OPTIGA™ Trust M1

Solution Reference Manual

About this document

Scope and purpose

The scope of this document is the OPTIGA™ Trust M[1] solution spanning from the device with its external interface to the enabler components used for integrating the device with a bigger system. Throughout this document the term OPTIGA™ is interchangable used for the particular OPTIGA™ Trust family member OPTIGA™ Trust M, which is subject of this document.

Intended audience

This document addresses the audience: development teams as well as customers, solution providers or system integrators who are interested in solution details.

[1] All references regarding the OPTIGA™ Trust M are given generically without indicating the dedicated version (e.g. M1, M2, ...)

Table of Contents

Table of Contents

Table of Contents

Table of Contents

Table of Contents

Table of Contents

Figures

Figures

Tables

Tables

1 Introduction

This chapter provides beyond others 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
CRL	Certificate Revocation List
DD	Device Driver
DO	Data Object
DTLS	Datagram Transport Layer Security
ESW	Embedded Software
GAD	Generic Authentication Device
LC / LCM	Life Cycle / Life Cycle Management
NVM	Non-Volatile Memory
OID	Object Identifier
OSD	Object Sequence Diagram
RAM	Random-Access Memory
SEC	Security Event Counter
SecMC	Secure Microcontroller
TLS	Transport Layer Security
UID	Unique Identifier
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™ Private Key for Authentication).

Abbreviation	Term
AUT	Authentication (Key)

Abbreviation	Term
CERT	Certificate
ECC	Elliptic Curve Crypto (Key)
ENC	Encryption (Key, confidentiality)
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
SYM	Symmetric (Key)

1.3 References

The shown references are either direct used throughout this document or worth to read for a better understanding of the eco-systems with which the OPTIGA™ interacts.

Name	Description
[AIS-31]	<https://www.bsi.bund.de/SharedDocs/Downloads/DE/BSI/Zertifizierung/Interpretationen/AIS_31_Functionality_classes_for_random_number_generators_e.pdf?__blob=publicationFile> A proposal for: Functionality classes for random number generators
[CBOR]	https://tools.ietf.org/html/rfc7049 Concise Binary Object Representation(CBOR)
[CDDL]	https://tools.ietf.org/html/draft-ietf-cbor-cddl-05 Concise data definition language (CDDL): a notational convention to express CBOR and JSON data structures [Draft version]
[CoAP]	https://tools.ietf.org/html/rfc7252 The Constrained Application Protocol (CoAP)
[COSE RSA]	https://tools.ietf.org/html/rfc8230 Using RSA Algorithms with CBOR Object Signing and Encryption messages
[COSE]	https://tools.ietf.org/html/rfc8152 CBOR Object Signing and Encryption
[Data Sheet M]	OPTIGA™ Trust M - Data Sheet

Name	Description
[DAVE]	http://www.infineon.com/cms/en/product/microcontroller/development-tools-software-and-kits/dave-tm-–-free-development-platform-for-code-generation/channel.html?channel=db3a30433580b37101359f8ee6963814
[FIPS PUB 140-2]	FIPS140-2 <http://csrc.nist.gov/publications/fips/fips140-2/fips1402.pdf> FIPS 140-2, Security Requirements for Cryptographic Modules (May 25, 2001; Change Notice 2, 12/3/2002)
[FIPS PUB 186-3]	https://www.nist.gov/publications/updated-digital-signature-standard-approved-federal-information-processing-standard Updated Digital Signature Standard Approved as Federal Information Processing Standard (FIPS)186-3
[FIPS PUB 186-4]	https://nvlpubs.nist.gov/nistpubs/fips/nist.fips.186-4.pdf Federal Information Processing Standards Publication: Digital Signature Standard (DSS)
[IANA]	http://www.iana.org/assignments/tls-parameters/tls-parameters.xhtml Transport Layer Security (TLS) Parameters
[IFX_I2C]	Infineon Technologies AG; IFX I2C Protocol Specification
[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
[MQTT]	http://docs.oasis-open.org/mqtt/mqtt/v3.1.1/os/mqtt-v3.1.1-os.html#_Toc398718013 OASIS Message Queuing Telemetry Transport version 3.1.1
[RFC2631]	https://tools.ietf.org/pdf/rfc2631.pdf Diffie-Hellman Key Agreement Method
[RFC2986]	https://tools.ietf.org/html/rfc2986 Certificate Request Syntax Specification Version 1.7
[RFC4108]	https://tools.ietf.org/html/rfc4108 Using Cryptographic Message Syntax (CMS) to Protect Firmware Packages
[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
[RFC5246]	https://tools.ietf.org/html/rfc5246 The Transport Layer Security (TLS) Protocol, Version 1.2, August 2008
[RFC5280]	https://tools.ietf.org/html/rfc5280 Internet X.509 Public Key Infrastructure Certificate and Certificate

Name	Description
	Revocation List (CRL) Profile
[RFC6347]	https://tools.ietf.org/html/rfc6347 Datagram Transport Layer Security Version 1.2
[RFC6655]	https://tools.ietf.org/html/rfc6655 AES-CCM Cipher Suites for Transport Layer Security (TLS)
[RFC7251]	https://tools.ietf.org/html/rfc7251 AES-CCM Elliptic Curve Cryptography (ECC) Cipher Suites for TLS
[RFC7301]	https://tools.ietf.org/html/rfc7301 Transport Layer Security (TLS) - Application-Layer Protocol Negotiation Extension
[RFC7925]	https://tools.ietf.org/html/rfc7925 Transport Layer Security (TLS)/ Datagram Transport Layer Security (DTLS) Profiles for the Internet of Thinks.
[RFC8017]	https://tools.ietf.org/html/rfc8017 PKCS #1: RSA Cryptography Specifications Version 2.2
[SP 800-38C]	http://csrc.nist.gov/publications/nistpubs/800-38C/SP800-38C.pdf Recommendation for Block Cipher Modes of Operation: the CCM Mode for Authentication and Confidentiality.
[SP 800-56A]	https://csrc.nist.gov/publications/detail/sp/800-56a/rev-3/final Recommendation for Pair-Wise Key Establishment Schemes Using Discrete Logarithm Cryptography
[SP 800-90A]	http://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.800-90Ar1.pdf Recommendation for Random Number Generation Using Deterministic Random Bit Generators (SP 800-90A Rev1)
[SUIT_DRAFTv2]	https://tools.ietf.org/html/draft-moran-suit-manifest-02 A CBOR-based Manifest Serialization Format [Draft version]
[SysML]	http://www.omg.org/spec/SysML/1.2/PDF/ Object Management Group: "**OMG Systems Modeling Language (OMG SysML™) - Version 1.2**", June 2010, formal/2010-06-01
[TLS]	[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

Name	Description
[WPC Auth]	open Wireless Power Charging 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.

1.4 Overview

The OPTIGA™ provides a cryptographic feature set particular supporting IoT use cases along with that it provides a number of key and arbitrary data objects which hold user/customer related keys and data.

The subsequent document is structured in the chapters Supported Use Cases, 'Enabler APIs', 'OPTIGA™ Trust M External Interface, OPTIGA™ Trust M Data Structures and 'Appendix'.

- Supported Use Cases provides the main use case expressed as sequence diagrams to maintain a better understanding of how to use the host side Enabler APIs.
- Enabler APIs provides the necessary details of the host side provided architectural APIs which are implemented based on the OPTIGA™ Trust M External Interface.
- OPTIGA™ Trust M External Interface provides the necessary details of the external interface provided to utilize the OPTIGA™ functionality.
- OPTIGA™ Trust M Data Structures provides details of the key and data structures and storage provided by the OPTIGA™.
- Appendix provides some useful information with regards to Command Coding Examples, Limitations, Certificate Parser Details, Security Guidance, Shielded Connection V1 Guidance, Protected Update: Manifest and Signature structures, Glossary, etc.

2 Supported Use Cases

In the Supported Use Cases chapter a collection of use cases are provided which are expressed as UML sequence diagrams to show how to utilize the OPTIGA™ enabler components (Enabler APIs) to achieve the target functionality of the solution. This chapter is intended to maintain a well understanding of the OPTIGA™ eco system components particular for system integrators who like to integrate the OPTIGA™ with their solution.

2.1 Architecture Decomposition

Contained Components

The architecture components contained in the shown solution architecture views are listed and briefly described in the table below.

Name	Description
local_host_application	The local_host_application is the embedded application implementing the local host functionality. For implementing that functionality it utilizes the APIs exposed by the service layer and the third_party_crypto libraries and optionally if necessary the Access Layer and the Abstraction Layer.
optiga_cmd	This module optiga_cmd exposes the main interface to interact with OPTIGA™. It is aware of the format of the command set provided by the OPTIGA™. The optiga_cmd converts API calls in the regarded (command / response) APDUs known by the OPTIGA™. The optiga_cmd APIs expose the same semantics provided by OPTIGA™. The optiga_cmd provides multiple instances of the API. Beyond exposing the APIs it arbitrates as well concurrent invocations of the APIs. Its usage characteristic is asynchronous, where the caller of an instance has to take care of the correct sequence of calls for a dedicated use case. In case an instance of the API requires multiple invocations to reliably implement a use case (strict sequence), the APIs allows locking out other instances from interacting with the OPTIGA™. As soon as those strict sequences are executed the locking must be released. The optiga_cmd interacts with optiga_comms_xxx (xxx stands for variants e.g. ifx_i2c, tc, ...) for reliable communication with OPTIGA™.
optiga_comms_ifx_i2c	optiga_comms_ifx_i2c implements the protocol used to turn-in communication between Local Host and OPTIGA™. The invoking component, in the given architecture is the optiga_cmd block through the pal. The optiga_cmd provides command APDUs to optiga_comms_ifx_i2c and receives response APDUs from the optiga_comms_ifx_i2c. The size of APDUs may vary between few bytes to kilo bytes. The protocol implementation is done in multiple layers and seamlessly handles data transfer from Local Host to OPTIGA™ and OPTIGA™ to Local Host. More details of the implemented protocol can

Supported Use Cases

Name	Description
	be found in [IFX_I2C]. optiga_comms_ifx_i2c usage characteristic is asynchronous, were the caller has to take care of the correct sequence of calls for a dedicated use case.
optiga_crypt	The optiga_crypt module provides cryptographic tool box functionality with the following characteristics: • Multiple instances could be created using optiga_crypt_create to allow concurrent access to the tool box. • Usage of the optiga_cmd module to interact with the OPTIGA™. • The optiga_cmd module might get locked for some consecutive invocations, which need to be executed atomic (strict).
optiga_util	The optiga_util module provides useful utilities to manage the application and data stores with the following characteristics: • Multiple instances could be created to allow concurrent access to other services. • Usage of the optiga_cmd module to interact with the OPTIGA™.
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 (platform_timer, platform_i2c, platform_socket, ...) to allow the modules calling it being platform agnostic. The pal is composed of hardware, software and an operating system abstraction part.
platform_crypto	Cryptographic functionalities are implemented either in software or in hardware or as a mixture of both. The functionality is provided in platform_crypto. platform_crypto is supplied by the platform vendor or a third party. This module is used multifold but not limited to: • On the authenticator site of the one-way authentication use case (refer to OPTIGA Trust IP Protection View and OPTIGA Trust Brand Protection View) • Supporting FW decryption at the local host. • Performing the key negotiation part for the platform binding and the communication protection at the local host
platform_i2c	The platform_i2c is the platform specific I2C device driver.
platform_timer	The platform_timer is the platform specific timer device driver.
third_party_crypto	Cryptographic functionalities are implemented in software and provided in third_party_crypto. The main cryptographic operations of interest for the local host are certificate parsing, signature

Supported Use Cases

Name	Description
	verification, signature generation key negotiation and certificate verification. third_party_crypto is supplied by third party. This module is used multifold but not limited to: • Supporting TLS/DTLS protocol either for client or server side. • Supporting bulk encryption in case the record protocol is performed at the local host.

Supported Use Cases

The class diagram OPTIGA Trust Communication Protection - toolbox - View shows the Communication Protection Solution Architecture in case the local host is invoking a third_party_crypto library (e.g. WolfSSL, OpenSSL, mbedTLS, ...) containing its main functional blocks. The OPTIGA™ is integrated via its toolbox functionality. The entities communicating across a protected channel are the Server and the Client (Host) and optionally the Client and the OPTIGA™ (OPTIGA™ Shielded Connection). This view is applied for toolbox based solution kind of use cases, where the involved blocks are represented as dedicated lifelines.
The color coding provides information of whether the functional block is:

- yellow: platform agnostic and provided by IFX or
- green: platform ported (subject of porting to a target platform) and provided by IFX as an example ported to the evaluation board or
- blue: platform specific provided by a third party.

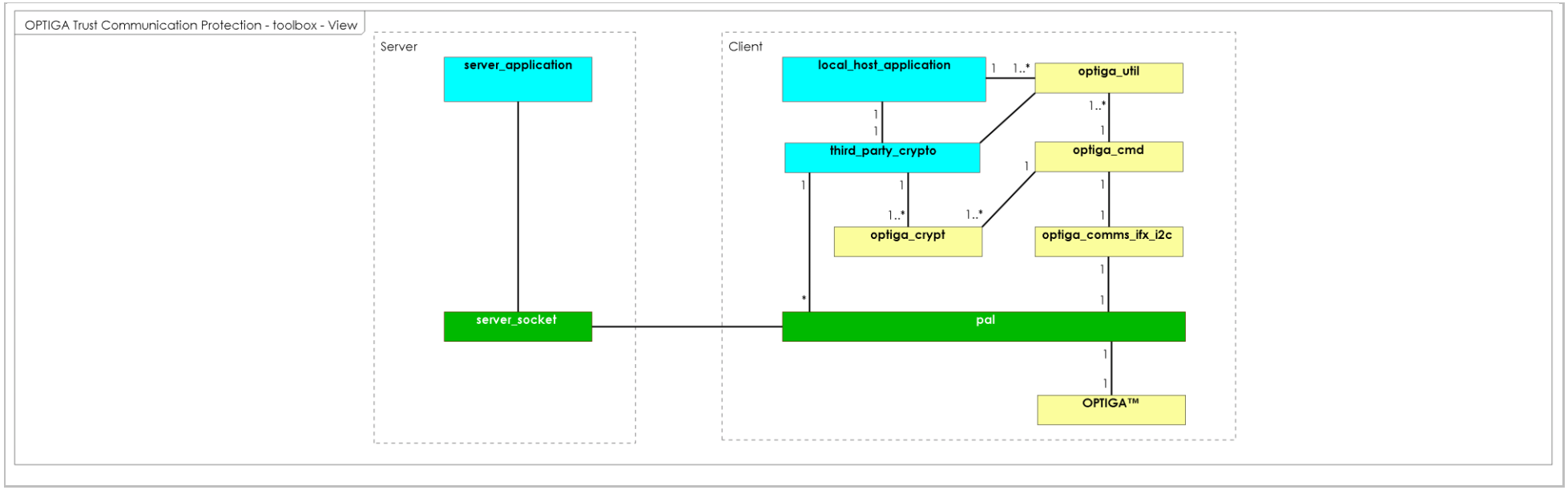

Figure 1 - OPTIGA Trust Communication Protection - toolbox - View

2.1.1 Host code size

The Table Host code size shows the footprint of the various host side configurations. The "Note" column specifies the components contained in the footprint calculation. All other components even shown by the architecture diagram are heavily project specific and provided by the system integrator. The values specified in the table 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 Communication Protection Architecture (Server <=> Client) is shown by OPTIGA Trust Communication Protection - toolbox - View.	5 / 15 KByte	[without the Shielded Connection] The components optiga_crypt, optiga_util, optiga_cmd, optiga_comms_ifx_i2c, pal are covered.
The Communication Protection Architecture (Server <=> Client) is shown by OPTIGA Trust Communication Protection - toolbox - View.	15 / 30 KByte	[with Shielded Connection] The components optiga_crypt, optiga_util, optiga_cmd, optiga_comms_ifx_i2c, pal and platform_crypto are covered. Here mbed TLS v2.7.0 is used as platform_crypto for the Shielded Connection cryptographic operations (e.g key derivation, encryption and decryption).

Table 1 - Host code size

2.2 Sequence Diagrams utilizing basic functionality

2.2.1 Use Case: Read General Purpose Data - data object

The local_host_application intends to read the content of a data object maintained by the OPTIGA™.
The sequence diagram Use Case: Read General Purpose Data - data object is provided to show the functions involved in reading a data object. The function is performed atomic (no other invocation of the optiga_cmd module will interrupt the execution).

Pre-condition:

- The *OPTIGA™* application is already launched
- The necessary access conditions for reading the target data object are satisfied.

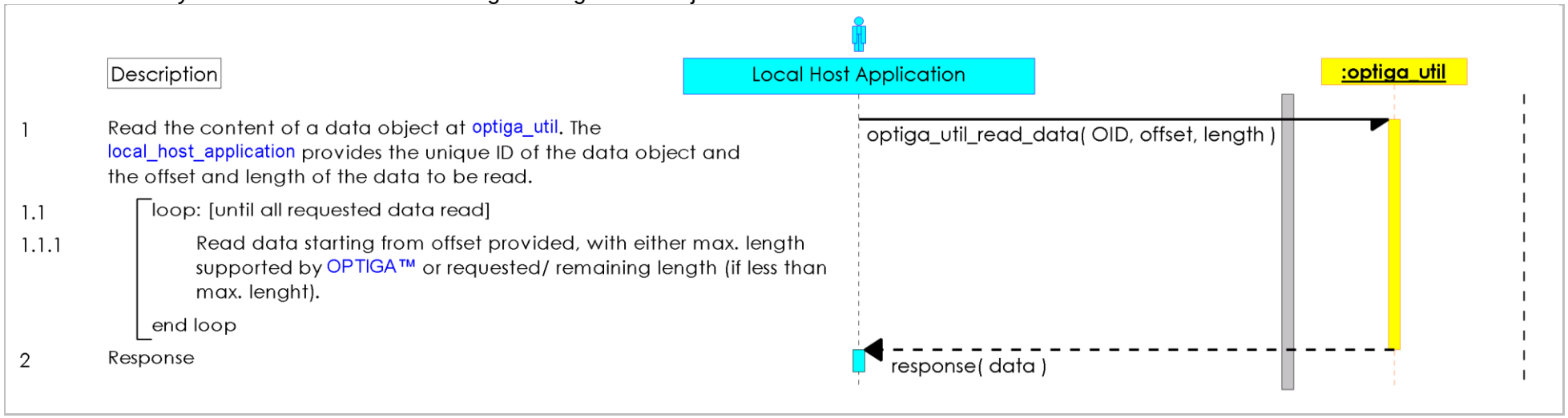

Figure 2 - Use Case: Read General Purpose Data - data object

2.2.2 Use Case: Read General Purpose Data - metadata

The local_host_application intends to read the metadata of a data object maintained by the OPTIGA™.
The sequence diagram Use Case: Read General Purpose Data - metadata is provided to show the functions involved in reading the metadata of a data object. The function is performed atomic (no other invocation of the optiga_cmd module will interrupt the execution).

Supported Use Cases

Pre-condition:

- The *OPTIGA™* application is already launched
- The necessary access conditions for reading the target data object are satisfied.

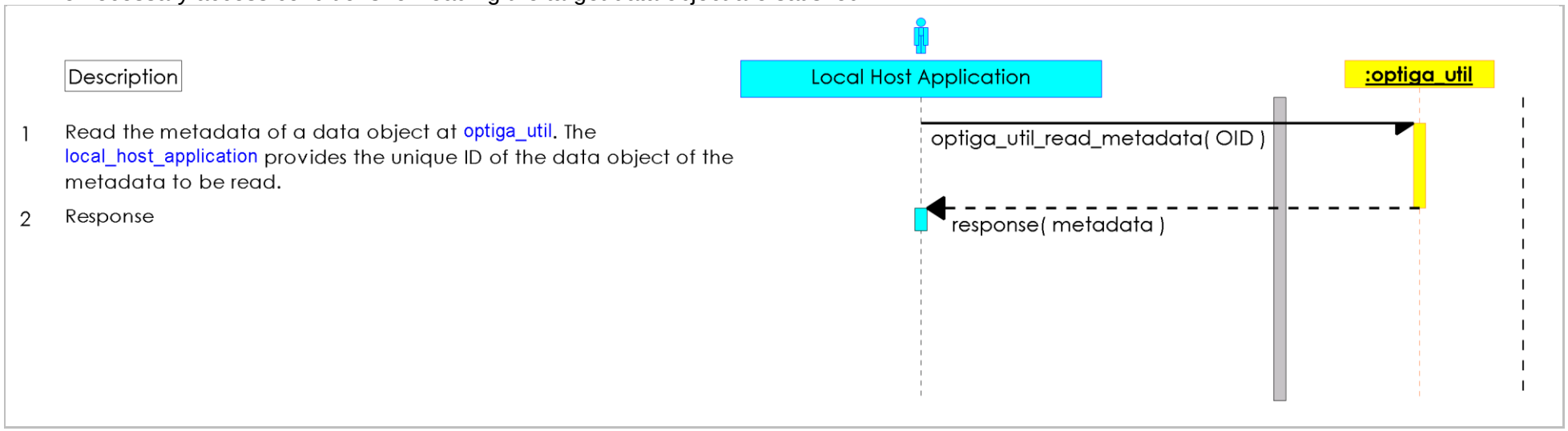

Figure 3 - Use Case: Read General Purpose Data - metadata

2.2.3 Use Case: Write General Purpose Data - data object

The local_host_application intends to update a data object maintained by the OPTIGA™.
The sequence diagram Use Case: Write General Purpose Data - data object is provided to show the functions involved in performing updating an data object by a single invocation of the optiga_cmd module. The function is performed atomic (no other invocation of the optiga_cmd module will interrupt the execution).

Pre-condition:

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

Post-condition:

- The target data object is updated

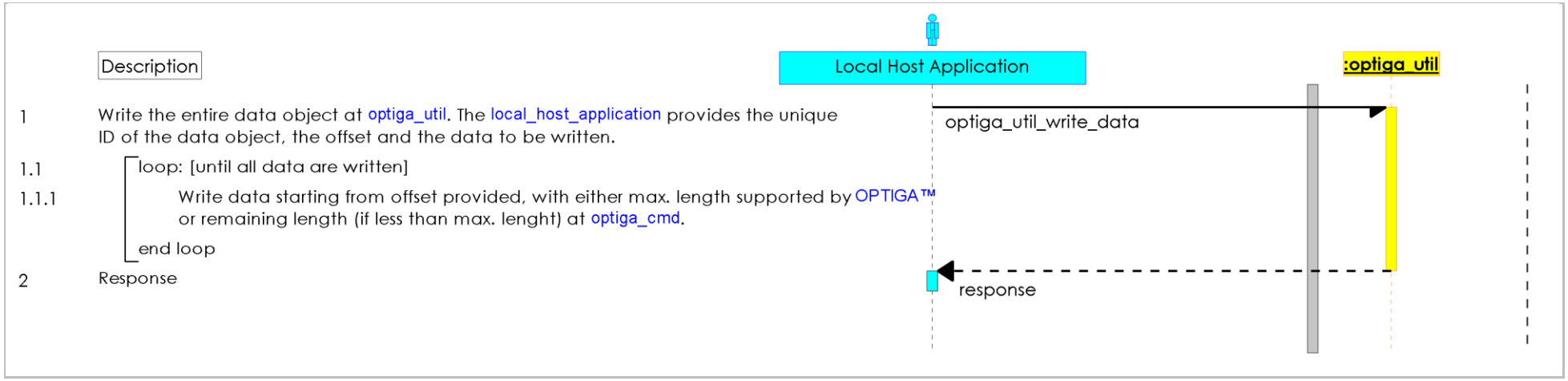

Figure 4 - Use Case: Write General Purpose Data - data object

2.2.4 Use Case: Write General Purpose Data - metadata

The local_host_application intends to update the metadata associated to a data object which is maintained by the OPTIGA™.
The sequence diagram Use Case: Write General Purpose Data - metadata is provided to show the functions involved in updating metadata associated to a data object.

Pre-condition:

- The *OPTIGA™* application is already launched
- The necessary access conditions for writing the metadata associated with a data object are satisfied.

Post-condition:

- The metadata associated to the target data object are updated

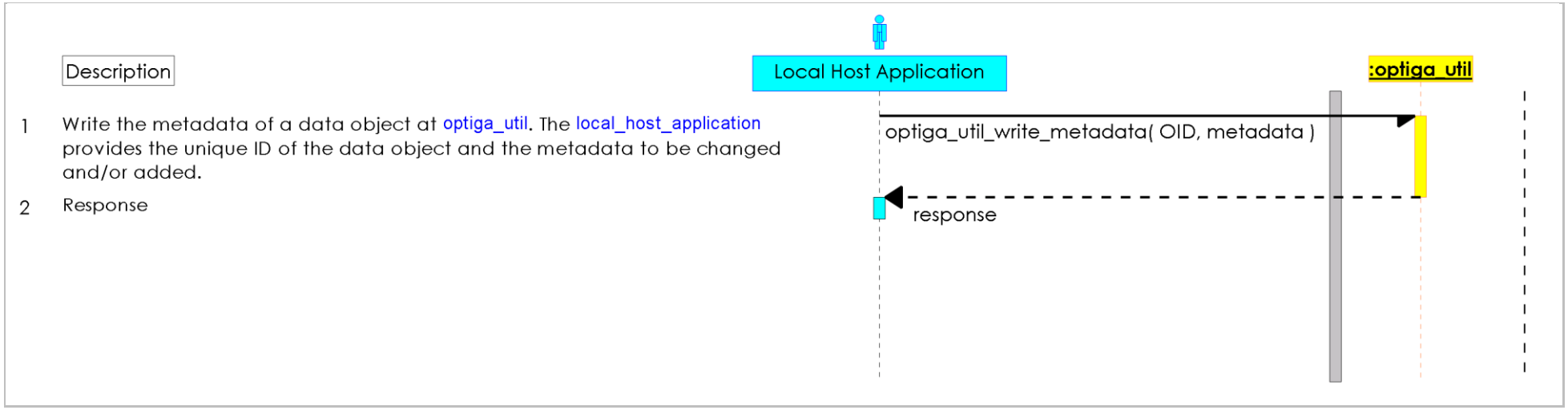

Figure 5 - Use Case: Write General Purpose Data - metadata

2.2.5 Use Case: Integrity Protected Update of Data Object

The Management Server intends to update a data object (e.g. a Trust Anchor) with integrity protected. The Management Server provides an update data set, which gets forwarded to the OPTIGA™. The OPTIGA™ checks and removes the protection and upon success updates the target data object.

Pre-condition(s):

- The OPTIGA™ application is already launched
- The Trust Anchor for management purpose is well formatted and available at the OPTIGA™.
- The access conditions of the target data object allow protected updating.

Post-condition:

- The target data object is updated.

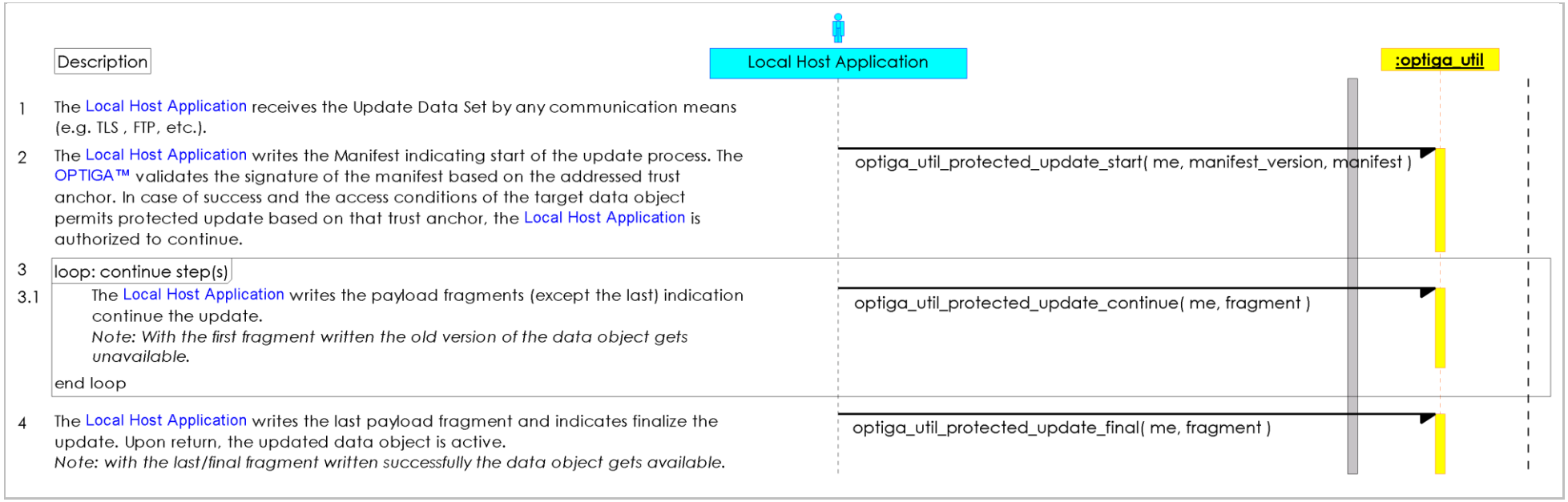

Figure 6 - Use Case: Integrity Protected Update of Data Object

2.2.6 Use Case: Local "data-at-rest" and "data-in-transit" protection

The data-at-rest protection use case is required in case the host needs to protect data against access by any third party. This Use Case: Local "data-at-rest" and "data-in-transit" protection is about protecting low volume of data at the Host. For that purpose OPTIGA™ stores the data at its embedded data store. The data store need to be configured in a way the protection (OPTIGA™ Shielded Connection) of data being transferred between data object and host is enforced by the respective access conditions defined as part of the metadata associated with the target data objects.

Pre-condition:

- Each data object to protect data at rest are configured in a way writing (AC CHA = Conf(0xE140)) or reading (AC RD = Conf(0xE140)) it must apply protection by OPTIGA™ Shielded Connection.

