_d, inout PpsResponse : sCmdResponse_d) : int32_t
Parameters	`in PpsGDVector : sGetData_d` Pointer to Get Data Object inputs
	`inout PpsResponse : sCmdResponse_d` Pointer to Response structure

3.1.10 CmdLib_SetDataObject

CmdLib_SetDataObject	
Description	CmdLib_SetDataObject writes data or metadata to the specified data object by issuing SetDataObject command based on input parameters. Notes: • Application on security chip must be opened using CmdLib_OpenApplication before using this operation. • The function does not verify if the write access permitted for the data object. • While writing metadata, the metadata must be specified in an already TLV encoded byte array format. For example, to set LcsO to operational the value passed by the user must be 0x20 0x03 0xC0, 0x01, 0x07. • The function does not validate if the provided input data bytes are correctly formatted. For example, while setting LcsO to operational, function does not verify if the value is indeed 0x07. • In case of failure,it is possible that partial data is written into the data object. In such a case, the user should decide if the data has to be re-written.
Note	Return values: CMD_LIB_OK, CMD_LIB_ERROR, CMD_LIB_INVALID_PARAM, CMD_LIB_INSUFFICIENT_MEMORY, CMD_DEV_ERROR, CMD_LIB_NULL_PARAM
Signature	CmdLib_SetDataObject (in PpsSDVector : sSetData_d) : int32_t
Parameters	`in PpsSDVector : sSetData_d` Pointer to Set Data Object inputs

3.1.11 CmdLib_SetOptigaCommsContext

CmdLib_SetOptigaCommsContext	
Description	CmdLib_SetOptigaCommsContext sets the optiga_comms_ifx_i2c context provided by the caller.
Note	Return values: CMD_LIB_OK, CMD_LIB_ERROR, CMD_LIB_NULL_PARAM
Signature	CmdLib_SetOptigaCommsContext (inout p_input_optiga_comms : optiga_comms_t) : int32_t
Parameters	`inout p_input_optiga_comms : optiga_comms_t` Pointer to optiga_comms_ifx_i2c context.

3.1.12 CmdLib_GetRandom

CmdLib_GetRandom	
Description	CmdLib_GetRandom returns random bytes generated by OPTIGA™ Trust X Security Chip. Notes: • Application on security chip must be opened using CmdLib_OpenApplication before using this operation. • Command chaining is not supported in this operation. • If the requested length of random bytes is either more than communication buffer size or more than the buffer size in PpsResponse, CMD_LIB_INSUFFICIENT_MEMORY error is returned.
Note	Return values: CMD_LIB_OK, CMD_LIB_ERROR, CMD_LIB_INSUFFICIENT_MEMORY, CMD_LIB_LENZERO_ERROR, CMD_DEV_ERROR, CMD_LIB_NULL_PARAM
Signature	CmdLib_GetRandom (in PpsRng : sRngOptions_d, inout PpsResponse : sCmdResponse_d) : int32_t
Parameters	`in PpsRng : sRngOptions_d` Pointer to sRngOptions_d to specify random number generation
	`inout PpsResponse : sCmdResponse_d` Pointer to sCmdResponse_d to store random number

3.1.13 CmdLib_GetSignature

CmdLib_GetSignature	
Description	CmdLib_GetSignature returns the signature generated by OPTIGA™ Trust X Security Chip. The message to be signed is provided by the caller. The following commands are issued in the sequence. • SetAuthScheme : To set authentication scheme and the private key to be used • SetAuthMsg : To write the message to Security Chip that must be digitally signed. • GetAuthMsg : To read the digitally signed message from Security Chip. Notes: • Application on security chip must be opened using CmdLib_OpenApplication before using this operation. • The operation just returns the signature without verifying it. • The private key to be used in set auth scheme is passed in sAuthMsg_d::wOIDDevPrivKey.

Enabler APIs

CmdLib_GetSignature	
	• The sAuthMsg_d::prgbRnd and sAuthMsg_d::wRndLength carry the challenge to be signed. • The length of challenge should be between 8 and 256 bytes. If the length of challenge is out of this range, CMD_LIB_INVALID_LEN error is returned.
Note	Return values: CMD_LIB_OK, CMD_LIB_ERROR, CMD_DEV_ERROR, CMD_LIB_INSUFFICIENT_MEMORY, CMD_LIB_NULL_PARAM
Signature	CmdLib_GetSignature (in PpsAuthMsg : sAuthMsg_d, inout PpsResponse : sCmdResponse_d) : int32_t
Parameters	`in PpsAuthMsg : sAuthMsg_d` Pointer to Get Signature Object inputs
	`inout PpsResponse : sCmdResponse_d` Pointer to Response structure

3.1.14 CmdLib_SetAuthScheme

CmdLib_SetAuthScheme	
Description	CmdLib_SetAuthScheme sets the Authentication Scheme by issuing SetAuthScheme command to OPTIGA™ Trust X Security Chip. Notes: • Application on security chip must be opened using CmdLib_OpenApplication before using this operation. • Currently only session OID (0xE100) is supported by the security chip.
Note	Return values: CMD_LIB_OK, CMD_LIB_ERROR, CMD_LIB_INVALID_PARAM, CMD_LIB_INSUFFICIENT_MEMORY, CMD_DEV_ERROR, CMD_LIB_NULL_PARAM
Signature	CmdLib_SetAuthScheme (in PpsAuthVector : sAuthScheme_d) : int32_t
Parameters	`in PpsAuthVector : sAuthScheme_d` Pointer to Authentication Scheme data

3.1.15 CmdLib_VerifySign

CmdLib_VerifySign	
Description	CmdLib_VerifySign verifies the signature over the input digest by using the Security chip. Input: • For eVerifyDataType eDataStream indicates that sPubKeyInput is considered for signature verification. eOIDData indicates that wOIDPubKey is considered for signature verification. Output: • Successful signature verification returns CMD_LIB_OK. Notes: • Application on security chip must be opened using CmdLib_OpenApplication before using this operation. • If the the data to be sent to security chip is more than communication buffer, CMD_LIB_INSUFFICIENT_MEMORY is returned.
Note	**Return values:** CMD_LIB_OK, CMD_LIB_ERROR, CMD_LIB_NULL_PARAM,

Enabler APIs

CmdLib_VerifySign	
	CMD_LIB_INSUFFICIENT_MEMORY, CMD_DEV_EXEC_ERROR, CMD_DEV_ERROR
Signature	CmdLib_VerifySign (in PpsVerifySign : sVerifyOption_d const*, inout PpsDigest : sbBlob_d const*, inout PpsSignature : sbBlob_d const*) : int32_t
Parameters	`in PpsVerifySign : sVerifyOption_d const*` Pointer to information for verifying signature
	`inout PpsDigest : sbBlob_d const*` pointer to a blob which holds the digest
	`inout PpsSignature : sbBlob_d const*` pointer to a blob which holds the Signature to be verified

3.1.16 CmdLib_GenerateKeyPair

CmdLib_GenerateKeyPair	
Description	CmdLib_GenerateKeyPair generates a key pair by issuing GenKeyPair command to Security chip. Input: • Provide the required option for exporting the generated keys. Use sKeyPairOption_d::eKeyExport eStorePrivKeyOnly indicates that only private key is stored in the OID and public key is exported. eExportKeyPair indicates that both public and private keys are exported. Output: • Successful execution, Public key is returned in sOutKeyPair_d::sPublicKey. Private key is returned in sOutKeyPair_d::sPrivateKey, if input is eExportKeyPair. Notes: • Application on security chip must be opened using CmdLib_OpenApplication before using this operation. • Values of eKeyUsage_d can be logically 'ORed' and passed to sKeyPairOption_d::eKeyUsage. • If the memory buffers in sOutKeyPair_d is not sufficient to store the generated keys, CMD_LIB_INSUFFICIENT_MEMORY is returned.
Note	**Return values:** CMD_LIB_OK, CMD_LIB_ERROR, CMD_LIB_NULL_PARAM, CMD_LIB_INSUFFICIENT_MEMORY, CMD_DEV_EXEC_ERROR, CMD_DEV_ERROR
Signature	CmdLib_GenerateKeyPair (in PpsKeyPairOption : sKeyPairOption_d const*, inout PpsOutKeyPair : sOutKeyPair_d) : int32_t
Parameters	`in PpsKeyPairOption : sKeyPairOption_d const*`
	`inout PpsOutKeyPair : sOutKeyPair_d`

3.1.17 CmdLib_CalculateSign

CmdLib_CalculateSign	
Description	CmdLib_CalculateSign calculates signature on a digest by using the Security Chip. Input:

Enabler APIs

CmdLib_CalculateSign	
	• Provide the signature scheme. Use sCalcSignOptions_d::eSignScheme. • Provide the digest to be signed. Use sCalcSignOptions_d::sDigestToSign. • Provide the OID of the private key. Use sCalcSignOptions_d::wOIDSignKey. Output: • Successful execution, Signature is returned in PpsSignature. Notes: • Application on security chip must be opened using CmdLib_OpenApplication before using this operation. • If the the data to be sent to security chip is more than communication buffer, CMD_LIB_INSUFFICIENT_MEMORY is returned. • If the memory buffer in PpsSignature is not sufficient to store the generated signature, CMD_LIB_INSUFFICIENT_MEMORY is returned.
Note	**Return values:** CMD_LIB_OK, CMD_LIB_ERROR, CMD_LIB_NULL_PARAM, CMD_LIB_INSUFFICIENT_MEMORY, CMD_DEV_EXEC_ERROR, CMD_DEV_ERROR
Signature	CmdLib_CalculateSign (in PpsCalcSign : sCalcSignOptions_d const*, inout PpsSignature : sbBlob_d *) : int32_t
Parameters	`in PpsCalcSign : sCalcSignOptions_d const*`
	`inout PpsSignature : sbBlob_d *`

3.1.18 CmdLib_CalculateSharedSecret

CmdLib_CalculateSharedSecret	
Description	CmdLib_CalculateSharedSecret calculates a shared secret by using the security chip. Input: • Provide the key agreement algorithm for generating shared secret. Use sCalcSSecOptions_d::eKeyAgreementType. • Provide the OID of private key. Use sCalcSSecOptions_d::wOIDPrivKey. • Provide the algorithm identifier of the public key. Use sCalcSSecOptions_d::ePubKeyAlgId. • Provide the public key. Use sCalcSSecOptions_d::sPubKey. • Provide the OID to store the shared secret. Use sCalcSSecOptions_d::wOIDSharedSecret. 0x0000 indicates that the shared secret is exported. Output: • Successful execution, Calculated shared secret is returned in PpsSecret if sCalcSSecOptions_d::wOIDSharedSecret is 0x0000. Notes: • Application on security chip must be opened using CmdLib_OpenApplication before using this operation. • If the the data to be sent to security chip is more than communication buffer, CMD_LIB_INSUFFICIENT_MEMORY is returned. • If the memory buffer in PpsSecret is not sufficient to store the calculated secret, CMD_LIB_INSUFFICIENT_MEMORY is returned.
Note	**Return values:** CMD_LIB_OK, CMD_LIB_ERROR, CMD_LIB_NULL_PARAM,

Enabler APIs

CmdLib_CalculateSharedSecret	
	CMD_LIB_INSUFFICIENT_MEMORY, CMD_DEV_EXEC_ERROR, CMD_DEV_ERROR
Signature	CmdLib_CalculateSharedSecret (in PpsCalcSSec : sCalcSSecOptions_d const*, inout PpsSecret : sbBlob_d*) : int32_t
Parameters	`in PpsCalcSSec : sCalcSSecOptions_d const*` Pointer to sCalcSSecOptions_d to provide input for shared secret calculation
	`inout PpsSecret : sbBlob_d*` Pointer to sbBlob_d that contains calculated shared secret

3.1.19 CmdLib_DeriveKey

CmdLib_DeriveKey	
Description	CmdLib_DeriveKey derives a key using the Security Chip for the host requested key derivation algorithm. Input: • Provide the key derivation method. Use sDeriveKeyOptions_d::eKDM. • Provide the OID of the shared secret. Use sDeriveKeyOptions_d::wOIDSharedSecret. • Provide the input seed. Use sDeriveKeyOptions_d::sSeed. • Provide the length for derived key. Use sDeriveKeyOptions_d::wDerivedKeyLen. • Provide the OID to store the derived key. Use sDeriveKeyOptions_d::wOIDDerivedKey. 0x0000 indicates that the derived key is exported. Output: • Successful execution, Derived key is returned in PpsKey if sDeriveKeyOptions_d::wOIDDerivedKey is 0x0000. Notes: • Application on security chip must be opened using CmdLib_OpenApplication before using this operation. • If the the data to be sent to security chip is more than communication buffer, CMD_LIB_INSUFFICIENT_MEMORY is returned. • If the memory buffer in PpsKey is not sufficient to store the derived key, CMD_LIB_INSUFFICIENT_MEMORY is returned.
Note	**Return values:** CMD_LIB_OK, CMD_LIB_ERROR, CMD_LIB_NULL_PARAM, CMD_LIB_INSUFFICIENT_MEMORY, CMD_DEV_EXEC_ERROR, CMD_DEV_ERROR
Signature	CmdLib_DeriveKey (in PpsDeriveKey : sDeriveKeyOptions_d const*, inout PpsKey : sbBlob_d *) : int32_t
Parameters	`in PpsDeriveKey : sDeriveKeyOptions_d const*` Pointer to sDeriveKeyOptions_d to provide inputs for key derivation
	`inout PpsKey : sbBlob_d *`

Enabler APIs

CmdLib_DeriveKey	
	Pointer to sbBlob_d that contains derived key

3.2 CryptoLib

The CryptoLib wraps the Cryptographic Library specific API to a neutral API to enable multiple sourcing of a 3rd Party Crypto Lib. This allows reuse of the components of the OPTIGA™ Trust X solution, which are consuming cryptographic functionalities. The CryptoLib exports only those API functions which are required by the OPTIGA™ Trust X solution.

3.2.1 CryptoLib_GenerateSeed

CryptoLib_GenerateSeed	
Description	CryptoLib_GenerateSeed generates seed for initializing the crypto pseudo random generator. • Reads 24 bytes of random data bytes from and stores it. The data bytes are read only once. • A subsequent requests for seed generation uses the stored random data bytes. • Concatenates counter to the random bytes and computes HASH (SHA256) on the concatenated value. • Maximum length of generated seed is 32 bytes.
Signature	CryptoLib_GenerateSeed (inout prgbSeed : puint8_t, in dwSeedLength : uint32_t) : int32_t
Parameters	`inout prgbSeed : puint8_t` Pointer to the seed
	`in dwSeedLength : uint32_t` Length of the seed

3.2.2 CryptoLib_GetRandom

CryptoLib_GetRandom	
Description	CryptoLib_GetRandom generates random data bytes of requested length. • If the requested length is less than 32 bytes, random data bytes of 32 bytes, will be generated and the requested number of bytes will be returned. • The crypto random number generator is initialized and seeded only once. • For subsequent seeding, the first 32 bytes of the generated random data bytes will be fed back as seed.
Signature	CryptoLib_GetRandom (in wRandomDataLen, inout psResponse : sCmdResponse_d*) : int32_t
Parameters	`in wRandomDataLen` Number of random bytes requested
	`inout psResponse : sCmdResponse_d*` Pointer to response structure

3.2.3 CryptoLib_ParseCertificate

CryptoLib_ParseCertificate	
Description	CryptoLib_ParseCertificate parses raw X509 v3 certificate into a custom defined certificate structure.

Enabler APIs

CryptoLib_ParseCertificate	
	• The raw certificate must be in DER encoded format. • The function does not allocate any memory. • Stores reference address/location from raw certificate data into the custom defined certificate structure. • The following details are parsed from the raw certificate : Public Key, Signature, Certificate Data
Signature	CryptoLib_ParseCertificate (in psRawCertificate : const sbBlob_d*, inout psCertificate : sCertificate_d *) : int32_t
Parameters	`in psRawCertificate : const sbBlob_d*` Pointer to structure containing raw certificate data and its length
	`inout psCertificate : sCertificate_d *` Structure which holds parsed certificate data

3.2.4 CryptoLib_VerifySignature

CryptoLib_VerifySignature	
Description	CryptoLib_VerifySignature verifies the signature using the given public key.
Signature	CryptoLib_VerifySignature (in psSignatureVector : const sSignatureVector_d *) : int32_t
Parameters	`in psSignatureVector : const sSignatureVector_d *` Pointer to structure which holds data for signature verification

3.3 IntegrationLib

IntegrationLib mainly implements use cases which user can use for reference, for eg. one-way authentication is a use case which is made up of several commands implemented in command lib.

