OPTIGA™ Trust M

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 version 1 and version 3 are given generically without indicating the dedicated version

Table of Contents

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
CERT	Certificate
CRL	Certificate Revocation List
DDK	Device Driver Kit
DO	Data Object
DoS	Denial of Service
DRNG	Deterministic Random Number Generator
DTLS	Datagram Transport Layer Security
EAL	Evaluation Assurance Level
ESW	Embedded Software
LC / LCM	Life Cycle / Life Cycle Management
NVM	Non-Volatile Memory
NW	Network
OID	Object Identifier
PKI	Public Key Infrastructure
PP	Protection Profile
RAM	Random-Access Memory
SecMC	Secure Microcontroller
SW	Software
TBD	To Be Defined
TBS	To Be Specified
TLS	Transport Layer Security

Abbreviation	Term
TRNG	True Random Number Generator
UID	Unique Identifier
µC / MCU	Microcontroller

Table 1 - Abbreviations

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)
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)

Table 2 – Naming Conventions

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
[DAVE]	https://infineoncommunity.com/dave-download_ID645
[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
[ISO 9797-1]	ISO/IEC 9797-1:2011 Information technology - Security techniques - Message Authentication Codes (MACs) Part 1 https://www.iso.org/standard/50375.html Information technology—Security techniques—Message Authentication Codes (MACs) -- Part 1: Mechanisms using a block cipher
[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
[RFC2104]	https://tools.ietf.org/pdf/rfc2104.pdf HMAC: Keyed-Hashing for Message Authentication
[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
[RFC3279]	https://tools.ietf.org/html/rfc3279 Algorithms and Identifiers for the Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile.
[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 Revocation List (CRL) Profile
[RFC5288]	https://tools.ietf.org/html/rfc5288 AES Galois Counter Mode (GCM) Cipher Suites for TLS
[RFC5869]	https://tools.ietf.org/html/rfc5869 HMAC-based Extract-and-Expand Key Derivation Function (HKDF)
[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

[RFC8032]	https://tools.ietf.org/html/rfc8032 Edwards-Curve Digital Signature Algorithm (EdDSA)
[SP 800-38A]	https://nvlpubs.nist.gov/nistpubs/Legacy/SP/nistspecialpublication800-38a.pdf
[SP 800-38B]	https://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.800-38b.pdf Recommendation for Block Cipher Modes of Operation: The CMAC Mode for Authentication
[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 800-38D]	https://nvlpubs.nist.gov/nistpubs/Legacy/SP/nistspecialpublication800-38d.pdf Recommendation for Block Cipher Modes of Operation: Galois/Counter Mode (GCM) and GMAC.
[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
[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
[WPC Auth]	open Wireless Power Charging Authentication Specification
[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 which in particular supporting IoT use cases and along with that it provides a number of key and data objects which hold user/customer related keys and data. The subsequent document is structured in the chapters Supported Use Cases, Enabler APIs, OPTIGA™

Introduction

Trust M External Interface, OPTIGA™ Trust M Data Structures and Appendix.

- Supported Use Cases provides the main use cases in a form of sequence diagrams which explains how the host side Enabler APIs are used in the respective use cases.
- Enabler APIs provides the necessary details of the host side 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 to utilize the OPTIGA™ functionality.
- OPTIGA™ Trust M Data Structures provides details of the key and data objects 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, 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 view (OPTIGA Trust Communication Protection - toolbox - View) 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, third_party_crypto libraries and optionally 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 lock acquired 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 kilobytes. 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].

Supported Use Cases

Name	Description
	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 toolbox. • Uses 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 OPTIGA™ (open/close) and data/key objects with the following characteristics: • Multiple instances could be created to allow concurrent access to other services. • Uses 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: • Supporting Firmware 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, which turns in communication with the OPTIGA™.
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 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. • Supporting cloud service specific adaptation

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.