Post-condition:

- The plain payload read or written was traveling on the I2C bus confidentiality protected.

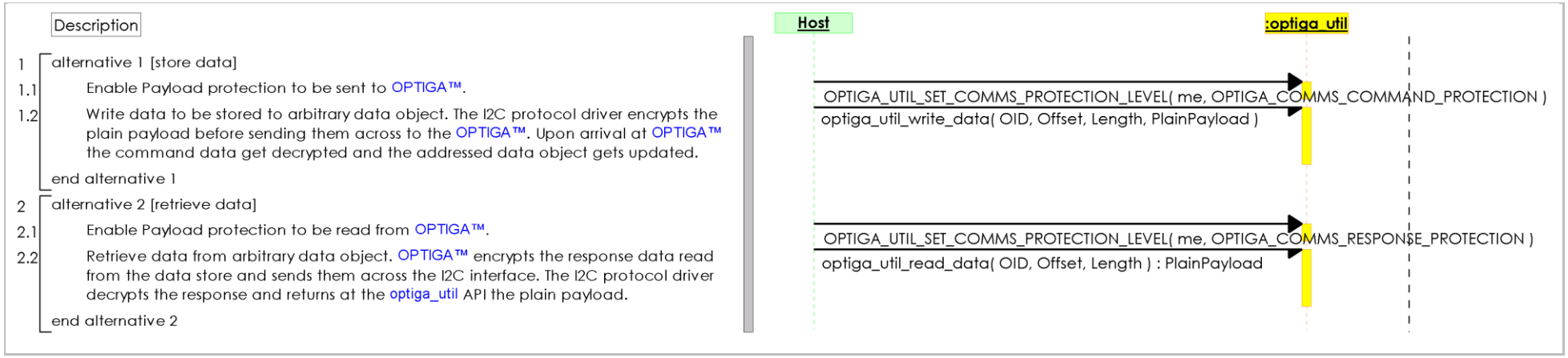

Figure 7 - Use Case: Local "data-at-rest" and "data-in-transit" protection

2.3 Sequence Diagrams utilizing cryptographic toolbox functionality

2.3.1 Use Case: Mutual Authentication establish session -toolbox- (TLS-Client)

The Server and the Client (on behalf of the User), which incorporates the OPTIGA™, intend to proof the authenticity of each other. Both the Server and OPTIGA™ 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, and the Client is authenticated as well.

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™. In the latter case the intermediate results shall be returned by OPTIGA™ and provided for continuing the hashing with further commands.

Pre-conditions:

- The OPTIGA™ application is already launched
- The public key pairs for authentication purpose and public key certificates are properly installed at the OPTIGA™.
- 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™.

Supported Use Cases

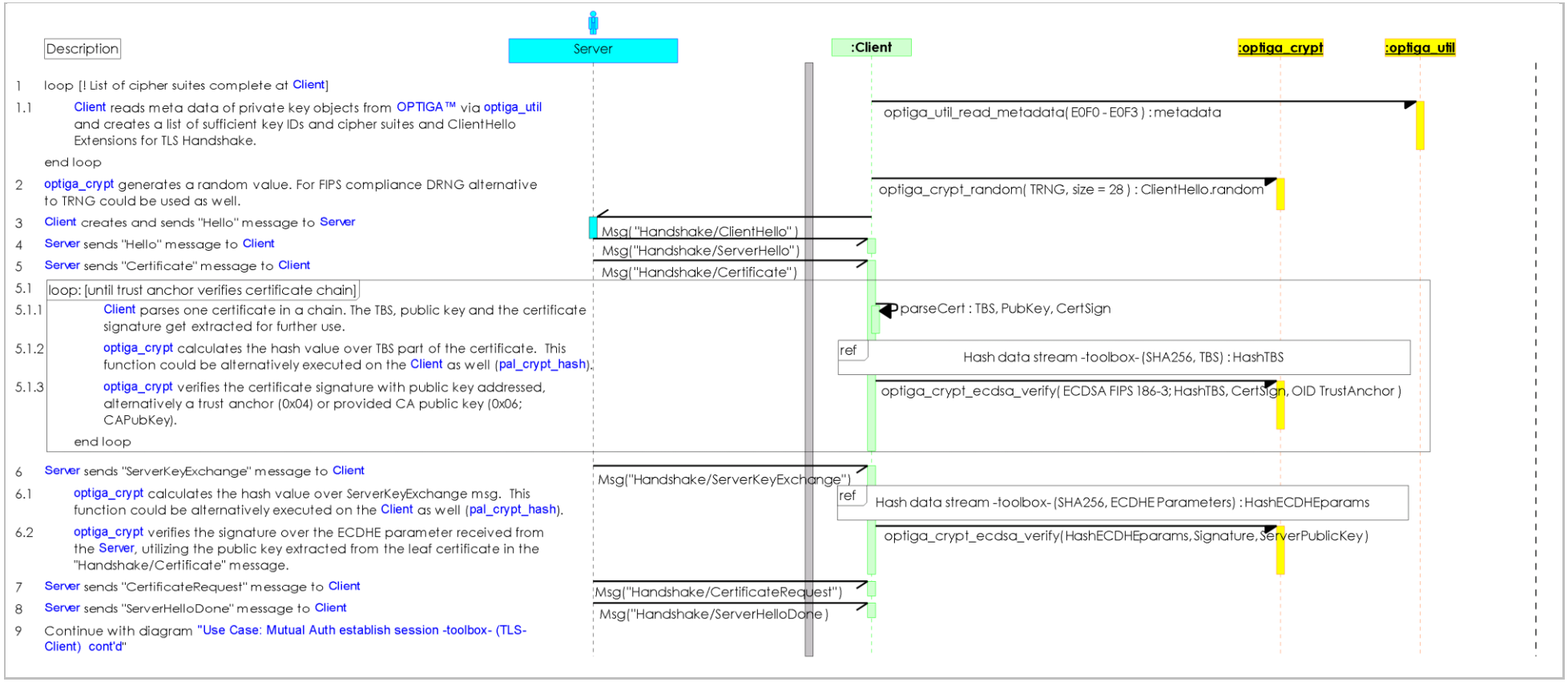

Figure 8 - Use Case: Mutual Authentication establish session -toolbox- (TLS-Client)

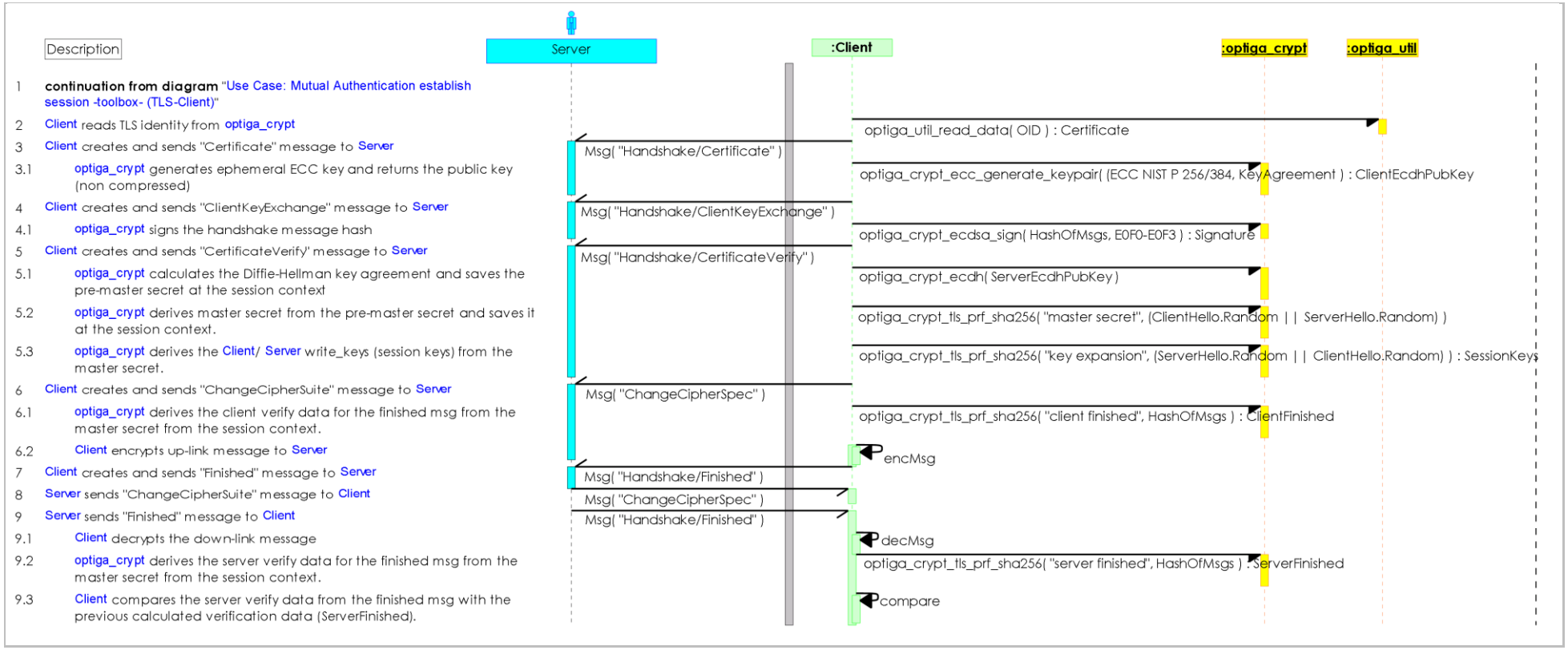

Figure 9 - Use Case: Mutual Auth establish session -toolbox- (TLS-Client) cont'd

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

The Server and the Client (on behalf of the User), which incorporates the OPTIGA™, intend to resume an established session. Both the Server and OPTIGA™ 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™. In the latter case the intermediate results shall be returned by OPTIGA™ and provided for continuing the hashing with further commands.

Supported Use Cases

Pre-conditions:

- The OPTIGA™ 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 key(s).
- The Client is able to hash all handshake messages without the help of OPTIGA™.

Post-condition:

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

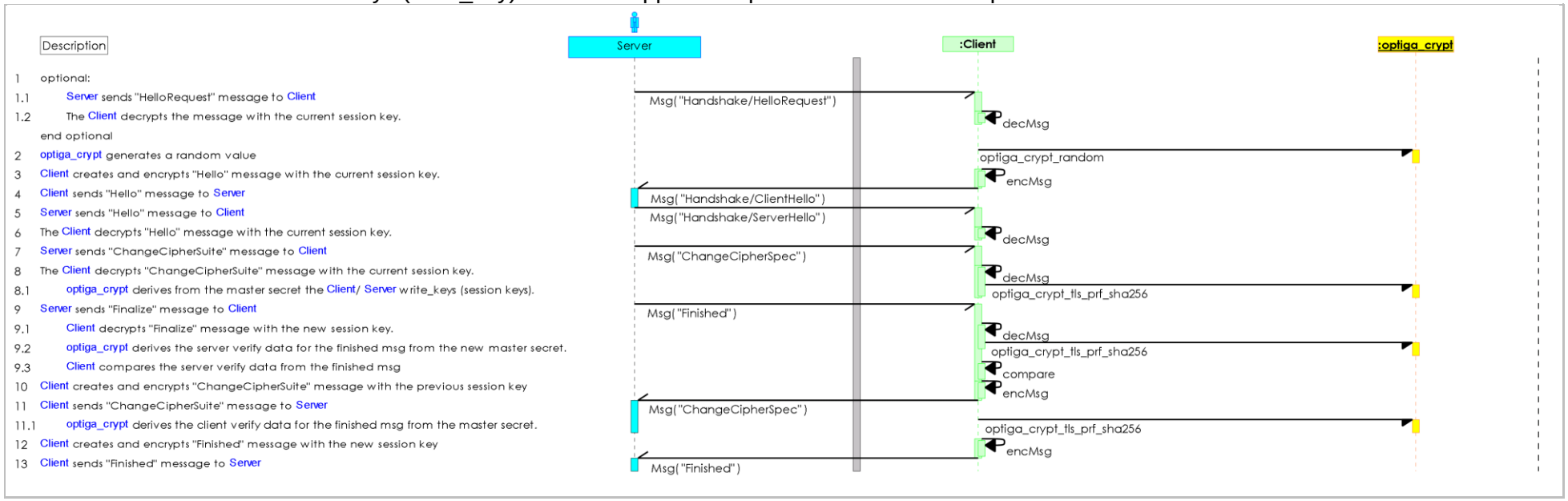

Figure 10 - Use Case: Abbreviated Handshake -toolbox- (TLS-Client)

2.3.3 Use Case: Host FW Update -toolbox-

The Host intends 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 functions involved in performing.

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.

Supported Use Cases

Post-condition:

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

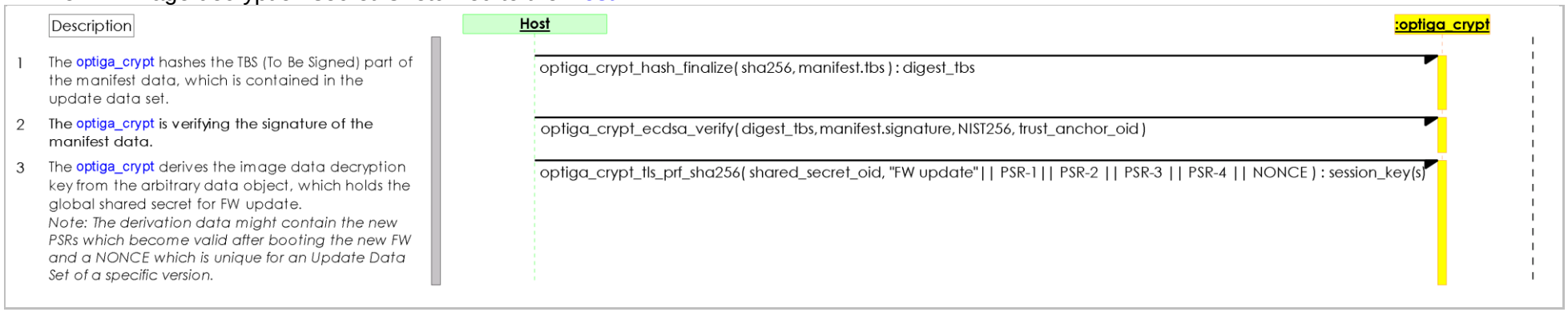

Figure 11 - Use Case: Host FW Update -toolbox-

2.3.4 Use Case: Local "data-at-rest" protection -toolbox-

The data-at-rest protection use case is required in case the host needs to protect data against access by any third party. This Use Case: Local "data-at-rest" protection -toolbox- is about encryption high volume of data at the Host. For that purpose OPTIGA™ and Host establishing a unique key for local data encryption/ decryption. It generates a random secret once and is using it for lifetime to derive the actual secret used for encrypt/decrypt of local data by the Host.

Pre-condition:

- Either there is at least one arbitrary data object (Data Structure Arbitrary data object) of type 3 (in this example OID = 0xF1D1) available at the OPTIGA™ to save the unique secret for local encryption.
- Or the unique secret for local encryption is already saved and locked.
- The OPTIGA™ Shielded connection is activated (presentation layer of the I2C protocol [IFX_I2C] is present) and is recommended to be used for all commands and responses carrying secret data.

Post-condition:

- The local secret for encryption is known by the Host

Supported Use Cases

Figure 12 - Use Case: Local "data-at-rest" protection -toolbox-

2.3.5 Use Case: Pair OPTIGA™ with Host (Pre-Shared Secret based)

The OPTIGA™ and Host establishing a protected communication channel, which provides integrity and confidentiality for data exchanged between both entities. This Use Case: Pair OPTIGA™ with Host (Pre-Shared Secret based) is about generation and exchange of those assets during production of the customer solution. The solution comprises at least of the Host and the OPTIGA™.

Pre-condition(s):

- The target data object is not locked (LcsO < op).

Post-conditions(s):

- The pre-shared secret is available and locked (read/write = NEV or read = NEV).

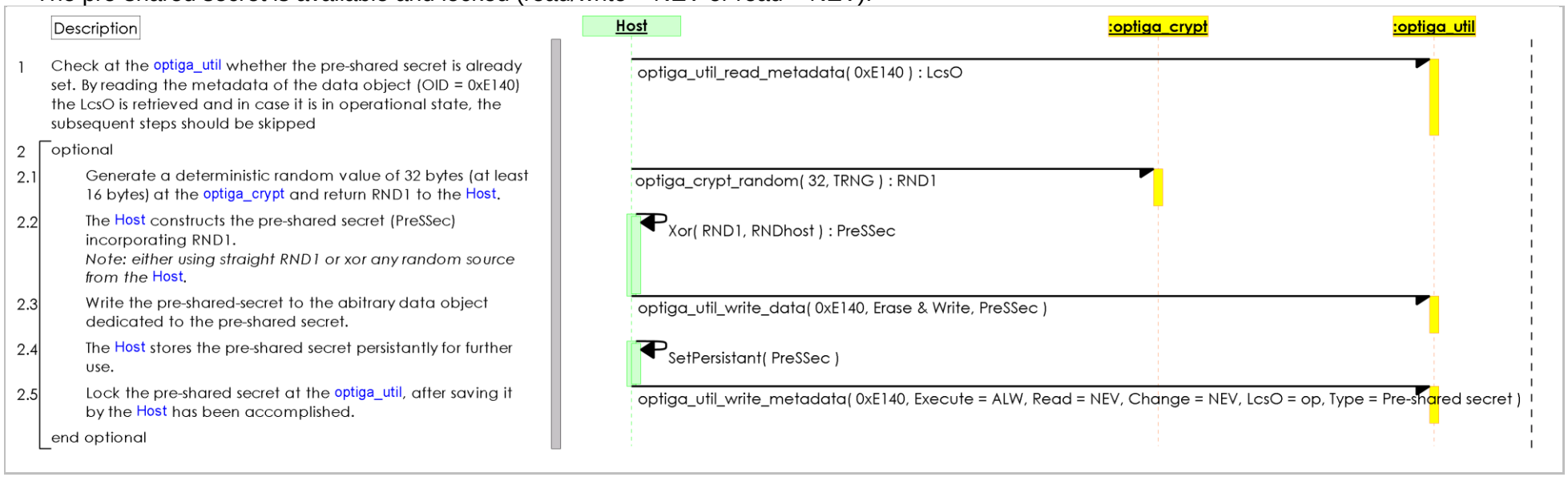

Figure 13 - Use Case: Pair OPTIGA™ with Host (Pre-Shared Secret based)

2.3.6 Use Case: Verified Boot -toolbox-

The Host system intends to verify the integrity of the host software image. The verification shall be done based on a public key signature scheme. The components involved are the immutable_boot_block, the primary_boot_loader, some further platform specific components integrated in the boot

process and the OPTIGA™.

Pre-conditions:

- The OPTIGA™ application is already launched
- The Trust Anchor for verifying the image hash is properly installed at the OPTIGA™ .

Post-condition:

- The Host software image is proven being integrity correct.

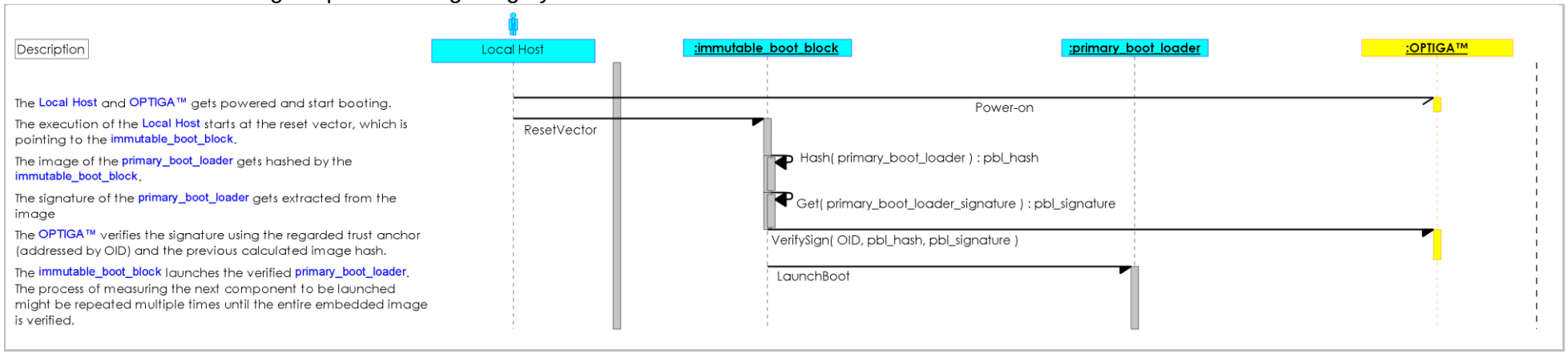

Figure 14 - Use Case: Verified Boot -toolbox-

2.3.7 Use Case: Update Platform Binding Secret during runtime (Pre-Shared Secret based)

This Use Case: Update Platform Binding Secret during runtime (Pre-Shared Secret based) is about generation and exchange of Platform Binding Secret using Shielded Connection during runtime. The solution comprises at least of the Host and the OPTIGA™.

Pre-condition(s):

- The Pairing of OPTIGA™ and Host (Pre-Shared secret based) is performed.
- The change access condition of Platform Binding Secret is enabled for the runtime protected update using Shielded Connection (e.g. CONF (E140)).

Post-conditions(s):

- The pre-shared secret is updated with the new secret.

Supported Use Cases

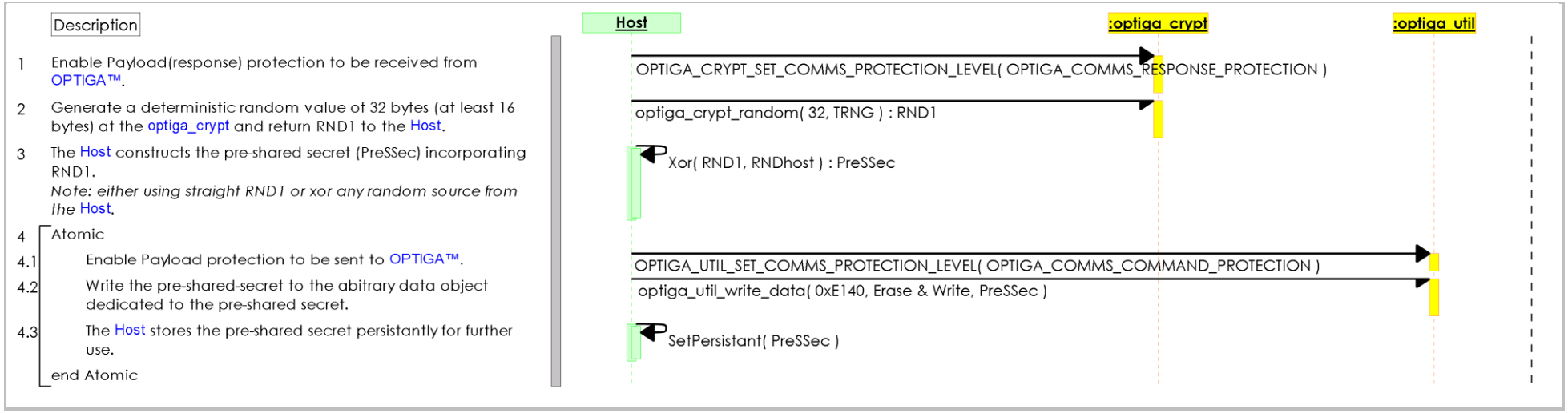

Figure 15 - Use Case: Update Platform Binding Secret during runtime (Pre-Shared Secret based)

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™ solution. The target platforms for those enabler components are embedded systems, Linux and Windows.
The class diagram OPTIGA Trust Enabler Software Overview shows the OPTIGA Trust Family Enabler SW Architecture V2 containing its main function blocks.
The color coding provides information of whether the function blocks are *platform agnostic*, *platform specific*, *platform ported* or *third party*.

- *Platform agnostic* components (yellow) could be reused on any target platform with just compiling the source code for a specific platform. The code is endian aware.
- *Platform specific* components (dark blue) are available for a specific platform. The component could be provided as source or in binary format.
- *Platform ported* components (green) are used to connect platform specific and platform agnostic components. Those components exposing a platform agnostic API and calling platform specific APIs.
- *Third Party* components (light blue) need to be ported to platform agnostic APIs.

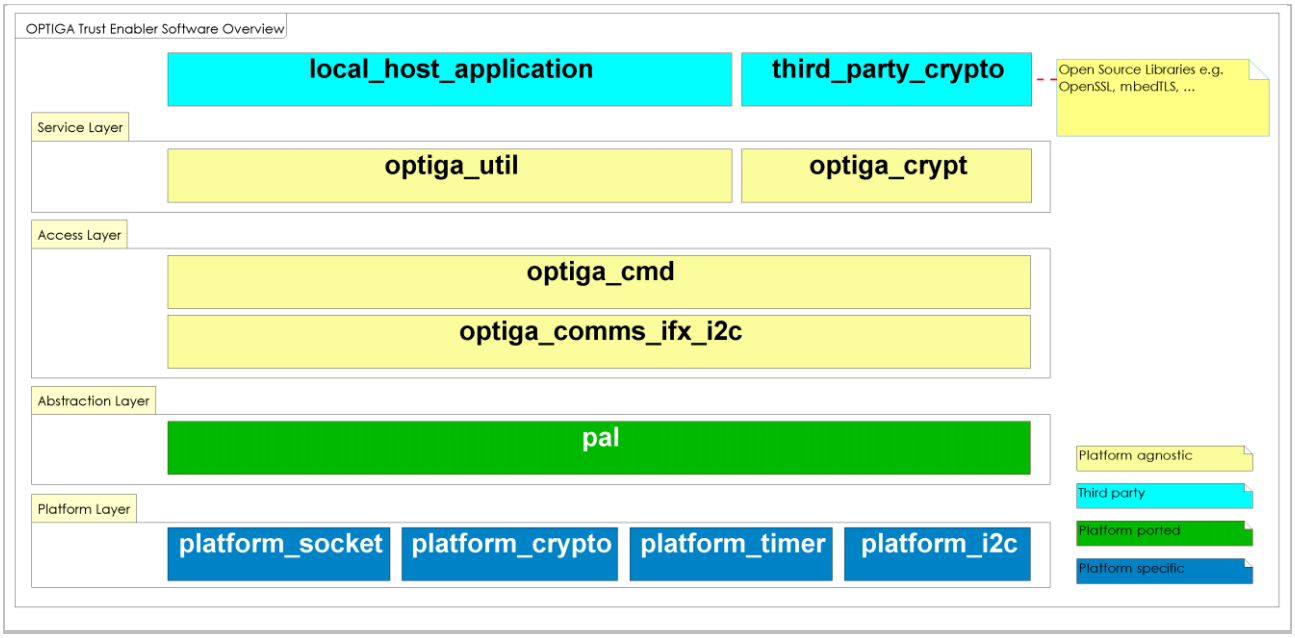

Figure 16 - OPTIGA Trust Enabler Software Overview

3.1 Service Layer Decomposition

The Service Layer Decomposition diagram shows the components providing the services at the main application interface.

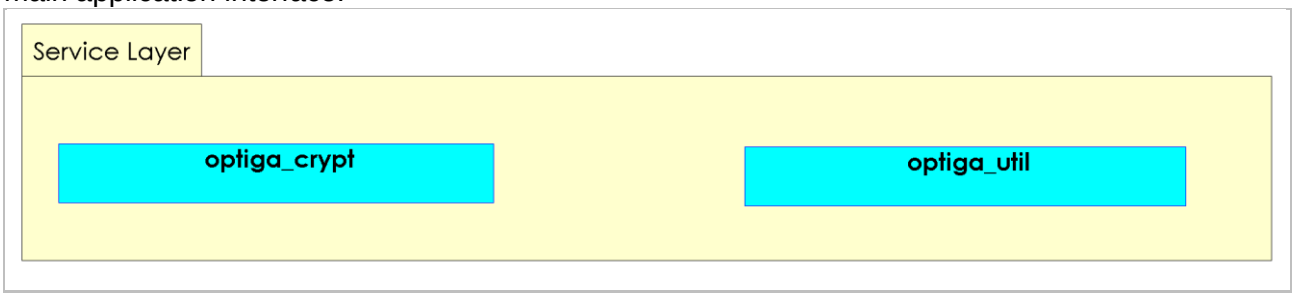

Figure 17 - Service Layer Decomposition

3.1.1 optiga_crypt

The optiga_crypt module provides cryptographic tool box functionality with the following characteristics:

- Multiple instances could be created using optiga_crypt_create to allow concurrent access to the tool box.
- Usage of the optiga_cmd module to interact with the OPTIGA™.
- The optiga_cmd module might get locked for some consecutive invocations, which need to be executed atomic (strict).

3.1.1.1 optiga_crypt_create

optiga_crypt_create	
Description	This operation creates an instance of optiga_crypt. The volatile memory gets allocated and initialized. This operation inherently creates an instance of optiga_cmd if available due to solution constraints (the number of optiga_cmd instances might be limited). Some of the optiga_crypt operations needs session context in OPTIGA™. In such a case, the instance of optiga_cmd of the respective optiga_crypt instances acquires one of the OPTIGA™ sessions before invoking the actual operation. For the protected communication (Shielded Connection) between Host and OPTIGA™, • This instance needs to be updated before invoking the respective optiga_util operations using OPTIGA_CRYPT_SET_COMMS_PROTECTION_LEVEL. • Based on the chosen protection level, the access layer activates the Shielded Connection between Host and OPTIGA™. These settings get automatically reset to OPTIGA_COMMS_DEFAULT_PROTECTION_LEVEL once after the operation is invoked. • The above specified settings for the shielded connection are applicable only if OPTIGA_COMMS_SHIELDED_CONNECTION is enabled/defined in configurations in the provided implementation. • The precondition for the Shielded Connection is pairing of OPTIGA™ and Local Host. The sequence of pairing is depicted in Use Case: Pair OPTIGA™ with Host (Pre-Shared Secret based).
Signature	optiga_crypt_create (in optiga_instance_id : uint8_t, in handler : callback_handler_t, in caller_context : void *) : p_optiga_crypt_t

optiga_crypt_create	
Parameters	`in optiga_instance_id : uint8_t` Instance id of the OPTIGA™ integrated on the host platform. The default value is 0.
	`in handler : callback_handler_t` Pointer to the callback function.
	`in caller_context : void *` Pointer to the caller context.