3.3.1 IntLib_ReadGpData

IntLib_ReadGpData	
Description	IntLib_ReadGpData reads the specified general purpose data object from the OPTIGA™ Trust X Security Chip. The function performs the following steps while reading the data object, • Reads the Application Life Cycle Status(LcsA) and Global Life Cycle Status(LcsG) • Reads the metadata of the data object. • Verifies the read access conditions of the data object. • Reads the data object, if read access is permitted.
Note	Return values: INT_LIB_OK, INT_LIB_NULL_PARAM, INT_LIB_INVALID_RESPONSE, INT_LIB_INVALID_AC, INT_LIB_ZEROLEN_ERROR, INT_LIB_ERROR, CMD_DEV_ERROR
Signature	IntLib_ReadGpData (in PpsGDVector : const sReadGPData_d*, inout PpsGPData : sbBlob_d *) : int32_t
Parameters	`in PpsGDVector : const sReadGPData_d*` Pointer to Get Data parameters
	`inout PpsGPData : sbBlob_d *`

Enabler APIs

IntLib_ReadGpData	
	Pointer to data buffer for response

3.3.2 IntLib_WriteGpData

IntLib_WriteGpData	
Description	IntLib_WriteGpData writes to the specified general purpose data object to the OPTIGA™ Trust X Security Chip. The function performs the following steps while writing to the data object, • Reads the Application Life Cycle Status(LcsA) and Global Life Cycle Status(LcsG) • Reads the metadata of the data object. • Verifies the write access conditions of the data object. • Writes to the data object, if write access is permitted.
Note	Return values: INT_LIB_OK, INT_LIB_NULL_PARAM, INT_LIB_INVALID_RESPONSE, INT_LIB_INVALID_AC, INT_LIB_ERROR, CMD_DEV_ERROR
Signature	IntLib_WriteGpData (in PpsSDVector : const sWriteGPData_d*) : int32_t
Parameters	`in PpsSDVector : const sWriteGPData_d*` Pointer to set data parameters

3.3.3 IntLib_Authenticate

IntLib_Authenticate	
Description	IntLib_Authenticate performs One Way Authentication to prove the authenticity of the device which incorporates OPTIGA™ Trust X Security Chip. The operation performs one way authentication in the following way: • Reads the Device Certificate from OPTIGA™ Trust X Security Chip specified by sOneWayAuth_d::wOIDDevCertificate • Verifies the Device Certificate signature using the Trust Anchor public key that is available within the CA certificate parameter sOneWayAuth_d::sCaCert. • Random number of the length specified by sOneWayAuth_d::wChallengeLen is generated on Host which is used as a message that will be sent to OPTIGA™ Trust X Security Chip. • Issues SetAuthScheme APDU command based on the private key provided in sOneWayAuth_d::wOIDDevPrivKey • Issues SetAuthMsg and GetAuthMsg APDU commands to Security Chip to get signature on the challenge • Verifies the signature using the public key extracted from the device certificate Notes: • CA certificate must be provided in DER encoded binary format. • The current implementation is based on ECC NIST P 256 bit key length. • The wChallengeLen must range from 8 to 256 bytes. It is recommended to use a minimum challenge length of 16 bytes. If the length is out of this range, INT_LIB_INVALID_LENGTH error is returned. • This API supports device certificate objects in "One-Way Authentication" and "TLS" identity format only. Identity validation failures will return INT_LIB_INVALID_CERTIFICATE_FORMAT error. • For TLS identity, certificates chaining must be encoded as per RFC-5246. • Under some erroneous conditions, error codes from Command Library and

Enabler APIs

IntLib_Authenticate	
	crypto Library can also be returned. • If the return code is CMD_DEV_EXEC_ERROR, it might indicate that the application on the security chip is either closed or a reset has occurred. In such a case, user must invoke CmdLib_OpenApplication before attempting any interaction with the security chip.
Note	Return values: INT_LIB_OK, INT_LIB_ERROR, INT_LIB_ZEROLEN_ERROR, INT_LIB_INVALID_PARAM, INT_LIB_NULL_PARAM, CMD_DEV_ERROR
Signature	IntLib_Authenticate (in PpsOneWayAuth : sOneWayAuth_d const*) : int32_t
Parameters	`in PpsOneWayAuth : sOneWayAuth_d const*`

3.4 OCP

OCP (**O**PTIGA **C**rypto & **P**rotected Comm Library) is the module providing application the interface to the cryptographic functionalities. OCP fulfills the cryptographic functionalities by calling the implementations of the underlying components.

Cryptographic functionalities are implemented in software or in hardware. The software implementation of cryptographic functionalities is supplied by 3rd party. The hardware implementation is provided by OPTIGA™ Trust X. Currently only HW crypto is supported.

For DTLS the OCP implements the Handshake state machine. The outgoing messages of Handshake layer are generated by OPTIGA™ Trust X. The incoming messages from server are processed by OPTIGA™ Trust X. The record layer is implemented in OCP. However the encryption and decryption functionalities are fulfilled by OPTIGA™ Trust X.

Additionally for DTLS, Windowing, Fragmentation and De-Fragmentation of handshake messages, time-out and retransmission of messages are implemented in OCP module.

OCP interacts with pal to send and receive DTLS records.

3.4.1 OCP_Connect

OCP_Connect	
Description	OCP_Connect connects to the server and performs a DTLS handshake protocol as per DTLS v1.2 Pre-conditions: • OCP_Init was successful and application context is available. • Server trust anchor must be available in the security chip. Details: • Connects to the server via the pal. • Invokes CmdLib_SetAuthScheme based on configuration. • Performs a DTLS Handshake. Input: • Caller must provide a valid PhAppOCPCtx handle otherwise OCP_LIB_SESSIONID_UNAVAILABLE is returned. Notes: • The default value of timeout for retransmission must be 2 seconds on the server side. • If a connection already exists on the given port and IP address, OCP_LIB_CONNECTION_ALREADY_EXISTS is returned. • Under some failure conditions, error codes from lower layers could also be returned.

Enabler APIs

OCP_Connect	
	• In case of a Failure other than OCP_LIB_CONNECTION_ALREADY_EXISTS and OCP_LIB_SESSIONID_UNAVAILABLE The Session gets closed automatically. The memory allocated in OCP_Init is freed. OCP handle will not be set to NULL. It is up to the user to check return code and take appropriate action. • If the return value is CMD_DEV_EXEC_ERROR, it might indicate that the application on the security chip is either closed or a reset has occurred.
Note	Return values: OCP_LIB_OK, OCP_LIB_ERROR, OCP_LIB_NULL_PARAM, OCP_LIB_CONNECTION_ALREADY_EXISTS
Signature	OCP_Connect (in PhAppOCPCtx : hdl_t) : int32_t
Parameters	`in PhAppOCPCtx : hdl_t` Handle to OCP context

3.4.2 OCP_Send

OCP_Send	
Description	OCP_Send sends application data to the DTLS server. Pre-conditions: • OCP_Connect was successful and application context is available. Details: • Sends application data to DTLS server. • Application data is sent only if the use case Mutual Authentication Public Key Scheme (DTLS) was successfully performed. • Encryption of the application data is done at the record layer. Input: • Caller must provide a valid PhAppOCPCtx handle. • Caller must provide the data to be sent and its length If the length of the data to be sent is greater than MAX_APP_DATALEN (PhAppOCPCtx), then OCP_LIB_INVALID_LEN is returned. If the length of the data to be sent is equal to zero, then OCP_LIB_LENZERO_ERROR is returned. Notes: • The maximum length of data that can be sent by the operation depends upon the PMTU value set during OCP_Init (This length can be obtained by MAX_APP_DATALEN (PhAppOCPCtx). • Fragmentation of data to be sent should be done by the application. This operation does not perform data fragmentation. • If the record sequence number has reached maximum value for epoch 1, then OCP_RL_SEQUENCE_OVERFLOW error is returned. User must call OCP_Disconnect in this condition. No Alert will be sent due to the unavailability of record sequence number. • Under some failure conditions, error codes from lower layers could also be returned. • In case of a Failure, Existing session remains open and memory allocated during OCP_Init is not freed. PhAppOCPCtx handle is not set to NULL. The operation does not send any alert to the server. • If the return value is CMD_DEV_EXEC_ERROR, it might indicate that the application on the security chip is either closed or a reset has occurred. In

Enabler APIs

OCP_Send	
	such a case, close the existing DTLS session using OCP_Disconnect.
Note	Return values: OCP_LIB_OK, OCP_LIB_ERROR, OCP_LIB_NULL_PARAM, OCP_LIB_SESSIONID_UNAVAILABLE, OCP_LIB_AUTHENTICATION_NOTDONE, OCP_LIB_MALLOC_FAILURE, OCP_LIB_LENZERO_ERROR, OCP_LIB_INVALID_LEN, OCP_RL_SEQUENCE_OVERFLOW
Signature	OCP_Send (in PhAppOCPCtx : hdl_t, in PprgbData : puint8_t, in PwLen : uint16_t) : int32_t
Parameters	`in PhAppOCPCtx : hdl_t` Handle to OCP context
	`in PprgbData : puint8_t` Pointer to data to be send
	`in PwLen : uint16_t` Length of data to be send

3.4.3 OCP_Receive

OCP_Receive	
Description	OCP_Receive receives application data from the DTLS server. Pre-conditions: • OCP_Connect was successful and application context is available. Details: • Receives application data from the DTLS server. • Application data is received only if the use case Mutual Authentication Public Key Scheme (DTLS) was successfully performed. • Received data is assumed to be encrypted and processed accordingly. Decryption of the application data is done at the record layer. • Total received application data length is updated in PpwLen. Input: • Caller must provide a valid PhAppOCPCtx handle. • Caller must provide the buffer where application data should be returned. • Caller must provide the length of the buffer. If the length of the buffer is equal to zero, then OCP_LIB_LENZERO_ERROR is returned. • Caller must provide the timeout value in milliseconds. The value should be greater than 0 and maximum up to (2^16)-1. If the timeout is zero OCP_LIB_INVALID_TIMEOUT is returned. Notes: • The maximum length of data that can be received by the API depends upon the PMTU value set during OCP_Init. This length is indicated by MAX_APP_DATALEN (PhAppOCPCtx). • If required, the re-assembly of received data should be done by the application. This operation does not perform data re-assembly. • Failure in decrypting data will return OCP_LIB_DECRYPT_FAILURE. • If a fatal alert with valid description is received, OCP_AL_FATAL_ERROR is returned. User must invoke OCP_Disconnect in this condition. Invoking OCP_Send or OCP_Receive will return OCP_LIB_OPERATION_NOT_ALLOWED. • If a valid Hello request is received, the operation internally sends a warning

Enabler APIs

OCP_Receive	
	alert with description "no-renegotiation" to the server and then waits for data until timeout occurs. • If the length of buffer provided by the caller is not sufficient to return received data, OCP_LIB_INSUFFICIENT_MEMORY is returned. This data will not be returned in subsequent invocation. • If timeout occurs, OCP_LIB_TIMEOUT is returned. • Under some failure conditions, error codes from lower layers could also be returned. • In case of a Failure, Existing session remains open and memory allocated during OCP_Init is not freed. PhAppOCPCtx handle is not set to NULL. The operation does not send any alert to the server. PpwLen is set to zero. • If the return value is CMD_DEV_EXEC_ERROR, it might indicate that the application on the security chip is either closed or a reset has occurred. In such a case, close the existing DTLS session using OCP_Disconnect.
Note	Return values: OCP_LIB_OK, OCP_LIB_ERROR, OCP_LIB_NULL_PARAM, OCP_LIB_SESSIONID_UNAVAILABLE, OCP_LIB_AUTHENTICATION_NOTDONE, OCP_LIB_MALLOC_FAILURE, OCP_LIB_LENZERO_ERROR, OCP_AL_FATAL_ERROR, OCP_LIB_INSUFFICIENT_MEMORY, OCP_LIB_INVALID_TIMEOUT, OCP_LIB_TIMEOUT
Signature	OCP_Receive (in PhAppOCPCtx : hdl_t, inout PprgbData : uint8_t, inout PpwLen : uint16_t, in PwTimeout : uint16_t) : int32_t
Parameters	`in PhAppOCPCtx : hdl_t` Handle to OCP context
	`inout PprgbData : uint8_t` Pointer to buffer where data is to be received
	`inout PpwLen : uint16_t` Pointer to the Length of buffer. Updated with actual length of received data
	`in PwTimeout : uint16_t` Timeout in milliseconds

3.4.4 OCP_Init

OCP_Init	
Description	OCP_Init initializes the OCP context based on the user inputs provided in PpsAppOCPConfig. Once the OCP context is initialised, PphAppOCPCtx is returned as a handle. PphAppOCPCtx handle is used for all interactions with the OCP operations OCP_Connect, OCP_Disconnect, OCP_Send and OCP_Receive. Pre-conditions: • Communication with the security chip is up and running. • optiga_comms_open must be successfully executed. • The optiga_comms context for command library is registered using CmdLib_SetOptigaCommsContext. Details: • Checks for an available session OID. • Opens application on the security chip using CmdLib_OpenApplication.

Enabler APIs

OCP_Init	
	• Allocates the memory for internal structures, initializes it and returns as a hdl_t* Input: • The caller must provide configuration information in sAppOCPConfig_d • eMode allows the caller to configure the OCP context as a client/server as per eMode_d. Currently server mode is not supported. • eConfiguration allows the caller to choose supported OCP context configuration as per eConfiguration_d. Currently eTLS_12_TCP_HWCRYPTO is not supported. • wOIDDevCertificate allows the caller to choose the supported certificate for client authentication. If there is no client certificate then set it to 0x0000. • wOIDDevPrivKey allows the caller to choose the private key used for client authentication. • psNetworkParams allows the caller to configure the port, IP Address and maximum PMTU required for transport layer connection. • Valid IP address and port number must be provided. The correctness of the IP address and port number will not be verified. • PMTU value should range from 296 to 1500, else OCP_LIB_UNSUPPORTED_PMTU error is returned. • Logger allows user to log data. The caller must provide the low level log writer through sLogger_d. • pfGetUnixTIme (fGetUnixTime_d) is a call-back function pointer that allows caller to provide 32-bit Unix time format. • If pfGetUnixTIme is set to NULL, the unix time will not be sent to security chip. • If pfGetUnixTIme is not set to NULL or valid function pointer, the behavior would be unexpected. • The call back function pfGetUnixTIme is expected to return status s as CALL_BACK_OK for success. Notes: • Currently, only 1 DTLS session is supported by security chip. • If user invokes OCP_Init, without disconnecting/closing the previous session/context (if available), will lead to error OCP_LIB_SESSIONID_UNAVAILABLE. • Under some failure conditions, error codes from lower layers could also be returned.
Note	Return values: OCP_LIB_OK, OCP_LIB_ERROR, OCP_LIB_NULL_PARAM, OCP_LIB_UNSUPPORTED_CONFIG, OCP_LIB_SESSIONID_UNAVAILABLE, OCP_LIB_UNSUPPORTED_PMTU, OCP_LIB_UNSUPPORTED_CERTIFICATE
Signature	OCP_Init (in PpsAppOCPConfig : sAppOCPConfig_d, inout PphAppOCPCtx : hdl_t) : int32_t
Parameters	`in PpsAppOCPConfig : sAppOCPConfig_d` Application configuration
	`inout PphAppOCPCtx : hdl_t` Handle to OCP context

3.4.5 OCP_Disconnect

OCP_Disconnect	
Description	OCP_Disconnect disconnects from server and closes the DTLS session.

Enabler APIs

OCP_Disconnect	
	Pre-conditions: • OCP_Init or OCP_Connect was successful. Details: • Applicable only if called after a successful OCP_Connect Closes DTLS session on security chip via CmdLib_CloseSession. Sends closure alert to the server via pal. Disconnects from the server via pal. • Applicable if called after successful OCP_Init or OCP_Connect Clear memory associated with PhAppOCPCtx handle. Clears the internal session reference Id registry. PhAppOCPCtx handle will not be set to NULL. It is up to the caller to check return code and take appropriate action. Input: • Caller must provide a valid PhAppOCPCtx handle. Notes: • If the record sequence number has reached maximum value for epoch 1, No Alert will be send due to the unavailability of record sequence number. • DTLS server will not be notified of the fault condition that leads to failure while sending Close_Notify alert.
Note	Return values: OCP_LIB_OK, OCP_LIB_NULL_PARAM, OCP_LIB_ERROR
Signature	OCP_Disconnect (in PhAppOCPCtx : hdl_t) : int32_t
Parameters	`in PhAppOCPCtx : hdl_t` Handle to OCP context

3.5 optiga_comms_ifx_i2c

I2C is the interface used to communicate between Host and OPTIGA™ Trust X. The application running on Host generates data in the form of APDUs to communicate with OPTIGA™ Trust X. Size of APDUs varies between few bytes to kilo bytes. IFX_I2C protocol handles this huge data transaction generated by different applications. The protocol implementation is done in multiple layers and seamlessly handles data transfer from Host to OPTIGA™ Trust X and OPTIGA™ Trust X to Host. More details of IFX I2C protocol can be found in [IFX_I2C].