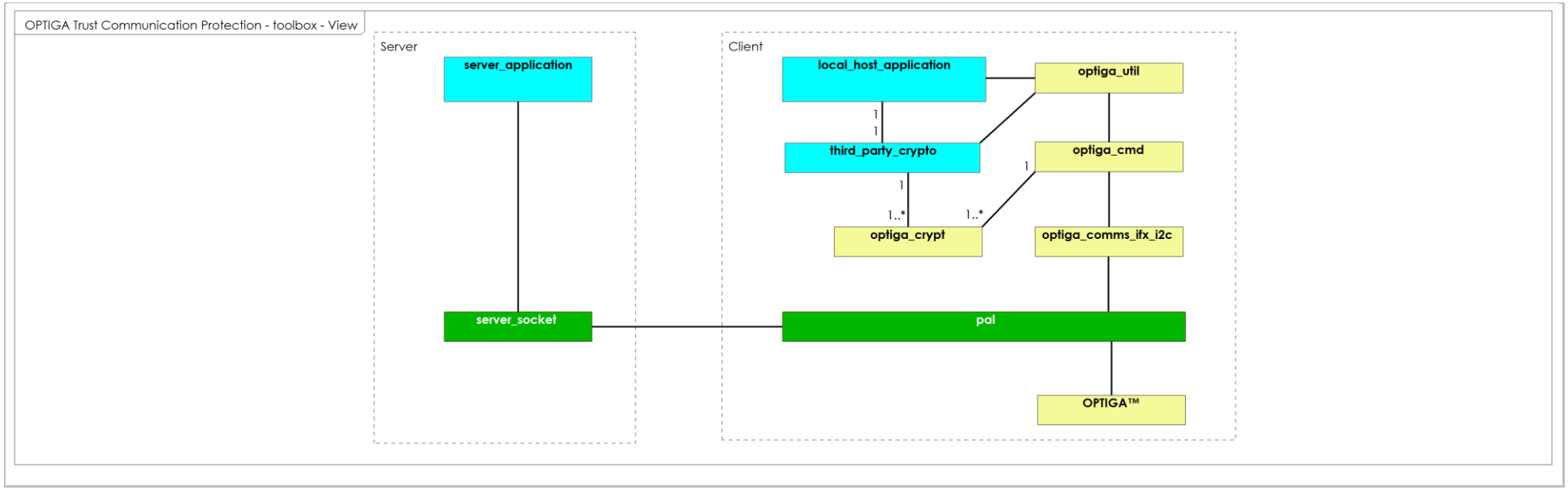


Figure 1 - OPTIGA Trust Communication Protection - toolbox - View

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 v5.25 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	OPTIGA™ Trust M V1		OPTIGA™ Trust M V3	
	RAM	CODE	RAM	CODE
[without the Shielded Connection] The components optiga_crypt, optiga_util, optiga_cmd, optiga_comms_ifx_i2c, and pal are covered.	5 KBytes	18 KBytes	18 KBytes	18 KBytes
[with Shielded Connection] The components optiga_crypt, optiga_util, optiga_cmd, optiga_comms_ifx_i2c, pal, and platform_crypto are covered. Here mbed TLS v2.16.0 is used as a reference for platform_crypto to perform the shielded connection cryptographic operations (e.g. key derivation, encryption and decryption).	15 KBytes	31 KBytes	31 KBytes	31 KBytes

Table 3 - Host code size

In addition, the *optiga_lib_config.h* file (in the host side library) can be updated to enable or disable the features based on the target usage to reduce the code consumption if compiler is not optimizing automatically.

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.