3.1.1.2 optiga_crypt_random

optiga_crypt_random	
Description	This operation generates random data using OPTIGA™.
Signature	optiga_crypt_random (in me : p_optiga_crypt_t, in rng_type : optiga_rng_type_t, inout random_data : uint8_t *, in random_data_length : uint16_t) : optiga_lib_status_t
Parameters	`in me : p_optiga_crypt_t` Pointer to one instance of optiga_crypt.
	`in rng_type : optiga_rng_type_t` Type (optiga_rng_type_t) of random to be generated.
	`inout random_data : uint8_t *` Pointer to a data buffer to store the random data generated. The caller must ensure that the size of the buffer must be of at least the length of random data (random_data_length) to be generated.
	`in random_data_length : uint16_t` Length of random data to be generated.

3.1.1.3 optiga_crypt_hash_start

optiga_crypt_hash_start	
Description	This operation initializes OPTIGA™ to hash the data further using optiga_crypt_hash_update.
Signature	optiga_crypt_hash_start (in me : p_optiga_crypt_t, inout hash_ctx : p_optiga_hash_context_t) : optiga_lib_status_t
Parameters	`in me : p_optiga_crypt_t` Pointer to one instance of optiga_crypt.
	`inout hash_ctx : p_optiga_hash_context_t` Pointer to the hash context optiga_hash_context_t.

3.1.1.4 optiga_crypt_hash_update

optiga_crypt_hash_update	
Description	This operation performs the hashing for the given data (could be either host or referring to a readable data object from OPTIGA™) and updates the hash context using OPTIGA™.

optiga_crypt_hash_update	
Signature	optiga_crypt_hash_update (in me : p_optiga_crypt_t, inout hash_ctx : p_optiga_hash_context_t, in source_of_data_to_hash : uint8_t, in data_to_hash : const void *) : optiga_lib_status_t
Parameters	`in me : p_optiga_crypt_t` Pointer to one instance of optiga_crypt.
	`inout hash_ctx : p_optiga_hash_context_t` Pointer to the hash context optiga_hash_context_t.
	`in source_of_data_to_hash : uint8_t` Specifies the source of data to be hashed which could be data from host or data from OPTIGA™ data objects.
	`in data_to_hash : const void *` Pointer to data to hash. If source_of_data_to_hash specifies the data is from host, data_to_hash is a pointer to a structure hash_data_from_host_t. If source_of_data_to_hash specifies the data is from OPTIGA™, data_to_hash is a pointer to a structure hash_data_in_optiga_t.

3.1.1.5 optiga_crypt_hash_finalize

optiga_crypt_hash_finalize	
Description	This operation finalizes the hash.
Signature	optiga_crypt_hash_finalize (in me : p_optiga_crypt_t, in hash_ctx : p_optiga_hash_context_t, inout hash_output : uint8_t *) : optiga_lib_status_t
Parameters	`in me : p_optiga_crypt_t` Pointer to one instance of optiga_crypt.
	`in hash_ctx : p_optiga_hash_context_t` Pointer to the hash context optiga_hash_context_t.
	`inout hash_output : uint8_t *` Pointer to a buffer to store the hash generated with a size of expected hash output length.

3.1.1.6 optiga_crypt_ecc_generate_keypair

optiga_crypt_ecc_generate_keypair	
Description	This operation generates ECC key pair using OPTIGA™. The generated private key could be either stored in OPTIGA™ (static private key from key store or volatile session based) or exported to host. In case of session based, the instance internally acquires one of the OPTIGA™ sessions before invoking the actual operation.
Signature	optiga_crypt_ecc_generate_keypair (in me : p_optiga_crypt_t, in curve_id : optiga_ecc_curve_t, in key_usage : uint8_t, in export_private_key : bool_t, inout private_key : void *, inout public_key : uint8_t *, inout public_key_length : uint16_t *) : optiga_lib_status_t
Parameters	`in me : p_optiga_crypt_t` Pointer to one instance of optiga_crypt

optiga_crypt_ecc_generate_keypair	
	`in curve_id : optiga_ecc_curve_t` Type of curve optiga_ecc_curve_t (e.g. ECC NIST P 256, or ECC NIST P 384)
	`in key_usage : uint8_t` Specifies the usage (optiga_key_usage_t) of the key generated
	`in export_private_key : bool_t` Specifies whether the private key to be exported to the host or not. = 0, stores the generated private key with in OPTIGA™. In this case, the private_key is to be a pointer to an OID of the private key (optiga_key_id_t). = 1 or non-zero, OPTIGA™ exports the generated private key. In this case, the private_key is a pointer to a buffer with at least a minimum length required based on curve_id chosen.
	`inout private_key : void *` Pointer to the private key reference (based on export_private_key parameter) In case of export_private_key = True, The examples for the format of the exported ECC private key is shown in ECC Private Key.
	`inout public_key : uint8_t *` Pointer to a buffer to store the public key. The examples for the format of the exported public key is shown in ECC Public Key.
	`inout public_key_length : uint16_t *` Pointer to the length of the public key buffer provided. This value gets updated with length of the public key information returned.

3.1.1.7 optiga_crypt_ecdsa_sign

optiga_crypt_ecdsa_sign	
Description	This operation generates signature (ECDSA) using a private key from OPTIGA™. The private key could be either from a static key store or acquired session.
Signature	optiga_crypt_ecdsa_sign (in me : p_optiga_crypt_t, in digest : const uint8_t *, in digest_length : uint8_t, in private_key : optiga_key_id_t, inout signature : uint8_t *, inout signature_length : uint16_t *) : optiga_lib_status_t
Parameters	`in me : p_optiga_crypt_t` Pointer to one instance of optiga_crypt
	`in digest : const uint8_t *` Pointer to the digest
	`in digest_length : uint8_t` Length of digest provided
	`in private_key : optiga_key_id_t` OID of the private key from OPTIGA™ to be used to sign the digest
	`inout signature : uint8_t *` Pointer to the signature. The examples for the format of the ECDSA signature returned by OPTIGA™ are provided in ECDSA Signature.

optiga_crypt_ecdsa_sign	
	`inout signature_length : uint16_t *` Length of the signature buffer provided. This parameter gets updated with the length of the data stored in signature after the completion of operation.

3.1.1.8 optiga_crypt_ecdsa_verify

optiga_crypt_ecdsa_verify	
Description	This operation verifies the signature (ECDSA) using OPTIGA™. The public key could be either sourced from host or referring to OID (data object which holds a certificate) in OPTIGA™.
Signature	optiga_crypt_ecdsa_verify (in me : p_optiga_crypt_t, in digest : const uint8_t *, in digest_length : uint8_t, in signature : const uint8_t *, in signature_length : uint16_t, in public_key_source_type : uint8_t, in public_key : const void *) : optiga_lib_status_t
Parameters	`in me : p_optiga_crypt_t` Pointer to one instance of optiga_crypt
	`in digest : const uint8_t *` Pointer to a digest
	`in digest_length : uint8_t` Length of digest provided
	`in signature : const uint8_t *` Pointer to the signature. The examples for the format of the ECDSA signature to be sent to OPTIGA™ are provided in ECDSA Signature.
	`in signature_length : uint16_t` Length of the signature
	`in public_key_source_type : uint8_t` To specify the source of public key (e.g. host or an OID from OPTIGA™ which holds a certificate).
	`in public_key : const void *` Pointer to the public key details. If the Public key is from host, This must be a pointer to a public key public_key_from_host_t If the public key is from a certificate OID from OPTIGA™, this must be a pointer to a uint16_t which holds a certificate OID. The examples for the format of public key (from host) are provided in ECC Public Key.

3.1.1.9 optiga_crypt_ecdh

optiga_crypt_ecdh	
Description	This operation generates shared secret. OPTIGA™ performs ECDH operation using the referred private key and provided public key. • Here the private key is from OPTIGA™ referring to a static key store OID or

optiga_crypt_ecdh	
	session based. In case of session based, the private key is used from the session acquired. • The public key has to be sourced from host. • The generated shared secret can be either exported to the host or stored in OPTIGA™'s session acquired by the respective optiga_crypt instance. The session gets acquired internally if not acquired already.
Signature	optiga_crypt_ecdh (in me : p_optiga_crypt_t, in private_key : optiga_key_id_t, in public_key : const p_public_key_from_host_t, in export_to_host : bool_t, inout shared_secret : uint8_t *) : optiga_lib_status_t
Parameters	`in me : p_optiga_crypt_t` Pointer to one instance of optiga_crypt
	`in private_key : optiga_key_id_t` OID of the private key from OPTIGA™ to be used.
	`in public_key : const p_public_key_from_host_t` Public key information The examples for the format of public key (from host) are provided in ECC Public Key.
	`in export_to_host : bool_t` Specifies whether the generated shared secret to be exported to the host or not. = 0, stores the generated secret in the acquired instance with in OPTIGA™. = 1 or non-zero, OPTIGA™ exports the generated shared secret. In this case, the shared_secret is a pointer to a buffer with at least a minimum length required based on curve.
	`inout shared_secret : uint8_t *` Pointer to the shared secret. In case of storing with in OPTIGA™, this parameter must be NULL.

3.1.1.10 optiga_crypt_rsa_generate_keypair

optiga_crypt_rsa_generate_keypair	
Description	This operation generates RSA key pair using OPTIGA™. The generated private key could be either stored in OPTIGA™ (static private key from key store) or exported to host.
Signature	optiga_crypt_rsa_generate_keypair (in me : p_optiga_crypt_t, in key_type : optiga_rsa_key_type_t, in key_usage : uint8_t, in export_private_key : bool_t, inout private_key : void *, inout public_key : uint8_t *, inout public_key_length : uint16_t *) : optiga_lib_status_t
Parameters	`in me : p_optiga_crypt_t` Pointer to one instance of optiga_crypt
	`in key_type : optiga_rsa_key_type_t` RSA key type (e.g. 1024, 2048,...) as specified in optiga_rsa_key_type_t.
	`in key_usage : uint8_t` Specifies the usage (optiga_key_usage_t) of the key generated. This parameter is not applicable in case of exporting the private key to host.

optiga_crypt_rsa_generate_keypair	
	`in export_private_key : bool_t` Specifies whether the private key to be exported to the host or not. = TRUE (1) or non-zero, OPTIGA™ exports the generated private key. In this case, the private_key is a pointer to a buffer with at least a minimum length required based on key_type chosen. = FALSE (0), stores the generated private key with in OPTIGA™. In this case, the private_key is to be pointer to the private key OID (optiga_key_id_t).
	`inout private_key : void *` Pointer to the private key reference (based on export_private_key).
	`inout public_key : uint8_t *` Pointer to a buffer to store the public key.
	`inout public_key_length : uint16_t *` Pointer to the length of the public key buffer provided. This value gets updated with length of the public key information returned by OPTIGA™.

3.1.1.11 optiga_crypt_rsa_sign

optiga_crypt_rsa_sign	
Description	This operation generates signature using RSA based static private key from key store (optiga_key_id_t) in OPTIGA™.
Signature	optiga_crypt_rsa_sign (in me : p_optiga_crypt_t, in signature_scheme : optiga_rsa_signature_scheme_t, in digest : const uint8_t *, in digest_length : uint8_t, in private_key : optiga_key_id_t, inout signature : uint8_t *, inout signature_length : uint16_t *, in salt_length : uint16_t) : optiga_lib_status_t
Parameters	`in me : p_optiga_crypt_t` Pointer to one instance of optiga_crypt
	`in signature_scheme : optiga_rsa_signature_scheme_t` Scheme to be used to generate the signature. Refer optiga_rsa_signature_scheme_t for scheme types.
	`in digest : const uint8_t *` Pointer to the digest
	`in digest_length : uint8_t` Length of digest provided
	`in private_key : optiga_key_id_t` OID of the RSA static private key (optiga_key_id_t) from OPTIGA™ key store to be used to sign the digest.
	`inout signature : uint8_t *` Pointer to the signature. The examples for the format of signature returned by OPTIGA™ are provided in RSA Signature.
	`inout signature_length : uint16_t *` Length of the signature buffer provided. This parameter gets updated with the length of the data stored in signature after the completion of operation.
	`in salt_length : uint16_t` Reserved for future use. The value provided is ignored.

3.1.1.12 optiga_crypt_rsa_verify

optiga_crypt_rsa_verify	
Description	This operation verifies the signature using OPTIGA™. The RSA public key could be either sourced from host or referring to OID (data object which holds a certificate) in OPTIGA™.
Signature	optiga_crypt_rsa_verify (in me : p_optiga_crypt_t, in signature_scheme : optiga_rsa_signature_scheme_t, in digest : const uint8_t *, in digest_length : uint8_t, in signature : const uint8_t *, in signature_length : uint16_t, in public_key_source_type : uint8_t, in public_key : const void *, in salt_length : uint16_t) : optiga_lib_status_t
Parameters	`in me : p_optiga_crypt_t` Pointer to one instance of optiga_crypt
	`in signature_scheme : optiga_rsa_signature_scheme_t` Scheme to be used to verify the signature. Refer optiga_rsa_signature_scheme_t for scheme types.
	`in digest : const uint8_t *` Pointer to a digest
	`in digest_length : uint8_t` Length of digest provided
	`in signature : const uint8_t *` Pointer to the signature. The examples for the format of signature are provided in RSA Signature.
	`in signature_length : uint16_t` Length of the signature
	`in public_key_source_type : uint8_t` To specify the source of public key (e.g. host or an OID from OPTIGA™ which holds a certificate).
	`in public_key : const void *` Pointer to the public key details. If the Public key is from host, This must be a pointer to a public key(public_key_from_host_t) If the public key is from a data object (holds a certificate) from OPTIGA™, this must be a pointer to a uint16_t which holds an OID. The examples for the format of public key (from host) are provided in RSA Public Key.
	`in salt_length : uint16_t` Reserved for future use. The value provided is ignored.

3.1.1.13 optiga_crypt_rsa_encrypt_message

optiga_crypt_rsa_encrypt_message	
Description	This operation encrypts the message or data provided using OPTIGA™. The RSA public key could be either sourced from host or referring to OID (data object

optiga_crypt_rsa_encrypt_message	
	which holds a certificate) in OPTIGA™. The message_length that can be encrypted is limited as per [RFC8017]. The caller of this operation has to take care of chaining of message if message length is more than supported in a single operation.
Signature	optiga_crypt_rsa_encrypt_message (in me : p_optiga_crypt_t, in encryption_scheme : optiga_rsa_encryption_scheme_t, in message : const uint8_t *, in message_length : uint16_t, in label : const uint8_t *, in label_length : uint16_t, in public_key_source_type : uint8_t, in public_key : const void *, inout encrypted_message : uint8_t *, inout encrypted_message_length : uint16_t *) : optiga_lib_status_t
Parameters	`in me : p_optiga_crypt_t` Pointer to one instance of optiga_crypt
	`in encryption_scheme : optiga_rsa_encryption_scheme_t` Scheme to be used to encrypt the given message. Refer optiga_rsa_encryption_scheme_t for scheme types.
	`in message : const uint8_t *` Pointer to a message to be encrypted
	`in message_length : uint16_t` Length of message provided
	`in label : const uint8_t *` This parameter is reserved for the future use and hence ignored.
	`in label_length : uint16_t` This parameter is reserved for the future use and hence ignored.
	`in public_key_source_type : uint8_t` To specify the source of public key (e.g. host or an OID from OPTIGA™ which holds a certificate).
	`in public_key : const void *` Pointer to the public key details. If the Public key is from host, This must be a pointer to a public key public_key_from_host_t If the public key is from a data object (holds a certificate) from OPTIGA™, this must be a pointer to a uint16_t which holds an OID. The examples for the format of public key (from host) are provided in RSA Public Key.
	`inout encrypted_message : uint8_t *` Pointer to the buffer to store the encrypted message.
	`inout encrypted_message_length : uint16_t *` Pointer to the length of the encrypted message buffer. This will be updated with the actual length of encrypted_message returned by OPTIGA™.

3.1.1.14 optiga_crypt_rsa_encrypt_session

optiga_crypt_rsa_encrypt_session	
Description	This operation encrypts the data from acquired session in OPTIGA™. The RSA public key could be either sourced from host or referring to OID (data object

optiga_crypt_rsa_encrypt_session	
	which holds a certificate) in OPTIGA™. If the shielded connection (OPTIGA_COMMS_SHIELDED_CONNECTION) is enabled, By default the optiga_cmd sends the command to OPTIGA™ with confidentiality protection.
Signature	optiga_crypt_rsa_encrypt_session (in me : p_optiga_crypt_t, in encryption_scheme : optiga_rsa_encryption_scheme_t, in label : const uint8_t *, in label_length : uint16_t, in public_key_source_type : uint8_t, in public_key : const void *, inout encrypted_message : uint8_t *, inout encrypted_message_length : uint16_t *) : optiga_lib_status_t
Parameters	`in me : p_optiga_crypt_t` Pointer to one instance of optiga_crypt
	`in encryption_scheme : optiga_rsa_encryption_scheme_t` Scheme to be used to encrypt the given message. Refer optiga_rsa_encryption_scheme_t for scheme types.
	`in label : const uint8_t *` This parameter is reserved for the future use and hence ignored.
	`in label_length : uint16_t` This parameter is reserved for the future use and hence ignored.
	`in public_key_source_type : uint8_t` To specify the source of public key (e.g. host or an OID from OPTIGA™ which holds a certificate).
	`in public_key : const void *` Pointer to the public key details. If the Public key is from host, This must be a pointer to a public key public_key_from_host_t If the public key is from a data object (holds a certificate) from OPTIGA™, this must be a pointer to a uint16_t which holds an OID. The examples for the format of public key (from host) are provided in RSA Public Key.
	`inout encrypted_message : uint8_t *` Pointer to a buffer to store the encrypted message.
	`inout encrypted_message_length : uint16_t *` Pointer to the length of the encrypted message buffer (encrypted_message). This will be updated with the actual length of encrypted_message returned by OPTIGA™.

3.1.1.15 optiga_crypt_rsa_decrypt_and_export

optiga_crypt_rsa_decrypt_and_export	
Description	This operation decrypts the provided encrypted message using a RSA private key from OPTIGA™ and exports the decrypted message to the host. The encrypted_message_length that can be decrypted is limited as per the specification. The caller of this operation has to take care of chaining of encrypted message if length is more than supported in a single operation.

optiga_crypt_rsa_decrypt_and_export	
	If the shielded connection is enabled, the decrypted data (message) is received with confidentiality protection from OPTIGA™.
Signature	optiga_crypt_rsa_decrypt_and_export (in me : p_optiga_crypt_t, in encryption_scheme : optiga_rsa_encryption_scheme_t, in encrypted_message : const uint8_t *, in encrypted_message_length : uint16_t, in label : const uint8_t *, in label_length : uint16_t, in private_key : optiga_key_id_t, inout message : uint8_t *, inout message_length : uint16_t *) : optiga_lib_status_t
Parameters	`in me : p_optiga_crypt_t` Pointer to one instance of optiga_crypt
	`in encryption_scheme : optiga_rsa_encryption_scheme_t` Scheme to be used to decrypt the given encrypted message. Refer optiga_rsa_encryption_scheme_t for scheme types.
	`in encrypted_message : const uint8_t *` Pointer to the encrypted message to be decrypted
	`in encrypted_message_length : uint16_t` Length of the encrypted_message buffer
	`in label : const uint8_t *` This parameter is reserved for the future use and hence ignored.
	`in label_length : uint16_t` This parameter is reserved for the future use and hence ignored.
	`in private_key : optiga_key_id_t` OID of the RSA static private key (optiga_key_id_t) from OPTIGA™ to be used to decrypt the encrypted_message.
	`inout message : uint8_t *` Pointer to a buffer to return the decrypted message.
	`inout message_length : uint16_t *` Pointer to the length of the message buffer. This will be updated with the actual length of message returned by OPTIGA™.

3.1.1.16 optiga_crypt_rsa_decrypt_and_store

optiga_crypt_rsa_decrypt_and_store	
Description	This operation decrypts the provided encrypted_message using the referred RSA static private key from OPTIGA™ and stores message in the acquired session. This operation acquires a session if not already acquired.
Signature	optiga_crypt_rsa_decrypt_and_store (in me : p_optiga_crypt_t, in encryption_scheme : optiga_rsa_encryption_scheme_t, in encrypted_message : const uint8_t *, in encrypted_message_length : uint16_t, in label : const uint8_t *, in label_length : uint16_t, in private_key : optiga_key_id_t) : optiga_lib_status_t
Parameters	`in me : p_optiga_crypt_t` Pointer to one instance of optiga_crypt
	`in encryption_scheme : optiga_rsa_encryption_scheme_t` Scheme to be used to decrypt the given encrypted message. Refer optiga_rsa_encryption_scheme_t for scheme types.
	`in encrypted_message : const uint8_t *`

optiga_crypt_rsa_decrypt_and_store	
	Pointer to the encrypted message to be decrypted
	`in encrypted_message_length : uint16_t` Length of the encrypted_message buffer
	`in label : const uint8_t *` This parameter is reserved for the future use and hence ignored.
	`in label_length : uint16_t` This parameter is reserved for the future use and hence ignored.
	`in private_key : optiga_key_id_t` OID of the RSA static private key (optiga_key_id_t) from OPTIGA™ to be used to decrypt the encrypted_message.

3.1.1.17 optiga_crypt_rsa_generate_pre_master_secret

optiga_crypt_rsa_generate_pre_master_secret	
Description	This operation generates pre-master secret (optional data || random stream) for RSA key exchange and stores in the acquired session. • This operation acquires a session if not already acquired. • The minimum size of random stream is 8 bytes. • The maximum size of pre-master secret allowed to store in the acquired session is 48 bytes.
Signature	optiga_crypt_rsa_generate_pre_master_secret (in me : p_optiga_crypt_t, in optional_data : const uint8_t *, in optional_data_length : uint16_t, in pre_master_secret_length : uint16_t) : optiga_lib_status_t
Parameters	`in me : p_optiga_crypt_t` Pointer to one instance of optiga_crypt
	`in optional_data : const uint8_t *` Pointer to optional data which gets prepended to the generated secret. If optional_data is NULL, This will be ignored.
	`in optional_data_length : uint16_t` Length of optional data. optional_data_length = 0, if optional data is not required to be provided.
	`in pre_master_secret_length : uint16_t` Length of pre-master secret.

3.1.1.18 optiga_crypt_tls_prf_sha256

optiga_crypt_tls_prf_sha256	
Description	This operation derives shared secret or key using OPTIGA™. OPTIGA™ performs PRF SHA256 (as specified in TLS RFC5246) operation using the referred data object ID or session holding a secret. Here secret is input OID which holds the input secret. secret = 0x0000, instance of optiga_cmd of respective optiga_crypt instance uses the acquired session. In case of session based, the exported derived key from OPTIGA™ is received with confidentiality protection if the shielded connection is enabled.

optiga_crypt_tls_prf_sha256	
Signature	optiga_crypt_tls_prf_sha256 (in me : p_optiga_crypt_t, in secret : uint16_t, in label : const uint8_t *, in label_length : uint16_t, in seed : const uint8_t *, in seed_length : uint16_t, in derived_key_length : uint16_t, in export_to_host : bool_t, inout derived_key : uint8_t *) : optiga_lib_status_t
Parameters	`in me : p_optiga_crypt_t` Pointer to one instance of optiga_crypt
	`in secret : uint16_t` OID of input secret. Could be a secret from a data object or acquired session.
	`in label : const uint8_t *` Pointer to label which is typically a constant string. If no label is used, then this parameter could be NULL.
	`in label_length : uint16_t` Length of label
	`in seed : const uint8_t *` Pointer to seed
	`in seed_length : uint16_t` Length of seed
	`in derived_key_length : uint16_t` Length of key to be derived
	`in export_to_host : bool_t` Specifies whether the derived key/secret to be exported to the host or not. = 0, stores the derived key in the acquired session by the instance with in OPTIGA™. = 1 or non-zero, OPTIGA™ exports the derived key. In this case, the derived_key is a pointer to a buffer with at least a minimum length of requested derived_key_length.
	`inout derived_key : uint8_t *` Pointer to a buffer to store the derived key in case of exporting the derived key from OPTIGA™. If not exporting, this could be a NULL pointer.

3.1.1.19 optiga_crypt_set_comms_params

optiga_crypt_set_comms_params	
Description	This operation sets the shielded connection(Encrypted communication between Host and OPTIGA™) parameters like version, protection level and etc.
Signature	optiga_crypt_set_comms_params (in me : p_optiga_crypt_t, in parameter_type : uint8_t, in value : uint8_t) : void
Parameters	`in me : p_optiga_crypt_t` Pointer to one instance of optiga_crypt.
	`in parameter_type : uint8_t` Shielded connection configuration parameter type. The possible shielded connection parameter types that can be set are version (e.g. pre-shared secret based) and protection level (e.g. command protection, response protection, both or none). OPTIGA_COMMS_PROTOCOL_VERSION

optiga_crypt_set_comms_params	
	OPTIGA_COMMS_PROTECTION_LEVEL
	`in value : uint8_t` Value to be configured to the respective parameter.

3.1.1.20 OPTIGA_CRYPT_SET_COMMS_PROTOCOL_VERSION

OPTIGA_CRYPT_SET_COMMS_PROTOCOL_VERSION	
Description	This operation sets the protection version of shielded connection.
Signature	OPTIGA_CRYPT_SET_COMMS_PROTOCOL_VERSION (in me : p_optiga_util_t, in protocol_version) : void
Parameters	`in me : p_optiga_util_t` Pointer to one instance of optiga_crypt.
	`in protocol_version` Shielded connection protocol version. The possible values are OPTIGA_COMMS_PROTOCOL_VERSION_PRE_SHARED_SECRET

3.1.1.21 OPTIGA_CRYPT_SET_COMMS_PROTECTION_LEVEL

OPTIGA_CRYPT_SET_COMMS_PROTECTION_LEVEL	
Description	This operation sets the protection level of shielded connection to be enabled while sending/receiving the data to OPTIGA™. For the protected/encrypted communication (Shielded Connection) between Host and OPTIGA™, the optiga_crypt instance needs to be updated before invoking the respective optiga_crypt operations using the below specified Macros. • To enable protection for the command (data to be sent to OPTIGA™), OPTIGA_CRYPT_SET_COMMS_PROTECTION_LEVEL(instance, OPTIGA_COMMS_COMMAND_PROTECTION) • To enable protection for the response (data to be received from OPTIGA™), OPTIGA_CRYPT_SET_COMMS_PROTECTION_LEVEL(instance, OPTIGA_COMMS_RESPONSE_PROTECTION) • To enable protection for both command and response, OPTIGA_CRYPT_SET_COMMS_PROTECTION_LEVEL(instance, OPTIGA_COMMS_FULL_PROTECTION) • To disable protection for both command and response, OPTIGA_CRYPT_SET_COMMS_PROTECTION_LEVEL(instance, OPTIGA_COMMS_NO_PROTECTION) The above specified settings for the shielded connection are applicable only if OPTIGA_COMMS_SHIELDED_CONNECTION is enabled in configurations (*optiga_lib_config.h*) in the provided host library. The precondition for the shielded connection is pairing of OPTIGA™ and Local

OPTIGA_CRYPT_SET_COMMS_PROTECTION_LEVEL	
	Host. The sequence of pairing is explained in Use Case: Pair OPTIGA™ with Host (Pre-Shared Secret based).
Signature	OPTIGA_CRYPT_SET_COMMS_PROTECTION_LEVEL (in me : p_optiga_util_t, in protection_level) : void
Parameters	`in me : p_optiga_util_t` Pointer to one instance of optiga_crypt.
	`in protection_level` Shielded connection protection level. The possible values are OPTIGA_COMMS_NO_PROTECTION OPTIGA_COMMS_COMMAND_PROTECTION OPTIGA_COMMS_RESPONSE_PROTECTION OPTIGA_COMMS_FULL_PROTECTION In addition to the above values, OPTIGA_COMMS_RE_ESTABLISH can be used along with one of the above (e.g. OPTIGA_COMMS_COMMAND_PROTECTION | OPTIGA_COMMS_RE_ESTABLISH), the access layer will clear the old shielded connection session and re-establish a new session.

3.1.1.22 optiga_crypt_destroy

optiga_crypt_destroy	
Description	This operation destructs an instance of optiga_crypt. The volatile memory gets freed. This operation inherently destructs the instance of optiga_cmd and releases the session(if acquired) which was owned by the destructed instance of optiga_crypt.
Signature	optiga_crypt_destroy (in me : p_optiga_crypt_t) : optiga_lib_status_t
Parameters	`in me : p_optiga_crypt_t` Pointer to one instance of optiga_crypt

3.1.2 optiga_util

The optiga_util module provides useful utilities to manage the application and data stores with the following characteristics:

- Multiple instances could be created to allow concurrent access to other services.
- Usage of the optiga_cmd module to interact with the OPTIGA™.