3.5.1 optiga_comms_close

optiga_comms_close	
Description	optiga_comms_close closes the IFX I2C protocol stack for the given context. Details: • A successful optiga_comms_open must be done before using this operation. • De-Initializes the I2C slave device. • Power downs the I2C slave. User Input: • The input p_ctx must not be NULL.
Note	Return values: OPTIGA_COMMS_SUCCESS, OPTIGA_COMMS_ERROR
Signature	optiga_comms_close (in p_ctx : optiga_comms_t)
Parameters	`in p_ctx : optiga_comms_t` Pointer to protocol context

3.5.2 optiga_comms_open

optiga_comms_open	
Description	optiga_comms_open initializes optiga_comms_ifx_i2c and provides a handle. Details: • Performs a reset sequence. • Initializes the I2C slave device. • Initializes the ifx i2c protocol stack and registers the event callbacks. • Negotiates the frame size and bit rate with the I2C slave. User Input: • The context pointer p_ctx must not be NULL. • The following parameters in p_ctx must be initialized with appropriate values ***slave address:*** Address of I2C slave ***frame_size:*** Frame size in bytes. Minimum supported value is 16 bytes. It is recommended not to use a value greater than the slave's frame size. The user specified frame size is written to I2C slave's frame size register.The frame size register is read back from I2C slave. This frame value is used by the ifx-i2c protocol even if it is not equal to the user specified value. ***frequency:*** Frequency/speed of I2C master in KHz. This must be lowest of the maximum frequency supported by the devices (master/slave) connected on the bus. Initial negotiation starts with a frequency of 100KHz. If the user specified frequency is more than 400 KHz, the I2C slave is configured to operate in "Fm+" mode, otherwise the I2C slave is configured for "SM & Fm" mode. If the user specified frequency frequency negotiation fails, the I2C master frequency remains at 100KHz ***upper_layer_event_handler:*** Upper layer event handler. This is invoked when optiga_comms_open is asynchronously completed. ***upper_layer_ctx:*** Context of upper layer. ***p_slave_vdd_pin:*** GPIO pin for VDD. If not set, cold reset is not done. ***p_slave_reset_pin:*** GPIO pin for Reset. If not set, warm reset is not done. Notes: • The values of registers MAX_SCL_FREQU and DATA_REG_LEN, read from slave are not validated.
Note	Return values: OPTIGA_COMMS_SUCCESS, OPTIGA_COMMS_ERROR
Signature	optiga_comms_open (in p_ctx : optiga_comms_t)
Parameters	`in p_ctx : optiga_comms_t` Pointer to protocol context

3.5.3 optiga_comms_reset

optiga_comms_reset	
Description	optiga_comms_reset resets the I2C slave and initializes the IFX I2C protocol stack for a given context. Details: • A successfull optiga_comms_open must be done before using this operation. • Resets the I2C slave. • Initializes the ifx i2c protocol stack. • Re-Initializes and negotiates the frame size and bit rate with the I2C slave.The values remain same as that in previous optiga_comms_open. Input: • The context pointer p_ctx must not be NULL. Notes: • For COLD and WARM reset type: If the gpio(vdd and/or reset) pins are not configured, the operation continues without any failure return status.
Note	Return values: OPTIGA_COMMS_SUCCESS, OPTIGA_COMMS_ERROR
Signature	optiga_comms_reset (in p_ctx : optiga_comms_t, inout reset_type : uint8_t)
Parameters	`in p_ctx : optiga_comms_t` Pointer to protocol context
	`inout reset_type : uint8_t` type of reset

3.5.4 optiga_comms_transceive

optiga_comms_transceive	
Description	optiga_comms_transceive used to send and receive data over I2C. Details: • A successful optiga_comms_open must be done before using this operation. • Transmit data(Command) to I2C slave. • Receive data(Response) from I2C slave. Input: • The context pointer p_ctx must not be NULL. • The following parameters in p_ctx must be initialized with appropriate values ***upper_layer_event_handler:*** Upper layer event handler, if it is different from that in optiga_comms_open. This is invoked when optiga_comms_transceive is asynchronously completed. ***upper_layer_ctx:*** Context of upper layer, if it is different from that in optiga_comms_transceive. Notes: • The actual number of bytes received is stored in p_rx_buffer_len. In case of error, p_rx_buffer_len is set to 0. • If the size of prgbReadBuffer is zero or insufficient to copy the response bytes then IFX_I2C_STACK_MEM_ERROR error is returned.
Note	Return values: OPTIGA_COMMS_SUCCESS, OPTIGA_COMMS_ERROR, IFX_I2C_MEM_ERROR
Signature	optiga_comms_transceive (in p_ctx : optiga_comms_t, in p_data : puint8_t, in p_data_length : puint16_t, inout p_buffer : puint8_t, inout p_buffer_len : puint16_t)

Enabler APIs

optiga_comms_transceive	
Parameters	`in p_ctx : optiga_comms_t` Pointer to protocol context
	`in p_data : puint8_t` Pointer to buffer to send data buffer
	`in p_data_length : puint16_t` Pointer to length of send data in buffer
	`inout p_buffer : puint8_t` Pointer to buffer to receive data
	`inout p_buffer_len : puint16_t` Pointer to length of receive data buffer

4 OPTIGA™ Trust X External Interface

The chapter OPTIGA™ Trust X External Interface provides the detailed definition of the OPTIGA™ Trust X commands available at its I/O interface.

4.1 Warm Reset

The Warm Reset (reset w/o power off/on cycle) of the OPTIGA™ Trust X might be triggered either by HW signal or by SW. In case of a HW triggered Warm Reset the RST pin must be set to low (for more details refer to [Data Sheet]). In case of a SW triggered Warm Reset the I2C master must write to the SOFT_RESET register (for more details refer to [IFX_I2C]).

4.2 Power Consumption

When operating, the power consumption of OPTIGA™ Trust X is limited to meet the requirements regarding the power limitation set by the Host. The power limitation is implemented by utilizing the current limitation feature of the underlying HW device in steps of 1 mA from 6mA to 15 mA with a precision of ±5% (for more details refer to Current limitation).

4.2.1 Low Power Sleep Mode

The OPTIGA™ Trust X automatically enters a low-power mode after a configurable delay. Once it has entered Sleep mode, the OPTIGA™ Trust X resumes normal operation as soon as its address is detected on the I2C bus.
In case no command is sent to the OPTIGA™ Trust X it behaves as shown in Figure "Go-to-Sleep diagram".
(1) As soon as the OPTIGA™ Trust X is idle it starts to count down the “delay to sleep” time (t_{SDY}).
(2) In case this time elapses the device enters the “go to sleep” procedure.
(3) The “go to sleep” procedure waits until all idle tasks are finished (e.g. counting down the SEC). In case all idle tasks are finished and no command is pending, the OPTIGA™ Trust X enters sleep mode.

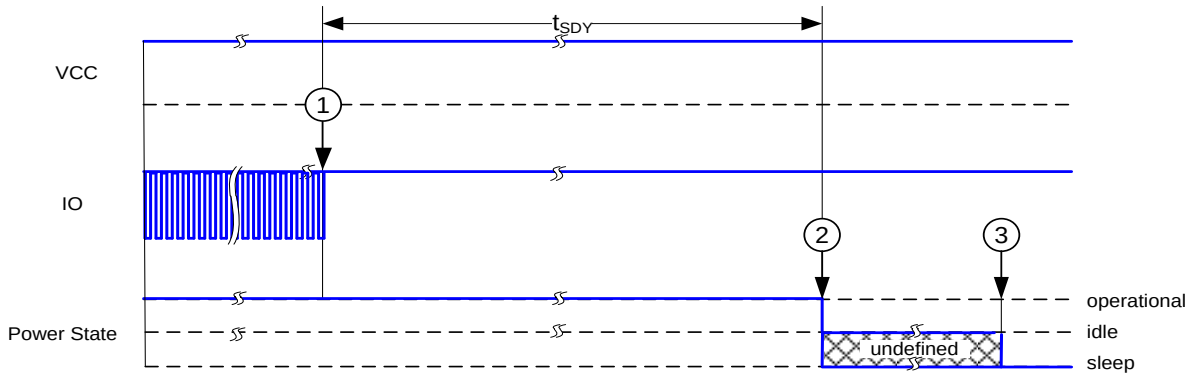

Figure 18 - Go-to-Sleep diagram

4.3 Protocol Stack

The OPTIGA™ Trust X is an I2C slave device. The protocol stack from the physical up to the application layer is specified in [IFX-I2C].
The protocol variation for the *OPTIGA™ Trust X* project is defined by Table "*Protocol Stack Variation*".

OPTIGA™ Trust X External Interface

Property	Value	Comment
MAX_PACKET_SIZE	0x110	
WIN_SIZE	1	
MAX_NET_CHAN	1	
CHAINING	TRUE	
TRANS_TIMEOUT	10	ms
TRANS_REPEAT	3	
PWR_SAVE_TIMEOUT		Not implemented
BASE_ADDR	0x30	I2C base address default
MAX_SCL_FREQU	1000	KHz
GUARD_TIME	50	µs
I2C_STATE		SOFT_RESET = 1; CONT_READ = 0; REP_START = 0; CLK_STRETCHING = 0

Table 2 - Protocol Stack Variation

4.4 Commands

This chapter provides the detailed description of the OPTIGA™ Trust X command coding and how those commands behave.

4.4.1 Command basic definitions

Name	Description
Cmd	Command code[2] as defined in Table "Command Codes"
Param	Parameter to control major variants of a command. For details refer to the particular command definition.
InLen	Length of the command data section
InData	Command data section
Sta	Response status code as defined in Table "*Response Status Codes*"
UnDef	Undefined value (contains any value 0x00-0xFF)
OutLen	Length of the response data section.
OutData	Response data section

Table 3 - APDU Fields

Command Codes

Command Code	Name	Short description
0x01 or 0x81	*GetDataObject*	Command to get (read) an data object

[2] In case the most significant bit of Cmd is set to '1', the Last Error Code XE "Last Error Code" gets flushed implicitly. This feature might be used to avoid an explicit read (with flush) of the Last Error Code XE "Last Error Code" . This feature has priority over any further command evaluation

OPTIGA™ Trust X External Interface

Command Code	Name	Short description
0x02 or 0x82	*SetDataObject*	Command to set (write) an data object
0x0C or 0x8C	*GetRandom*	Command to generate a random stream
0x10 or 0x90	SetAuthScheme	Command to set the authentication scheme which gets used subsequently
0x18 or 0x98	GetAuthMsg	Command to get (receive from *OPTIGA™ Trust X*) an authentication message
0x19 or 0x99	SetAuthMsg	Command to set (send to OPTIGA™ Trust X) an authentication message
0x1A or 0x9A	ProcUpLinkMsg	Command to process an up-link message (receive from OPTIGA™ Trust X)
0x1B or 0x9B	*ProcDownLinkMsg*	Command to process a down-link message (send to *OPTIGA™ Trust X*)
0x30 or 0xB0	CalcHash	Command to calculate a Hash
0x31 or 0xB1	CalcSign	Command to calculate a signature
0x32 or 0xB2	VerifySign	Command to verify a signature
0x33 or 0xB3	CalcSSec	Command to execute a Diffie-Hellmann key agreement
0x34 or 0xB4	DeriveKey	Command to derive keys
0x38 or 0xB8	GenKeyPair	Command to generate public key pairs
0x70 or 0xF0	OpenApplication	Command to launch an application

Table 4 - Command Codes

Response Status Codes

Response Status Code	Name	Short description
0x00	NO ERROR	Command was executed successfully
0xFF	(GENERAL) ERROR	Command execution failed due to an error. The more specific error indication is available at the Last Error Code data object (Refer to Table "Error Codes"). In this case the OutData field is absent.

Table 5 - Response Status Codes

4.4.2 Error Codes

The possible error codes are listed in Table Error Codes. If multiple commands happen to produce subsequent errors then only the highest-numbered error code is stored.

Field	Code	Description
No error	0x00	No Error
Invalid OID	0x01	Invalid OID
Invalid Param field	0x03	Invalid Param field in command

OPTIGA™ Trust X External Interface

Field	Code	Description
Invalid length field	0x04	Invalid Length field in command
Invalid parameter in data field	0x05	Invalid parameter in command data field
Internal process error	0x06	Internal process error
Access conditions not satisfied	0x07	Access conditions are not satisfied
Data object boundary exceeded	0x08	The sum of offset and data provided (offset + data length) exceeds the max length of the data object
Metadata truncation error	0x09	Metadata truncation error
Invalid command field	0x0A	Invalid command field
Command out of sequence	0x0B	Command or message out of sequence.
Command not available	0x0C	• due to termination state of the application • due to Application closed
Insufficient buffer/ memory	0x0D	Insufficient memory to process the command APDU
Invalid Handshake message	0x21	Illegal parameters in (D)TLS Handshake message, either in header or data.
Version mismatch	0x22	Protocol or data structure version mismatch (e.g. DTLS version, X.509 Version, ...).
Insufficient/Unsupported Cipher Suite	0x23	Cipher suite mismatch between client and server.
Unsupported extension/ identifier	0x24	• An unsupported extension found in the message • Unsupported keyusage/Algorithm extension/identifier for the usage of Private key
Unsupported Parameters	0x25	At least one parameter received in the handshake message is not supported.
Invalid Trust Anchor	0x26	The Trust Anchor is either not loaded or the loaded Trust Anchor is invalid (e.g. not well formed X.509 certificate, public key missing, ...).
Trust Anchor Expired	0x27	The Trust Anchor loaded at OPTIGA™ Trust X is expired.
Unsupported Trust Anchor	0x28	• The cryptographic algorithms specified in Trust Anchor loaded are not supported by OPTIGA™ Trust X.
Invalid certificate format	0x29	Invalid certificate(s) in certificate message with the following reasons. • Invalid format • Invalid chain of certificates • Signature verification failure
Unsupported certificate algorithm	0x2A	At least one cryptographic algorithm specified in the certificate is not supported (e.g. hash or sign algorithms).
Certificate expired	0x2B	The certificate or at least one certificate in a certificate chain received is expired.

OPTIGA™ Trust X External Interface

Field	Code	Description
Signature verification failure	0x2C	Signature verification failure.
MAC validation failure	0x2D	Message Integrity validation failure (e.g. during CCM decryption).

Table 6 - Error Codes

4.4.3 Command/Response Definitions

4.4.3.1 OpenApplication

The OpenApplication is used to open an application on the OPTIGA™ Trust X. Since after cold or warm Reset all applications residing on the OPTIGA™ Trust X are closed, an application has to be opened before using it. This command instantiates[3] and initializes the application context.

Field	Offset [direct]	Description
Cmd	0 [in]	**0x70 or 0xF0**[4] Command Code
Param	1 [in]	**0x00** Initialize a clean application context
InLen	2 [in]	**0xXXXX** Length of *InData*
InData	4 [in]	**0x00-0xFF** Unique Application Identifier (refer to Table "*Data Structure "Unique Application Identifier"*")
Sta	0 [out]	**0x00 | 0xFF** Response Status Code
UnDef	1 [out]	**0x00-0xFF** Undefined Value
OutLen	2 [out]	**0x0000** Length of *OutData*
OutData	4 [out]	Absent

Table 7 - OpenApplication Coding

[3] An application might be multiple times available on the OPTIGA™ Trust X. Each of them are referenced to as an instance of that application, which owns a dedicated application context, thus are independent but sharing the same code and NVM data associated to that application. In case an application has only a single instance available it is called a Singleton Application.
[4] In case of 0xF0 the Last Error Code gets flushed

4.4.3.2 GetDataObject

The GetDataObject command is used to read data objects from the OPTIGA™ Trust X. The field "Param" contains the type of data accessed. The field "InData" contains the OID of the data object, and optional the offset within the data object and maximum length to be returned with the response APDU.
Note: This command supports chaining through partial read applying offset & length as appropriate.

Field	Offset [direct]	Description
Cmd	0 [in]	**0x01 or 0x81**[5] Command Code
Param	1 [in]	• **0x00** Read data • **0x01** Read metadata
InLen	2 [in]	• **0x0006** Length of Data in case "Param = 0x00" • **0x0002** Length of Data in case "Param = 0x00" and the entire data of the data object starting at offset 0 shall be returned • **0x0002** Length of Data in case "Param = 0x01"
InData	4 [in]	**0x0000-0xFFFF** OID of data object to be read (refer to 'TLV-Coding and Access Conditions (AC)') **0x0000-0xLLLL** Offset within the data object (0xLLLL denotes the length of the data object - 1) **0x0001-0xFFFF** Number of Data bytes to be read. In case the length is longer than the available data the length will be adapted to the maximum possible length[6] and returned with the response APDU. (e.g. 0xFFFF indicates all data from offset to the end of the data object)
Sta	0 [out]	**0x00 | 0xFF** Response Status Code
UnDef	1 [out]	**0x00-0xFF** Undefined Value
OutLen	2 [out]	**0x0000-0xFFFF** Length of Data
OutData	4 [out]	**0x00-0xFF** Data object or metadata

Table 8 - GetDataObject Coding

[5] In case of 0x81 the Last Error Code gets flushed
[6] considering the offset and used data length

4.4.3.3 SetDataObject

The SetDataObject command is used to write data objects to the OPTIGA™ Trust X. The field "Param" contains the type of data accessed. The field "InData" contains the OID of the data object, the offset within the data object, and the data to be written.
Note: This command supports chaining through partial write applying offset & length as appropriate.