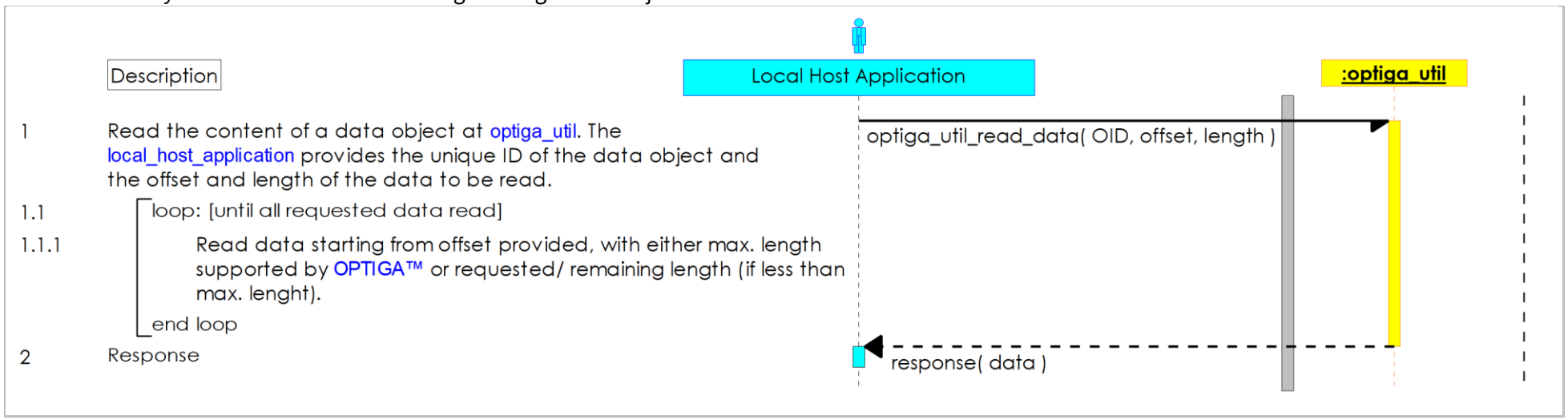


Figure 2 - Use Case: Read General Purpose Data - data object

2.2.2 Use Case: Read General Purpose Data - metadata

The local_host_application intends to read the metadata of a data/key 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/key 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