3.1.2.1 optiga_util_create

optiga_util_create	
Description	This operation creates an instance of optiga_util. The volatile memory gets allocated and initialized. This operation inherently creates an instance of optiga_cmd if available due to solution constraints (the number of optiga_cmd instances might be limited). For the protected communication (Shielded Connection) between Host and OPTIGA™,

optiga_util_create	
	• This instance needs to be updated before invoking the respective optiga_util operations using OPTIGA_UTIL_SET_COMMS_PROTECTION_LEVEL. • Based on the chosen protection level, the access layer activates the Shielded Connection between Host and OPTIGA™. These settings get automatically reset to OPTIGA_COMMS_DEFAULT_PROTECTION_LEVEL once after the operation is invoked. • The above specified settings for the shielded connection are applicable only if OPTIGA_COMMS_SHIELDED_CONNECTION is enabled/defined in configurations in the provided implementation. • The precondition for the shielded connection is pairing of OPTIGA™ and Local Host. The sequence of pairing is depicted in Use Case: Pair OPTIGA™ with Host (Pre-Shared Secret based).
Signature	optiga_util_create (in optiga_instance_id : uint8_t, in handler : callback_handler_t, in caller_context : void *) : p_optiga_util_t
Parameters	`in optiga_instance_id : uint8_t` Instance id of the OPTIGA™ integrated on the host platform. The default value is 0.
	`in handler : callback_handler_t` Pointer to the callback function
	`in caller_context : void *` Pointer to the caller context.

3.1.2.2 optiga_util_open_application

optiga_util_open_application	
Description	This operation initializes or restores (from a hibernate state if performed) the application on OPTIGA™. Since after cold or warm reset, all applications residing on the OPTIGA™ are closed, an application has to be opened before using it. This operation initializes the application context on OPTIGA™. This operation must be issued once at least before invoking any other operations either from optiga_util or optiga_crypt. This operation with perform_restore = True, restores the already stored/hibernated (using optiga_util_close_application) application on OPTIGA™. The stored application context on OPTIGA™ gets flushed once after this operation is invoked (irrespective of perform_restore value chosen) and the same will not be allowed for reuse.
Signature	optiga_util_open_application (in me : p_optiga_util_t, in perform_restore : bool_t) : optiga_lib_status_t
Parameters	`in me : p_optiga_util_t` Pointer to one instance of optiga_util.
	`in perform_restore : bool_t` Specifies to perform restore from hibernate state. = 1 (True), restores the hibernated application on OPTIGA™ = 0 (false), initializes the clean/new application on OPTIGA™

3.1.2.3 optiga_util_close_application

optiga_util_close_application	
Description	This operation closes the application or on OPTIGA™. With the hibernate option (perform_hibernate = True), the OPTIGA™ stores the current context of application and restores with next optiga_util_open_application(perform_restore = True). With this option, the host can power off the OPTIGA™ when not in use and restore with optiga_util_open_application when required to avoid the power consumption by OPTIGA™ during the unused period. In this operation, after OPTIGA™ confirms the storing of state/context (commnd is successfully executed), the Access Layer switches off the OPTIGA™ provided the GPIOs are configured during control the Vcc connected to OPTIGA™. After successful completion of this operation, OPTIGA™ will not perform any other operations until the next successful optiga_util_open_application operation. Note: • In case of Security Event Counter (SEC) > 0, OPTIGA™ doesn't allow the hibernate operation. Hence this operation leads to failure.
Signature	optiga_util_close_application (in me : p_optiga_util_t, in perform_hibernate : bool_t) : optiga_lib_status_t
Parameters	`in me : p_optiga_util_t` Pointer to one instance of optiga_util.
	`in perform_hibernate : bool_t` Specifies to perform hibernate. = 1 (True), hibernates the application on OPTIGA™. This application can be restored using optiga_util_open_application. = 0 (False), closes the application on OPTIGA™, restore is not possible.

3.1.2.4 optiga_util_read_data

optiga_util_read_data	
Description	This operation reads the data from the specified data object in OPTIGA™.
Signature	optiga_util_read_data (in me : p_optiga_util_t, in optiga_oid : uint16_t, in offset : uint16_t, inout buffer : uint8_t *, inout length : uint16_t *) : optiga_lib_status_t
Parameters	`in me : p_optiga_util_t` Pointer to one instance of optiga_util.
	`in optiga_oid : uint16_t` OID of the data object from OPTIGA™
	`in offset : uint16_t` Offset in the data object from which the data has to be read
	`inout buffer : uint8_t *` Pointer to the buffer to store the data read
	`inout length : uint16_t *` Length of the data to be read

3.1.2.5 optiga_util_read_metadata

optiga_util_read_metadata	
Description	This operation reads the metadata from the specified data object in OPTIGA™.
Signature	optiga_util_read_metadata (in me : p_optiga_util_t, in optiga_oid : uint16_t, inout buffer : uint8_t *, inout length : uint16_t *) : optiga_lib_status_t
Parameters	`in me : p_optiga_util_t` Pointer to one instance of optiga_util.
	`in optiga_oid : uint16_t` OID of the data object from OPTIGA™
	`inout buffer : uint8_t *` Pointer to the buffer to store the metadata read
	`inout length : uint16_t *` Length of the metadata to be read

3.1.2.6 optiga_util_write_data

optiga_util_write_data	
Description	This operation writes data to the specified data object in OPTIGA™.
Signature	optiga_util_write_data (in me : p_optiga_util_t, in optiga_oid : uint16_t, in write_type : uint8_t, in offset : uint16_t, in buffer : const uint8_t *, in length : uint16_t) : optiga_lib_status_t
Parameters	`in me : p_optiga_util_t` Pointer to one instance of optiga_util.
	`in optiga_oid : uint16_t` OID of the data object from OPTIGA™ to which the data to be written.
	`in write_type : uint8_t` Type of write operation - (Erase & Write) or Write. OPTIGA_UTIL_ERASE_AND_WRITE (Erase & Write) - Erases the complete data object and writes the given data starting from the specified offset OPTIGA_UTIL_WRITE_ONLY (Write) - Writes the given data starting from the specified offset.
	`in offset : uint16_t` Offset from which the data to be written.
	`in buffer : const uint8_t *` Pointer to the buffer which holds data to be written.
	`in length : uint16_t` Length of the data to be written.

3.1.2.7 optiga_util_write_metadata

optiga_util_write_metadata	
Description	This operation writes metadata to the specified data object in OPTIGA™.
Signature	optiga_util_write_metadata (in me : p_optiga_util_t, in optiga_oid : uint16_t, in buffer : const uint8_t *, in length : uint16_t) : optiga_lib_status_t

optiga_util_write_metadata	
Parameters	`in me : p_optiga_util_t` Pointer to one instance of optiga_util.
	`in optiga_oid : uint16_t` OID of the data object from OPTIGA™ to which the metadata has to be written
	`in buffer : const uint8_t *` Pointer to the buffer which holds metadata to be written
	`in length : uint16_t` Length of the metadata to be written

3.1.2.8 optiga_util_update_count

optiga_util_update_count	
Description	This operation updates counter data object optiga_counter_oid with the provided count value in OPTIGA™. The counter in counter data object optiga_counter_oid gets incremented/decremented up to the threshold value depending on the counter type set. Any further attempts after reaching the threshold value, the OPTIGA™ returns an error.
Signature	optiga_util_update_count (in me : p_optiga_util_t, in optiga_counter_oid : uint16_t, in count : uint8_t) : optiga_lib_status_t
Parameters	`in me : p_optiga_util_t` Pointer to one instance of optiga_util.
	`in optiga_counter_oid : uint16_t` OID of the counter data object to be updated in OPTIGA™. Preconditions: The data object referred here must be a counter type.
	`in count : uint8_t` Count value (1 to 255) that to be incremented/decremented from the counter in the specified optiga_counter_oid.

3.1.2.9 optiga_util_protected_update_start

optiga_util_protected_update_start	
Description	This operation initiates the protected update of data objects in OPTIGA™. The manifest provided will be validated by OPTIGA™. Upon the successful completion of this operation, The fragments (which contain the data to be updated) are to be sent using optiga_util_protected_update_continue and/or optiga_util_protected_update_final. The protected update needs to be performed in a strict sequence. The optiga_cmd acquires the lock for the strict sequence to execute the protected update in atomic way. The lock acquired gets released either by the successful completion of optiga_util_protected_update_final or any failure until the optiga_util_protected_update_final is completed. Once the optiga_util instance is used with optiga_util_protected_update_start successfully, the same instance is not supposed to be used until the

optiga_util_protected_update_start	
	optiga_util_protected_update_final completed or the protected update sequence failed with other operations. The other operations (optiga_util, optiga_crypt) invoked with different instances during the protected update, will be scheduled automatically after the lock is released. The shielded connection protection level chosen for optiga_util_protected_update_start, will also be applied for optiga_util_protected_update_continue and optiga_util_protected_update_final implicitly.
Signature	optiga_util_protected_update_start (in me : p_optiga_util_t, in manifest_version : uint8_t, in manifest : const uint8_t *, in manifest_length : uint16_t) : optiga_lib_status_t
Parameters	`in me : p_optiga_util_t` Pointer to one instance of optiga_util.
	`in manifest_version : uint8_t` Version of the manifest format. Default/Supported version = 0x01.
	`in manifest : const uint8_t *` Pointer to the buffer which holds the manifest.
	`in manifest_length : uint16_t` Length of manifest provided.

3.1.2.10 optiga_util_protected_update_continue

optiga_util_protected_update_continue	
Description	This operation sends the fragments to OPTIGA™. If the protected update contains a single fragment, then the fragment has to be sent using the optiga_util_protected_update_final and the optiga_util_protected_update_continue is skipped. If the protected update contains multiple fragments (payload is split into multiple fragments), then all the fragments (except last) are to be sent using optiga_util_protected_update_continue and the last fragment to be sent using optiga_util_protected_update_final. E.g., The number of fragments are n, n = 1, the fragment must be sent using optiga_util_protected_update_final and optiga_util_protected_update_continue is not used. n > 1, the first and up to (n-1) fragments must be sent using optiga_util_protected_update_continue and the last fragment must be sent using optiga_util_protected_update_final. Notes: • The local_host_application must take care of sending the fragments in the correct order to OPTIGA™ as each fragment contains the integrity of the next fragment. • The fragment size must be 640 bytes.
Signature	optiga_util_protected_update_continue (in me : p_optiga_util_t, in fragment : const uint8_t *, in fragment_length : uint16_t) : optiga_lib_status_t

optiga_util_protected_update_continue	
Parameters	`in me : p_optiga_util_t` Pointer to one instance of optiga_util.
	`in fragment : const uint8_t *` Pointer to the buffer which holds the fragment to be sent to OPTIGA™.
	`in fragment_length : uint16_t` Length of fragment.

3.1.2.11 optiga_util_protected_update_final

optiga_util_protected_update_final	
Description	This operation sends the last fragment and finalizes the protected update of OPTIGA™ data object and the optiga_cmd releases the strict lock acquired. The size of the fragment can be up to 640 bytes.
Signature	optiga_util_protected_update_final (in me : p_optiga_util_t, in fragment : const uint8_t *, in fragment_length : uint16_t) : optiga_lib_status_t
Parameters	`in me : p_optiga_util_t` Pointer to one instance of optiga_util.
	`in fragment : const uint8_t *` Pointer to the buffer which holds the fragment to be sent to OPTIGA™.
	`in fragment_length : uint16_t` Length of fragment.

3.1.2.12 optiga_util_set_comms_params

optiga_util_set_comms_params	
Description	This operation sets the shielded connection(Encrypted communication between Host and OPTIGA™) parameters like version, protection level and etc.
Signature	optiga_util_set_comms_params (in me : p_optiga_util_t, in parameter_type : uint8_t, in value : uint8_t) : void
Parameters	`in me : p_optiga_util_t` Pointer to one instance of optiga_util.
	`in parameter_type : uint8_t` Shielded connection configuration parameter type. The possible shielded connection parameter types that can be set are version (e.g. pre-shared secret based) and protection level (e.g. command protection, response protection, both or none). OPTIGA_COMMS_PROTOCOL_VERSION OPTIGA_COMMS_PROTECTION_LEVEL
	`in value : uint8_t` Value to be configured to the respective parameter.

3.1.2.13 OPTIGA_UTIL_SET_COMMS_PROTOCOL_VERSION

OPTIGA_UTIL_SET_COMMS_PROTOCOL_VERSION	
Description	This operation sets the protection version of shielded connection.
Signature	OPTIGA_UTIL_SET_COMMS_PROTOCOL_VERSION (in me : p_optiga_util_t, in protocol_version) : void
Parameters	`in me : p_optiga_util_t` Pointer to one instance of optiga_util.
	`in protocol_version` Shielded connection protocol version. The possible values are OPTIGA_COMMS_PROTOCOL_VERSION_PRE_SHARED_SECRET

3.1.2.14 OPTIGA_UTIL_SET_COMMS_PROTECTION_LEVEL

OPTIGA_UTIL_SET_COMMS_PROTECTION_LEVEL	
Description	This operation sets the protection level of shielded connection to be enabled while sending/receiving the data to OPTIGA™. For the protected/encrypted communication (Shielded Connection) between Host and OPTIGA™, the optiga_util instance needs to be updated before invoking the respective optiga_util operations using the below specified Macros. • To enable protection for the command (data to be sent to OPTIGA™), OPTIGA_UTIL_SET_COMMS_PROTECTION_LEVEL(instance, OPTIGA_COMMS_COMMAND_PROTECTION) • To enable protection for the response (data to be received from OPTIGA™), OPTIGA_UTIL_SET_COMMS_PROTECTION_LEVEL(instance, OPTIGA_COMMS_RESPONSE_PROTECTION) • To enable protection for both command and response, OPTIGA_UTIL_SET_COMMS_PROTECTION_LEVEL(instance, OPTIGA_COMMS_FULL_PROTECTION) • To disable protection for both command and response, OPTIGA_UTIL_SET_COMMS_PROTECTION_LEVEL(instance, OPTIGA_COMMS_NO_PROTECTION) The above specified settings for the shielded connection are applicable only if OPTIGA_COMMS_SHIELDED_CONNECTION is enabled in configurations (*optiga_lib_config.h*) in the provided host library. The precondition for the shielded connection is pairing of OPTIGA™ and Local Host. The sequence of pairing is explained in Use Case: Pair OPTIGA™ with Host (Pre-Shared Secret based).
Signature	OPTIGA_UTIL_SET_COMMS_PROTECTION_LEVEL (in me : p_optiga_util_t, in protection_level) : void
Parameters	`in me : p_optiga_util_t` Pointer to one instance of optiga_util.
	`in protection_level` Shielded connection protection level.

OPTIGA_UTIL_SET_COMMS_PROTECTION_LEVEL	
	The possible values are OPTIGA_COMMS_NO_PROTECTION OPTIGA_COMMS_COMMAND_PROTECTION OPTIGA_COMMS_RESPONSE_PROTECTION OPTIGA_COMMS_FULL_PROTECTION In addition to the above values, OPTIGA_COMMS_RE_ESTABLISH can be used along with one of the above (e.g. OPTIGA_COMMS_COMMAND_PROTECTION | OPTIGA_COMMS_RE_ESTABLISH), the access layer will clear the old shielded connection session and re-establish a new session.

3.1.2.15 optiga_util_destroy

optiga_util_destroy	
Description	This operation destructs an instance of optiga_util. The volatile memory gets freed. This operation inherently destructs the instance of optiga_cmd which was owned by the destructed instance of optiga_util.
Signature	optiga_util_destroy (in me : p_optiga_util_t) : optiga_lib_status_t
Parameters	`in me : p_optiga_util_t` Pointer to one instance of optiga_util.

3.2 Access Layer Decomposition

The Access Layer Decomposition diagram shows the components providing and managing access to the command interface of the OPTIGA™.

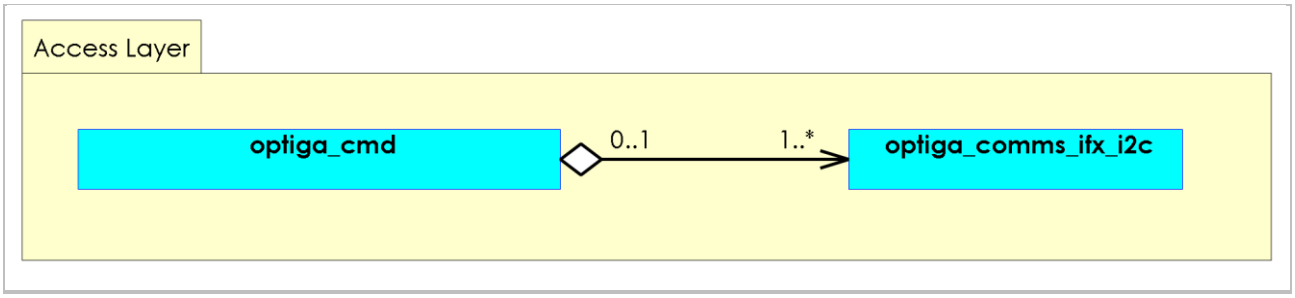

Figure 18 - Access Layer Decomposition

3.2.1 optiga_cmd

This module optiga_cmd exposes the main interface to interact with OPTIGA™. It is aware of the format of the command set provided by the OPTIGA™. The optiga_cmd converts API calls in the regarded (command / response) APDUs known by the OPTIGA™. The optiga_cmd APIs expose the same semantics provided by OPTIGA™.

The optiga_cmd provides multiple instances of the API. Beyond exposing the APIs it arbitrates as well concurrent invocations of the APIs. Its usage characteristic is asynchronous, where the caller of an instance has to take care of the correct sequence of calls for a dedicated use case. In case an instance of the API requires multiple invocations to reliably implement a use case (strict sequence), the APIs allows locking out other instances from interacting with the OPTIGA™. As soon as those strict sequences are executed the locking must be released. The optiga_cmd interacts with optiga_comms_xxx (xxx stands for variants e.g. ifx_i2c, tc, ...) for reliable

communication with OPTIGA™.

3.2.1.1 optiga_cmd_create

optiga_cmd_create	
Description	This operation creates an instance of optiga_cmd. The volatile memory gets allocated and initialized. This operation inherently creates an instance of optiga_comms_ifx_i2c or optiga_comms_tc if available due to solution constraints (the number of optiga_comms_ifx_i2c or optiga_comms_tc instances might be limited).
Signature	optiga_cmd_create (in optiga_instance_id : uint8_t, in handler : callback_handler_t, in caller_context : void *) : p_optiga_cmd_t
Parameters	`in optiga_instance_id : uint8_t` Instance id of the OPTIGA™ integrated on the host platform. The default value is 0.
	`in handler : callback_handler_t` Pointer to the callback function.
	`in caller_context : void *` Pointer to the caller context.

3.2.1.2 optiga_cmd_open_application

optiga_cmd_open_application	
Description	This operation exposes the OpenApplication command at the optiga_cmd API.
Signature	optiga_cmd_open_application (inout me : p_optiga_cmd_t, in cmd_param : uint8_t) : optiga_lib_status_t
Parameters	`inout me : p_optiga_cmd_t` Pointer to one instance of optiga_cmd.
	`in cmd_param : uint8_t` Command APDU param value.

3.2.1.3 optiga_cmd_close_application

optiga_cmd_close_application	
Description	This operation exposes the CloseApplication command at the optiga_cmd API.
Signature	optiga_cmd_close_application (inout me : p_optiga_cmd_t, in cmd_param : uint8_t) : optiga_lib_status_t
Parameters	`inout me : p_optiga_cmd_t` Pointer to one instance of optiga_cmd.
	`in cmd_param : uint8_t` Command APDU param value.

3.2.1.4 optiga_cmd_get_data_object

optiga_cmd_get_data_object	
Description	This operation exposes the GetDataObject command at the optiga_cmd API. • Implements command chaining to abstract away the maximum buffer size of the OPTIGA™.
Signature	optiga_cmd_get_data_object (inout me : p_optiga_cmd_t, in cmd_param : uint8_t, inout params : p_optiga_get_data_object_params_t) : optiga_lib_status_t
Parameters	`inout me : p_optiga_cmd_t` Pointer to one instance of optiga_cmd.
	`in cmd_param : uint8_t` Command APDU param value.
	`inout params : p_optiga_get_data_object_params_t` Pointer to the command APDU parameters.

3.2.1.5 optiga_cmd_set_data_object

optiga_cmd_set_data_object	
Description	This operation exposes the SetDataObject command at the optiga_cmd API. • Implements command chaining to abstract away the maximum buffer size of the OPTIGA™.
Signature	optiga_cmd_set_data_object (inout me : p_optiga_cmd_t, in cmd_param : uint8_t, inout params : p_optiga_set_data_object_params_t) : optiga_lib_status_t
Parameters	`inout me : p_optiga_cmd_t` Pointer to one instance of optiga_cmd.
	`in cmd_param : uint8_t` Command APDU param value.
	`inout params : p_optiga_set_data_object_params_t` Pointer to the command APDU parameters.

3.2.1.6 optiga_cmd_set_object_protected

optiga_cmd_set_object_protected	
Description	This operation exposes the SetObjectProtected command at the optiga_cmd API.
Signature	optiga_cmd_set_object_protected (inout me : p_optiga_cmd_t, in cmd_param : uint8_t, inout params : p_optiga_set_object_protected_params_t) : optiga_lib_status_t
Parameters	`inout me : p_optiga_cmd_t` Pointer to one instance of optiga_cmd.
	`in cmd_param : uint8_t` Command APDU param value.
	`inout params : p_optiga_set_object_protected_params_t` Pointer to the command APDU parameters.

3.2.1.7 optiga_cmd_get_random

optiga_cmd_get_random	
Description	This operation exposes the GetRandom command at the optiga_cmd API.
Signature	optiga_cmd_get_random (inout me : p_optiga_cmd_t, in cmd_param : uint8_t, inout params : p_optiga_get_random_params_t) : optiga_lib_status_t
Parameters	`inout me : p_optiga_cmd_t` Pointer to one instance of optiga_cmd.
	`in cmd_param : uint8_t` Command APDU param value.
	`inout params : p_optiga_get_random_params_t` Pointer to the command APDU parameters.

3.2.1.8 optiga_cmd_calc_hash

optiga_cmd_calc_hash	
Description	This operation exposes the CalcHash command at the optiga_cmd API. • Implements command chaining to abstract away the maximum buffer size of the OPTIGA™.
Signature	optiga_cmd_calc_hash (inout me : p_optiga_cmd_t, in cmd_param : uint8_t, inout params : p_optiga_calc_hash_params_t) : optiga_lib_status_t
Parameters	`inout me : p_optiga_cmd_t` Pointer to one instance of optiga_cmd.
	`in cmd_param : uint8_t` Command APDU param value.
	`inout params : p_optiga_calc_hash_params_t` Pointer to the command APDU parameters.

3.2.1.9 optiga_cmd_gen_keypair

optiga_cmd_gen_keypair	
Description	This operation exposes the GenKeyPair command at the optiga_cmd API.
Signature	optiga_cmd_gen_keypair (inout me : p_optiga_cmd_t, in cmd_param : uint8_t, inout params : p_optiga_gen_keypair_params_t) : optiga_lib_status_t
Parameters	`inout me : p_optiga_cmd_t` Pointer to one instance of optiga_cmd.
	`in cmd_param : uint8_t` Command APDU param value.
	`inout params : p_optiga_gen_keypair_params_t` Pointer to the command APDU parameters.

3.2.1.10 optiga_cmd_calc_sign

optiga_cmd_calc_sign	
Description	This operation exposes the CalcSign command at the optiga_cmd API.
Signature	optiga_cmd_calc_sign (inout me : p_optiga_cmd_t, in cmd_param : uint8_t, inout params : p_optiga_calc_sign_params_t) : optiga_lib_status_t
Parameters	`inout me : p_optiga_cmd_t` Pointer to one instance of optiga_cmd.
	`in cmd_param : uint8_t` Command APDU param value.
	`inout params : p_optiga_calc_sign_params_t` Pointer to the command APDU parameters.

3.2.1.11 optiga_cmd_verify_sign

optiga_cmd_verify_sign	
Description	This operation exposes the VerifySign command at the optiga_cmd API.
Signature	optiga_cmd_verify_sign (inout me : p_optiga_cmd_t, in cmd_param : uint8_t, inout params : p_optiga_verify_sign_params_t) : optiga_lib_status_t
Parameters	`inout me : p_optiga_cmd_t` Pointer to one instance of optiga_cmd.
	`in cmd_param : uint8_t` Command APDU param value.
	`inout params : p_optiga_verify_sign_params_t` Pointer to the command APDU parameters.

3.2.1.12 optiga_cmd_calc_ssec

optiga_cmd_calc_ssec	
Description	This operation exposes the CalcSSec command at the optiga_cmd API.
Signature	optiga_cmd_calc_ssec (inout me : p_optiga_cmd_t, in cmd_param : uint8_t, inout params : p_optiga_calc_ssec_params_t) : optiga_lib_status_t
Parameters	`inout me : p_optiga_cmd_t` Pointer to one instance of optiga_cmd.
	`in cmd_param : uint8_t` Command APDU param value.
	`inout params : p_optiga_calc_ssec_params_t` Pointer to the command APDU parameters.

3.2.1.13 optiga_cmd_derive_key

optiga_cmd_derive_key	
Description	This operation exposes the DeriveKey command at the optiga_cmd API.

optiga_cmd_derive_key	
Signature	optiga_cmd_derive_key (inout me : p_optiga_cmd_t, in cmd_param : uint8_t, inout params : p_optiga_derive_key_params_t) : optiga_lib_status_t
Parameters	`inout me : p_optiga_cmd_t` Pointer to one instance of optiga_cmd.
	`in cmd_param : uint8_t` Command APDU param value.
	`inout params : p_optiga_derive_key_params_t` Pointer to the command APDU parameters.

3.2.1.14 optiga_cmd_encrypt_asym

optiga_cmd_encrypt_asym	
Description	This operation exposes the EncryptAsym command at the optiga_cmd API.
Signature	optiga_cmd_encrypt_asym (inout me : p_optiga_cmd_t, in cmd_param : uint8_t, inout params : p_optiga_encrypt_asym_params_t) : optiga_lib_status_t
Parameters	`inout me : p_optiga_cmd_t` Pointer to one instance of optiga_cmd.
	`in cmd_param : uint8_t` Command APDU param value.
	`inout params : p_optiga_encrypt_asym_params_t` Pointer to the command APDU parameters.

3.2.1.15 optiga_cmd_decrypt_asym

optiga_cmd_decrypt_asym	
Description	This operation exposes the DecryptAsym command at the optiga_cmd API.
Signature	optiga_cmd_decrypt_asym (inout me : p_optiga_cmd_t, in cmd_param : uint8_t, inout params : p_optiga_decrypt_asym_params_t) : optiga_lib_status_t
Parameters	`inout me : p_optiga_cmd_t` Pointer to one instance of optiga_cmd.
	`in cmd_param : uint8_t` Command APDU param value.
	`inout params : p_optiga_decrypt_asym_params_t` Pointer to the command APDU parameters.

3.2.1.16 optiga_cmd_destroy

optiga_cmd_destroy	
Description	This operation destructs an instance of optiga_cmd. The volatile memory gets freed. This operation inherently releases the session if acquired and destructs the instance of optiga_comms_ifx_i2c or optiga_comms_tc which was owned by the destructed instance of optiga_cmd.
Signature	optiga_cmd_destroy (inout me : p_optiga_cmd_t) : optiga_lib_status_t

optiga_cmd_destroy	
Parameters	`inout me : p_optiga_cmd_t` Pointer to one instance of optiga_cmd.

3.2.2 optiga_comms_ifx_i2c

optiga_comms_ifx_i2c implements the protocol used to turn-in communication between Local Host and OPTIGA™. The invoking component, in the given architecture is the optiga_cmd block through the pal. The optiga_cmd provides command APDUs to optiga_comms_ifx_i2c and receives response APDUs from the optiga_comms_ifx_i2c. The size of APDUs may vary between few bytes to kilo bytes. The protocol implementation is done in multiple layers and seamlessly handles data transfer from Local Host to OPTIGA™ and OPTIGA™ to Local Host. More details of the implemented protocol can be found in [IFX_I2C]. optiga_comms_ifx_i2c usage characteristic is asynchronous, were the caller has to take care of the correct sequence of calls for a dedicated use case.

3.2.2.1 optiga_comms_create

optiga_comms_create	
Description	This operation creates an instance of optiga_comms_ifx_i2c.
Signature	optiga_comms_create (in handler : callback_handler_t, in caller_context : void *) : p_optiga_comms_t
Parameters	`in handler : callback_handler_t` Pointer to the callback function.
	`in caller_context : void *` Pointer to the caller context.

3.2.2.2 optiga_comms_open

optiga_comms_open	
Description	This operation initializes the communication with OPTIGA™.
Signature	optiga_comms_open (inout me : p_optiga_comms_t) : optiga_lib_status_t
Parameters	`inout me : p_optiga_comms_t` Pointer to one instance of optiga_comms_ifx_i2c.

3.2.2.3 optiga_comms_transceive

optiga_comms_transceive	
Description	This operation sends command and receives response APDUs.
Signature	optiga_comms_transceive (inout me : p_optiga_comms_t, in p_tx_data : const uint8_t *, in tx_data_length : uint16_t, in p_rx_data : uint8_t *, in p_rx_data_length : uint16_t *) : optiga_lib_status_t
Parameters	`inout me : p_optiga_comms_t` Pointer to one instance of optiga_comms_ifx_i2c.
	`in p_tx_data : const uint8_t *`

optiga_comms_transceive	
	Pointer to the data to be sent.
	`in tx_data_length : uint16_t` Length of data to be sent.
	`in p_rx_data : uint8_t *` Pointer to the data to be received.
	`in p_rx_data_length : uint16_t *` Pointer to the length of the data to be received. This gets updated with the length of the data received.

3.2.2.4 optiga_comms_close

optiga_comms_close	
Description	This operation closes the communication with OPTIGA™.
Signature	optiga_comms_close (inout me : p_optiga_comms_t) : optiga_lib_status_t
Parameters	`inout me : p_optiga_comms_t` Pointer to one instance of optiga_comms_ifx_i2c.

3.2.2.5 optiga_comms_destroy

optiga_comms_destroy	
Description	This operation destructs an instance of optiga_comms_ifx_i2c.
Signature	optiga_comms_destroy (in me : p_optiga_comms_t) : optiga_lib_status_t
Parameters	`in me : p_optiga_comms_t` Pointer to one instance of optiga_comms_ifx_i2c.

3.3 Abstraction Layer Decomposition

The Abstraction Layer Decomposition diagram shows the components providing an agnostic interface to the underlying HW and SW platform functionality used by the higher level components of the architecture.