Field	Offset [direct]	Description
Cmd	0 [in]	**0x02 or 0x82**[7] Command Code
Param	1 [in]	• **0x00** Write data • **0x01** Write metadata • **0x40** Erase & write data
InLen	2 [in]	**0xXXXX** Length of Data +4
InData	4 [in]	**0x0000-0xFFFF** OID of data object to be written (refer to '*TLV-Coding and Access Conditions (AC)*') **0x0000-0xLLLL** Offset within the data object (0xLLLL denotes the length of the data object - 1) **0x00-0xFF** Data bytes to be written starting from the offset within the data object.
Sta	0 [out]	**0x00 | 0xFF** Response Status Code
UnDef	1 [out]	**0x00-0xFF** Undefined Value
OutLen	2 [out]	**0x0000** Length of Data
OutData	4 [out]	Absent

Table 9 - SetDataObject Coding

[7] In case of 0x82 the Last Error Code gets flushed

4.4.3.4 GetRandom

The *GetRandom* command is used to generate a random stream to be used by various security schemes. The field "*Param*" contains the type of random stream. The field "*InData*" contains the length of the random stream to be returned with the response APDU.

Field	Offset [direct]	Description
Cmd	0 [in]	**0x0C or 0x8C**[8] Command Code
Param	1 [in]	• **0x00** Random number from TRNG (according *[AIS-31]*) • **0x01** Random number from DRNG (according *[SP 800-90A]*)
InLen	2 [in]	**0x0002** Length of Data
InData	4 [in]	**0x0008-0x0100** length of random stream to be returned
Sta	0 [out]	**0x00 | 0xFF** Response Status Code
UnDef	1 [out]	**0x00-0xFF** Undefined Value
OutLen	2 [out]	**0x0000-0xFFFF** Length of Data
OutData	4 [out]	**0x00-0xFF** Random stream.

Table 10 - GetRandom Coding

[8] In case of 0x8C the Last Error Code gets flushed

4.4.3.5 SetAuthScheme

The *SetAuthScheme* command is used to set the authentication scheme and private key to be used subsequently.

Field	Offset [direct]	Description
Cmd	0 [in]	**0x10 or 0x90**[9] Command Code
Param	1 [in]	**0xXX** Authentication Scheme (refer to chapter *Authentication Schemes*)
InLen	2 [in]	**0xXXXX** Length of *InData*
InData	4 [in]	*0xE0F0*, *0xE0F1-0xE0F3* OID of the device private key (PriKey) to be used. *0xE100* OID of the session context, where the SesKey to be negotiated gets stored (is only provided in case the authentication scheme supports session key negotiation).
Sta	0 [out]	**0x00 | 0xFF** Response Status Code
UnDef	1 [out]	**0x00-0xFF** Undefined Value
OutLen	2 [out]	**0x0000** Length of *OutData*
OutData	4 [out]	Absent

Table 11 - SetAuthScheme Coding

[9] In case of 0x90 the Last Error Code gets flushed

4.4.3.6 GetAuthMsg

The GetAuthMsg command is used to read an authentication message from the OPTIGA™ Trust X. Those authentication messages and their sequence of exchange between the OPTIGA™ Trust X and its host are defined in detail in Chapter "Supported Use Cases".

Field	Offset [direct]	Description
Cmd	0 [in]	**0x18 or 0x98**[10] Command Code
Param	1 [in]	**0xXX** Authentication Message Type (refer to chapter Authentication Message Definitions)
InLen	2 [in]	**0x0000** Length of InData
InData	4 [in]	Absent
Sta	0 [out]	**0x00 | 0xFF** Response Status Code
UnDef	1 [out]	**0x00-0xFF** Undefined Value
OutLen	2 [out]	**0x0000-0xFFFF** Length of *OutData*
OutData	4 [out]	**0x00-0xFF** Authentication message

Table 12 - GetAuthMsg Coding

[10] In case of 0x98 the Last Error Code gets flushed

4.4.3.7 SetAuthMsg

The SetAuthMsg is used to write an authentication message to the OPTIGA™ Trust X. Those authentication messages and their sequence of exchange between the OPTIGA™ Trust X and the connected host are defined in detail in Chapter "Supported Use Cases".

Field	Offset [direct]	Description
Cmd	0 [in]	**0x19 or 0x99**[11] Command Code
Param	1 [in]	**0xXX** Authentication Message Type (refer to Authentication Message Definitions)
InLen	2 [in]	**0xXXXX** Length of *InData*
InData	4 [in]	**0x00-0xFF** Authentication Message
Sta	0 [out]	**0x00 | 0xFF** Response Status Code
UnDef	1 [out]	**0x00-0xFF** Undefined Value
OutLen	2 [out]	**0x0000** Length of OutData
OutData	4 [out]	Absent

Table 13 - SetAuthMsg Coding

[11] In case of 0x99 the Last Error Code gets flushed

4.4.3.8 ProcUpLinkMsg

The ProcUpLinkMsg command is used to process an uplink message by the OPTIGA™ Trust X. Those messages and their sequence of exchange between the OPTIGA™ Trust X and its host are defined in detail in Chapter "Supported Use Cases".

Field	Offset [direct]	Description
Cmd	0 [in]	**0x1A or 0x9A**[12] Command Code
Param	1 [in]	**0xXX** Authentication Message Type (refer to chapter Authentication Message Definitions)
InLen	2 [in]	**0xXXXX** Length of *InData*
InData	4 [in]	Authentication Message InData[InLen] 0xE100-0xE103 Session reference (Session context OID used in regarded SetAuthScheme command) TLV Record (optional one or multiple) • 0x31, 0x0004, xx yy zz ww sRnd.gmt_unix_time; valid for PARAM = 0x01 • 0x32, 0x0002, xx yy CertificateOID; valid for PARAM = 0x0B • 0x6y, 0xXXXX, 0x00-0xFF (start/final y = 1) Record data to be protected; valid for PARAM = 0x61
Sta	0 [out]	**0x00 | 0xFF** Response Status Code
UnDef	1 [out]	**0x00-0xFF** Undefined Value
OutLen	2 [out]	**0xXXXX** Length of OutData
OutData	4 [out]	Authentication Message OutData[OutLen] TLV Record (alternative one) • 0x5y, 0xXXXX, 0x00-0xFF (start/final y = 1) Record data protected • 0x6y, 0xXXXX, 0x00-0xFF (start/final y = 1) handshake message

Table 14 - ProcUpLinkMsg Coding

[12] In case of 0x9A the Last Error Code gets flushed

4.4.3.9 ProcDownLinkMsg

The ProcDownLinkMsg is used to process a downlink authentication message by the OPTIGA™ Trust X. Those authentication messages and their sequence of exchange between the OPTIGA™ Trust X and the connected host are defined in detail in Chapter "Supported Use Cases".

Field	Offset [direct]	Description
Cmd	0 [in]	**0x1B or 0x9B**[13] Command Code
Param	1 [in]	**0xXX** Authentication Message Type (refer to Authentication Message Definitions)
InLen	2 [in]	**0xXXXX** Length of *InData*
InData	4 [in]	Authentication Message *InData*[*InLen*] *0xE100-0xE103* Session reference (Session context OID used in regarded *SetAuthScheme* command) TLV Record (alternative one) • 0x5y, 0xXXXX, 0x00-0xFF (start/final y = 1) Record data protected • 0x6y, 0xXXXX, 0x00-0xFF (start/final y = 1) handshake message
Sta	0 [out]	**0x00 | 0xFF** Response Status Code
UnDef	1 [out]	**0x00-0xFF** Undefined Value
OutLen	2 [out]	**0xXXXX** Length of OutData
OutData	4 [out]	Authentication Message OutData[OutLen] TLV Record (optional) • 0x6y, 0xXXXX, 0x00-0xFF (start/final y = 1) Record data unprotected

Table 15 - ProcDownLinkMsg Coding

[13] In case of 0x9B the Last Error Code gets flushed

4.4.3.10 CalcHash

The *CalcHash* is used calculating a digest of a message by the *OPTIGA™ Trust X*. The message to be hashed gets either provided by the *External World* or could be one data object, or a part of a data object, or parts of multiple data objects, hosted by the *OPTIGA™ Trust X* whose read access rights are met.
In case the Intermediate hash data (context of the hash sequence which allows continuing it) is returned, the hash calculation can be continued regardless whether another hash function is executed in-between. However, the in-between hash function must be finalized or it gets terminated upon continuing the exported (context) sequence.
Note: Once the hash calculation is started (y=0) and not finalized (y=1/3/4) each command starting a new hash (e.g. CalcHash with start hashing; Handshake protocol using ProcUpLinkMsg or ProcDownLinkMsg) will terminate the currently running hash calculation and drop the result.

Field	Offset [direct]	Description
Cmd	0 [in]	**0x30 or 0xB0**[14] Command Code
Param	1 [in]	**0xXX** Hash Algorithm Identifier (refer to table '*Algorithm Identifier*')
InLen	2 [in]	**0xXXXX** Length of *InData*
InData	4 [in]	Hash Input InData[InLen] (alternative one) • 0x0y, Length[15], Message data (start y = 0, start&final y = 1, continue y = 2, final y=3, final and keep intermediate hash y=5[16]) • 0x04, 0x0000 - To terminate the hash sequence in case initialized already. • 0x1y, 0x0006, OID[17], Offset, Length[18] (start y = 0, start&final y = 1, continue y = 2, final y=3, final and keep intermediate hash y=5) (optional one or multiple) (only allowed in conjunction with continue (y=2) or final (y=3) or final and keep intermediate hash (y=5) indication) • 0x06, Length, Intermediate hash context data (only allowed in conjunction with start (y=0) or continue (y=2) indication) 0x07, 0x0000 indicate exporting the Intermediate hash context via the external interface *Note: allowed sequences are "start-(zero to n-times continue)-final" or "start&final" (atomic) or "start-(zero to n-times continue)-terminate"*
Sta	0 [out]	**0x00 | 0xFF** Response Status Code
UnDef	1 [out]	**0x00-0xFF** Undefined Value
OutLen	2 [out]	**0xXXXX** Length of *OutData*
OutData	4 [out]	Digest or intermediate hash context data 0x01, Length, Hash/Digest

[14] In case of 0xB0 the Last Error Code gets flushed
[15] Length can be 0 in case of y= 0 or 2 or 3 or 5; else it must be > 0
[16] keeping the current Intermediate hash context valid and return the hash
[17] The OID might vary throughout the hash chaining (start to final)
[18] Offset + Length must not exceed the used length of the data object addressed by OID

OPTIGA™ Trust X External Interface

Field	Offset [direct]	Description
		0x06, Length, Intermediate Hash context data Note 1: Digest is only returned in case of the final part of the message (y = 1/3) was indicated with the command. In all other cases the Digest is absent. Note 2: Intermediate hash context is only returned if indicated by *InData*.

Table 16 - CalcHash Coding

4.4.3.11 CalcSign

The *CalcSign* is used to calculate a signature over the message digest provided with the *InData*. This command is notifying the security event *Private Key Use*.

Field	Offset [direct]	Description
Cmd	0 [in]	**0x31 or 0xB1**[19] Command Code
Param	1 [in]	**0xXX** Signature Scheme (refer to *Signature Schemes*)
InLen	2 [in]	**0xXXXX** Length of *InData*
InData	4 [in]	Signature Input *InData*[*InLen*] • 0x01, Length[20], Digest to be signed • 0x03, 0x0002, OID of signature key[21] Note: The key usage of the addressed key must be set to *Sign* or *Auth*; refer to *Key Usage Identifier*
Sta	0 [out]	**0x00 | 0xFF** Response Status Code
UnDef	1 [out]	**0x00-0xFF** Undefined Value
OutLen	2 [out]	**0xXXXX** Length of OutData
OutData	4 [out]	**0x00-0xFF** Signature[22] Note: The length of the signature is derived from the applied key and signature scheme.

Table 17 - CalcSign Coding

[19] In case of 0xB1 the Last Error Code gets flushed
[20] Shall be 10 bytes up to the length of the addressed signature key
[21] The addressed signing key shall be a private key
[22] The signature pair (r,s) is encoded as two DER "INTEGER"

4.4.3.12 VerifySign

The *VerifySign* is used to verify a signature over a given digest provided with the *InData*.

Field	Offset [direct]	Description
Cmd	0 [in]	**0x32 or 0xB2**[23] Command Code
Param	1 [in]	**0xXX** Signature Scheme (refer to *Signature Schemes*)
InLen	2 [in]	**0xXXXX** Length of *InData*
InData	4 [in]	Signature Input InData[InLen] • 0x01, Length[24], Digest • 0x02, Length[25], Signature over Digest[26] (alternative one) • 0x04, 0x0002, OID of Public Key Certificate[27] • 0x05, 0x0001, Algorithm Identifier (of the Public Key), 0x06, Length, Public Key[28]
Sta	0 [out]	**0x00 | 0xFF** Response Status Code
UnDef	1 [out]	**0x00-0xFF** Undefined Value
OutLen	2 [out]	**0x0000** Length of *OutData*
OutData	4 [out]	Absent

Table 18 - VerifySign Coding

[23] In case of 0xB2 the Last Error Code gets flushed
[24] The length of the digest must be up to the key size used for the signature (e.g. ECC256 = 32) and its max. length is 64 bytes
[25] The length is limited to max. 520 bytes
[26] The ECC signature pair (r,s) is encoded as two DER "INTEGER"
[27] Must be a single certificate (DER coded) with the key usage either digitalSignature or keyCertSign according [RFC5280]. The first byte of the object must be 0x30 (which is the start byte of a DER encoded certificate)
[28] PubKey is encoded as DER "BIT STRING"

4.4.3.13 GenKeyPair

The GenKeyPair is used to generate a key pair. The Public Key gets returned to the caller. The Private Key gets stored at the provided OID of a Key or it gets returned to the caller in case no OID is provided.

Field	Offset [direct]	Description
Cmd	0 [in]	**0x38 or 0xB8**[29] Command Code
Param	1 [in]	**0xXX** Algorithm Identifier (ref to *Algorithm Identifier*) of the key to be generated
InLen	2 [in]	**0xXXXX** Length of *InData*
InData	4 [in]	Generate Key Pair Input InData[InLen] (alternative one) • 0x01, 0x0002, OID of Private Key[30] to be generated and stored as indicated by the OID (no Private Key export!). The Public Key gets exported in plain. 0x02, 0x0001, key usage (ref to Key Usage Identifier) • 0x07, 0x0000 (export key pair in plain)
Sta	0 [out]	**0x00 | 0xFF** Response Status Code
UnDef	1 [out]	**0x00-0xFF** Undefined Value
OutLen	2 [out]	**0xXXXX** Length of OutData
OutData	4 [out]	• 0x01, Len, PrivKey[31] • 0x02, Len, PubKey[32]

Table 19 - GenKeyPair Coding

[29] In case of 0xB8 the Last Error Code gets flushed
[30] Private Key can either be a non-volatile Device Private Key OR a Session Context for volatile Device Private Key, in which case the generated Key has to be stored in the respective Session Context and can later be addressed.
[31] PrivKey is encoded as DER "OCTET STRING"
[32] PubKey is encoded as DER "BIT STRING"

4.4.3.14 CalcSSec

The *CalcSSec* command calculates a shared secret, applying the algorithm defined by *Param*. The session context addressed in *InData* (tag 0x08) gets flushed and the agreed shared secret is stored there for further use or returned as requested by *InData* (tag 0x07).

Field	Offset [direct]	Description
Cmd	0 [in]	**0x33 or 0xB3**[33] Command Code
Param	1 [in]	**0xXX** Key agreement primitive (refer to *Key Agreement/ Encryption Primitives*)
InLen	2 [in]	**0xXXXX** Length of *InData*
InData	4 [in]	• 0x01, 0x0002, OID of Private Key • 0x05, 0x0001, *Algorithm Identifier*, 0x06, Length, Public Key[34] (alternative one) • 0x07, 0x0000[35] • 0x08, 0x0002, OID of Shared Secret[36]
Sta	0 [out]	**0x00 | 0xFF** Response Status Code
UnDef	1 [out]	**0x00-0xFF** Undefined Value
OutLen	2 [out]	**0xXXXX** Length of OutData
OutData	4 [out]	**0x00-0xFF** Shared secret Note 1: Shared secret is only returned in case it is requested by *InData* (0x07, 0x0000)

Table 20 - CalcSSec Coding

[33] In case of 0xB3 the Last Error Code gets flushed
[34] PubKey is encoded as DER "BIT STRING"
[35] Indicates exporting the shared secret via the external interface
[36] The shared secret becomes part of the session context and can be addressed until the session context gets flushed

4.4.3.15 DeriveKey

The *DeriveKey* command derives a key from a shared secret. The derived key is returned or saved as part of the addressed session context. The key which is stored as part of the session context can be further used as shared secret until it gets flushed.