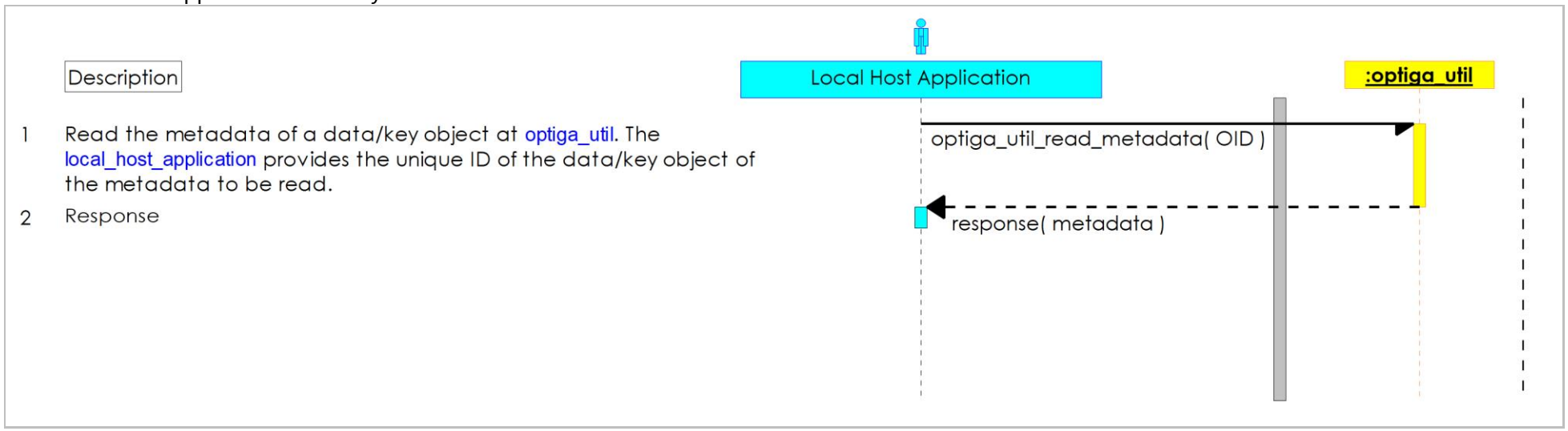


Figure 3 - Use Case: Read General Purpose Data - metadata

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

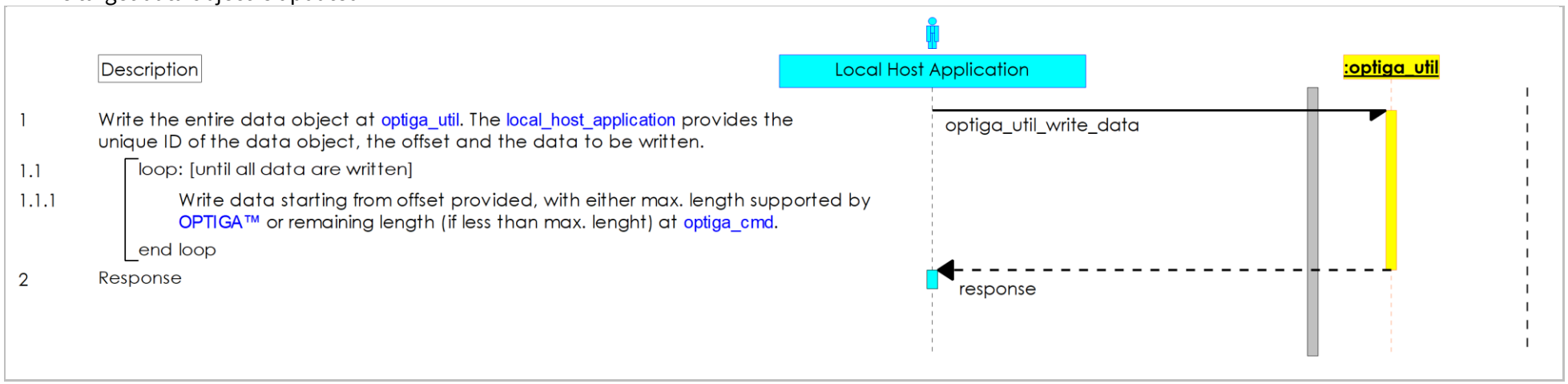


Figure 4 - Use Case: Write General Purpose Data - data object

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 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/key object are satisfied.

Post-condition:

- The metadata associated to the target data/key object is updated

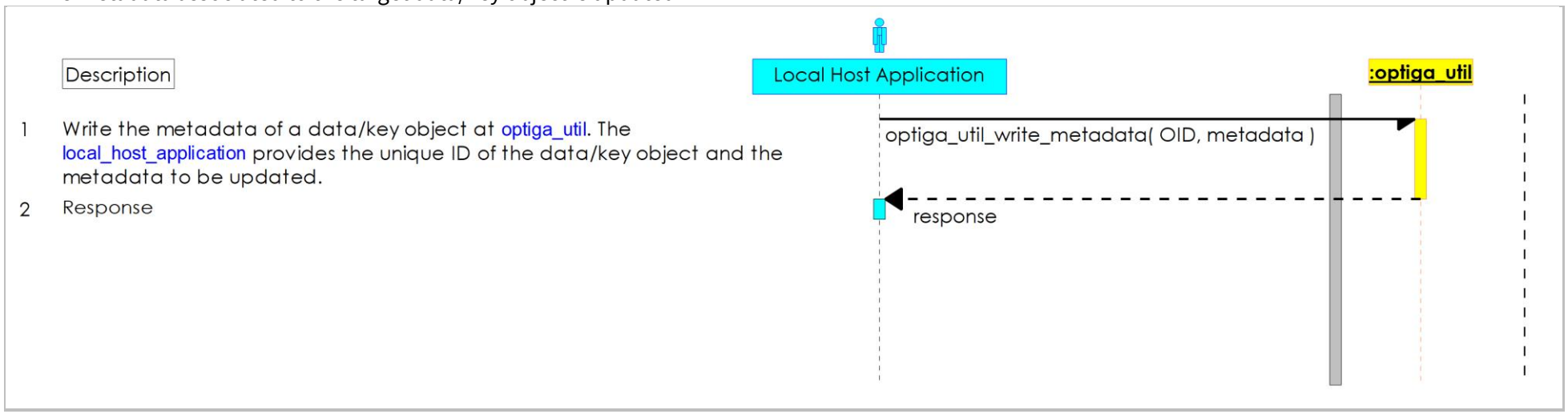


Figure 5 - Use Case: Write General Purpose Data - metadata

2.2.5 Use Case: Integrity Protected Update of a 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 is forwarded to the OPTIGA™. The OPTIGA™ checks and removes the protection and upon success updates the target key or 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 update.