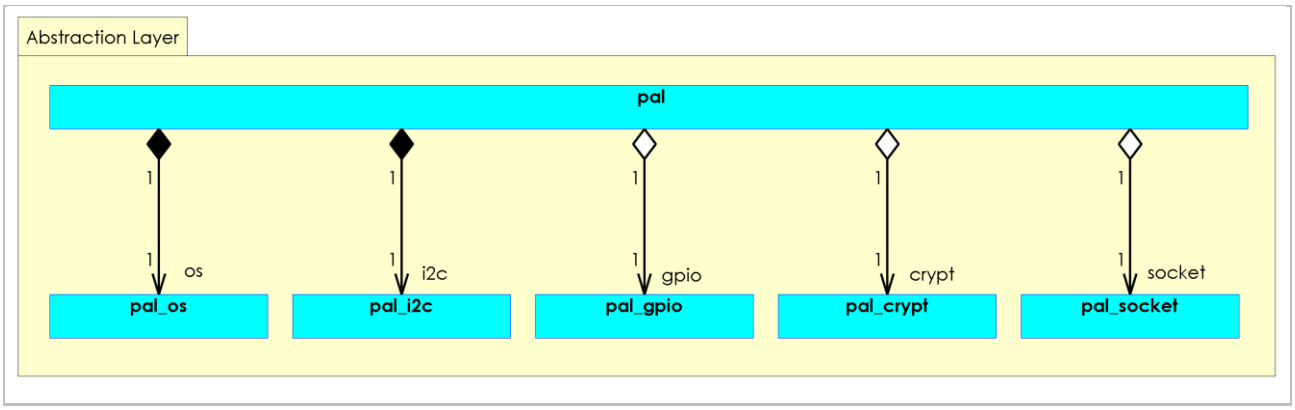

Figure 19 - Abstraction Layer Decomposition

3.3.1 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 (platform_timer, platform_i2c, platform_socket, ...) to allow the modules calling it being platform agnostic. The pal is composed of hardware, software and an operating system abstraction part.

3.3.1.1 pal_init

pal_init	
Description	This operation initializes the pal and aggregated pal modules (e.g. pal_i2c_init, pal_gpio_init, pal_os_init, etc).
Signature	pal_init () : pal_status_t

3.3.1.2 pal_deinit

pal_deinit	
Description	This operation deinitializes the pal and aggregated pal modules (e.g. pal_i2c_deinit, pal_gpio_deinit, pal_os_deinit, etc).
Signature	pal_deinit () : pal_status_t

3.3.2 pal_crypt

The pal_crypt module provides the platform specific migration of platform-specific cryptographic functionality (either SW libraries or HW) and is exposing cryptographic primitives invoked by platform agnostic modules.

3.3.2.1 pal_crypt_tls_prf_sha256

pal_crypt_tls_prf_sha256	
Description	This operation derives the secret using the TLS PRF SHA256 for a given shared secret.
Signature	pal_crypt_tls_prf_sha256 (inout p_pal_crypt : pal_crypt_t *, in p_secret : const uint8_t *, in secret_length : uint16_t, in p_label : const uint8_t *, in label_length : uint16_t, in p_seed : const uint8_t *, in seed_length : uint16_t, inout p_derived_key : uint8_t *, in derived_key_length : uint16_t) : pal_status_t
Parameters	`inout p_pal_crypt : pal_crypt_t *` Pointer to pal_crypt context.
	`in p_secret : const uint8_t *` Pointer to input secret key to be used to derive the key.
	`in secret_length : uint16_t` Input secret key length.
	`in p_label : const uint8_t *` Pointer to label which is typically a constant string. If no label is used, then this parameter could be NULL.

pal_crypt_tls_prf_sha256	
	`in label_length : uint16_t` Length of label.
	`in p_seed : const uint8_t *` Pointer to the seed.
	`in seed_length : uint16_t` Length of seed
	`inout p_derived_key : uint8_t *` Pointer to derived key.
	`in derived_key_length : uint16_t` Length of key to be derived.

3.3.2.2 pal_crypt_encrypt_aes128_ccm

pal_crypt_encrypt_aes128_ccm	
Description	This operation encrypts the given plain text using the provided encryption key and nonce.
Signature	pal_crypt_encrypt_aes128_ccm (inout p_pal_crypt : pal_crypt_t *, inout p_plain_text : uint8_t *, in plain_text_length : uint16_t, inout p_encrypt_key : const uint8_t *, inout p_nonce : const uint8_t *, in nonce_length : uint16_t, inout p_associated_data : const uint8_t *, in associated_data_length : uint16_t, in mac_size : uint8_t, inout p_cipher_text : uint8_t *) : pal_status_t
Parameters	`inout p_pal_crypt : pal_crypt_t *` Pointer to pal_crypt context
	`inout p_plain_text : uint8_t *` Pointer to plain text to be encrypted
	`in plain_text_length : uint16_t` Length of plain text to be encrypted
	`inout p_encrypt_key : const uint8_t *` Pointer to encryption key
	`inout p_nonce : const uint8_t *` Pointer to nonce
	`in nonce_length : uint16_t` Length of nonce
	`inout p_associated_data : const uint8_t *` Pointer to associated data
	`in associated_data_length : uint16_t` Length of associated data
	`in mac_size : uint8_t` Size of MAC
	`inout p_cipher_text : uint8_t *` Pointer to Cipher text

3.3.2.3 pal_crypt_decrypt_aes128_ccm

pal_crypt_decrypt_aes128_ccm	
Description	This operation decrypts the given cipher text using the provided decryption key and nonce. This operation validates the MAC internally and provides the plain text if MAC is successfully validated.
Signature	pal_crypt_decrypt_aes128_ccm (inout p_pal_crypt : pal_crypt_t *, inout p_cipher_text : uint8_t *, in cipher_text_length : uint16_t, inout p_decrypt_key : const uint8_t *, inout p_nonce : const uint8_t *, in nonce_length : uint16_t, inout p_associated_data : const uint8_t *, in associated_data_length : uint16_t, in mac_size : uint8_t, inout p_plain_text : uint8_t *) : pal_status_t
Parameters	`inout p_pal_crypt : pal_crypt_t *` Pointer to pal_crypt context
	`inout p_cipher_text : uint8_t *` Pointer to cipher text
	`in cipher_text_length : uint16_t` Length of cipher text including size of MAC
	`inout p_decrypt_key : const uint8_t *` Pointer to decryption key
	`inout p_nonce : const uint8_t *` Pointer to nonce
	`in nonce_length : uint16_t` length of nonce
	`inout p_associated_data : const uint8_t *` Pointer to associated data
	`in associated_data_length : uint16_t` Length of associated data
	`in mac_size : uint8_t` Size of MAC
	`inout p_plain_text : uint8_t *` Pointer to plain text

3.3.3 pal_gpio

This Module provides APIs to set GPIO high/low to perform below operations.

- Power on/off
- HW Reset on/off

3.3.3.1 pal_gpio_init

pal_gpio_init	
Description	This operation initializes the lower level driver of gpio.
Signature	pal_gpio_init (in p_gpio_context : const pal_gpio_t *) : pal_status_t
Parameters	`in p_gpio_context : const pal_gpio_t *` Pointer to a pal gpio context.

3.3.3.2 pal_gpio_deinit

pal_gpio_deinit	
Description	This operation de-initializes the lower level driver of gpio.
Signature	pal_gpio_deinit (in p_gpio_context : const pal_gpio_t *) : pal_status_t
Parameters	`in p_gpio_context : const pal_gpio_t *` Pointer to a pal gpio context

3.3.3.3 pal_gpio_set_high

pal_gpio_set_high	
Description	This operation sets the gpio pin state to high.
Signature	pal_gpio_set_high (in p_gpio_context : const pal_gpio_t *) : pal_status_t
Parameters	`in p_gpio_context : const pal_gpio_t *` Pointer to a pal gpio context

3.3.3.4 pal_gpio_set_low

pal_gpio_set_low	
Description	This operation sets the gpio pin state to low.
Signature	pal_gpio_set_low (in p_gpio_context : const pal_gpio_t *) : pal_status_t
Parameters	`in p_gpio_context : const pal_gpio_t *` Pointer to a pal gpio context

3.3.4 pal_i2c

The pal_i2c module is a platform ported module and provides the platform specific migration of HW based I2C functionality. The pal_i2c gets invoked as a platform agnostic security device communication API by platform agnostic modules. It is assumed that multiple callers are invoking its API concurrently. Therefore the implementation of each API function is atomic and stateless (except the initialization).

3.3.4.1 pal_i2c_init

pal_i2c_init	
Description	This operation initializes the lower level driver of i2c.
Signature	pal_i2c_init (in p_i2c_context : const pal_i2c_t *) : pal_status_t
Parameters	`in p_i2c_context : const pal_i2c_t *` Pointer to a pal i2c context

3.3.4.2 pal_i2c_deinit

pal_i2c_deinit	
Description	This operation de-initializes the lower level driver of i2c.
Signature	pal_i2c_deinit (in p_i2c_context : const pal_i2c_t *) : pal_status_t
Parameters	`in p_i2c_context : const pal_i2c_t *` Pointer to a pal i2c context

3.3.4.3 pal_i2c_read

pal_i2c_read	
Description	This operation reads the data from I2C bus.
Signature	pal_i2c_read (in p_i2c_context : const pal_i2c_t *, inout p_data : uint8_t *, in length : uint16_t)
Parameters	`in p_i2c_context : const pal_i2c_t *` Pointer to a pal i2c context.
	`inout p_data : uint8_t *` Pointer to a buffer to store the read data.
	`in length : uint16_t` Length of data to be read.

3.3.4.4 pal_i2c_write

pal_i2c_write	
Description	This operation writes the data to I2C bus.
Signature	pal_i2c_write (in p_i2c_context : const pal_i2c_t *, in p_data : const uint8_t *, in length : uint16_t)
Parameters	`in p_i2c_context : const pal_i2c_t *` Pointer to a pal i2c context.
	`in p_data : const uint8_t *` Pointer to a data buffer to be written on I2C bus.
	`in length : uint16_t` Length of data to be written.

3.3.4.5 pal_i2c_set_bitrate

pal_i2c_set_bitrate	
Description	This operation sets the bit rate of I2C master.
Signature	pal_i2c_set_bitrate (in p_i2c_context : const pal_i2c_t *, in Parameter2)
Parameters	`in p_i2c_context : const pal_i2c_t *` Pointer to a pal i2c context (pal_i2c_t).
	`in Parameter2`

pal_i2c_set_bitrate	
	Bitrate to be used by I2C master in KHz.

3.3.5 pal_os

The pal_os module provides the platform specific migration of operating system (e.g. RTOS) based functionality which gets invoked by platform agnostic modules.

3.3.5.1 pal_os_datastore_read

pal_os_datastore_read	
Description	This operation abstracts the reading of data from the specified location in the host platform.
Signature	pal_os_datastore_read (in datastore_id : uint16_t, inout p_buffer : uint8_t *, inout p_buffer_length : uint16_t *) : pal_status_t
Parameters	`in datastore_id : uint16_t` Reference ID for the data to be read on the host platform memory
	`inout p_buffer : uint8_t *` Pointer to the buffer to store the data read.
	`inout p_buffer_length : uint16_t *` Pointer to the Length of buffer. This gets updated with the actual length read from the data store.

3.3.5.2 pal_os_datastore_write

pal_os_datastore_write	
Description	This operation abstracts the writing of data to the specified location in the host platform.
Signature	pal_os_datastore_write (in datastore_id : uint16_t, in p_buffer : const uint8_t *, in length : uint16_t) : pal_status_t
Parameters	`in datastore_id : uint16_t` Reference ID for the location on the host platform memory to write/store the data.
	`in p_buffer : const uint8_t *` Pointer to the data to be written
	`in length : uint16_t` Length of data to be written

3.3.5.3 pal_os_event_create

pal_os_event_create	
Description	This operation initializes (creates optionally) returns context to the event for the later use.
Signature	pal_os_event_create () : p_pal_os_event_t

3.3.5.4 pal_os_event_register_callback_oneshot

pal_os_event_register_callback_oneshot	
Description	This operation registers the callback and context. The callback will invoked by pal_os_event_register_callback_oneshot with the given context after the provided time.
Signature	pal_os_event_register_callback_oneshot (in p_pal_os_event : p_pal_os_event_t, in callback : register_callback, in callback_args : void *, in time_us : uint32_t) : pal_status_t
Parameters	`in p_pal_os_event : p_pal_os_event_t` Pointer to the pal_os_event_t (created using pal_os_event_create)
	`in callback : register_callback` Callback handler to be registered
	`in callback_args : void *` Context which could be used by callback when triggered after the timeout.
	`in time_us : uint32_t` Timeout in microseconds.

3.3.5.5 pal_os_event_trigger_registered_callback

pal_os_event_trigger_registered_callback	
Description	This operation invokes the registered callback with the given context once the time out is triggered.
Signature	pal_os_event_trigger_registered_callback () : void

3.3.5.6 pal_os_event_destroy

pal_os_event_destroy	
Description	This operation destroys the event.
Signature	pal_os_event_destroy (in p_pal_os_event : p_pal_os_event_t) : void
Parameters	`in p_pal_os_event : p_pal_os_event_t` Pointer to the pal_os_event_t (created using pal_os_event_create)

3.3.5.7 pal_os_timer_init

pal_os_timer_init	
Description	This operation initializes the timer on the host platform.
Signature	pal_os_timer_init () : pal_status_t

3.3.5.8 pal_os_timer_get_time_in_milliseconds

pal_os_timer_get_time_in_milliseconds	
Description	This operation provides the current time stamp in milliseconds.

pal_os_timer_get_time_in_milliseconds	
Signature	pal_os_timer_get_time_in_milliseconds () : uint32_t

3.3.5.9 pal_os_timer_get_time_in_microseconds

pal_os_timer_get_time_in_microseconds	
Description	This operation provides the current time stamp in microseconds.
Signature	pal_os_timer_get_time_in_microseconds () : uint32_t

3.3.5.10 pal_os_timer_deinit

pal_os_timer_deinit	
Description	This operation de-initializes the timer on the host platform.
Signature	pal_os_timer_deinit () : pal_status_t

3.3.5.11 pal_os_lock_create

pal_os_lock_create	
Description	This operation initializes (creates optionally) the lock.
Signature	pal_os_lock_create (in p_lock : pal_os_lock_t *, in lock_type : uint8_t) : void
Parameters	`in p_lock : pal_os_lock_t *` Pointer to the lock
	`in lock_type : uint8_t` Type of lock

3.3.5.12 pal_os_lock_acquire

pal_os_lock_acquire	
Description	This operation acquires the lock.
Signature	pal_os_lock_acquire (in p_lock : pal_os_lock_t *) : pal_status_t
Parameters	`in p_lock : pal_os_lock_t *` Pointer to the lock

3.3.5.13 pal_os_lock_release

pal_os_lock_release	
Description	This operation releases the lock.
Signature	pal_os_lock_release (in p_lock : pal_os_lock_t *) : void
Parameters	`in p_lock : pal_os_lock_t *` Pointer to the lock

3.3.5.14 pal_os_lock_destroy

pal_os_lock_destroy	
Description	This operation destroys the lock.
Signature	pal_os_lock_destroy (in p_lock : pal_os_lock_t *) : void
Parameters	`in p_lock : pal_os_lock_t *` Pointer to the lock

3.3.5.15 pal_os_lock_enter_critical_section

pal_os_lock_enter_critical_section	
Description	This operation allows to enter critical section.
Signature	pal_os_lock_enter_critical_section () : void

3.3.5.16 pal_os_lock_exit_critical_section

pal_os_lock_exit_critical_section	
Description	This operation allows to exit from critical section.
Signature	pal_os_lock_exit_critical_section () : void

3.3.5.17 pal_os_malloc

pal_os_malloc	
Description	This operation allocates memory.
Signature	pal_os_malloc (in block_size : uint32_t) : void *
Parameters	`in block_size : uint32_t` Size of the block

3.3.5.18 pal_os_calloc

pal_os_calloc	
Description	This operation allocates a clean (set to all 0's) memory.
Signature	pal_os_calloc (in number_of_blocks : uint32_t, in block_size : uint32_t) : void
Parameters	`in number_of_blocks : uint32_t` Number of blocks of memory with a block_size to be allocated
	`in block_size : uint32_t` Size of the block

3.3.5.19 pal_os_free

pal_os_free	
Description	This operation frees the memory.
Signature	pal_os_free (in block : void *) : void
Parameters	`in block : void *` Pointer to the memory block to be freed

3.4 Data Types

This section defines the data types used by the service layer operations specified in Enabler APIs.

3.4.1 Enumerations

optiga_ecc_curve_t	
Description	Types of ECC Curves supported by OPTIGA™
Enumeration literals	OPTIGA_ECC_CURVE_NIST_P_256 Curve type - ECC NIST P-256
	OPTIGA_ECC_CURVE_NIST_P_384 Curve type - ECC NIST P-384

optiga_hash_type_t	
Description	Types of digest generation supported by OPTIGA™
Enumeration literals	OPTIGA_HASH_TYPE_SHA_256 Generate digest using SHA256

optiga_key_id_t	
Description	Key slot IDs in OPTIGA™
Enumeration literals	OPTIGA_KEY_ID_E0F0 Key from key store (non-volatile). Supports only ECC Type.
	OPTIGA_KEY_ID_E0F1 Key from key store (non-volatile). Supports only ECC Type.
	OPTIGA_KEY_ID_E0F2 Key from key store (non-volatile). Supports only ECC Type.
	OPTIGA_KEY_ID_E0F3 Key from key store (non-volatile). Supports only ECC Type.
	OPTIGA_KEY_ID_E0FC Key from key store (non-volatile). Supports only RSA Type.
	OPTIGA_KEY_ID_E0FD Key from key store (non-volatile). Supports only RSA Type.
	OPTIGA_KEY_ID_SESSION_BASED

	Key from session context (volatile).

optiga_key_usage_t	
Description	Types of Key usage. The multiple key usage types can be selected based on the requirement and key type. For example, if the private key from OPTIGA™ to be used for key agreement (Diffie-Hellmann) and signature generation purpose, then the key usage can be chosen as (OPTIGA_KEY_USAGE_SIGN | OPTIGA_KEY_USAGE_KEY_AGREEMENT).
Enumeration literals	OPTIGA_KEY_USAGE_AUTHENTICATION Allows to use the private key for the signature generation as part of authentication and sign commands
	OPTIGA_KEY_USAGE_ENCRYPTION Allows to use the private key for decrypt operations. This key usage is applicable for RSA key type only.
	OPTIGA_KEY_USAGE_SIGN Allows to use the private key for the signature generation as part of sign command
	OPTIGA_KEY_USAGE_KEY_AGREEMENT Allows to use the private key for key agreement (e.g. ECDH operations)

optiga_rng_type_t	
Description	Types of random number generation supported by OPTIGA™
Enumeration literals	OPTIGA_RNG_TYPE_TRNG Generate Random number using TRNG
	OPTIGA_RNG_TYPE_DRNG Generate Random number using DRNG

optiga_rsa_encryption_scheme_t	
Description	RSA Encryption schemes supported by OPTIGA™ for encryption and decryption.
Enumeration literals	OPTIGA_RSAES_PKCS1_V15 Encryption scheme - RSAES PKCS1-v1_5

optiga_rsa_key_type_t	
Description	Types of RSA keys supported by OPTIGA™
Enumeration literals	OPTIGA_RSA_KEY_1024_BIT_EXPONENTIAL RSA Key type - 1024 Bit exponential
	OPTIGA_RSA_KEY_2048_BIT_EXPONENTIAL RSA Key type - 2048 Bit exponential

Enabler APIs

optiga_rsa_signature_scheme_t	
Description	RSA Signature schemes supported by OPTIGA™ for sign and verify
Enumeration literals	OPTIGA_RSASSA_PKCS1_V15_SHA256 Signature scheme - RSA SSA PKCS1-v1_5 with SHA256 digest
	OPTIGA_RSASSA_PKCS1_V15_SHA384 Signature scheme - RSA SSA PKCS1-v1_5 with SHA384 digest

3.4.2 Structures

optiga_hash_context_t	
Description	Structure to the Hash context details managed by OPTIGA™.
Attributes	`context_buffer : uint8_t * [1]` Pointer a Buffer to hold the hash context information
	`context_buffer_length : uint32_t [1]` Length of context buffer
	`hash_algo : uint8_t [1]` hash algorithm as specified in optiga_hash_type_t

hash_data_in_optiga_t	
Description	Structure to provide the details of data to be hashed from OPTIGA™.
Attributes	`oid : uint16_t [1]` OID of the data object to be hashed
	`offset : uint16_t [1]` Offset of data in the data object
	`length : uint16_t [1]` Number of data bytes starting from the offset

hash_data_from_host_t	
Description	Structure to provide the details of data to be hashed from host.
Attributes	`buffer : const uint8_t * [1]` Pointer to data to hash
	`length : uint32_t [1]` Length of data to be hashed

public_key_from_host_t	
Description	Structure to provide the Public key and its details.
Attributes	`public_key : uint8_t * [1]` Pointer to Public Key
	`length : uint16_t [1]` Length of public key buffer
	`key_type : uint8_t [1]` Public key type details. For ECC, refer optiga_ecc_curve_t For RSA, refer optiga_rsa_key_type_t

Enabler APIs

p_hash_data_from_host_t	
Description	Pointer to structure hash_data_from_host_t
Data type	hash_data_from_host_t

p_hash_data_in_optiga_t	
Description	Pointer to structure hash_data_in_optiga_t
Data type	hash_data_in_optiga_t

p_optiga_hash_context_t	
Description	Pointer to structure optiga_hash_context_t
Data type	optiga_hash_context_t

p_public_key_from_host_t	
Description	Pointer to structure public_key_from_host_t
Data type	public_key_from_host_t

4 OPTIGA™ Trust M External Interface

The chapter OPTIGA™ Trust M External Interface provides the detailed definition of the OPTIGA™ device commands and responses available at its [I²C] interface.

4.1 Warm Reset

The Warm Reset (reset w/o power off/on cycle) of the OPTIGA™ 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 M]). 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™ 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% (refer to table Common data objects with TAG's and AC's OID '0xE0C4').

4.2.1 Low Power Sleep Mode

The OPTIGA™ automatically enters a low-power mode after a configurable delay. Once it has entered Sleep mode, the OPTIGA™ resumes normal operation as soon as its address is detected on the I2C bus.
In case no command is sent to the OPTIGA™ it behaves as shown in Figure "Go-to-Sleep diagram".
(1) As soon as the OPTIGA™ 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™ enters sleep mode.

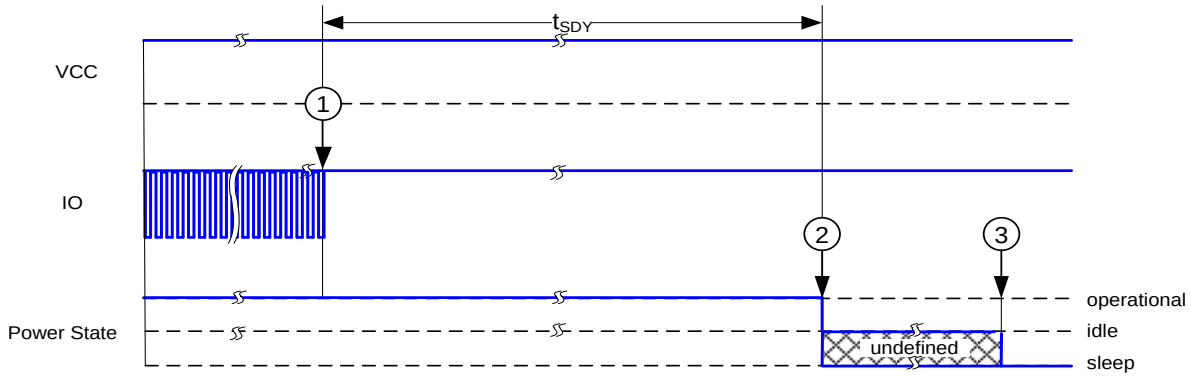

Figure 20 - Go-to-Sleep diagram

4.3 Protocol Stack

The OPTIGA™ is an I2C slave device. The protocol stack from the physical up to the application layer is specified in [IFX_I2C]. The protocol is defined for point-to-point connection and a multi-layer approach with low failure rate. It is optimized for minimum RAM usage and minimum overhead to achieve maximum bandwidth, but also offers error handling, flow control, chaining and optional communication protection.

OPTIGA™ Trust M External Interface

The used ISO/OSI layers are Physical, Data Link, Network, Transport, Presentation and Application layer as the figure below depicts.

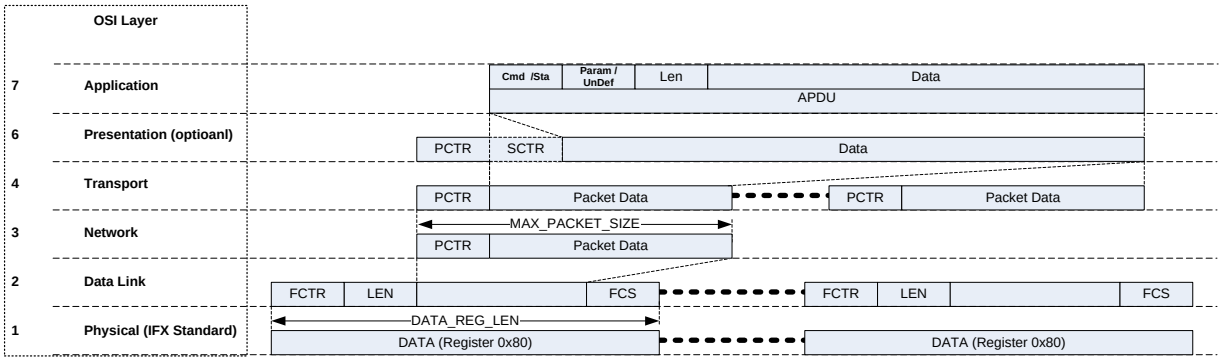

Figure 21 - Overview protocol stack used

The **Physical Layer** is entirely defined in [I²C]. Only a subset of those definitions is used for this protocol:

- Support of 7-Bit Addressing only (only 1 Address value)
- Single-Master / Multi-Slave configuration
- Speed (Fast Mode (Fm) up to 400 KHz; optional (Fm+) up to 1000 KHz)
- IFX standardized register interface.

The **Data Link Layer** provides reliable transmission of data packets.
The **Network Layer** provides the routing of packets to different channels.
The **Transport Layer** provides data packet chaining in case the upper layer consists of more data as the maximum packet size of the Data Link Layer supports.
The **Presentation Layer** is optional and provides the communication protection (integrity and confidentiality) according to the OPTIGA™ Shielded Connection technology specified by [IFX_I2C].
The OPTIGA™ Shielded Connection technology gets controlled by the Enabler APIs through its Service Layer components.
The **Application Layer** provides the functionality of the OPTIGA™ as defined in chapter Commands of this document.

The protocol variation for the OPTIGA™ is defined by Table "Protocol Stack Variation".

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

Property	Value	Comment
MAX_SCL_FREQU	1000[2]	KHz
GUARD_TIME	50	µs
I2C_STATE		SOFT_RESET = 1; CONT_READ = 0; REP_START = 0; CLK_STRETCHING = 0; PRESENT_LAYER = 1;

Table 2 - Protocol Stack Variation

4.4 Commands

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

4.4.1 Command definitions

Command Codes

Table 'Command Codes' lists the command codes for the functionality provided by the OPTIGA™.

Command Code	Name	Short description
0x01 or 0x81	GetDataObject	Command to get (read) an data object
0x02 or 0x82	SetDataObject	Command to set (write) an data object
0x03 or 0x83	SetObjectProtected	Command to set (write) a data object protected.
0x0C or 0x8C	GetRandom	Command to generate a random stream
0x1E or 0x9E	EncryptAsym	Command to encrypt a message based on an asymmetric key scheme
0x1F or 0x9F	DecryptAsym	Command to decrypt a message based on an asymmetric key scheme
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
0x71 or 0xF1	CloseApplication	Command to close/terminate an application

Table 3 - Command Codes

Table 'APDU Fields' lists the fields contained in a command and response APDU.

[2] The default setting is 400 KHz

Name	Description
Cmd	Command code[3] 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 4 - APDU Fields

The Generic Source and Destination definition allows providing and returning of command and response data from or to three types of objects which are defined within the InData part of the command definition. Each object is defined by an associated TLV object. For commands the source of data fed in the command execution could be actual input data, the data or key store, or a session context.
The ***input data*** are represented by a tag, the actual length of the data and the data itself.
The ***data or key store*** is represented by a tag, the length (=2) of the regarded identifier and the OID of the data or key object.
The ***session context*** is represented by a tag, the length (=2) of the regarded identifier and the OID of the session context. The session context behaves as a temporary volatile storage space where various intermediate data might be buffered or retrieved from. Those data could be:

- an ephemeral key
- a shared session secret
- etc.

The Session context could be addressed as part of the command definition being input to a command definition or target for the response or parts of the response.

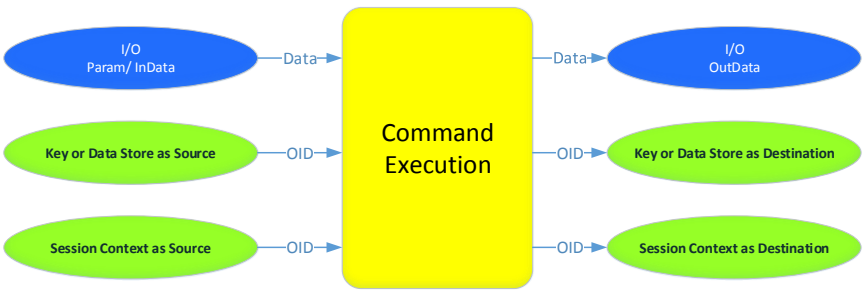

Figure 22 - Generic Source and Destination Definition

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

Response Status Codes

Table 'Response Status Codes' lists the status codes provided by an response APDU.