Field	Offset [direct]	Description
Cmd	0 [in]	**0x34 or 0xB4**[37] Command Code
Param	1 [in]	**0xXX** Key derivation method (refer to *Key Derivation Method*)
InLen	2 [in]	**0xXXXX** Length of *InData*
InData	4 [in]	Key Derivation Parameter *InData*[*InLen*] • 0x01, 0x0002, OID of Shared Secret to derive the new secret from[38] • 0x02, Len, Secret derivation data[39] • 0x03, 0x0002, Length of the key to be derived[40] (alternative one) • 0x07, 0x0000[41] • 0x08, 0x0002, OID of derived key[42]
Sta	0 [out]	**0x00 | 0xFF** Response Status Code
UnDef	1 [out]	**0x00-0xFF** Undefined Value
OutLen	2 [out]	**0xXXXX** Length of OutData
OutData	4 [out]	**0x00-0xFF** Derived data Note 1: Derived data is only returned in case it is requested by *InData* (0x07, 0x0000)

Table 21 - DeriveKey Coding

4.4.4 Authentication Scheme Definitions

The Table Authentication Schemes shows the supported authentication schemes and their coding used for the SetAuthScheme command.

Value	Description
0x91	The *ECDSA FIPS 186-3 sign SHA256 hash* scheme as specified within FIPS 186-3 • ECC key length is determined by the *External World* through the public key certificate • Hash algorithm used is SHA256
0x99	The **DTLS-Client** side of the DTLS_ECDHE_ECDSA_WITH_AES_128_CCM_8 cipher suite as defined by *[RFC6347]* {0xC0,0xAE}

Table 22 - Authentication Schemes

[37] In case of 0xB4 the Last Error Code gets flushed
[38] The source of the shared secret could be a session context or data object. The used size of the data object must be max. 64 bytes.
[39] min. Length = 8 byte; max. length = 1024 byte
[40] min. Length = 16 byte; max. length = 48 byte in case of session reference; max. length = 256 byte in case of returned secret
[41] Indicates exporting the derived key via the external interface
[42] The key becomes part of the session context and can be addressed as shared secret until the session context gets flushed

4.4.5 Authentication Message Definitions

The Table DTLS Handshake client sequence shows the sequence of commands and the regarded parameters to execute a mutual authentication utilizing the DTLS protocol.

Command	Param	CmdData	RespData	Note
SetAuthScheme	0x99	OID of PriKey OID of SesKey	-	refer to Table "Authentication Schemes"
ProcUpLinkMsg	0x01	0xE100-0xE103 SesID 0x31, len, sRnd.gmt_unix_time	0x61, len, [sHeaderDTLS, sHello]	
ProcDownLinkMsg	0x03	0xE100-0xE103 SesID 0x61, len, [sHeaderDTLS, sHelloVerifyRequest]		
ProcUpLinkMsg	0x03	0xE100-0xE103 SesID	0x61, len, [sHeaderDTLS, sClientHelloWithCookie]	
ProcDownLinkMsg	0x02	0xE100-0xE103 SesID 0x61, len, [sHeaderDTLS, sHello]	-	
ProcDownLinkMsg	0x0B	0xE100-0xE103 SesID 0x60/0x61/0x62, len, [sHeaderDTLS, sCertificate]	-	In case the length of the certificate chain structure leads to a command size exceeding the max. buffer size, the External World applies command level chaining.
ProcDownLinkMsg	0x0C	0xE100-0xE103 SesID 0x61, len, [sHeaderDTLS, sServerKeyExchange]	-	
ProcDownLinkMsg	0x0D	0xE100-0xE103 SesID 0x61, len, [sHeaderDTLS, sCertificateRequest]	-	
ProcDownLinkMsg	0x0E	0xE100-0xE103 SesID 0x61, len, [sHeaderDTLS,	-	

OPTIGA™ Trust X External Interface

Command	Param	CmdData	RespData	Note
		sServerHelloDone]		
ProcUpLinkMsg	0x0B	0xE100-0xE103 SesID 0x32, 0x0002, OID of certificate data object	0x60/0x61/0x62, len, [sHeaderDTLS, sCertificate]	In case the length of the certificate chain structure exceeds the max. buffer size, the External World has to repeat the same command until the tag value in the response indicates final.
ProcUpLinkMsg	0x10	0xE100-0xE103 SesID	0x61, len, [sHeaderDTLS, sClientKeyExchange]	
ProcUpLinkMsg	0x0F	0xE100-0xE103 SesID	0x61, len, [sHeaderDTLS, sCertificateVerify]	
ProcUpLinkMsg	0x14	0xE100-0xE103 SesID	0x61, len, [sHeaderDTLS, sFinished]	
ProcUpLinkMsg	0x61	*0xE100-0xE103* SesID 0x61, len, unprotected Record(*sRecordDTLS* (*sHeaderDTLS, sFinished*))	0x51, len, protected Record(*sRecordDTLS*)	The finished message generated in the previous step to be encrypted and send to the server.
ProcDownLinkMsg	0x51	*0xE100-0xE103* SesID 0x51, len, protected Record(*sRecordDTLS*)	0x61, len, unprotected Record(sRecordDTLS)	
ProcDownLinkMsg	0x14	*0xE100-0xE103* SesID 0x61, len, [*sHeaderDTLS, sFinished*]	-	
ProcUpLinkMsg	0x71	0xE100-0xE103 SesID	-	Close the session as indicated by SesID

Table 23 - DTLS Handshake client sequence

The Table One-way authentication sequence shows the sequence of commands and the regarded parameters to execute a one-way authentication.

Command	Param	CmdData	RespData	Note
SetAuthScheme	0x91	OID of PriKey	-	refer to Table "Authentication

OPTIGA™ Trust X External Interface

Command	Param	CmdData	RespData	Note
				Schemes"
SetAuthMsg	0x01	Message to be signed	-	Length of Message must be in a range of 8 to 256 bytes
GetAuthMsg	0x02	-	Response (SIGN)	Length of SIGN depends on key length

Table 24 - One-way authentication sequence

4.4.6 Crypto Performance

The performance metrics for various schemes are provided by Table '*Crypto Performance Metrics*'

Scheme	Sequence	Execution Time[43]	Note
0x91: Authentication Schemes	@ OPTIGA™ Trust X external interface. • SetAuthMsg • GetAuthMsg	ECC 256 < 80 ms	Refer to "strict" container @ Use Case: One-way Authentication - PubKey signature scheme [osd].
0x99: Authentication Schemes	@ OPTIGA™ Trust X external interface. • ProcDownLinkMsg • ProcUpLinkMsg	ECC 256 < 1000 ms	Refer to "strict" container without "Wait" executed as of Use Case: Mutual Authentication establish session (DTLS-Client) [osd]
AES 128	@ OPTIGA™ Trust X external interface. • ProcDownLinkMsg or • ProcUpLinkMsg	CCM 15 KByte/s	Measured with 1KByte of payload processed at once.

Table 25 - Crypto Performance Metrics

4.5 Security Monitor

The Security Monitor is a central component which enforces the security policy of the OPTIGA™ Trust X. It consumes security events sent by security aware parts of the OPTIGA™ Trust X embedded SW and takes actions accordingly.

4.5.1 Security Events

The Table "*Security Events*" provides the definition of not permitted security events considered by the *OPTIGA™ Trust X* implementation.

Name	Description
Decryption Failure	The *Decryption Failure* event occurs in case a decryption and/ or integrity check of provided data lead to an integrity failure.

[43] Execution of the entire sequence, except the External World timings, with I2C@400KHz & current limitation max. value

OPTIGA™ Trust X External Interface

Name	Description
Private Key Use	The Private Key Use event occurs in case the internal services are going to use a OPTIGA™ Trust X hosted private key.
Suspect System Behavior	The Suspect System Behavior event occurs in case the embedded software detects inconsistencies with the expected behavior of the system. Those inconsistencies might be redundant information which doesn't fit to their counterpart.

Table 26 - Security Events

4.5.2 Security Policy

This paragraph provides all details of the policy chosen for the Generic Authentication Device project.
In order to mitigate exhaustive testing of the *OPTIGA™ Trust X* private keys, secret keys and passwords, and to limit the possible number of failure attacks targeting disclosure of those assets, the Security Monitor judges the notified security events regarding the number of occurrence over time and in case those violate the permitted usage profile of the system takes actions to throttle down the performance and thus the possible frequency of attacks.
The permitted usage profile is defined as:

1. One protected operation (refer to *Security Events*) events per t_{max} period.
2. A Suspect System Behavior event is never permitted and will cause setting the SEC to its maximum.
3. t_{max} is set to 5 seconds (± 5%).

The Security Monitor must enforce, in exhaustive testing scenarios, that the maximum permitted usage profile is not violated.

With other words it must not allow more than one out of the protected operations per t_{max} period (worst case, ref to bullet 1. above). This condition must be stable, at least after 500 uninterrupted executions of protected operations.

4.5.3 Characteristics

This paragraph provides the throttle down characteristics for the protected operations implemented by the Security Monitor. The Security Monitor uses the SEC to count Security Events in order to figure out not permitted usage profiles. Figure "*Throttling down profile*" depicts the characteristic (dotted line) of the dependence between the value of the SEC and the implemented delay for protected operations. The value of SEC gets decreased by one every t_{max} period. With other words, the delay starts as soon as SEC reaches the value of 128 and will be t_{max} in case the SEC reaches its maximum value of 255.

OPTIGA™ Trust X External Interface

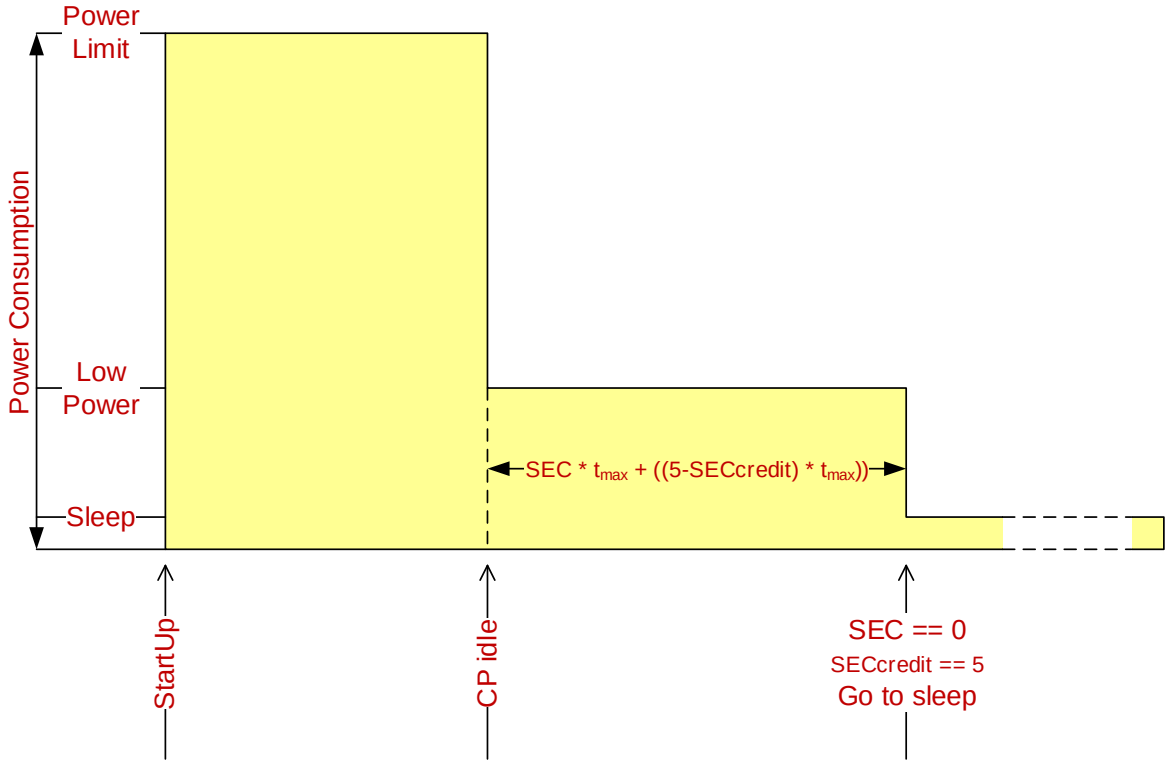

Figure 19 Power profile

Figure "*Power Profile*" depicts the power profile of a regular startup sequence, caused either by PowerUP, Warm Reset, or Security Reset. The OPTIGA™ Trust X starts up with its maximum power consumption limit set by the Current limitation data object (refer to Table Common data structures). As soon as the OPTIGA™ Trust X enters idle state (nothing to compute or communicate) the OPTIGA™ Trust X reduces its power consumption to the low power limit (System Halt Power Consumption). In case a time period of t_{max} is elapsed the SEC gets decremented by one. As soon as the SEC reaches the value of 0, the SECcredit counter reaches its maximum value, and the OPTIGA™ Trust X is in idle state, the OPTIGA™ Trust X enters the sleep mode to achieve maximum power saving. It is recommended not to switch off the power before the SEC becomes 0. In order to avoid power consumption at all, VCC could be switched off while keeping the I2C bus connected. However, before doing that the SEC value should have reached 0, to avoid accumulated SEC values which might lead to throttling down the OPTIGA™ Trust X performance (ref to Figure "*Throttling down profile*") for functionalities which potentially triggering Security Events. However, the method of switching VCC off and on is limited to 200000 times over lifetime.

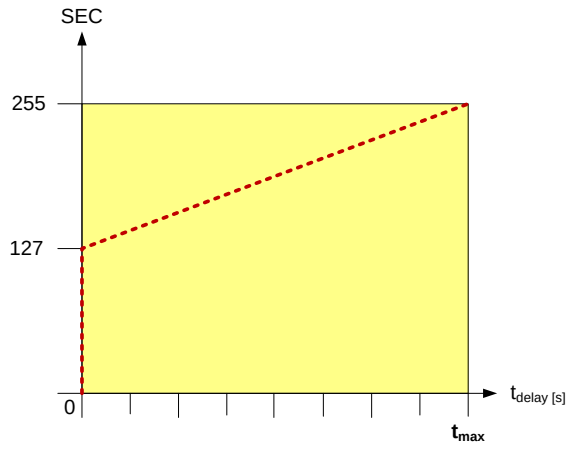

Figure 20 Throttling down profile

The SEC Credit methodology is introduced in order to reduce stress for the NVM cells hosting the SEC. For that purpose the device collects SECcredit over time (residing in RAM).

- After power-up or restart the SECcredit is cleared.
- In case the t_{max} elapses without a Security Event and the SEC is > 0 the SEC gets decreased by one.
- In case t_{max} elapses without a Security Event and the SEC is =0, the SECcredit gets increased by one to a maximum limit of 5.
- In case a Security Event occurs and the SECcredit is > 0 the SECcredit gets decreased by one.
- In case the SECcredit is = 0 and a Security Event occurs the SEC hosted in NVM gets increased.

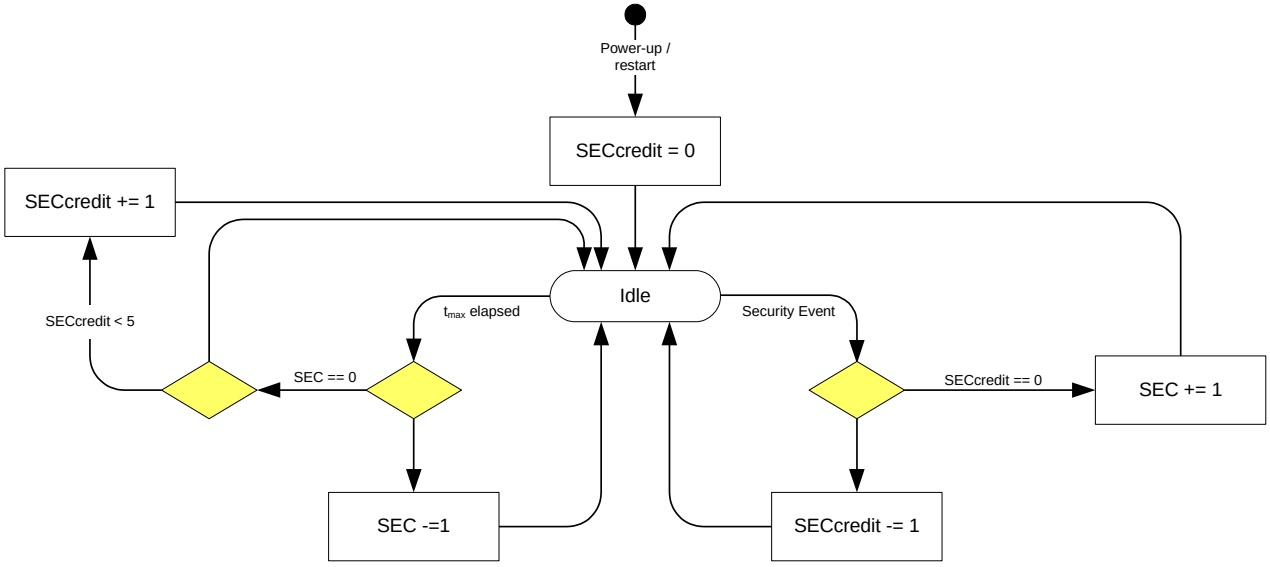

Figure 21 Security Event Counter Characteristics

4.6 Data Structures

4.6.1 Access Conditions

At each level of the data structure, Access Conditions (AC's) are defined. The ACs are defined for commands acting upon data. The ACs must be fulfilled before the data can be accessed through the regarded commands.