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

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 Password	0x02	Invalid Password
Invalid Param field	0x03	Invalid Param field in command
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
Counter threshold limit exceeded	0x0E	Counter value crossed the threshold limit and further counting is denied.
Invalid Manifest	0x0F	• The Manifest version provided is not supported or the Payload Version in Manifest has MSB set (Invalid Flag=1) • Invalid or un-supported manifest values or formats including CBOR parsing errors.

Field	Code	Description
Invalid/Wrong Payload Version	0x10	The Payload Version provided in the Manifest is not greater than the version of the target object, or the last update was interrupted and the restarted/retried update has not the same version.
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. 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 key usage / Algorithm extension/identifier for the usage of Private key
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™ is expired.
Unsupported Trust Anchor	0x28	The cryptographic algorithms specified in Trust Anchor loaded are not supported by OPTIGA™.
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.
Signature verification failure	0x2C	Signature verification failure.
Integrity validation failure	0x2D	Message Integrity validation failure (e.g. during CCM decryption).
Decryption Failure	0x2E	Decryption Failure

Table 6 - Error Codes

4.4.1.1 OpenApplication

The OpenApplication is used to open an application on the OPTIGA™. Since after cold or warm Reset all applications residing on the OPTIGA™ are closed, an application has to be opened before using it. This command initializes the application context. This command might be issued multiple times as well to re-initialize an already opened application context. Optionally a previous saved application context could be restored. In any case a saved context gets invalidated/ flushed as soon as the application context is initialized. In case an invalid context handle is used with the restore function the application context gets flushed and an error gets returned.
Note: The OpenApplication (restore) after restoring the context, enforces the presentation layer of the communication stack to be enabled if the CloseApplication (hibernate) was performed with presentation layer enabled.

Field	Offset [direct]	Description
Cmd	0 [in]	**0x70 | 0xF0**[4] Command Code
Param	1 [in]	**0x00** Initialize a clean application context **0x01** Restore the application context from the previously saved context.
InLen	2 [in]	**0xXXXX** Length of *InData*
InData	4 [in]	**Param = 0x00** • **0x00-0xFF** Unique Application Identifier (refer to Table 'Data Structure Unique Application Identifier') **Param = 0x01** • **0x00-0xFF** Unique Application Identifier (refer to Table 'Data Structure Unique Application Identifier') • **0x00-0xFF** (8 Bytes) Context handle as returned by the CloseApplication command with Param = 0x01.
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

[4] In case of 0xF0 the Last Error Code gets flushed

4.4.1.2 CloseApplication

The CloseApplication is used to close an application on the OPTIGA™. The application to be closed gets addressed by communication means like a dedicated Network channel. The application context becomes invalid and all resources allocated at OpenApplication and during the execution of the application get released to the OS for further reuse. After the CloseApplication command is successful executed no further commands specified for the closed application, except OpenApplication, is available. Optionally, this command might save the application context persistently. This allows surviving power-lost by keeping the achieved security state and session contexts of the application. This application context could be restored once by the next OpenApplication command.

Field	Offset [direct]	Description
Cmd	0 [in]	**0x71 | 0xF1**[5] Command Code
Param	1 [in]	**0x00** Close the application instance without saving the application context. **0x01** Saving the application context, close the application instance, and return the random (TRNG) context handle.[6]
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** or **0x0008** Length of *OutData*
OutData	4 [out]	**Param = 0x00** Absent **Param = 0x01** • **0x00-0xFF** (8 Bytes) Context handle to be used by the OpenApplication command as reference for restoring the context (Param = 0x01).

Table 8 - CloseApplication Coding

[5] In case of 0xF1 the Last Error Code gets flushed
[6] Saving the context is only possible when the Security Event Counter (SEC) XE "Security Event Counter (SEC)" value is zero, otherwise it returns an error Command out of sequence XE "Command out of sequence" to the application

4.4.1.3 GetDataObject

The GetDataObject command is used to read data objects from the OPTIGA™. 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 | 0x81**[7] 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[8] 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 9 - GetDataObject Coding

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

4.4.1.4 SetDataObject

The SetDataObject command is used to write data objects to the OPTIGA™. 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 | 0x82**[9] Command Code
Param	1 [in]	• **0x00** Write data • **0x01** Write metadata[10] • **0x02** Count data object [11] [12] [13] • **0x40** Erase & write data
InLen	2 [in]	**0xXXXX** Length of InData
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. In case of Param = "Count data object", the count value represented in Data bytes must be a single byte non-zero value.
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 10 - SetDataObject Coding

[9] In case of 0x82 the Last Error Code gets flushed
[10] In this case, the offset must be 0x0000 and the constructed metadata is provided in the Data field. However, only those metadata tags, which are going to be changed must be contained
[11] The offset given in InData must be ignored
[12] The counter value gets counted by the provided value (offset 4 in InData XE "InData"). As soon as the counter reaches the threshold (either exact or beyond) the counter gets set to the threshold value and any further count attempts will return an error. The change (CHA) access condition allows writing the counter and threshold values like a Byte String XE "Byte String" type data object.
[13] The execute (EXE) access condition is considered for counting

4.4.1.5 SetObjectProtected

The SetObjectProtected command is used to write data objects integrity protected to the OPTIGA™. The field "Param" contains the manifest version of the update data set. The field "InData" contains the protected update data set to be written. The contained manifest addresses the protection keys and the target object.

Note:

- This command supports chaining (start, continue, finalize) through partial write.
- This command does not support the data objects specified below.
  - Life Cycle Status Global
  - Life Cycle Stats Application
  - Security Status Application
  - Security Status Global
  - Coprocessor UID
  - Sleep mode activation delay
  - Current Limitation
  - Security Event Counter
  - Last Error Code
  - Maximum Com Buffer Size
- The Trust Anchor data object used to enable the integrity protection (in the metadata access conditions of target data object) and the target data object to be updated must not be same.
- The manifest provided as part of InData must follow the strict cbor [CBOR] encoding as specified in Manifest and Signature format (Refer Appendix).
- For "WriteType = Write" (refer Manifest CDDL format), if the command execution fails during either Continue or Final and payload version is already invalidated, reattempt with "WriteType = Write" is allowed only with the same payload version until the target data object gets successfully updated.

Field	Offset [direct]	Description
Cmd	0 [in]	**0x03 | 0x83**[14] Command Code
Param	1 [in]	• **0x01** manifest format (CDDL CBOR)
InLen	2 [in]	**0xXXXX** Length of InData
InData	4 [in]	**0x3y, 0xXXXX, 0x00-0xFF** (start y = 0, final y = 1, continue y = 2) start => manifest of update data set[15] continue => first to n-1th payload block of update data set[16] final => last payload block of update data set[17]
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 11 - SetObjectProtected Coding

[14] In case of 0x83 the Last Error Code gets flushed
[15] Start will terminate and clear any not completed sequence (final not executed)
[16] the length must be 640 bytes
[17] The length must be in a range of 1 to 640 bytes

4.4.1.6 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 generated. The random stream could be returned with the response APDU or saved at a session context for later use as e.g. Pre-Master-Secret within the DeriveKey command.

Field	Offset [direct]	Description
Cmd	0 [in]	**0x0C | 0x8C**[18] Command Code
Param	1 [in]	• **0x00** Random number from TRNG (according [AIS-31]) • **0x01** Random number from DRNG (according [SP 800-90A]) • **0x04** Pre-Master Secret to be temporarily stored in the addressed session context[19]
InLen	2 [in]	**0xXXXX** Length of InData
InData	4 [in]	**0x0008-0x0100** length of random stream to be generated • In case of Param **= 0x04** additional: **0xE100-0xE103** OID of session Context 1-4 **0x41**, Length, prepending optional data[20] [21]
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. • In case of Param = **0x00-0x01**, the random stream gets returned. *Note: absent in case of Param = 0x04 (OutLen = 0x0000).*

Table 12 - GetRandom Coding

[18] In case of 0x8C the Last Error Code gets flushed
[19] The DRNG mode gets used to generate the random value
[20] The pre-pending optional data plus the requested length of the random value shall not exceed 48 bytes
[21] Length could be 0x0000

4.4.1.7 EncryptAsym

The EncryptAsym is used to protect an arbitrary message by the OPTIGA™, based on a public key scheme.
Note: In case the shared secret from a session is used and the cryptogram gets returned and the presentation layer protection is enabled the command must be protected. The response protection is up to the caller.

Field	Offset [direct]	Description
Cmd	0 [in]	**0x1E | 0x9E**[22] Command Code
Param	1 [in]	**0xXX** cipher suite as specified by table Asymmetric Cipher Suite Identifier.
InLen	2 [in]	**0xXXXX** Length of InData
InData	4 [in]	Encryption Input InData[InLen] • alternate one:{ (0x61, Length[23], Message) (0x02, 0x0002, OID of session context to be used in case a Pre-Master-Secret from GetRandom gets encrypted.)} • alternate one:{ (0x04, 0x0002, OID of Public Key Certificate[24], (0x05, 0x0001, Algorithm Identifier (of the Public Key), 0x06, Length, Public Key[25])
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]	• 0x61, 0xXXXX, 0x00-0xFF Message data protected

Table 13 - EncryptAsym Coding

[22] In case of 0x9E the Last Error Code gets flushed
[23] The length of the message must be up to the key size minus the minimum size of the applied padding scheme
[24] Must be a single certificate (DER coded) with the key usage as encryption according [RFC5280]. The first byte of the object must be 0x30 (which is the start byte of a DER encoded certificate)
[25] PubKey is encoded as two DER INTEGER (Modulus || Public Exponent) contained in a DER "BIT STRING"

4.4.1.8 DecryptAsym

The DecryptAsym is used to unprotect an arbitrary message by the OPTIGA™, based on a public key scheme.
Note: In case the presentation layer protection is enabled the response must be protected in case the unprotected data get exported. The command protection is up to the caller.

Field	Offset [direct]	Description
Cmd	0 [in]	**0x1F | 0x9F**[26] Command Code
Param	1 [in]	**0xXX** cipher suite as specified by table Asymmetric Cipher Suite Identifier.
InLen	2 [in]	**0xXXXX** Length of InData
InData	4 [in]	Decryption Input InData[InLen] • 0x61, Length[27], Protected message • 0x03, 0x0002, OID of decryption key[28] *Note: The key usage of the addressed key must be set to Enc; refer to Key Usage Identifier.* • optional:{ (0x02, 0x0002, OID of session context to store the decrypted data.)[29]}
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]	• 0x61, 0xXXXX, 0x00-0xFF Message data unprotected *Note: Absent in case targeting the session context (*OID of session context *provided in InData).*

Table 14 - DecryptAsym Coding

[26] In case of 0x9F the Last Error Code gets flushed
[27] The length of the message must match the key size
[28] The addressed decryption key shall be a RSA private key
[29] the length of the decrypted data shall not exceed 48 bytes and the usage is limited as input shared secret in DeriveKey XE "DeriveKey" command

4.4.1.9 CalcHash

The CalcHash is used calculating a digest of a message by the OPTIGA™. 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™ 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) will terminate the currently running hash calculation and drop the result.

Field	Offset [direct]	Description
Cmd	0 [in]	**0x30 | 0xB0**[30] 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[31], Message data (start y = 0, start&final y = 1, continue y = 2, final y=3, final and keep intermediate hash y=5[32]) • 0x04, 0x0000 - To terminate the hash sequence in case initialized already. • 0x1y, 0x0006, OID[33], Offset, Length[34] (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

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

Field	Offset [direct]	Description
		0x01, Length, Hash/Digest 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 15 - CalcHash Coding

4.4.1.10 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 | 0xB1**[35] 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[36], Digest to be signed • 0x03, 0x0002, OID of signature key[37] 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[38] Note: The length of the signature is derived from the applied key and signature scheme.

Table 16 - CalcSign Coding

[35] In case of 0xB1 the Last Error Code gets flushed
[36] For ECC shall be 10 bytes up to the length of the addressed signature key; RSA case: must be exactly equal to the the output lengh of the hash algorithm used
[37] The addressed signing key shall be a private key
[38] ECC case: The signature pair (r,s) is encoded as two DER "INTEGER"; RSA case: The signature encoding is OCTET STRING

4.4.1.11 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 | 0xB2**[39] 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[40], Digest • 0x02, Length[41], Signature over Digest[42] • alternate one:{ (0x04, 0x0002, OID of Public Key Certificate[43], (0x05, 0x0001, Algorithm Identifier (of the Public Key), 0x06, Length, Public Key[44])}
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 17 - VerifySign Coding

[39] In case of 0xB2 the Last Error Code gets flushed
[40] ECC case: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; RSA case: must be exactly equal to the output length of the hash algorithm used
[41] The length is limited to max. 520 bytes
[42] ECC case: The signature pair (r,s) is encoded as two DER "INTEGER"; RSA case: The signature encoding is OCTET STRING
[43] 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)
[44] PubKey is encoded as DER "BIT STRING"

4.4.1.12 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 | 0xB8**[45] 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[46] 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]	• Alternative one[{ECC key 0x01, Len, PrivKey[47] 0x02, Len, PubKey[48]} {RSA Key 0x01, Len, PrivKey[49] 0x02, Len, PubKey[50]}]

Table 18 - GenKeyPair Coding

[45] In case of 0xB8 the Last Error Code gets flushed
[46] 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.
[47] PrivKey is encoded as DER "OCTET STRING"
[48] PubKey is encoded as DER "BIT STRING"
[49] PrivKey is encoded as DER "OCTET STRING"
[50] PubKey is encoded as two DER INTEGER (Modulus || Public Exponent) contained in a DER "BIT STRING"

4.4.1.13 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 | 0xB3**[51] Command Code
Param	1 [in]	**0xXX** Key agreement primitive (refer to *Key Agreement Schemes*)
InLen	2 [in]	**0xXXXX** Length of InData
InData	4 [in]	• 0x01, 0x0002, OID of Private Key • 0x05, 0x0001, Algorithm Identifier, 0x06, Length, Public Key[52] (alternative one) • 0x07, 0x0000[53] • 0x08, 0x0002, OID of Shared Secret[54]
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 [55] Note 1: Shared secret is only returned in case it is requested by InData (0x07, 0x0000)

Table 19 - CalcSSec Coding

[51] In case of 0xB3 the Last Error Code gets flushed
[52] Public Key is encoded as DER "BIT STRING"
[53] Indicates exporting the shared secret via the external interface
[54] The shared secret becomes part of the session context and can be addressed until the session context gets flushed
[55] The shared secret is encoded as OCTET STRING

4.4.1.14 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.
Note: In case the shared secret from a session is used and the presentation layer protection is enabled the command must be protected. The response protection is up to the caller.

Field	Offset [direct]	Description
Cmd	0 [in]	**0x34 | 0xB4**[56] 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[57] • 0x02, Len, Secret derivation data[58] • 0x03, 0x0002, Length of the key to be derived[59] (alternative one) • 0x07, 0x0000[60] • 0x08, 0x0002, OID of derived key[61]
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 20 - DeriveKey Coding

[56] In case of 0xB4 the Last Error Code gets flushed
[57] 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.
[58] min. Length = 8 byte; max. length = 1024 byte
[59] min. Length = 16 byte; max. length = 48 byte in case of session reference; max. length = 256 byte in case of returned secret
[60] Indicates exporting the derived key via the external interface
[61] The key becomes part of the session context and can be addressed as shared secret until the session context gets flushed

4.4.2 Command Parameter Identifier

Table 'Algorithm Identifier' lists the algorithm identifier supported by the OPTIGA™.

Value	Description
0x03	Elliptic Curve Key on NIST P256 curve.
0x04	Elliptic Curve Key on NIST P384 curve
0x41	RSA Key 1024 bit exponential format
0x42	RSA Key 2048 bit exponential format
0xE2	SHA 256

Table 21 - Algorithm Identifier

Table 'Key Usage Identifier' lists the key usage identifier supported by the OPTIGA™.

Value	Description
0x01	*Auth* (Authentication)
0x02	Enc (Encryption, Decryption, Key Transport)
0x10	*Sign*
0x20	*KeyAgree* (Key Agreement)

Table 22 - Key Usage Identifier

Table 'Asymmetric Cipher Suite Identifier' lists the supported asymmetric cipher suites used in public key schemes and their coding.

Value	Description
0x11	Cipher suite PKCS#1v2.2 RSAES-PKCS1-v1.5 (RSA key pair with PKCS1 v1.5 padding) according to [RFC8017]

Table 23 - Asymmetric Cipher Suite Identifier

Table 'Key Agreement Schemes' lists the key agreement schemes supported by the OPTIGA™.

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

Table 24 - Key Agreement Schemes

Table 'Key Derivation Method' lists the key derivation method supported by the OPTIGA™.

Value	Description
0x01	TLS PRF SHA256 according to [RFC5246].

Table 25 - Key Derivation Method

Table 'Signature Schemes' lists the signature schemes supported by the OPTIGA™.

Value	Description
0x01	PKCS#1v2.2 RSASSA-PKCS1-v1.5_SHA256 according to [RFC8017] w/o hash
0x02	PKCS#1v2.2 RSASSA-PKCS1-v1.5_SHA384 according to [RFC8017] w/o hash
0x11	*ECDSA FIPS 186-3* w/o hash

Table 26 - Signature Schemes

4.4.3 Crypto Performance

The performance metrics for various schemes are provided by Table 'Crypto Performance Metrics'. If not particular mentioned the performance is measured @ OPTIGA™ I/O interface including data transmission with:

- I2C FM mode (400KHz,)
- Without power limitation,
- @ 25°C,
- VCC = 3.3V

Scheme	Sequence	Execution Time[62]	Note
Calculate signature	CalcSign (ECDSA FIPS 186-3)	ECC NIST P256 < 60ms	Toolbox function w/o message hashing
Calculate signature	CalcSign (PKCS#1v2.2 RSASSA-PKCS1-v1.5_SHA256)	RSA 2048 bit exponential < 310ms	Toolbox function w/o message hashing
Verify signature	VerifySign (ECDSA FIPS 186-3)	ECC NIST P256 < 85ms	Toolbox function w/o message hashing
Verify signature	VerifySign (PKCS#1v2.2 RSASSA-PKCS1-v1.5_SHA256)	RSA 2048 bit exponential < 45ms	Toolbox function w/o message hashing
Diffie Hellman key agreement	CalcSSec (NIST P256)	ECC NIST P256 < 65ms	Based on ephemeral key pair
Key pair generation	GenKeyPair	ECC NIST P256 < 80ms	
Key pair generation	GenKeyPair	RSA 2048 bit exponential minimum 2900 ms	
Key derivation	DeriveKey (TLS v1.2 PRF SHA256)	AES128 CCM < 140ms	
Hash calculation	CalcHash	SHA256 ~ 5Kbyte/s	In blocks of 500 bytes

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

Scheme	Sequence	Execution Time[62]	Note
RSA Encryption	EncryptAsym(PKCS# 1v2.2 RSAES-PKCS1-v1.5)	< 50ms	
RSA Decryption	DecryptAsym(PKCS# 1v2.2 RSAES-PKCS1-v1.5)	< 315ms	

Table 27 - Crypto Performance Metrics

4.5 Security Policy

A Security Policy is a crucial concept for turning a generic cryptographic device into an optimally tailored device for the respective customer needs. The essential components are the Policy-Enforcement-Point and Policy-Attributes.

4.5.1 Overview

In order to define a project specific security set-up the OPTIGA™ provides a set of Policy Attributes and a Policy Enforcement Point. The Policy Enforcement Point hosted by the key and data store, combines the Policy Attributes with the access conditions associated with each key or data object and finally judges whether the intended access is permitted or not.

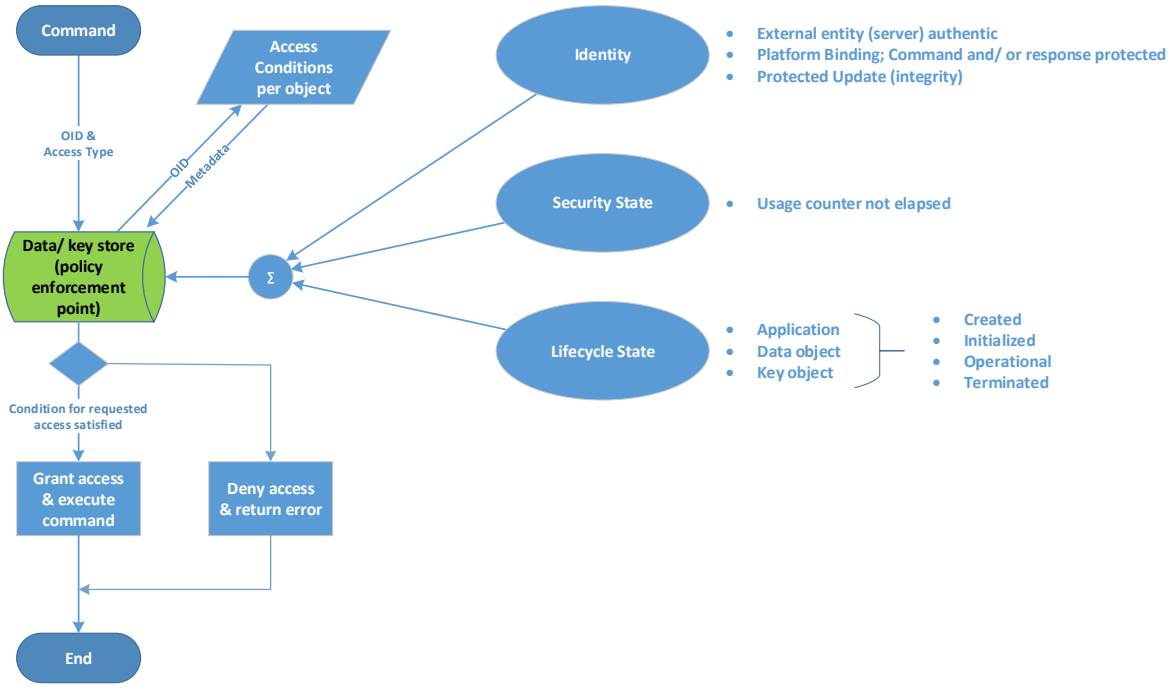

Figure 23 - Security Policy Architecture

4.5.2 Policy Attributes

The Policy Attributes are grouped in Identity, Security State, Platform State and Life Cycle State

based attributes.

- Identity (e.g. Identity of the Host: platform binding, namely OPTIGA Shielded Connection technology, is used for command/response)
- Security State (e.g. Usage counter of a data object or key is not elapsed)
- Life Cycle State (Lcs); e.g. Lcs for an object is in initialization state

4.5.3 Policy Enforcement Point

The key and data store implementation acts as Policy Enforcement Point. The enforcement is expressed by granting or denying a type of access (read, change, execute) to a dedicated key or data object, which is addressed by its unique object identifier (OID). The diagram below depicts the flow which leads to granting or denying the respective access.

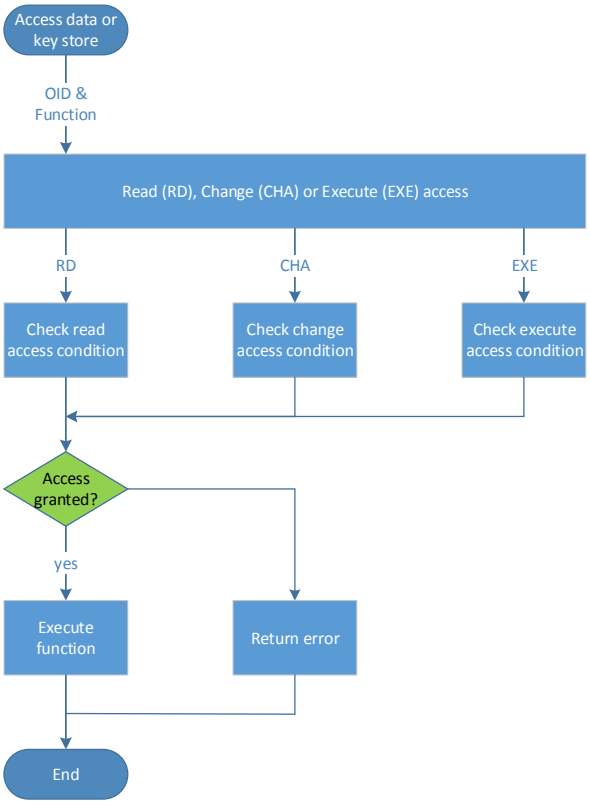

Figure 24 - Policy Enforcement Flow

4.6 Security Monitor

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

4.6.1 Security Events

Table 'Security Events' provides the definition of not permitted security events considered by the OPTIGA™ implementation.

Name	Description
Key Derivation	The Key Derivation event occurs in case the DeriveKey command gets applied on a persistent data object (not volatile data object as session context). In that case the persistent data object gets used as pre-shared secret.
Private Key Use	The Private Key Use event occurs in case the internal services are going to use a OPTIGA™ hosted private key, except the temporary keys from session context used for key agreement like ECDH.
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 28 - Security Events

4.6.2 Security Monitor Policy

This paragraph provides all details of the policy chosen for the OPTIGA™ project.
In order to mitigate exhaustive testing of the OPTIGA™ 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 it 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.

4.6.3 Security Monitor 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.

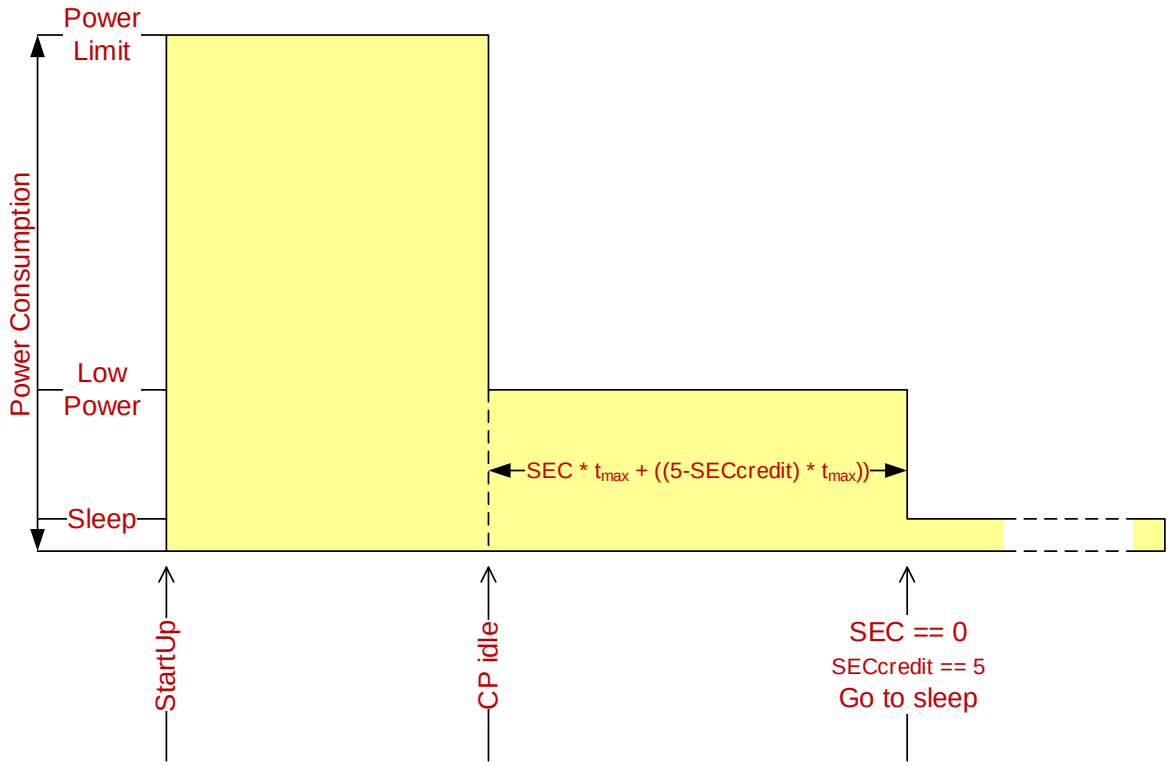

Figure 25 - Power profile

Figure "*Power Profile*" depicts the power profile of a regular startup sequence, caused either by PowerUP, Warm Reset, or Security Reset. The OPTIGA™ 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™ enters idle state (nothing to compute or communicate) the OPTIGA™ reduces its power consumption to the low power limit (ref for details to “System Halt Power Consumption” in [SLE70PRM]). 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™ is in idle state, the OPTIGA™ 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™ 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 200.000 times over lifetime.

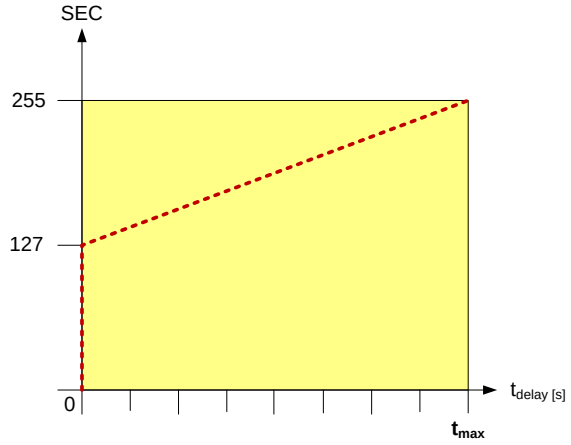

Figure 26 - Throttling down profile

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.