The following access types are used in this document:

RD reading a data or key object by an external command (e.g. GetDataObject)

CHA changing (writing or flushing) a data or key object by an external command (e.g. SetDataObject)

DEL deleting a data or key object by an external command (no command defined yet)

EXE utilizing a data or key object implicitly by executing a complex command (e.g. CalcSign, GenKeyPair, ...)

The following ACs are used in this document:

ALW the action is ***always*** possible. It can be performed without any restrictions.

OPTIGA™ Trust X External Interface

NEV the action is ***never*** possible. It can only be performed internally.

LcsG(X) the action is only possible in case the global Lifecycle Status meets the condition given by X.

LcsA(X) the action is only possible in case the application-specific Lifecycle Status meets the condition given by X.

LcsO(X) the action is only possible in case the data object-specific Lifecycle Status meets the condition given by X.

Table 'Access Condition Identifier and Operators' defines the Access Condition Identifier and Operands to be used, to define ACs associated with data objects. Access Condition Identifier must be used with the commands trying to achieve associated ACs.
There are ***simple*** and ***complex*** Access Condition expressions defined.

- A **Simple AC (sAC)** expression consists just of an access type tag (e.g. read, change, increment, decrement, delete), the length of the condition, and a single condition (e.g. ALW, NEV, LcsO < 0x04 …) which must be satisfied to grand access for that access type.
- A **Complex AC (cAC)** expression consists of multiple simple expressions combined by && and/or || operators. Where …
  ... && operators combining sACs to an access token (AT)
  AT = sAC_1 … && sAC_n (n = 1…7)
  ... || operators combining multiple ATs to a cAC
  cAC = AT_1 … || AT_m (m = 1…3; ((n_1+ … +n_m) * m) > 1)

Notes:

- *An AT evaluates TRUE in case all contained simple AC evaluate TRUE (logical AND).*
- *In case one of the AT evaluates TRUE, the regarded access becomes granted (logical OR).*
- *ALW and NEV are not allowed in cACs*

Remark: With the rules given above it doesn't matter whether starting the evaluation of a complex expression from the beginning or the end.
The access conditions which could be associated to OPTIGA™ Trust X data and key objects are defined by Table "Access Condition Identifier and Operators".

AC ID	Operator	Value	Description
ALW	-	0x00	1 byte; Value
LcsG	-	0x70	3 byte; Value, Qualifier, Reference (e.g. LcsG < op → 0x70, 0xFC, 0x04)
LcsA	-	0xE0	3 byte; Value, Qualifier, Reference (e.g. LcsA > in → 0xE0, 0xFB, 0x03)
LcsO	-	0xE1	3 byte; Value, Qualifier, Reference (e.g. LcsO < op → 0xE1, 0xFB, 0x03)
-	==	0xFA	equal
-	>	0xFB	greater than
-	<	0xFC	less than
-	&&	0xFD	logical AND
-	||	0xFE	logical OR
NEV	-	0xFF	1 byte; Value

Table 27 - Access Condition Identifier and Operators

OPTIGA™ Trust X External Interface

4.6.2 Application-specific data structures

Data element	Value	Coding	Length (Bytes)
Registered application provider identifier (RID)	0xD276000004[44]	HexNumber 5	5
Proprietary application identifier extension (PIX)	'GenAuthAppl' 0x47656E417574684170706C	Ascii11	11

Table 28 - Data Structure "Unique Application Identifier"

Data element	Value	Coding	Length (Bytes)
Arbitrary data object type 1	0x00 - 0xFF	application-specific	100
Arbitrary data object type 2	0x00 - 0xFF	application-specific	1500

Table 29 - Data Structure "Arbitrary data object"

4.6.3 Common data structures

Table '*Common data structures*' shows all common data structures defined for the *OPTIGA™ Trust X*.

Data element	Value	Coding	Length (Bytes)	Description
Life Cycle State (refer to Table '*Life Cycle Status*')		BinaryNumber 8	1	The LCS is implemented in a way that the four primary states only progress in one direction from a lower value to a higher value (e.g. initialization => operational state, but not vice versa). The application-specific states, if used at all, are managed by the particular application.
Security State (refer to Table '*Security Status*')		BinaryNumber 8	1	The device and each application may have a security status associated. The device security status is further referenced to by “Global security status” and the application specific status by “Application-specific security status”.
Last Error Code (refer to Table '*Error Codes*')		BinaryNumber 8	1	The Last Error Code stores the most recent error code generated since the data object was last cleared. The

[44] RID of former Siemens HL will be reused for IFX

OPTIGA™ Trust X External Interface

Data element	Value	Coding	Length (Bytes)	Description
				availability of a Last Error Code is indicated by the (GENERAL) ERROR (refer to Table '*Response Status Codes*'), returned from a failed command execution. The error code is cleared (set to 0x00 = "no error") after it is read or in case the MSB bit is set in the Cmd field of a Command APDU (ref to Table *'Command Codes*'). The possible error codes are listed in Table '*Error Codes*'. If multiple commands happen to produce subsequent errors then only the highest-numbered error code is stored.
Sleep Mode Activation Delay in ms	0x14 - 0xFF	BinaryNumber 8	1	The Sleep Mode Activation Delay holds the delay time in milliseconds starting from the last communication until the device enters its power saving sleep mode. The allowed values are 20-255 (ms). Its default content is 20.
Current limitation in mA	0x06 - 0x0F	BinaryNumber 8	1	The Current limitation holds the maximum value of current allowed to be consumed by the OPTIGA™ Trust X across all operating conditions. The allowed values are 6-15 (mA). This register resides in Non-Volatile Memory (NVM) and will be restored upon power up or reset. Its default content is 6mA. **Note:** *15mA will cause best case performance. 9 mA will cause roughly 60% of the best case performance.* Even the maximum communication speed might be degraded by Current limitation (How the max. possible communication speed gets indicated to the I2C master, please refer to [IFX_I2C]).
Buffer size in number of bytes	0x0200 - 0xFFFF	BinaryNumber 16	2	The Maximum Com Buffer Size holds the maximum size of APDUs transferred through the communication buffer.

OPTIGA™ Trust X External Interface

Data element	Value	Coding	Length (Bytes)	Description
				Note: In case higher data volumes need to be transferred, command chaining must be applied.
Security Event Counter (SEC)	0x00 - 0xFF	0x00 - 0xFF	1	The SEC holds the current value of the Security Event Counter as described in chapter Security Monitor.
Public Key Certificate	0x00 - 0xFF	x.509	1728 (max.)	The *Public Key Certificate* data object holds one or multiple of X.509 Certificate (refer to [RFC5280]). The certificate was issued by IFX or a Customer. A host utilizes it to authenticate the *OPTIGA™ Trust X* within the regarded PKI domain (IFX or Customer). • **One-Way Authentication Identity:** Certificate DER coded The first byte of the DER encoded certificate is **0x30** and is **used as Tag** to differentiate from other *Public Key Certificate* formats defined below. • **TLS Identity**: **Tag** = 0xC0 **Length** = Value length (2 Bytes) **Value** = Certificate Chain[45] • **USB Type-C Identity:** **Tag** = 0xC2 **Length** = Value length (2 Bytes) **Value** = USB Type-C Certificate Chain *[USB Auth]*[46]
Root CA Public Key Certificate aka "Trust Anchor"	0x00 - 0xFF	x.509 (maybe self signed certificate)	1024 (max.)	The Root CA Public Key Certificate data object holds the Public Key Certificate of the IFX or Customer Root or Intermediate Certification Authority.

Table 30 - Common data structures

1. Creation state
2. Initialization state

[45] Format of a "Certificate Structure Message" used in TLS Handshake
[46] Format as defined in Section 3.2 of the USB Type-C Authentication Specification.

OPTIGA™ Trust X External Interface

3. Operational state
4. Termination state

The LCS is implemented in a way that the four primary states only progress in one direction from a lower value to a higher value (e.g. initialization => operational state, but not vice versa). The application-specific states, if used at all, are managed by the particular application.

The life cycle status shall be interpreted according to Table 'Life Cycle Status'.

Table '*Life Cycle Status*' shows all coding of the Life Cycle Status defined for the *OPTIGA™ Trust X*.

Bit 8-5	Bit 4-1	Description
0 0 0 0	x x x x	RFU
0 0 0 0	0 0 0 1	Creation state (abbreviation = cr)
0 0 0 0	0 0 1 1	Initialization state (abbreviation = in)
0 0 0 0	0 1 1 1	Operational state (abbreviation = op)
0 0 0 0	1 1 1 1	Termination state (abbreviation = te)[47]

Table 31 - Life Cycle Status

Table 'Security Status' shows the security status defined for the device either global or application-specific. The default is 0x00 after reset for the global and after OpenApplication for the application-specific Security Status [48] [49].

Bit 8-7	Bit 6-1	Description
x x	x x x x x x	RFU

Table 32 - Security Status

Table 'Data structure Coprocessor UID OPTIGA™ Trust X' shows UID definition for the OPTIGA™ Trust X.

Offset	Data Type	Name	Description
0	uint8_t	bCimIdentifier	CIM Identifier
1	uint8_t	bPlatformIdentifier	Platform Identifier
2	uint8_t	bModelIdentifier	Model identifier
3	uint16_t	wROMCode	ID of ROM mask
5	uint8_t	rgbChipType[6]	Chip type
11	uint8_t	rgbBatchNumber[6]	Batch number
17	uint16_t	wChipPositionX	Chip position on wafer: X-coordinate
19	uint16_t	wChipPositionY	Chip position on wafer: Y-coordinate
21	uint32_t	dwFirmwareIdentifier	Firmware Identifier
25	uint8_t	rgbESWBuild[2]	ESW build number, BCD coded

[47] this state is not applicable for the LcsA
[48] bit = 0 status not satisfied
[49] bit = 1 status satisfied

Table 33 - Data structure Coprocessor UID OPTIGA™ Trust X

4.6.4 TLV-Coding and Access Conditions (AC)

Table '*Common data objects with TAG's and AC's*' shows all common data structures defined for the *OPTIGA™ Trust X* with its TAG's and AC's.

Tag	Structure definition	Default Value	ACs CHA/DEL/RD (default LcsO)
0xE0C0	Data Structure global '*Life Cycle Status*' (LcsG)	0x07	ALW/ NEV/ ALW (op)
0xE0C1	Data Structure global '*Security Status*'	0x00	ALW[50]/ NEV/ ALW (op)
0xE0C2	Data structure 'Data structure Coprocessor UID OPTIGA™ Trust X'		NEV/ NEV/ ALW (op)
0xE0C3	Data structure '*Sleep Mode Activation Delay*' (refer to '*Common data structures*')	0x14	ALW/ NEV/ ALW (op)
0xE0C4	Data structure '*Current limitation*' (refer to '*Common data structures*')	0x06	ALW/ NEV/ ALW (op)
0xE0C5	Data structure '*Security Event Counter (SEC)*' (refer to '*Common data structures*')		NEV/ NEV/ ALW (op)
0xE0C6	Data structure 'Maximum Com Buffer Size' (refer to 'Common data structures')		NEV/ NEV/ ALW (op)
0xE0E0	Device *Public Key Certificate* issued by IFX (refer to '*Common data structures*')		LcsO < op/ NEV/ ALW (op)
0xE0E1-0xE0E3	Project-specific device Public Key Certificate 1-3[51]. (refer to 'Common data structures')	0x00	LcsO < op/ NEV/ ALW (cr)
0xE0E8	Root CA Public Key Certificate 1[52] (refer to 'Common data structures')		LcsO < op/ NEV/ ALW (cr)
0xE0EF	Root CA Public Key Certificate 8[53]. This trust anchor is assigned to platform integrity use cases (refer to 'Common data structures').	0x00	LcsO < op/ NEV/ ALW (cr)

Table 34 - Common data objects with TAG's and AC's

Table 'Common key objects with TAG's and AC's' shows all common Keys defined for the OPTIGA™ Trust X with its TAG's and AC's.

[50] It is only possible to reset an achieved security status
[51] due to its size the certificate is not written in an atomic way. With other words in case the write gets terminated by a tearing event like power lost or reset, the write might be incomplete - the certificate is just partly written
[52] due to its size the public key or certificate is not written in an atomic way. With other words in case the write gets terminated by a tearing event like power lost or reset, the write might be incomplete - the certificate is just partly written.
[53] due to its size the public key or certificate is not written in an atomic way. With other words in case the write gets terminated by a tearing event like power lost or reset, the write might be incomplete - the certificate is just partly written.

OPTIGA™ Trust X External Interface

Tag	Structure definition	Default Value	ACs EXE/CHA/DEL/RD (default LcsO)
0xE0F0	Device Private Key 1		ALW/ NEV/ NEV/ NEV (op)
0xE0F1-0xE0F3	Device Private Key 2-4; The *GetDataObject*, *SetDataObject* commands are not allowed for the data part of the key object even if the metadata state the access rights differently.		ALW/ LcsO < op/ NEV/ NEV (cr)
0xE100-0xE103	Session context 1-4 (OID to address one of the four session contexts e.g. (D)TLS connection state). These OIDs are not applicable for the *GetDataObject*, *SetDataObject* commands. The session context holds either Private key or Shared secret or is target for toolbox commands like (*GenKeyPair*, *CalcSSec*, *DeriveKey*).		ALW/ NEV/ NEV/ NEV (op)

Table 35 - Common key objects with TAG's and AC's

Table '*Authentication application-specific data objects with TAG's and AC's*' shows all data structures defined for the *OPTIGA™ Trust X* Authentication Application with its TAGs and ACs.

Tag	Structure definition	Default Value	ACs CHA/DEL/RD (default LcsO)
0xF1C0	Data Structure application '*Life Cycle Status*' (LcsA)	0x01	ALW/ NEV/ ALW (op)
0xF1C1	Data Structure application '*Security Status*'	0x00	ALW[54]/ NEV/ ALW (op)
0xF1C2	Error codes (refer to Table '*Error Codes*')		NEV[55]/ NEV/ ALW (op)
0xF1D0-0xF1DF	Data Structure "Arbitrary data object" type 1.	0x00	app-specific/ NEV/ app-specific (cr)
0xF1E0-0xF1E1	Data Structure "Arbitrary data object" type 2.	0x00	app-specific/ NEV/ app-specific (cr)

Table 36 - Authentication application-specific data objects with TAG's and AC's

Metadata associated with data / key objects are expressed as constructed TLV data objects. The metadata itself are expresses as simple TLV-Objects contained within the metadata constructed TLV-Object. The following table provides a collection of the possible metadata types as data attributes (e.g. LCS, max length …) and access conditions (read, change …) to those data objects. The access conditions expressed in Table *Access Condition Identifier and Operators* describing under which condition metadata itself could be accessed (GetDataObject or SetDataObject; Param == 0x01).

Implicit rules:

- *In case the entry for an access condition (tag = 0xD?) is absent, the regarded access condition is defined NEV.*
- *In case the LcsO is absent, the access conditions of the regarded data object is considered as operational (op) and couldn't be changed.*

[54] It is only possible to reset an achieved security status
[55] cleared on read

OPTIGA™ Trust X External Interface

- *In case the Used size (Tag 0xC5) is absent, the used size is same as max. size.*
- *The changed metadata get effective as soon as the change gets consolidated at the data object.*

Table '*Metadata associated with data and key objects*' shows all common data structures defined for the *OPTIGA™ Trust X* with its TAG's and AC's.

Tag	Structure definition	Default Value	ACs CHA/DEL/RD (default LcsO)
0x20	Metadata constructed TLV-Object		
0xC0	Life Cycle State of the data object (LcsO) (refer to Table '*Life Cycle Status*')		ALW/NEV/ALW (n.a.)
0xC4	Max. size of the data object		NEV/NEV/ALW (n.a.)
0xC5	Used size of the data object		auto[56]/NEV/ALW (n.a.)
0xD0	Change Access Condition descriptor		LcsO < op/NEV/ALW (n.a.)
0xD1	Read Access Condition descriptor		LcsO < op/NEV/ALW (n.a.)
0xD2	Delete Access Condition descriptor		LcsO < op/NEV/ALW (n.a.)
0xE0	Algorithm associated with key container (refer to Table 'Algorithm Identifier')		auto[57]/NEV/ALW (n.a.)
0xE1	Key usage associated with key container (refer to Table 'Key Usage Identifier')		LcsO < op/NEV/ALW (n.a.)

Table 37 - Metadata associated with data and key objects

Table 'Algorithm Identifier' shows the coding of algorithm identifier used by the OPTIGA™ Trust X.

Value	Description
0x03	Elliptic Curve Key on NIST P256 curve.
0x04	Elliptic Curve Key on NIST P384 curve
0xE2	SHA 256

Table 38 - Algorithm Identifier

Table 'Key Usage Identifier' the coding of key usage identifier used by the OPTIGA™ Trust X.