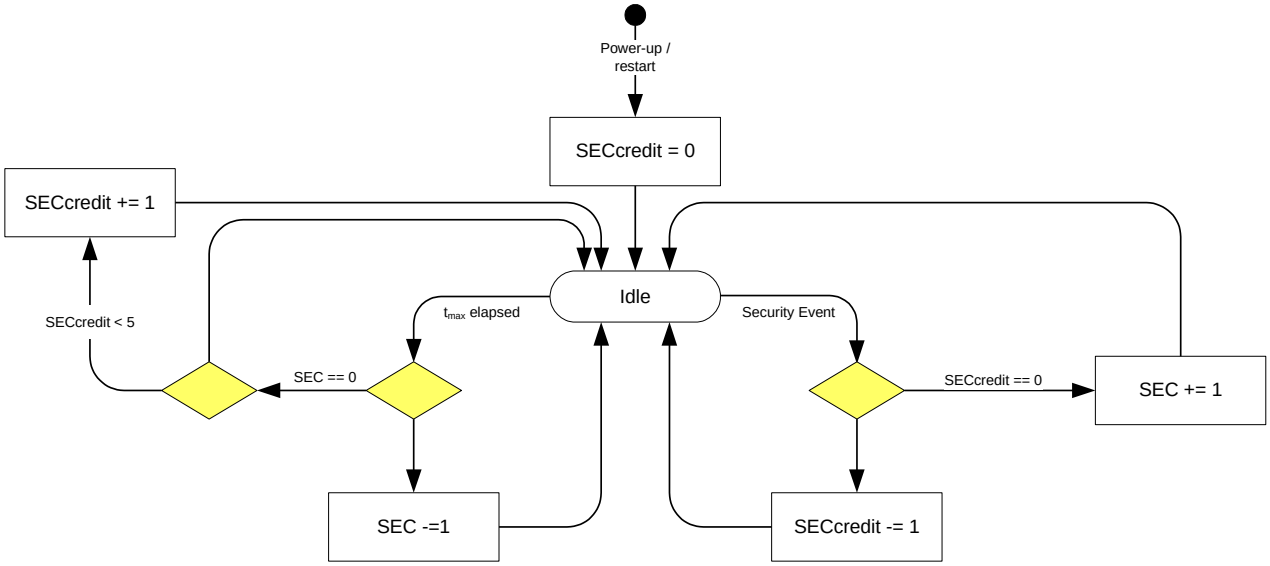

Figure 27 - Security Event Counter Characteristics

5 OPTIGA™ Trust M Data Structures

5.1 Overview Data and Key Store

The picture below provides an overview of all data and key objects hosted by the OPTIGA™.

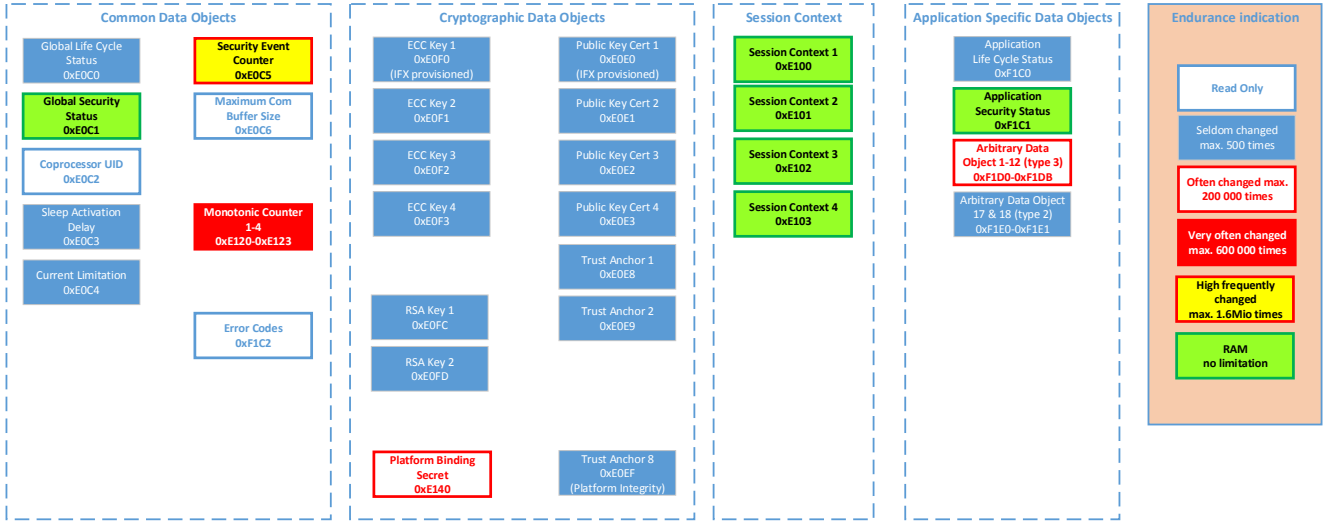

Figure 28 - Overview Data and Key Store

5.2 Access Conditions (ACs)

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)

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.

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.

Conf(X) the action is only possible in case the data involved (to be read/write) are confidentiality protected with key given by X.

Int(X) the action is only possible in case the data involved (to be read/write) are integrity protected with key given by X.

The following access driven behavior definition is used in this document:

Luc(x) in case of EXE accessing the object, the linked counter defined by X gets advanced by 1 and the action is allowed in case the count value did not reach its threshold value.

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 (Examples how to code are given in chapter Metadata expression).

- 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 combine sACs to an access token (AT)
  AT = sAC_1 … && sAC_n (n = 1…7)
  ... || operators combine 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™ data and key objects are defined by Table 'Access Condition Identifier and Operators'.

AC ID	Operator	Value	Description
ALW	-	0x00	1 byte; Value
Conf	-	0x20	3 byte; Value, Key Reference (e.g. Conf first Session Key → 0x20, 0xF1, 0xF0) (e.g. Read, Conf, Binding Secret → 0xD1, 0x03, 0x20, 0xE1, 0x40) In case of reading a data object (e.g. using GetDataObject), the shielded connection must be established already using the specified pre-shared secret (0xE140) and the response is requested with protection (encrypted). (e.g. Change, Conf, Binding Secret → 0xD0, 0x03, 0x20, 0xE1, 0x40) In case of writing a data object (e.g. using SetDataObject), the shielded connection must be established already using the specified pre-shared secret (0xE140) and the command is sent with protection (encrypted). (e.g. Execute, Conf, Binding Secret → 0xD3, 0x03, 0x20, 0xE1, 0x40) In case of using a data object with an internal operation (e.g. using DeriveKey from a pre-shared secret), the shielded connection must be established already using the specified pre-shared secret (0xE140) and the command is sent protection (encrypted).
Int	-	0x21	3 byte; Value, Key Reference (e.g. Int first Session Key → 0x21, 0xF1, 0xF0) (e.g. Read, Int, Binding Secret → 0xD1, 0x03, 0x21, 0xE1, 0x40) In case of reading a data object (e.g. using GetDataObject), the shielded connection must be established already using the specified pre-shared secret (0xE140) and the response is requested with

AC ID	Operator	Value	Description
			protection (MAC). (e.g. Change, Int, Binding Secret → 0xD0, 0x03, 0x21, 0xE1, 0x40) In case of writing a data object (e.g. using SetDataObject), the shielded connection must be established already using the specified pre-shared secret (0xE140) and the command is sent with protection (MAC). (e.g. Change, Int, Trust Anchor → 0xD0, 0x03, 0x21, 0xE0, 0xEF) In case of writing a data object (e.g. using SetObjectProtected), the signature associated with the meta data must be verified with the addressed trust anchor (e.g. 0xE0EF). (e.g. Execute, Int, Binding Secret → 0xD3, 0x03, 0x21, 0xE1, 0x40) In case of using a data object with an internal operation (e.g. using DeriveKey from a pre-shared secret), the shielded connection must be established already using the specified pre-shared secret (0xE140) and the command is sent with protection (MAC).
Luc	-	0x40	3 byte; Value, Counter Reference (e.g. Linked Counter 1 → 0x40, 0xE1, 0x20) For example, The arbitrary data object holds a pre-shared secret and this secret is allowed to be used for key derivation (DeriveKey) operations to a limited number of times. To enable this, choose a counter object (updated with maximum allowed limit) and assign the counter data object in the EXE access condition of arbitrary data object as shown below. (e.g. EXE, Luc, Counter Object → 0xD3, 0x03, 0x40, 0xE1, 0x20) The counter data objects gets updated (counter value gets incremented by 1 up to maximum limit) automatically when the DeriveKey command is performed.

AC ID	Operator	Value	Description
LcsG	-	0x70	3 byte; Value, Qualifier, Reference (e.g. LcsG < op → 0x70, 0xFC, 0x07)
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, 0xFC, 0x07)
-	==	0xFA	equal
-	>	0xFB	greater than
-	<	0xFC	less than
-	&&	0xFD	logical AND
-	||	0xFE	logical OR
NEV	-	0xFF	1 byte; Value

Table 29 - Access Condition Identifier and Operators

Table 'Data Object Types' lists the various types of key and data objects supported by OPTIGA™.

Name	Value	Description
BSTR	0x00	The Byte String data object type is represented by a sequence of bytes, which could be addressed by offset and length.
UPCTR	0x01	The Up-counter data type implements a counter with a current value which could be increased only and a threshold terminating the counter.
TA	0x11	The Trust Anchor data type contains a single X.509 certificate which could be used in various commands requiring a root of trust.
DEVCERT	0x12	The Device Identity data type contains a single X.509 certificate or a chain of certificates (TLS, USB-Type C, ...) which was issued to vouch for the cryptographic identity of the end-device.
PRESSEC	0x21	The Pre-shared Secret contains a binary data string which makes up a pre-shared secret for various

Name	Value	Description
		purposes (FW-decryption, ...).
PTFBIND	0x22	The Platform Binding contains a binary data string which makes up a pre-shared secret for platform binding (e.g. used for OPTIGA™ Shielded Connection).
UPDATSEC	0x23	The Protected Update Secret contains a binary data string which makes up a pre-shared secret for confidentiality protected update of data or key objects.

Table 30 - Data Object Types

5.3 Life Cycle State

The device, the application, and key and data objects have a life cycle state associated; the life cycle status (LCS) allows to identify the different states of the associated logical units throughout the OPTIGA™ lifetime. To support flexible management of the life cycle, four primary states (Bit 2^3 - 2^0) are defined in the following order:

1. Creation state (cr)
2. Initialization state (in)
3. Operational state (op)
4. Termination state (te)

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 (in) => operational (op) state, but not vice versa). The application-specific part of the LCS, if used at all, are managed by the particular application.

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

5.4 Common and application specific objects and ACs

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

OPTIGA™ Trust M Data Structures

Tag	Structure definition	Default Value	Execute	Change	Read	Note
0xE0C0	Data Structure global '*Life Cycle Status*' (LcsG)	0x07	NEV	ALW	ALW	default LcsO = op
0xE0C1	Data Structure global '*Security Status*'	0x00	NEV	ALW[63]	ALW	default LcsO = op
0xE0C2	Data structure 'Coprocessor UID OPTIGA™ Trust Family'		NEV	NEV	ALW	default LcsO = op
0xE0C3	Data structure '*Sleep Mode Activation Delay*' (refer to '*Common data structures*')	0x14	NEV	ALW	ALW	default LcsO = op
0xE0C4	Data structure '*Current limitation*' (refer to '*Common data structures*')	0x06	NEV	ALW	ALW	default LcsO = op
0xE0C5	Data structure '*Security Event Counter (SEC)*' (refer to '*Common data structures*')		NEV	NEV	ALW	default LcsO = op
0xE0C6	Data structure 'Maximum Com Buffer Size' (refer to 'Common data structures')		NEV	NEV	ALW	default LcsO = op
0xE0E0	Device *Public Key Certificate* issued by IFX (refer to '*Common data structures*')		ALW	NEV	ALW	default LcsO = cr; default Data Type is "Device Identity"
0xE0E1-0xE0E3	Project-specific device Public Key Certificate 1-3[64]. (refer to 'Common data structures')	0x00	ALW	LcsO < op	ALW	default LcsO = cr; default Data Type is "Device Identity"
0xE0E8-0xE0E9	Root CA Public Key Certificate 1-2[65] (refer to 'Common data structures')		ALW	LcsO < op	ALW	default LcsO = cr; default Data Type is "Trust Anchor"
0xE0EF	Root CA Public Key Certificate 8[66]. This trust anchor is assigned to platform	0x00	ALW	LcsO < op	ALW	default LcsO = cr; default Data Type is "Trust Anchor"

[63] It is only possible to reset an achieved security status
[64] 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
[65] 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.
[66] 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.

Tag	Structure definition	Default Value	Execute	Change	Read	Note
	integrity use cases (refer to 'Common data structures').					
0xE120-0xE123	Monotonic Counter 1-4		ALW	LcsO < op	ALW	default LcsO = in; This monotonic counters could be used as general purpose counters or getting linked (via AC Linked Usage Counter) to another data object. In case of a linked characteristics the change (CHA) AC shall be Never avoiding DoS attacks.
0xE140	Shared Platform Binding Secret.		ALW	LcsO < op || Conf (0xE140)	LcsO < op	default LcsO = cr; This data object holds the shared secret for the OPTIGA™ Shielded Connection technology, which establishes a cryptographic binding between the OPTIGA™ and the Host.

Table 31 - Common data objects with TAG's and AC's

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

Tag	Structure definition	Default Value	Execute	Change	Read	Note
0xE0F0	Device Private ECC Key 1		ALW	NEV	NEV	default LcsO = cr
0xE0F1-0xE0F3	Device Private ECC 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	default LcsO = cr
0xE0FC-0xE0FD	Device Private RSA Key 1-2; The SetDataObject commands is not allowed		ALW	LcsO < op	NEV	default LcsO = cr

Tag	Structure definition	Default Value	Execute	Change	Read	Note
	for the data part of the key object even if the metadata state the access rights differently. The GetDataObject returns the public key components if the RD access condition permits it.					
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	default LcsO = op

Table 32 - Common key objects with TAG's and AC's

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

Tag	Structure definition	Default Value	Execute	Change	Read	Note
0xF1C0	Data Structure application '*Life Cycle Status*' (LcsA)	0x01	NEV	ALW	ALW	default LcsO = op
0xF1C1	Data Structure application '*Security Status*'	0x00	NEV	ALW[67]	ALW	default LcsO = op
0xF1C2	Error codes (refer to Table '*Error Codes*')		NEV	NEV[68]	ALW	default LcsO = op

[67] It is only possible to reset an achieved security status
[68] cleared on read

Tag	Structure definition	Default Value	Execute	Change	Read	Note
0xF1D0-0xF1DB	Data Structure Arbitrary data object type 3.	0x00	app-specific	app-specific	app-specific	default LcsO = cr
0xF1E0-0xF1E1	Data Structure Arbitrary data object type 2.	0x00	app-specific	app-specific	app-specific	default LcsO = cr

Table 33 - Authentication application-specific data objects with TAG's and AC's

5.5 Metadata expression

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.*
- *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' lists all common data structures defined for the OPTIGA™ with its TAG's and AC's.

Tag	Structure definition	Default Value	Execute	Change	Read	Note
0x20	Metadata constructed TLV-Object	n.a.	n.a.	n.a.	ALW	
0xC0	Life Cycle State of the data object (LcsO)	n.a.	n.a.	ALW	ALW	refer to Table 'Life Cycle Status'

Tag	Structure definition	Default Value	Execute	Change	Read	Note
0xC1	Version information of the data or key object. The version is represented by 15 bits and the MSB (invalid flag) is indicating whether the object is temporarily invalid. The invalid flag is only controlled by command execution.	n.a.	n.a.	LcsO < op	ALW	0xC1, 0x02, 0x00, 0x00 In case the version tag is absent, this default version is 0x0000. The version is used and updated by the protected update use case (SetObjectProtected) and gets created if not already present. The most significant bit is the object status flag and is masked out for the version info itself. It indicates the data object is valid (0) or invalid (1).
0xC4	Max. size of the data object	n.a.	n.a.	NEV	ALW	
0xC5	Used size of the data object	n.a.	n.a.	auto	ALW	The used length gets updated automatically in case of change (CHA) access. In case it is not already present, it gets created.
0xD0	Change Access Condition descriptor	n.a.	n.a.	LcsO < op	ALW	
0xD1	Read Access Condition descriptor	n.a.	n.a.	LcsO < op	ALW	
0xD3	Execute Access Condition descriptor	n.a.	n.a.	LcsO < op	ALW	
0xE0	Algorithm associated with key container	n.a.	n.a.	auto[69]	ALW	refer to Table 'Algorithm Identifier'
0xE1	Key usage associated with key container	n.a.	n.a.	LcsO < op	ALW	refer to Table 'Key Usage Identifier'
0xE8	Data object Type	n.a.	n.a.	LcsO < op	ALW	Refer to table Data Object Types. In case this tag is not present the data object is represented by an unrestricted array of bytes (described by tags 0xC4

[69] As part of the key generation this tag will be updated automatically

Tag	Structure definition	Default Value	Execute	Change	Read	Note
						and 0xC5).

Table 34 - Metadata associated with data and key objects

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, 0x8C,              // TLV max size = 140
   0xC5, 0x01, 0x0A,              // TLV used size = 10
   0xD1, 0x01, 0x00,              // TLV Read = ALW

   0xD0, 0x03, 0xE1, 0xFC, 0x07   // TLV Change = LcsO < op
```

Note: in case of NEV the AC term for that kind of AC could be absent.

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

```
0x20, 0x13,                       // TL metadata TLV-Object
   0xC0, 0x01, 0x03,              // TLV LcsO = in
   0xC4, 0x02, 0x06, 0xC0,        // TLV max size = 1728
   0xC5, 0x02, 0x03, 0x40,        // TLV used size = 832
   0xD1, 0x01, 0x00,              // TLV Read = ALW
   0xD0, 0x03, 0xE1, 0xFC, 0x07; // TLV Change = LcsO < op
```

Note: there is no ordering rule for metadata tags

OPTIGA™ Trust M Data Structures

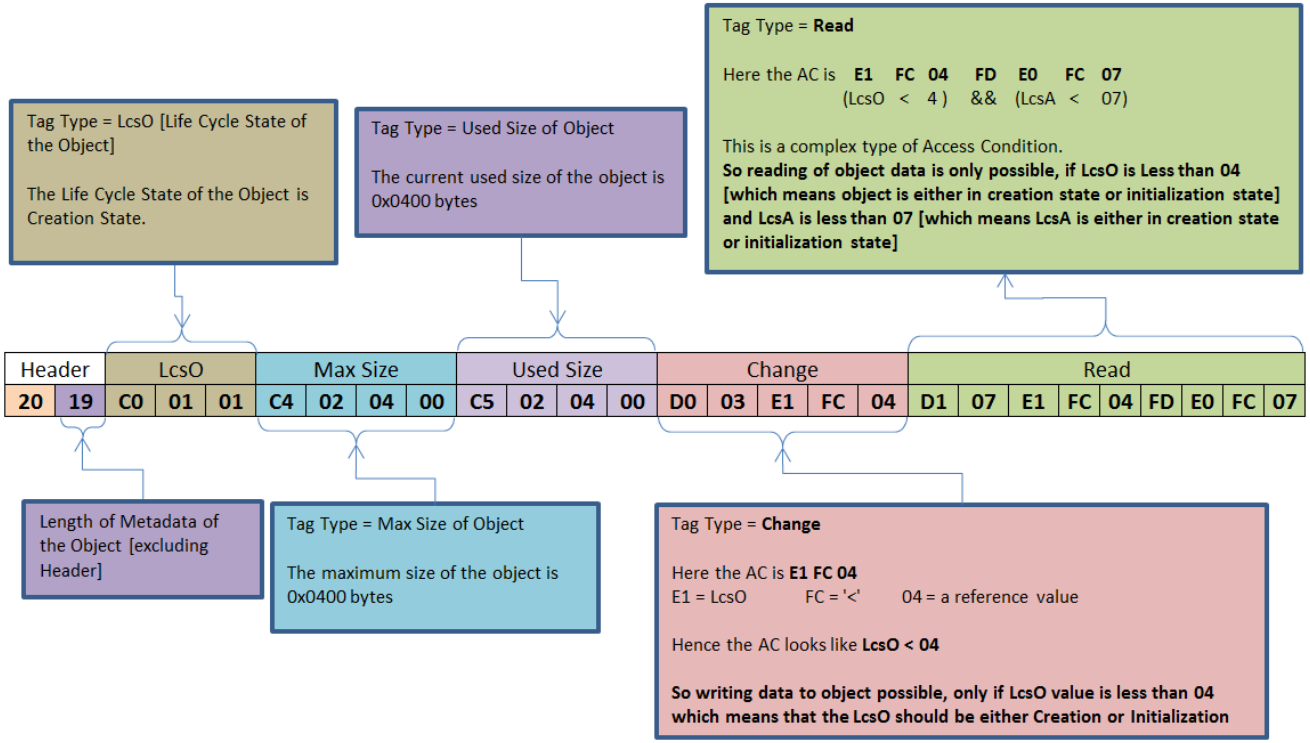

Figure 29 - Metadata sample

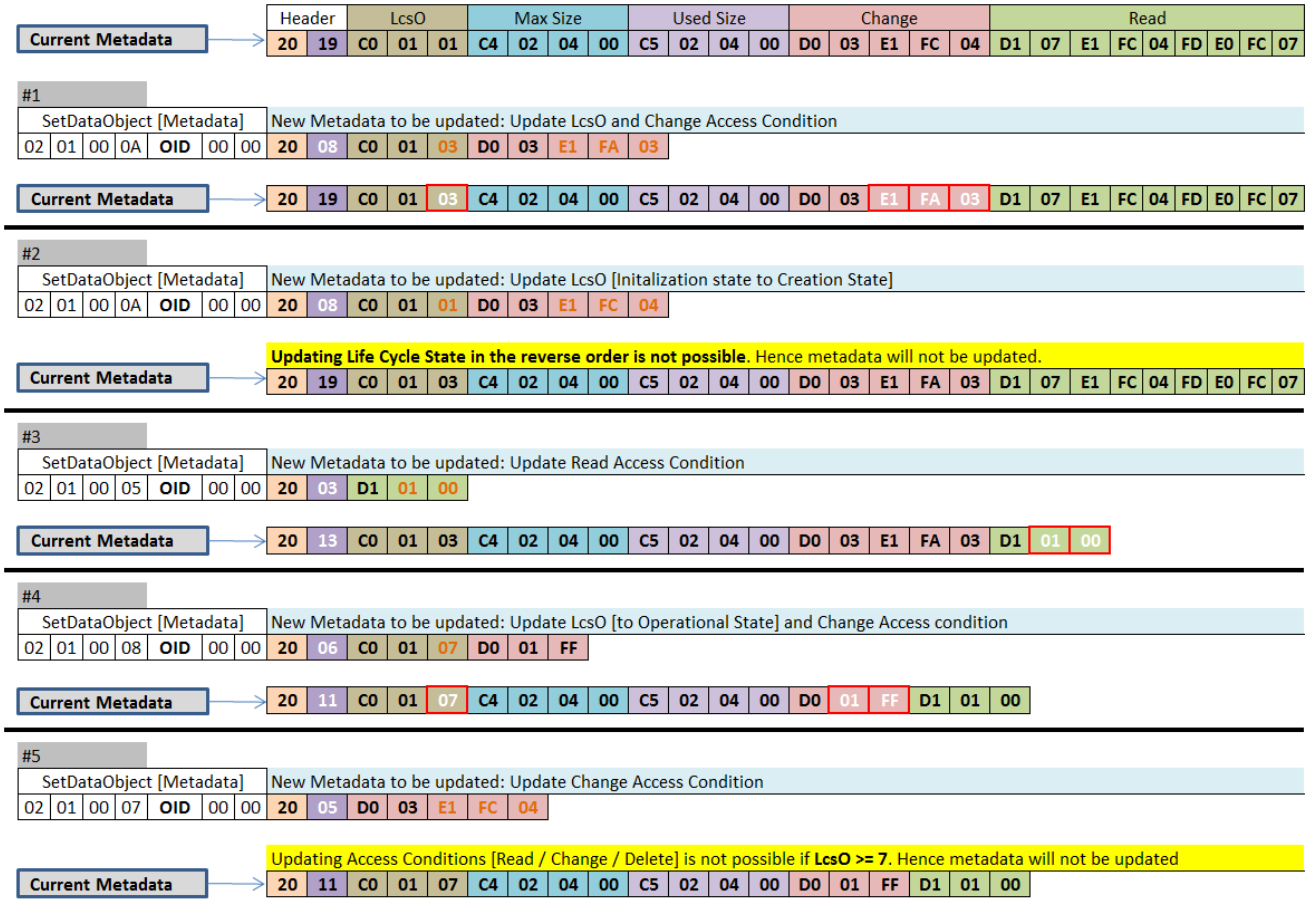

Figure 30 - SetDataObject (Metadata) examples

Note: The values specified in Figure 29 and Figure 30 are in HEX format.

5.6 Common data structures

Table 'Common data structures' lists all common data structures defined for the OPTIGA™.

Data element	Value	Coding	Length (Bytes)	Description
Coprocessor UID OPTIGA™ Trust Family	0x00 - 0xFF		27	Unique ID of the OPTIGA™.
Life Cycle State (refer to Table '*Life Cycle Status*')		BinaryNumber8	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 (in) => operational (op) 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*')		BinaryNumber8	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*')		BinaryNumber8	1	The Last Error Code stores the most recent error code generated since the data object was last cleared. The 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.

Data element	Value	Coding	Length (Bytes)	Description
Sleep Mode Activation Delay in ms	0x14 - 0xFF	BinaryNumber8	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	BinaryNumber8	1	The Current limitation holds the maximum value of current allowed to be consumed by the OPTIGA™ 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	BinaryNumber16	2	The Maximum Com Buffer Size holds the maximum size of APDUs transferred through the communication buffer. *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

Data element	Value	Coding	Length (Bytes)	Description
				a Customer. An external Entity (host, server) utilizes it as device identity to authenticate the OPTIGA™ 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[70] • **USB Type-C Identity:** **Tag** = 0xC2 **Length** = Value length (2 Bytes) **Value** = USB Type-C Certificate Chain[USB Auth][71] • **WPC Compressed Identity:** **Tag** = 0xC4 **Length** = Value length (2 Bytes) **Value** = WPC compressed Certificate Chain [WPC Auth][72]
Root CA Public Key Certificate aka "Trust Anchor"	0x00 - 0xFF	x.509 (maybe self signed certificate)	1200 (max.)	The Root CA Public Key Certificate data object holds the Public Key Certificate of the IFX or Customer Root or Intermediate Certification

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

Data element	Value	Coding	Length (Bytes)	Description
				Authority. It is used as "Trust Anchor" to authenticate external entities (e.g. verify server signature).
Platform Binding Secret		0x01 ... 0x40	64	The Platform Binding Secret data object holds the shared secret used during the handshake key agreement as part of the OPTIGA™ Shielded Connection protocol. It shall be 64 bytes and LcsO set to operational (op) and access condition set to CHA = NEV and RD = NEV. If appropriate it is recommended to set the LcsO to initialized (in) during executing the FactoryReset command, allowing newly binding the security ship with the host.
Counter		BinaryNumber64	8	The Counter is a data object consisting of the concatenated counter value (offset 0-3) and the regarded threshold (offset 4-7). The fixed length is 8 Byte. There are two types of monotonic counter the Up-counter and the Down-counter. The Up-counter is only allowed to increase and the Down-counter is only allowed to decrease. The 1 byte value provided gets added or respective subtracted from the counter value (offset 0-3). As soon as the counter reaches or exceeds the threshold, the counter gets set to the threshold value and frozen and returns an error upon attempting to further count.

Table 35 - Common data structures

Table 'Life Cycle Status' lists all coding of the Life Cycle Status defined for the OPTIGA™.

Bit 8-5	Bit 4-1	Description
0 0 0 0	x x x x	RFU (Application-specific states)
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)[73]

Table 36 - Life Cycle Status

Table 'Security Status' shows the security status defined for the device either global or application-specific. The default is set after reset for the global and after OpenApplication for the application-specific Security Status[74] [75].

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

Table 37 - Security Status

Table 'Coprocessor UID OPTIGA™ Trust Family' shows UID definition for the OPTIGA™.

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

[73] this state is not applicable for the LcsA
[74] bit = 0 status not satisfied
[75] bit = 1 status satisfied

Offset	Data Type	Name	Description
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

Table 38 - Coprocessor UID OPTIGA™ Trust Family

5.7 Application-specific data structures

Table 'Data Structure Unique Application Identifier' shows unique identifier for the OPTIGA™ application which gets used with the OpenApplication command.

Data element	Value	Coding	Length (Bytes)
RID	0xD276000004[76]	HexNumber5	5
PIX	'GenAuthAppl' 0x47656E417574684170706C	Ascii11	11

Table 39 - Data Structure Unique Application Identifier

Table 'Data Structure Arbitrary data object' lists the supported types of arbitrary data objects.

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

Data element	Value	Coding	Length (Bytes)
ADO2	0x00 - 0xFF	application-specific	1500
ADO3	0x00 - 0xFF	application-specific	140

Table 40 - Data Structure Arbitrary data object

5.8 Protected Update Data Set

This section provides the definition and some useful information of update data sets for data objects which are used to update those in a integrity protected way.
The figure below shows the high level structure of the update data set for data objects. It consists of a manifest and the connected binary data. The coding of the structure is according to [CBOR] as specified in Chapter Protected Update: Manifest and Signature structures which provides more detailed information regarding the format of manifest consumed.
The **Manifest** is a top level construct that ties all other structures together and is signed by an authorized entity whose identity is represented by a trust anchor installed at the OPTIGA™. The trust anchor is addressed by its unique ID (OID), which is contained in the metadata of the manifest. Manifest consists of the metadata in plain text, the payload binding and the signature over metadata and payload binding.
The **Metadata** provide information enabling interpretation and manage the update data set by the OPTIGA™. It contains:

- Version number
- Unique identifier (OID) of the trust anchor to be used for verifying the metadata signature.
- Unique identifier (OID) of the object to be updated
- Cipher suite specifying all cryptographic algorithms (signature, hash, ...) used during executing the update.
- Offset within the target object and length of the object data.

The **Integrity Protection** of the object data is based on the hash value of the first block of object data which is protected by the successful verification of the signature over the metadata. Each block of object data, except the last block, carries the hash value of the next block of object data.

OPTIGA™ Trust M Data Structures

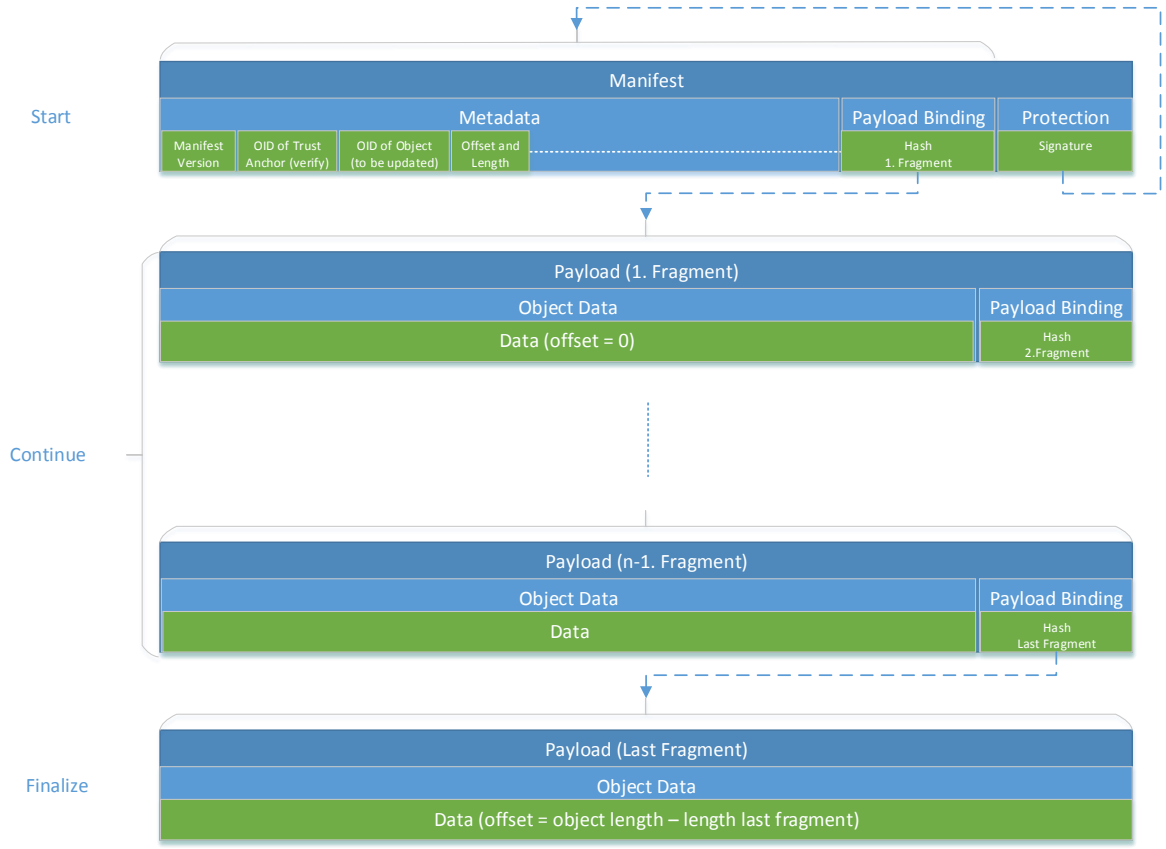

Figure 31 - Protected Update Data Set structure

6 Appendix

6.1 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 32 - GetDataObject [Read data] example

Note: The values specified in Figure 32 *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 33 - SetDataObject [Write data] example

Note: The values specified in Figure 33 *are in HEX format.*

6.2 Limitations

6.2.1 Memory Constraints

- Maximum command InData length is 1553. Any attempt to exceed these limit will lead to "Invalid length field" error.
- Maximum response OutData length is 1553. Any attempt to exceed these limit will lead to "Insufficient buffer/ memory" error.

6.3 Certificate Parser Details

6.3.1 Parameter Validation

Following are the basic parameter checks performed while parsing the certificate,

- X.509 DER coding and length checks
- Must be v3 only
- Maximum length of the serial number is 20 bytes
- Public Key
  - Only uncompressed Format is supported
  - Algorithms supported
    - ECC NIST P 256/384
    - RSA 1024/2048 Exponential
- The maximum length of Key Identifier considered is 32 bytes
- Issuer DN must not be Empty.
- Basic Constraints
  - CA - Flag to indicate CA certificate (TRUE for CA)
  - Path Length Constraint
    - Exists only for CA certificate

[RFC5280] specifies a number of fields (specified as MUST/SHOULD etc.). All of these will not be validated for correct formation and correctness of data. The fields verified for correctness are explicitly mentioned above.

6.4 Security Guidance

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

6.4.1 Use Case: Mutual Authentication -toolbox-

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

6.4.2 Use Case: Host FW Update -toolbox-

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

Appendix

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.

[RFC4108] Security considerations (chapter 6) shall be reviewed and taken as a guideline with regards to:

- Private signature key protection
- Firmware decryption key protection
- Cryptographic algorithms become weaker with time
- Randomly generated keys
- Stale firmware version number

6.4.3 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.

6.4.4 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.

6.4.5 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.
- Restrict the maximum usage of the secret using the monotonic counters to 2048 times is recommended

Appendix

- EXE access condition tied with Platform Binding shared secret (CONF E140), enforces the shielded connection for the usage of pre-shared secret in OPTIGA during the session key derivation which doesn't allow the usage with unknown Hosts.

6.4.6 Shielded Connection

- The security level of the Shielded connection is as high as typical microcontroller/Host side hardware security level.
- The recommended length of Platform binding shared secret is minimum 32 bytes.
- Updating the Platform Binding shared secret during the run time using the shielded connection is recommended.
  - enable the monotonic counter and reduce the max number of usages to XYZ using monotonic counter and update the secret on chip periodically.
  - To enforce this, the write access conditions for the Platform binding data object must be set accordingly.
- Secure binding (using the Platform binding shared secret) and usage restriction using the Monotonic counters enables additional usage restriction for the critical asset (e.g. RSA keys and shared secrets) if assets are not intended to use extremely.
- It is recommended to use the shielded connection for EncryptAsym (which uses a session context) command based operations which enforces the usage of session context.

6.4.7 Algorithm usage

- The recommendation is to use RSA 2048 against RSA 1024.

6.5 Shielded Connection V1 Guidance

The OPTIGA™ Shielded Connection enables a protected (Integrity and Confidentiality) communication between the OPTIGA™ and a corresponding Host platform as depicted in figure below.

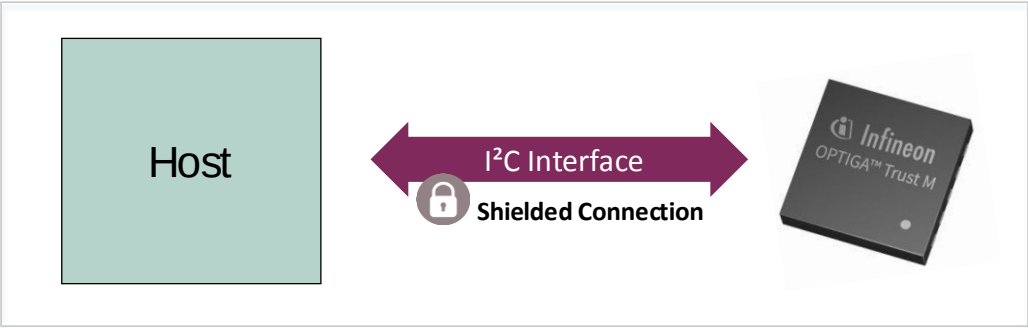

Figure 34 - Overview OPTIGA™ Shielded Connection

This section provides information regarding the set up and usage of Shielded Connection in the target device application.
The OPTIGA™ supports the Shielded Connection using a pre-shared secret (Platform Binding Secret) between the OPTIGA™ and a corresponding host platform. [IFX_I2C] explains internal details of establishing the shielded connection; e.g., negotiation and crypto algorithms used for the protection in the section Presentation Layer.

Appendix

6.5.1 Setup

Preconditions to establish the Shielded Connection is to pair the OPTIGA™ with a host. The pre-shared secret is established during first boot/initialization sequence. The Use Case: Pair OPTIGA™ with Host (Pre-Shared Secret based) depicts the pairing process.
The pal_os_datastore is an abstraction layer for writing and reading a platform binding secret to/from the host platform. This has to be adapted to the particular host platform needs.
During the Shielded Connection establishment, the optiga_comms_ifx_i2c module invokes pal_os_datastore_read function.

6.5.2 Usage

In the OPTIGA™ Host Library, the Shielded Connection feature can be enabled/disabled using the macro (OPTIGA_COMMS_SHIELDED_CONNECTION in optiga_lib_config.h) with the required default protection level (OPTIGA_COMMS_DEFAULT_PROTECTION_LEVEL in optiga_lib_config.h).
For the protected communication (Shielded Connection) between a host and the OPTIGA™, an instance of optiga_util or optiga_crypt needs to be updated with the required protection level before invoking the operations provided by optiga_util or optiga_crypt using OPTIGA_UTIL_SET_COMMS_PROTECTION_LEVEL and OPTIGA_CRYPT_SET_COMMS_PROTECTION_LEVEL respectively.
For example, to enable a full: i.e. command and response, protection for deriving the decryption keys in FW update use case using the pre-shared secret from OPTIGA™
Invoke OPTIGA_CRYPT_SET_COMMS_PROTECTION_LEVEL(me_crypt, OPTIGA_COMMS_FULL_PROTECTION)

- Invoke optiga_crypt_tls_prf_sha256 (me_crypt, shard_secret_oid, …)

In case of re-establishing the Shielded Connection periodically, the protection_level |OPTIGA_COMMS_RE_ESTABLISH can be set to the instance using above specified. The access layer will take care of rescheduling the shielded connection internally.
For example, to enable re-establishing the shielded connection with response protection for the data object read data operation

- Invoke OPTIGA_UTIL_SET_COMMS_PROTECTION_LEVEL(me_util, OPTIGA_COMMS_RESPONSE_PROTECTION|OPTIGA_COMMS_RE_ESTABLISH)
- Invoke optiga_util_read_data (me_util, oid, …)

Based on the above settings, the access layer activates the Shielded Connection between a host and the OPTIGA™. These settings reset automatically to the default protection level; i.e. OPTIGA_COMMS_DEFAULT_PROTECTION_LEVEL once after the operation is invoked.
The Use Case: Update Platform Binding Secret during runtime (Pre-Shared Secret based) depicts a process of updating the platform binding secret periodically using the shielded connection at runtime.

6.6 Protected Update: Manifest and Signature structures

This section provides the [CDDL] coding considered for manifest and signature structures for the protected update feature.

6.6.1 Manifest structures

This paragraph provides the structure considered for the manifest.