Value	Description
0x01	*Auth* (Authentication)
0x02	*Enc* (Encryption, Key Transport)
0x04	*HostFwUpdate*
0x08	*DevMgmt* (Device Management)
0x10	*Sign*

[56] if the used size is present
[57] As part of the key generation this tag will be updated automatically

OPTIGA™ Trust X External Interface

Value	Description
0x20	*KeyAgree* (Key Agreement)

Table 39 - Key Usage Identifier

Table '*Key Agreement/ Encryption Primitives*' shows the coding of key agreement/ encryption primitives used by the *OPTIGA™ Trust X*.

Value	Description
0x01	Elliptic Curve Diffie-Hellman shared secret agreement according to NIST SP-800 56A.

Table 40 - Key Agreement/ Encryption Primitives

Table '*Key Derivation Method*' the coding of key derivation method used by the *OPTIGA™ Trust X*.

Value	Description
0x01	IETF 5246 TLS PRF SHA256

Table 41 - Key Derivation Method

Table '*Signature Schemes*' shows the coding of signature schemes used by the *OPTIGA™ Trust X*.

Value	Description
0x11	ECDSA FIPS 186-3 w/o hash

Table 42 - Signature Schemes

Examples of commonly used access conditions:

- Arbitrary Data Record @ shipping to customer

```
0x20, 0x11,                         // TL metadata TLV-Object
   0xC0, 0x01, 0x03,                // TLV LcsO = in
   0xC4, 0x01, 0x64,                // TLV max size = 100
   0xC5, 0x01, 0x0A,                // TLV used size = 10
   0xD1, 0x01, 0x00,                // TLV Read = ALW
   0xD0, 0x03, 0xE1, 0xFC, 0x04     // TLV Change = LcsO < op
                                    // TLV Delete = NEV (absent)
```

Note: in case of NEV the AC term for that kind of AC could be absent. In this example for "delete"

- Project-Specific device Public Key Certificate @ shipping to customer

```
0x20, 0x16,                         // TL metadata TLV-Object
   0xC0, 0x01, 0x03,                // TLV LcsO = in
   0xC4, 0x02, 0x04, 0x00,          // TLV max size = 1024
   0xC5, 0x02, 0x03, 0x40,          // TLV used size = 832
   0xD1, 0x01, 0x00,                // TLV Read = ALW
   0xD0, 0x03, 0xE1, 0xFC, 0x04;    // TLV Change = LcsO < op
   0xD2, 0x01, 0xFF                 // TLV Delete = NEV
```

Note: there is no ordering rule for metadata tags

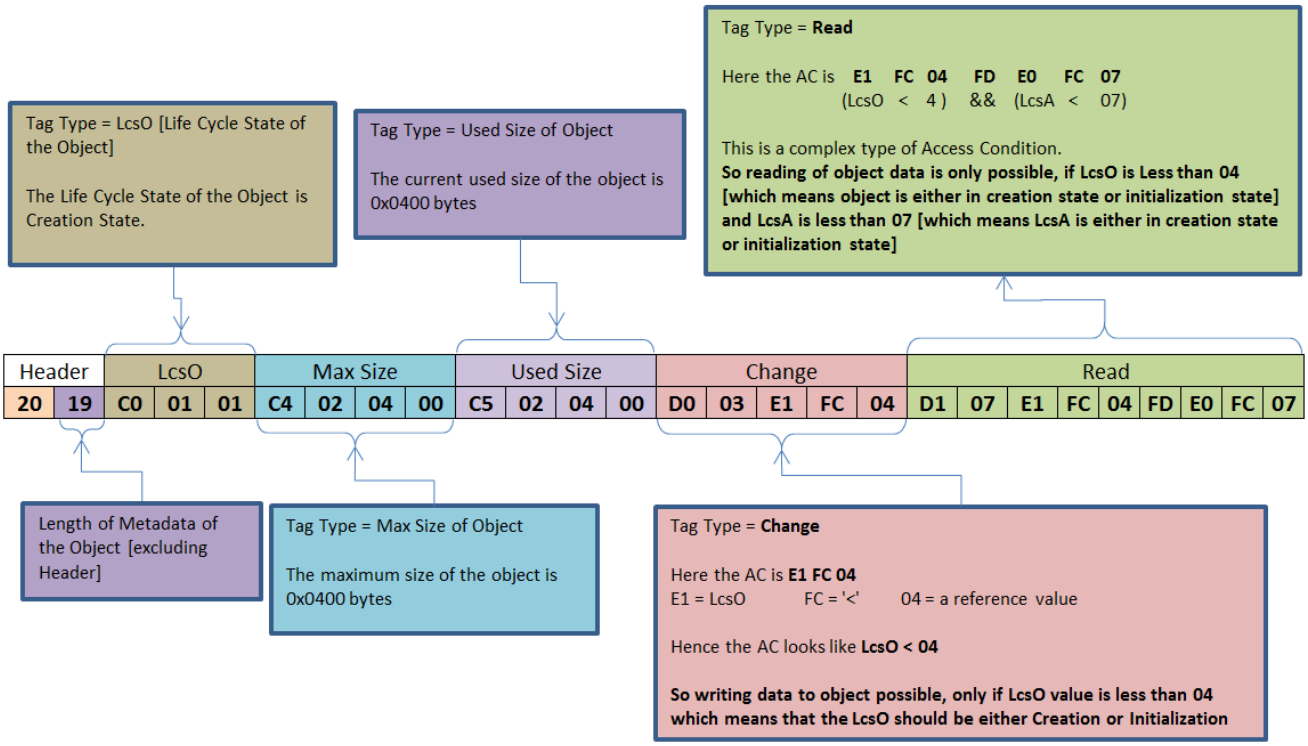

Figure 22 Metadata sample

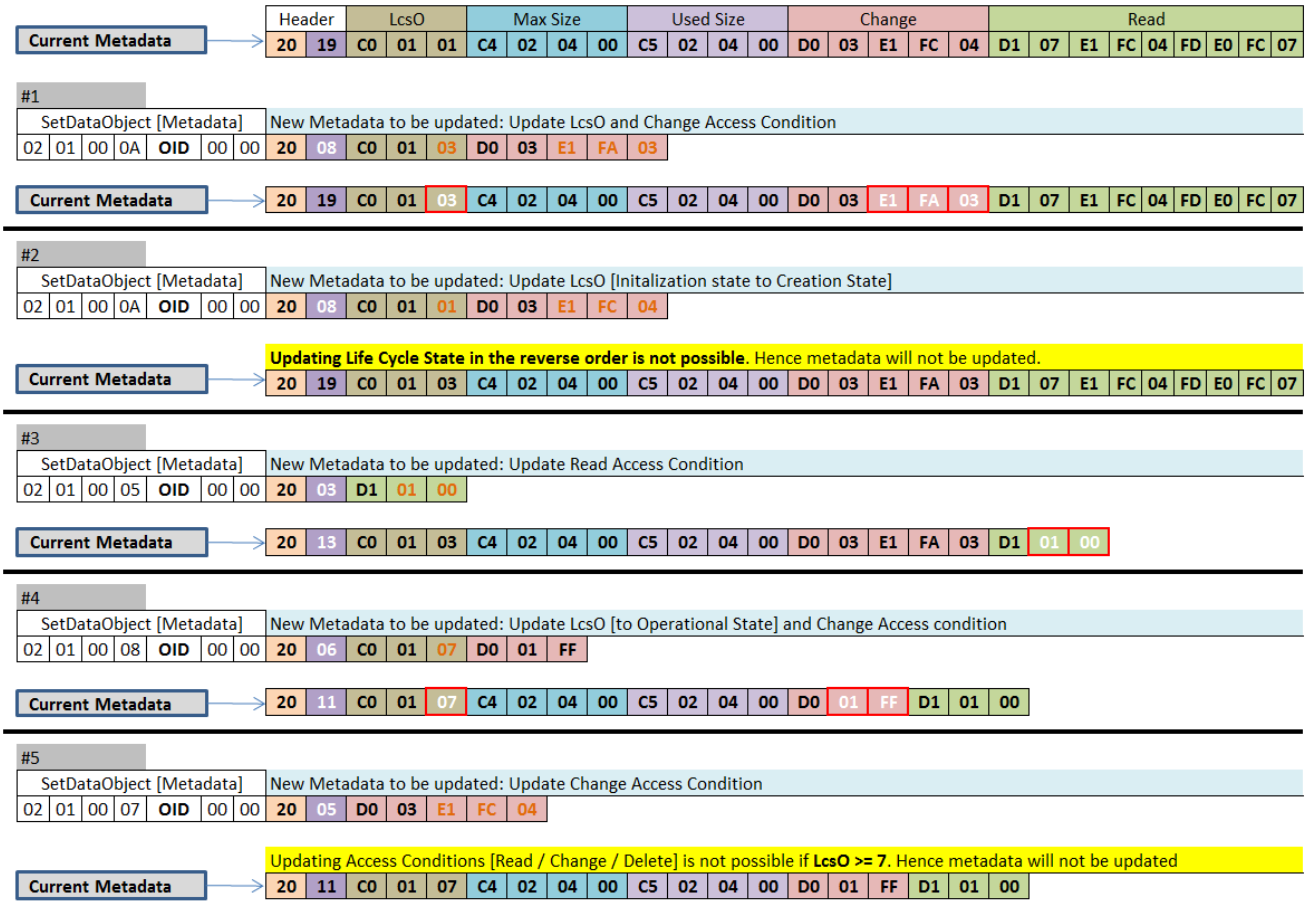

Figure 23 SetDataObject (Metadata) examples

Note: The values specified in Figure 22 and Figure 23 are in HEX format.

5 Appendix

5.1 Command Coding Examples

- **GetDataObject**

For Example, The GetDataObject command to read 5 bytes of Coprocessor UID data object starting from offset 2 is as shown below.

	CMD	PARAM	LEN		OID		Offset		No. of Bytes to be Read	
Offset	00	01	02		04		06		08	
APDU	**01**	**00**	**00**	**06**	**E0**	**C2**	**00**	**02**	**00**	**05**

RESPONSE	**00**	**00**	**00**	**05**	**xx**	**xx**	**xx**	**xx**	**xx**
	STA	**UnDef**	LEN		Data [5 Bytes]				

Figure 24 GetDataObject [Read data] example

Note: The values specified in Figure 24 *are in HEX format.*

- **SetDataObject**

For Example, The SetDataObject command to write 8 bytes of data to arbitrary data object 0xF1D0 starting from offset 9 is as shown below.

	CMD	PARAM	LEN		OID		Offset		data to be written to object [8 Bytes]							
Offset	00	01	02		04		06		08							
APDU	**02**	**00**	**00**	**0C**	**F1**	**D0**	**00**	**09**	**xx**	**xx**	**xx**	**xx**	**xx**	**xx**	**xx**	**xx**

RESPONSE	**00**	**00**	**00**	**00**
	STA	**UnDef**	LEN	

Figure 25 SetDataObject [Write data] example

Note: The values specified in Figure 25 *are in HEX format.*

5.2 (D)TLS Protocol Details

5.2.1 TLS_ECDHE_ECDSA_WITH_AES_128_CCM_8

The TLS_ECDHE_ECDSA_WITH_AES_128_CCM_8 implementation details are provided below:
DTLS is intentionally very similar to TLS. Therefore, [RFC6347] specifies DTLS as a series of deltas from TLS 1.2. Where [RFC6347] does not explicitly call out differences, DTLS is the same as TLS.

Verification of Server Signature in ServerKeyExchange Message:
[RFC5246] specifies the signature_algorithms Extensions indicating Hash Algorithm and Signature Algorithm used. Since in the absence of the signature_algorithms the default value is {sha1,ecdsa} this Extension shall be included in the Client Hello Message indicating {sha256,ecdsa} .

Generation of CertificateVerify Message:
NISTP-256 shall be supported as Signature Key. Therefore the Client Certificate Message shall contain an NISTP-256 ECDSA capable Public Key. Public Key Points MUST be in uncompressed Format. SHA256 and ECDSA for the Signature Algorithm shall be supported. Hence the Client shall only accept Certificate Request Message, which will indicate {sha256,ecdsa} as

Appendix
supported_signature_algorithms and ecdsa_sign as ClientCertificateType.

PRF Usage and Hash:
The PRF shall be based on SHA-256 as specified in [RFC7251] section 2. These cipher suites make use of the default TLS 1.2 PseudorandomFunction (PRF), which uses HMAC with the SHA-256 hash function. For Hashing of Handshake Messages used in the Verify Message SHA256 shall be supported. It shall be the same as for the PRF: The Hash MUST be the Hash used as the basis for the PRF

CCM Specification Details:
AEAD_AES_128_CCM as specified in [RFC5116] shall be used.
Nonce Generation as specified in [RFC6655] section 3 shall be used:

```
struct {
    uint32 client_write_IV; // low order 32-bits
    uint64 seq_num;         // TLS sequence number
} CCMClientNonce;
```

5.3 (D)TLS Messages

This section is provided just for information and not considered to be entirely implemented as shown here.

5.3.1 (D)TLS Record Protocol message

The Record Protocol takes messages to be transmitted, fragments the data into manageable blocks, optionally compresses the data, applies a MAC, encrypts and transmits the result. Received data is decrypted, verified, decompressed, reassembled, and then delivered to higher-level layers.
The four protocols that use the record protocol are handshake protocol, alert protocol, change cipher spec protocol, application data protocol and heartbeat protocol.
This chapter shows the record layer structure.

eRecContentType	
Description	The eRecContentType specifies the enumeration for all the (D)TLS Record Layer content types.
Enumeration literals	eChangeCipherSuite
	eAlert
	eHandshake
	eApplication
	eHeartbeat[58]

5.3.1.1 (D)TLS Handshake messages

This chapter provides detailed information regarding the (D)TLS Handshake messages. Handshake Protocol is used to negotiate the secure attributes of a session. Handshake messages are sent to record layer, where they are encapsulated within one or more TLSPlaintext structures, which are processed and transmitted as specified by the current active session state.
Note: This section covers the full standard. However, the implementation of the OPTIGA™ Trust X is

[58] Content Type not supported

Appendix

in some extend limited (for detailed information refer to the "Limitations" section).

eHandshakeType	
Description	The eHandshakeType specifies the enumeration for all the DTLS handshake messages.
Enumeration literals	eHello_request
	eClient_hello
	eServer_hello
	eHello_verify_request
	eCertificate
	eServer_key_exchange
	eCertificate_request
	eServer_hello_done
	eCertificate_verify
	eClient_key_exchange
	eFinished

5.3.1.1.1 Hello Request

This message is sent by server at any time. It is a notification that the client has to begin negotiation a new session. In response, the client should send ClientHello message when convenient. This message may be ignored by the client if it is negotiating a session and also if it does not wish to renegotiate a session and the client may also respond with a no-renegotiation alert message.
This message must not be included in the message hash. Servers do not send this message immediately upon the client's initial connection and are not expected to repeat the request until the subsequent handshake negotiation is complete.

5.3.1.1.2 Hello

Client sends Client_hello message after reception of *Hello Request* message from server or to initiate a session spontaniously. This message has the list of ciphersuites supported by the client, a random number and protocol version and extensions for ECC based ciphersuites.
In response the server sends a Server_Hello message whose structure is the same as specified by *sHello*.

eCompressionMethod	
Description	List of compression methods supported by the client/ server
Enumeration literals	NULL (0)
	eCompressionMax

eECPointFormat	
Description	Specifies the different types of point formats
Enumeration literals	eUncompressed
	eAnsiX962_compressed_prime
	eECPointFormatMax

Appendix

eExtensionType	
Description	Extension types are for extended functionality. Size is 2 bytes
Enumeration literals	eElliptic_curves Specifies the type of extension is an elliptic curve
	eEc_point_formats Specifies the type of extension is a point format
	eSignature_algorithms Specifies the type of extension is a signature algorithm
	eExtensionTypeMax

eNamedCurve	
Description	List of all the ECC curves, size is 2 bytes
Enumeration literals	eSect163k1
	eSecp256r1
	eArbitrary_explicit_prime_curves
	eArbitrary_explicit_char2_curves
	eNamedCurveMax

rgbCipherSuite	
Description	Cipher Suite is specified as 2 byte parameter. For eg: TLS_ECDHE_ECDSA_WITH_AES_128_CCM = {0xC0,0xAC}. All the cipher suites used in TLS can be found in [IANA].
Data type	uint8_t

5.3.1.1.3 HelloVerifyRequest

The client expects this message from the server after it sends the ClientHello Message. If the server's message is lost, the client knows that either the ClientHello or the HelloVerifyRequest is lost and retransmits. When the server receives retransmission, it knows to retransmit. The server also maintains retransmission timer and retransmits when timer expires.

5.3.1.1.4 ClientHelloWithCookie

The ClientHello message is sent twice. Once during beginning of session establishment, and second after it receives the HelloVerifyRequest from the server. The second ClientHello includes the cookie which is sent by server as part of HelloVerifyRequest.

5.3.1.1.5 Certificate

Server sends Certificate message when certificates are used for authentication. This is sent after Server Hello Message.
Client sends the Certificate message when requested by server through Certificate Request message

5.3.1.1.6 Server Key Exchange

This message conveys cryptographic information to allow the client to communicate the pre-master secret: a Diffie-Hellman public key used by the client to complete a key exchange. This message is sent when using ephemeral key exchange algorithms like ECDHE_ECDSA, ECDHE_RSA and

Appendix
ECDHE_anon.

eECCurveType	
Enumeration literals	eExplicit_prime
	eExplicit_char2
	eNamed_curve
	eECCurveTypeMax

5.3.1.1.7 Certificate Request

This is an optional message sent by server to request client certificate for authentication.

eClientCertificateType	
Enumeration literals	eEcdsa_sign
	eRsa_fixed_ecdh
	eEcdsa_fixed_ecdh
	eClientCertificateTypeMax

eHashAlgorithm	
Enumeration literals	eNone
	eMD5
	eSHA1
	eSHA224
	eSHA256
	eSHA384
	eSHA512
	eHashAlgorithmMax

eSignatureAlgorithm	
Enumeration literals	eANONYMOUS
	eRSA
	eDSA
	eECDSA
	eSignatureAlgorithmMax

5.3.1.1.8 Server Hello Done

The ServerHelloDone is sent by server to indicate end of Server Hello and associated messages. After sending this message server will wait for client response.

5.3.1.1.9 Client Key Exchange

This message is sent after ServerHelloDone. The message is used by client to communicate Diffie-Hellman public value to the server

5.3.1.1.10 Certificate Verify

This message is used to provide explicit verification of a client certificate. It constitutes the digitally signed handshake message. The Hash and Signature Algorithm used in the signature must be one of those present in supported_signature_algorithms field of CertificateRequest Message.

5.3.1.1.11 Finished

This message is sent after receiving change cipher spec message. It is sent to verify that the key exchange and authentication processes were successful.

5.3.1.1.11.1 (D)TLS ChangeCipherSpec message

The (D)TLS ChangeCipherSpec message is sent by the client to notify the receiving party that subsequent records will be protected under the newly negotiated CipherSpec and keys.
On sending this message to Server the Client takes the following actions:

- Instruct the record layer to make the write pending state as write active state

On receipt of Server ChangeCipherSpec message the Client takes the following actions:

- Instruct record layer to immediately copy the read pending state into the read current state

eCipherSpec	
Enumeration literals	eCipherSpecMax

5.4 Limitations

5.4.1 Memory/ Environment Constraints

- Maximum size of handshake message payload is limited to 1536 bytes. A value larger than this would lead to an "Internal Error" Alert. This implicitly limits the size of TLS-Identity certificates used in handshake massages.
- Maximum PMTU set by the user can range from 296 to 1500. Any attempt to set the value beyond these limits will lead to "Unsupported PMTU" error.
- The maximum UDP payload can range from (296-28) to (1500-28) bytes. This value is derived considering the header of IP and UDP.

5.4.2 DTLS-Protocol

This section highlights the limitations and deviations from the [RFC6347]

- Record sequence number validation for HelloVerifyRequest and ServerHello against that of ClientHello is not done, this applies even if HelloVerifyRequest is received in fragments.
- Multiple messages in a record are consumed only if they belong to the same flight. Each

message is processed one after the other. Processing stops at an invalid message, Flight completion or end of record.
- Multiple records in a datagram are processed sequentially.
- HelloRequest message when received, is fully ignored without considering its message sequence number. Since the HelloRequest message is ignored, the design allows HelloVerifyRequest to have message sequence number of either 0 or 1.
- Decode error and Handshake failure will lead to "illegal Parameter" alert message.
- Each message is sent in a separate record, even if the PMTU can accommodate more number of messages.
- Certificate chains are not supported with the Client Certificate message.
- USB Type C certificate chains are not compliant with the DTLS implementation of the Client Certificate message. Those would lead to an error condition at server side.

5.5 Certificate (Chain) Validation

This chapter specifies all the considerations done for implementation of certificate (chain) validation as specified by [RFC5280].

5.5.1 Parameter Validation

Following are the Basic Parameter Checks performed on all the Certificates in a Chain.
- Leaf Certificate Should NOT be a CA certificate
- Public Key
  i. Only uncompressed Format is supported
  ii. Length check
- Key usage
  i. Leaf Certificate -- DigitalSignature (if present)
  ii. CA Certificate -- KeyCertSign (mandatory)
- Hash Algorithm - Is supported [SHA256]
- Signature Algorithm - Is supported [ECDSA]
- Certificate Validity
  i. NotBefore < CurrentTime < NotAfter (if Current Time is supplied by Host)
  ii. NotBefore < NotAfter (If Current Time is NOT Supplied by Host)
  iii. Certificate well formed
- Basic Constraints
  i. cA - Flag to indicate cA certificate (TRUE for cA)

Following are the Basic Parameter Checks performed on Trust Anchor certificates
- The following ARE are validated
  i. Public Key Info
  ii. Supported Curve types(NIST ECC P-256 and NIST ECC P-384)
  iii. Length
  iv. Compression type (Uncompressed)
  v. Signing Algorithm (SHA-256, ECDSA)
  vi. CA (Must)
  vii. Key Usage - keyCertSign
- The following are NOT Validated
  i. Validity
  ii. Signature
  iii. Subject DN and issuer DN (For Equality)
  iv. Authority Key identifier and Subject Key identifiers (For Equality)
  v. Processing of Certificate Revocation List and policy mapping check is not done

Appendix
[RFC5280] specifies a number of fields (specified as MUST/SHOULD etc), which will not be validated for Correct Formation and Correctness of Data. Whichever field is verified for correctness are explicitly mentioned above.

5.5.2 Path Validation

For certificate path validation the OPTIGA™ Trust X performs Signature Verification back to the pre-installed Trust Entity (Trusted Root CA) on the OPTIGA™ Trust X and verification of Correct Key Usage, Algorithms, and Certificate Type (End Entity, CA Certificates). However full certificate path validation as defined in RFC5280 MUST be performed on the customer host system.
Note: If the Host System does not perform Certificate Path Validation it could for example inadvertently permit the use of expired or revoked certificates, etc. or if someone had an email signing certificate that chains up to the trusted root CA, it could probably be misused to set up a TLS connection to the OPTIGA™ Trust X*.*

The following checks are executed or not executed by the OPTIGA™ Trust X:
- Signature Validation
- Path Validation Ends at the Trust Anchor
- Missing CAs between the last certificate in the chain and the Trust Anchor
  i. Path Construction will NOT be DONE
  ii. Will be considered as invalid/failure case
- Certificate repetition in a chain against the serial number is not validated

5.6 Security Guidance

The security guidance provides useful information how to use and configure the customer solution in a reasonable manner.

5.6.1 Use Case: Host FW Update -toolbox-

The shared secret size shall be 64 byte to render a brute force useless.

FW Shared secret which is stored in one of data objects could be modified and read out by an attacker. By reading the global shared secret, the attacker can create faulty image containing malicious code which could be multicast installed to all nodes of the same type. By writing the global shared secret, the attacker can create faulty image containing malicious code which could be installed on the modified node.
- After setting the shared secret, the regarded **data object access conditions RD and CHA** shall be configured with **never allowed** (NEV) and the **lifecycle state** of the data object (LcsO) shall be **set to operational** (op) to prevent changes to the access conditions.

Platform integrity trust anchor which is stored in a data object can be modified by an attacker. By doing that the attacker can create new metadata which will be accepted by the modified node. This might be used to undermine the rollback prevention of the solution and could lead to installing known security flaws.
- After installing the trust anchor, the regarded **data object access conditions CHA** shall be configured with **never allowed** (NEV) and the **lifecycle state** of the data object (LcsO) shall be **set to operational** (op) to prevent changes to the access conditions.

5.6.2 Use Case: Mutual Authentication (DTLS-Client)

DTLS server trust anchor which is stored in a data object can be modified by an attacker. Doing

Appendix
that the attacker can mimic the legitimate server and misuse the services provided or consumed by the nodes.

- After installing the trust anchor, the regarded **data object access conditions CHA** shall be configured with **never allowed** (NEV) and the **lifecycle state** of the data object (LcsO) shall be **set to operational** (op) to prevent changes to the access conditions.

5.6.3 Key usage associated to toolbox functionality

Key usage which is stored in a key object metadata can be modified by an attacker. Doing that the attacker can misuse the key in not intended schemes, which could enable crypto analysis or brute force attacks.

- The regarded **key object usage** shall be configured with the **least usage profile** (in most cases just one) as required by the target host application
- After setting the key usage, the **lifecycle state** of the key object (LcsO) shall be **set to operational** (op) to prevent changes to the key usage.

5.6.4 Key pair generation associated to toolbox functionality

The generated key pair and the associated public key certificate are stored in key object and public key certificate data object. The attacker attempts to re-generate the key pair. Doing that the attacker is dropping the identity which was associated to that key pair and could be considered as DoS attack.
Note: A similar result could be achieved in case only the certificate data object gets corrupted.

- After installing the identity, the regarded **key object and public key certificate access conditions CHA** shall be configured with **never allowed** (NEV) and the **lifecycle state** of the key and data object (LcsO) shall be **set to operational** (op) to prevent changes to the access conditions.

5.6.5 Shared secret for key derivation associated to toolbox functionality

A shared secret which gets fed in a key derivation function, either from the session context or from a data object shall be at least of a size of 16 bytes.

5.6.6 Use Case: One-way Authentication

Authentication trust anchor on the host shall be protected against unintended or unpermitted change. By changing the attacker can counterfeit the legitimate device and participate in an eco-system without license.

5.7 Glossary

The Glossary provides a consistent set of definitions to help avoid misunderstandings. It is particular important to **Developers**, who make use of the terms in the Glossary when designing and implementing, and **Analysts**, who use the Glossary to capture project-specific terms, and to ensure that all kind of specifications make correct and consistent use of those terms**.**

Term	Description	Abbreviation
computer data storage	*computer data storage*, often called storage or memory, is a technology consisting of computer components and recording media used to retain	

Appendix

Term	Description	Abbreviation
	digital data. It is a core function and fundamental component of computers.	
Datagram Transport Layer Security	**D**atagram **T**ransport **L**ayer **S**ecurity (DTLS) protocol provides communications privacy for datagram protocols. The protocol allows client/server applications to communicate in a way that is designed to prevent eavesdropping, tampering or message forgery. The DTLS protocol is based on Transport Layer Security (TLS) protocol and provides equivalent security guarantees.	DTLS
designed for re-use	*designed for re-use* is synonym for designing / developing reusable components.	
embedded system	An *embedded system* is a computer system with a dedicated function within a larger mechanical or electrical system, often with real-time computing constraints	
Generic Authentication Device	The *Generic Authentication Device* provides a set of functionalities required for authentication use cases. The term "generic" means the user of the device has a certain flexibility to adapt the device for his particular purpose.	GAD
Microcontroller	*Microcontroller* is a small computer on a single integrated circuit containing a processor core, memory, and programmable input/output peripherals.	µC / MCU
Non-Volatile Memory	*Non-Volatile Memory*, NVM or non-volatile storage is a *computer data storage* that can get back stored information even when not powered. Examples of non-volatile memory include read-only memory (ROM), electrical erasable programmable read-only memory (EEPROM), **flash memory** (the most popular for *Secure Microcontroller*), ferroelectric RAM (F-RAM), most types of magnetic computer storage devices (e.g. hard disks , floppy disks, and magnetic tape, optical discs, and early computer storage methods such as paper tape and punched cards.	NVM
Random-Access Memory	*Random-Access Memory* is a form of a *computer data storage*. A *Random-Access Memory* device allows data items to be read and written in roughly the same amount of time regardless of the order in which data items are accessed.	RAM
Restriction of Hazardous Substances	Directive on the restriction of the use of certain hazardous substances in electrical and electronic equipment 2002/95/EC	RoS
Secure Microcontroller	*Secure Microcontroller* is a *Microcontroller* particular designed for embedded security applications and is hardened against a huge variety of attacks which threaten the contained assets.	SecMC

Appendix

Term	Description	Abbreviation
stereotype	A stereotype is in [UML] a modeling element that extends the semantics of the metamodel. Stereotypes must be based on certain existing types or classes in the metamodel. Stereotypes may extend the semantics, but not the structure of pre-existing types and classes.	
system	A system is a set of interacting or interdependent components forming an integrated whole. Every system is circumscribed by its spatial and temporal boundaries, surrounded and influenced by its environment, described by its structure and purpose and expressed in its functioning.	
Transport Layer Security	**T**ransport **L**ayer **S**ecurity (TLS) protocol provides communications privacy for IP based (e.g. TCP/IP) protocols. The protocol allows client/server applications to communicate in a way that is designed to prevent eavesdropping, tampering or message forgery.	TLS

Table 43 - Terms of OPTIGA™ Trust X Solution Reference Manual

5.8 Change History

Version	Date	Description
1.00	01-Dec-2016	First document release
1.10	14-Feb-2017	• Error code "0x0D; Insufficient communication buffer size" added. • API refinements; CmdLib_OpenApplication, IntLib_Authenticate, OCP_Init, OCP_Connect, OCP_Send, OCP_Receive, OCP_Disconnect • API operation added; CmdLib_CalcHash • Table DTLS Handshake client sequence refined (sHeaderDTLS added where appropriate). • Minor refinements
1.15	22-Feb-2017	Review feedback to version 1.10 incorporated
1.20	27-Feb-2017	• Memory/ Environment Constraints refined with hint to max. size of TLS-Identity certificate. • Some minor refinements.
1.25	01-Mar-2017	Minor refinements: references, typo, grammar, etc.
1.26	18-Apr-2017	• Tables *Common key objects with TAG's and AC's*, *Common data objects with TAG's and AC's*, *Authentication application-specific data objects with TAG's and AC's*, adapted to toolbox requirements. • Table *DTLS Session Key* removed. • Commands *CalcSign*, *VerifySign*, *GenKeyPair*, *DeriveKey* added. • Command *CalcHash* refined.
1.27	23-Apr-2017	• Table *Common data objects with TAG's and AC's, Common key objects with TAG's and AC's, Metadata associated with data and key objects*, *Algorithm Identifier* refined. • Table *Key Usage Identifier* added.
1.28	02-May-2017	• Commands *CalcHash*, *GenKeyPair*, *CalcSign* refined.
1.29	03-Jun-2017	• *Security Policy* rephrased • Table *Error Codes.Unsupported extension/ identifier*, *Common key objects with TAG's and AC's* refined. • Commands *GetRandom*, *CalcSign*, *VerifySign*, *GenKeyPair*, *CalcSSec*, *DeriveKey* refined.
1.30	28-Jun-2017	• Minor changes (typo, etc.) • Table *Key Usage Identifier* updated. • Chapter *Enabler APIs* updated.
1.31	31-Jul-2017	• Chapter *Toolbox based Sequence Diagrams* added. • Command *DeriveKey* updated.
1.32	02-Aug-2017	• Sequence diagram *Use Case: Host FW Update -toolbox-* updated. • Command *DeriveKey* updated. • Chapter *Security Guidance* added
1.33	02-Jan-2018	• Chapter Enabler APIs reworked with changed architecture. • Block definition diagrams OPTIGA Trust X IP Protection View [bdd], OPTIGA Trust X Brand Protection View [bdd], OPTIGA Trust X Communication Protection View [bdd], OPTIGA Trust X Communication Protection View -toolbox- [bdd] reworked, with regards to changed architecture. • Table Host Code Size updated.

Appendix

		• VerifySign footnotes revised.
1.34	08-Jan-2018	• Diagram OPTIGA Trust X Communication Protection View [bdd], OPTIGA Trust X Communication Protection View -toolbox- [bdd] refined. • Some language refined (optiga_comms_reset, CmdLib_SetAuthScheme, OpenApplication).
1.35	29-Jan-2018	• Review feedback from application engineering incorporated.

Edition January 29, 2018
Published by

Infineon Technologies AG

81726 Munich, Germany

Do you have a question about this document?

Email: erratum@infineon.com

Document reference
ifx1

IMPORTANT NOTICE

The information given in this document shall in no event be regarded as a guarantee of conditions or characteristics ("Beschaffenheitsgarantie") .

With respect to any examples, hints or any typical values stated herein and/or any information regarding the application of the product, Infineon Technologies hereby disclaims any and all warranties and liabilities of any kind, including without limitation warranties of non-infringement of intellectual property rights of any third party.

In addition, any information given in this document is subject to customer's compliance with its obligations stated in this document and any applicable legal requirements, norms and standards concerning customer's products and any use of the product of Infineon Technologies in customer's applications.

The data contained in this document is exclusively intended for technically trained staff. It is the responsibility of customer's technical departments to evaluate the suitability of the product for the intended application and the completeness of the product information given in this document with respect to such application.

For further information on the product, technology, delivery terms and conditions and prices please contact your nearest Infineon Technologies office (**www.infineon.com**).

WARNINGS

Due to technical requirements products may contain dangerous substances. For information on the types in question please contact your nearest Infineon Technologies office.

Except as otherwise explicitly approved by Infineon Technologies in a written document signed by authorized representatives of Infineon Technologies, Infineon Technologies' products may not be used in any applications where a failure of the product or any consequences of the use thereof can reasonably be expected to result in personal injury.