```
; OPTIGA(TM) Trust (Family Trust X, Trust M...) Manifest (v1)
;
; This file contains a CDDL (CBOR Data Definition Language) definition for a Manifest used within
OPTIGA(TM) Trust Protected update feature.
; It describes the serialization format of a manifest as CBOR.
;
```

Appendix

```
; The manifest contains metadata about an OPTIGA(TM) Trust data/key object image (Payload Resource)
and to which target object it applies.
; The manifest itself is protected via a COSE Signature Container.
; Optionally the data object image is encrypted via a COSE Encryption Container.
;
; The Processing Steps how to apply the Payload Resource to the Target are described as separate
Processing Array.
;
; Naming Conventions:
; (1) Existing CDDL Definitions profiled for OPTIGA(TM) Trust are enhanced
;     at the END with _Trust
;     e.g. COSE_Sign1_Trust
; (2) Definitions specific for OPTIGA(TM) Trust are defined with Trust_ in the
;     beginning, the same ways as done in COSE (where COSE_ is at the beginning)
;     e.g. Trust_PayloadVersion
; (3) Defintion used by OPTIGA(TM) Trust and OPTIGA(TM) TPM are defined with
;     IFX_ in the beginning
;     e.g. IFX_DigestSize
;
; Links:
; [CBOR]: https://tools.ietf.org/html/rfc7049
; [CDDL]: https://tools.ietf.org/html/draft-ietf-cbor-cddl-05
; [COSE]: https://tools.ietf.org/html/rfc8152
; [SUIT_DRAFTv2]: https://tools.ietf.org/html/draft-moran-suit-manifest-02
; [SUIT_DRAFTv3]: https://tools.ietf.org/html/draft-moran-suit-manifest-03
; [COSE_RSA]: https://tools.ietf.org/html/rfc8230
;
; The range of proprietary values to be used [-65700.....-65899].
; RSA-SSA-PKCS1-V1_5-SHA-256 = -65700
; IFX_KDF-TLS12              = -65720
;
; Untagged COSE Sign and know from Context, e.g. Trust implicitly knows
; input to update command MUST be a COSE-Sign1 Message Type
; signed_Trust = #6.18(COSE_Sign1_Trust); see Table 1 in [COSE]

; Define Signing Container with One Signer (COSE_Sign1)
COSE_Sign1_Trust = [
    protected: bstr .cbor protected-signed-header-Trust,
    unprotected: unprotected-signed-header-Trust,
    payload: bstr .cbor Trust_manifest,
    signature: bstr .size Trust_SignatureSize,
]

Trust_SignatureSize = &(
    ECC_256: 64,                            ;Raw signature: ECDSA (r|s) acc. to FIPS 186-4
                                            ;E.g. for ECC NIST P 256, size is 64 Bytes
                                            ;(the 0's must be prepended if r/s component is less
than key size)
    ECC_384: 96,                            ;Raw signature: ECDSA (r|s) acc. to FIPS 186-4
                                            ;E.g. for ECC NIST P 384, size is 96 Bytes
                                            ;(the 0's must be prepended if r/s component is less
than key size)
    RSA_1024_EXP: 128,                      ;RSA Raw Signature for the key length 1024 bits
    RSA_2048_EXP: 256,                      ;RSA Raw Signature for the key length 2048 bits
)

Trust_SignAlgorithms = &(
    ES-256: -7,                             ;[COSE]; ECDSA acc. to FIPS 186-4 and not deterministic
version as suggested in Section 8.1 in COSE
    RSA-SSA-PKCS1-V1_5-SHA-256: -65700,     ;[TRUST_PROP]; RSASSA PKCS#1 V1.5 w/ SHA-256
)

IFX_DigestSize = &(
    SHA2-256-digest-size: 32,
)

DigestAlgorithms = &(
    SHA-256: 41,                            ;SHA2
) /uint .size 1

Trust_PayloadType  = &(                     ;[TRUST_PROP]
    Payload_Data: -1,                       ;[TRUST_PROP]; To send the data to OPTIGA(TM) Trust Data
object
)

Trust_AddInfo_WriteType = &(                ;[TRUST_PROP]
```

Appendix

```
    Write: 1,                           ;Updates/writes the data in the target oid, starting
from the offset specified
                                        ;(Used Length gets updated to (Offset + Length of the
data provided),
                                        ;if this value is greater than the current value in the
object metadata)

    EraseAndWrite: 2,                   ;Erases the complete data in the target oid, writes the
data from the offset specified
                                        ;(Used Length of the data object = Offset + Length of
the data provided)
) /uint .size 1

protected-signed-header-Trust = {
    1 => Trust_SignAlgorithms,          ;[COSE] and [TRUST_PROP]
}

unprotected-signed-header-Trust = {
    4 => bstr .size 2,                  ;[COSE]; key identifier:Root of Trust, Trust Anchor OID
}

Trust_manifest = [                      ;[TRUST_PROP]; OPTIGA(TM) Trust Manifest
    manifestVersion: 1,                 ;[SUIT_DRAFTv2]; Manifest data model version
    ;digestInfo: DigestInfo,            ;[SUIT_DRAFTv2]; Digest algorithm used within manifest
    ;textReference: bstr,               ;[SUIT_DRAFTv2]; Digest of the associated text map
    ;nonce: bstr .size 8,               ;[SUIT_DRAFTv2]; Nonce 8 byte (IFX_PROP)
    ;sequence: uint,                    ;[SUIT_DRAFTv2]; SequenceNumber;
                                        ;current UTC time as unix epoch (Unix timestamp in
seconds);
                                        ;Store as 8 byte value internally (IFX_PROP)
    preConditions: nil,                 ;[SUIT_DRAFTv2]; Array with preconditions, not used
    postConditions: nil,                ;[SUIT_DRAFTv2]; Array with postconditions, not used
    ;directives: [],                    ;[SUIT_DRAFTv2]; Array with directives, not used
    resources: Trust_resource,          ;[TRUST_PROP]; No Array as [SUIT_DRAFTv2], 1 Element for
Data for Payload
    processors: Trust_processors,       ;[SUIT_DRAFTv2]; Array with 2 optional processing steps
    targets: Trust_target,              ;[TRUST_PROP]; No Array as [SUIT_DRAFTv2] 1 Element for
Target to update
    ;extensions : {},                   ;[SUIT_DRAFTv2]; Map with extensions, not used via empty
list
]

IFX_DigestInfo = [
    digestAlgorithm: DigestAlgorithms,  ;[SUIT_DRAFTv3]; Digest algorithms
    digest: bstr .size IFX_DigestSize   ;[IFX_PROP]; size depending the IFX_DigestAlgorithm
]

Trust_resource  = [                     ;[TRUST_PROP]
    type: Trust_PayloadType,            ;[SUIT_DRAFTv2]; Custom Types are indicated via negative
Value
    ;indicator: {},                     ;[SUIT_DRAFTv2]; = UriList: where to find the resource;
    size: uint .size 2,                 ;[SUIT_DRAFTv2]; Size of the resource; Length of payload
to be updated in the target object
    ;digest: DigestInfo,                ;[SUIT_DRAFTv2]; Digest of complete payload. Not
considered/supported

    ;IFX Extensions
    Trust_PayloadVersion: uint .size 2, ;[TRUST_PROP] Payload Version - up to (2^15 - 1 = 32767)

    ;Additional Info has a different set of information, based on Trust_PayloadType chosen.
    AdditionalInfo: Trust_AddInfo_Data, ;(TRUST_PROP); for Trust_PayloadType = Payload_Data
]

Trust_AddInfo_Data = [                      ;[TRUST_PROP]; Additional information for the
payload type = Payload_Data
    offset: uint .size 2,                   ;[TRUST_PROP]; Offset from which the data to be
updated
    write_type: Trust_AddInfo_WriteType     ;[TRUST_PROP]; Specifies the type of update/write
in the target
]

Trust_target = [                            ;[TRUST_PROP] based on  [SUIT_DRAFTv2]
    componentIdentifier: bstr .size 0,      ;[SUIT_DRAFTv2]; 0 Bytes for the Broadcast
operations
                                            ;Reserved for future use to enable uni-cast
operations.
```

Appendix

```
    storageIdentifier: bstr .size 2,              ;[SUIT_DRAFTv2]; Update Target of the Payload;
OPTIGA(TM) Object ID (IFX_PROP)
    ;encoding: bstr .size 1,                      ;[SUIT_DRAFTv2]; the format of the resource;
]

Trust_processors = [                              ;[SUIT_DRAFTv2]; = [ * ProcessingStep ]: Array of
processing steps
    ProcessingStep1: ProcessingStep_integrity,    ;[SUIT_DRAFTv2]; = first processing step is to
check the Fragment Integrity
    ProcessingStep2: nil,                         ;[SUIT_DRAFTv2]; = second processing step is not
defined. This is reserved for future use for confidentiality purpose.
]

ProcessingStep_integrity = [                      ;[SUIT_DRAFTv2]; = Custom ProcessingStep
    process: -1,                                  ;[TRUST_PROP]; Check Integrity of first fragment
    parameters: bstr .cbor IFX_DigestInfo,        ;[TRUST_PROP]; Digest of first fragment
]
```

6.6.2 Signature structures

This paragraph provides the structure considered for the signature.

```
; OPTIGA(TM) Trust (Family Trust X, Trust M...) Signature structure
;
; This file contains a CDDL (CBOR Data Definition Language) definition for a Manifest used within
OPTIGA(TM) Trust Protected update feature.
; It describes the serialization format of a manifest as CBOR.
;
;
; Naming Conventions:
; (1) Existing CDDL Definitions profiled for OPTIGA(TM) Trust are enhanced
;     at the END with _Trust
;     e.g. COSE_Sign1_Trust
; (2) Definitions specific for OPTIGA(TM) Trust are defined with Trust_ in the
;     beginning, the same ways as done in COSE (where COSE_ is at the beginning)
;     e.g. Trust_PayloadVersion
; (3) Defintion used by OPTIGA(TM) Trust and OPTIGA(TM) TPM are defined with
;     IFX_ in the beginning
;     e.g. IFX_DigestSize
;
; Links:
; [CBOR]: https://tools.ietf.org/html/rfc7049
; [CDDL]: https://tools.ietf.org/html/draft-ietf-cbor-cddl-05
; [COSE]: https://tools.ietf.org/html/rfc8152
; [SUIT_DRAFTv2]: https://tools.ietf.org/html/draft-moran-suit-manifest-02
; [SUIT_DRAFTv3]: https://tools.ietf.org/html/draft-moran-suit-manifest-03
; [COSE_RSA]: https://tools.ietf.org/html/rfc8230
;
; The range of proprietary values to be used [-65700.....-65899].
; RSA-SSA-PKCS1-V1_5-SHA-256 = -65700
; IFX_KDF-TLS12              = -65720
;
;
;
; [COSE] 4.4. Signing and Verification Process
; Define Container for the signature structure
Sig_structure = [
    context : "Signature1",
    body_protected : bstr .cbor protected-signed-header-Trust,
    external_aad : bstr .size 0,
    payload: bstr .cbor Trust_manifest,
]
Trust_SignatureSize = &(
    ECC_256: 64,                          ;Raw signature: ECDSA (r|s) acc. to FIPS 186-4
                                          ;E.g. for ECC NIST P 256, size is 64 Bytes
                                          ;(the 0's must be prepended if r/s component is less
than key size)
    ECC_384: 96,                          ;Raw signature: ECDSA (r|s) acc. to FIPS 186-4
                                          ;E.g. for ECC NIST P 384, size is 96 Bytes
                                          ;(the 0's must be prepended if r/s component is less
than key size)
    RSA_1024_EXP: 128,                    ;RSA Raw Signature for the key length 1024 bits
    RSA_2048_EXP: 256,                    ;RSA Raw Signature for the key length 2048 bits
```

Appendix

```
)
Trust_SignAlgorithms = &(
    ES-256: -7,                             ;[COSE]; ECDSA acc. to FIPS 186-4 and not detereministic
version as suggested in Section 8.1 in COSE
    RSA-SSA-PKCS1-V1_5-SHA-256: -65700,     ;[TRUST_PROP]; RSASSA PKCS#1 V1.5 w/ SHA-256
)
IFX_DigestSize = &(
    SHA2-256-digest-size: 32,
)
DigestAlgorithms = &(
    SHA-256: 41,                            ;SHA2
) /uint .size 1
Trust_PayloadType  = &(                     ;[TRUST_PROP]
    Payload_Data: -1,                       ;[TRUST_PROP]; To send the data to OPTIGA(TM) Trust Data
object
)
Trust_AddInfo_WriteType = &(                ;[TRUST_PROP]
    Write: 1,                               ;Updates/writes the data in the target oid, starting
from the offset specified
                                            ;(Used Length gets updated to (Offset + Length of the
data provided),
                                            ;if this value is greater than the current value in the
object metadata)
    EraseAndWrite: 2,                       ;Erases the complete data in the target oid, writes the
data from the offset specified
                                            ;(Used Length of the data object = Offset + Length of
the data provided)
) /uint .size 1
protected-signed-header-Trust = {
    1 => Trust_SignAlgorithms,              ;[COSE] and [TRUST_PROP]
}
unprotected-signed-header-Trust = {
    4 => bstr .size 2,                      ;[COSE]; key identifier:Root of Trust, Trust Anchor OID
}
Trust_manifest = [                          ;[TRUST_PROP]; OPTIGA(TM) Trust Manifest
    manifestVersion: 1,                     ;[SUIT_DRAFTv2]; Manifest data model version
    ;digestInfo: DigestInfo,                ;[SUIT_DRAFTv2]; Digest algorithm used within manifest
    ;textReference: bstr,                   ;[SUIT_DRAFTv2]; Digest of the associated text map
    ;nonce: bstr .size 8,                   ;[SUIT_DRAFTv2]; Nonce 8 byte (IFX_PROP)
    ;sequence: uint,                        ;[SUIT_DRAFTv2]; SequenceNumber;
                                            ;current UTC time as unix epoch (Unix timestamp in
seconds);
                                            ;Store as 8 byte value internally (IFX_PROP)
    preConditions: nil,                     ;[SUIT_DRAFTv2]; Array with preconditions, not used
    postConditions: nil,                    ;[SUIT_DRAFTv2]; Array with postconditions, not used
    ;directives: [],                        ;[SUIT_DRAFTv2]; Array with directives, not used
    resources: Trust_resource,              ;[TRUST_PROP]; No Array as [SUIT_DRAFTv2], 1 Element for
Data for Payload
    processors: Trust_processors,           ;[SUIT_DRAFTv2]; Array with 2 optional processing steps
    targets: Trust_target,                  ;[TRUST_PROP]; No Array as [SUIT_DRAFTv2] 1 Element for
Target to update
    ;extensions : {},                       ;[SUIT_DRAFTv2]; Map with extensions, not used via empty
list
]
IFX_DigestInfo = [
    digestAlgorithm: DigestAlgorithms,      ;[SUIT_DRAFTv3]; Digest algorithms
    digest: bstr .size IFX_DigestSize       ;[IFX_PROP]; size depending the IFX_DigestAlgorithm
]
Trust_resource  = [                         ;[TRUST_PROP]
    type: Trust_PayloadType,                ;[SUIT_DRAFTv2]; Custom Types are indicated via negative
Value
    ;indicator: {},                         ;[SUIT_DRAFTv2]; = UriList: where to find the resource;
    size: uint .size 2,                     ;[SUIT_DRAFTv2]; Size of the resource; Length of payload
to be updated in the target object
    ;digest: DigestInfo,                    ;[SUIT_DRAFTv2]; Digest of complete payload. Not
considered/supported
    ;IFX Extensions
    Trust_PayloadVersion: uint .size 2,     ;[TRUST_PROP] Payload Version - up to (2^15 - 1 = 32767)
    ;Additional Info has a different set of information, based on Trust_PayloadType chosen.
    AdditionalInfo: Trust_AddInfo_Data,     ;(TRUST_PROP); for Trust_PayloadType = Payload_Data
]
Trust_AddInfo_Data = [                            ;[TRUST_PROP]; Additional information for the
payload type = Payload_Data
    offset: uint .size 2,                         ;[TRUST_PROP]; Offset from which the data to be
updated
    write_type: Trust_AddInfo_WriteType           ;[TRUST_PROP]; Specifies the type of update/write
in the target
```

Appendix

```
]
Trust_target = [                                    ;[TRUST_PROP] based on  [SUIT_DRAFTv2]
    componentIdentifier: bstr .size 0,              ;[SUIT_DRAFTv2]; 0 Bytes for the Broadcast
operations
                                                    ;Reserved for future use to enable uni-cast
operations.
    storageIdentifier: bstr .size 2,                ;[SUIT_DRAFTv2]; Update Target of the Payload;
OPTIGA(TM) Object ID (IFX_PROP)
    ;encoding: bstr .size 1,                        ;[SUIT_DRAFTv2]; the format of the resource;
]
Trust_processors = [                                ;[SUIT_DRAFTv2]; = [ * ProcessingStep ]: Array of
processing steps
    ProcessingStep1: ProcessingStep_integrity,      ;[SUIT_DRAFTv2]; = first processing step is to
check the Fragment Integrity
    ProcessingStep2: nil,                           ;[SUIT_DRAFTv2]; = second processing step is not
defined. This is reserved for future use for confidentiality purpose.
]
ProcessingStep_integrity = [                        ;[SUIT_DRAFTv2]; = Custom ProcessingStep
    process: -1,                                    ;[TRUST_PROP]; Check Integrity of first fragment
    parameters: bstr .cbor IFX_DigestInfo,          ;[TRUST_PROP]; Digest of first fragment
]
```

6.6.3 CDDL Tool

The details of CDDL tool are provided as part of [CDDL] Appendix.

6.7 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 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. [RFC6347]	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	GAD

Appendix

Term	Description	Abbreviation
	device has a certain flexibility to adapt the device for his particular purpose.	
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
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
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 41 - Terms of OPTIGA™ Vocabulary

6.8 Change History

Version	Date	Description
1.00	04-Feb-2019	Initial release version

Trademarks of Infineon Technologies AG
µHVIC™, µIPM™, µPFC™, AU-ConvertIR™, AURIX™, C166™, CanPAK™, CIPOS™, CIPURSE™, CoolDP™, CoolGaN™, COOLiR™, CoolMOS™, CoolSET™, CoolSiC™, DAVE™, DI-POL™, DirectFET™, DrBlade™, EasyPIM™, EconoBRIDGE™, EconoDUAL™, EconoPACK™, EconoPIM™, EiceDRIVER™, eupec™, FCOS™, GaNpowIR™, HEXFET™, HITFET™, HybridPACK™, iMOTION™, IRAM™, ISOFACE™, IsoPACK™, LEDrivIR™, LITIX™, MIPAQ™, ModSTACK™, my-d™, NovalithIC™, OPTIGA™, OptiMOS™, ORIGA™, PowIRaudio™, PowIRStage™, PrimePACK™, PrimeSTACK™, PROFET™, PRO-SIL™, RASIC™, REAL3™, SmartLEWIS™, SOLID FLASH™, SPOC™, StrongIRFET™, SupIRBuck™, TEMPFET™, TRENCHSTOP™, TriCore™, UHVIC™, XHP™, XMC™

Trademarks updated November 2015

Other Trademarks
All referenced product or service names and trademarks are the property of their respective owners.

Edition February 4, 2019
Published by

Infineon Technologies AG

81726 Munich, Germany

© 2019 Infineon Technologies AG. All Rights Reserved.

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.