Post-condition:

- The target data object is updated.

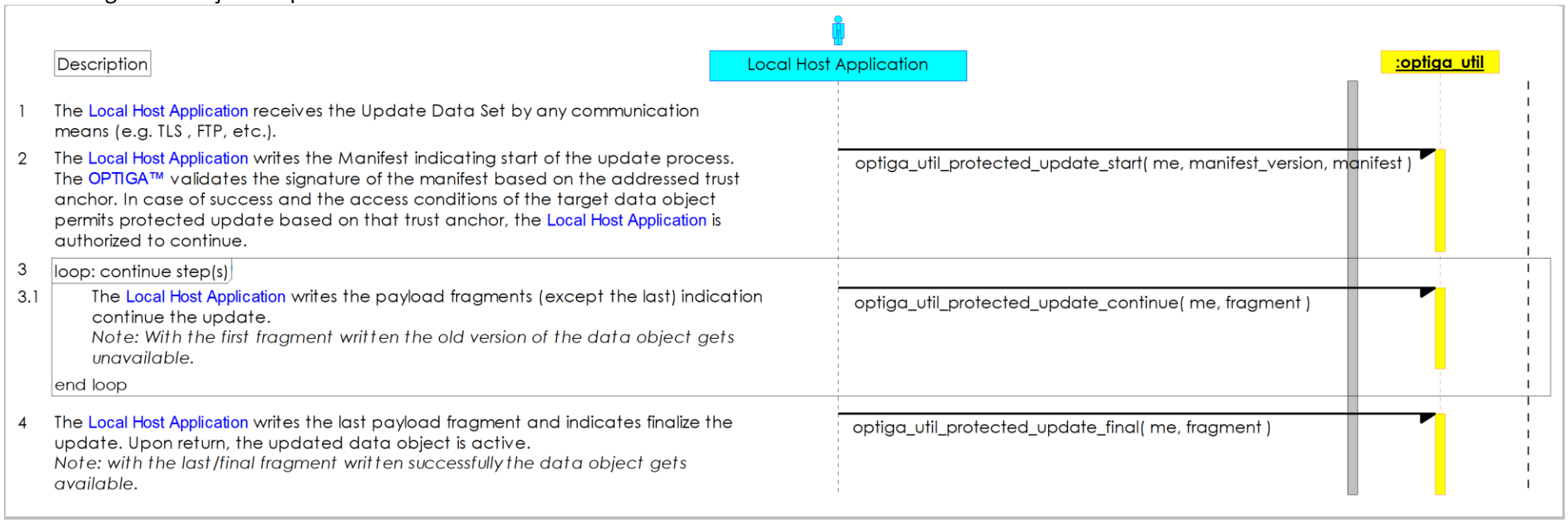


Figure 6 - Use Case: Integrity Protected Update of a data object

2.2.6 Use Case: Confidentiality Protected Update of key or a data object

The Management Server intends to update a key or a data object (e.g. Pre-shared Secret) with integrity and confidentiality protected. The Management Server provides an update data set, which is forwarded to the OPTIGA™. The OPTIGA™ checks and removes the protection and upon success updates the target data/key object.

Note: OPTIGA™ Trust M Version 1 doesn't support confidentiality and update of keys & metadata as part of protected update.

Pre-condition(s):

- The OPTIGA™ application is already launched
- The Trust Anchor for management purpose is well formatted and available at the OPTIGA™.
- The protected update secret for management purpose (to enable confidentiality) is available at OPTIGA™.
- The access conditions of the target data/key object allow protected updating.

Post-condition:

- The target data object is updated.

Supported Use Cases

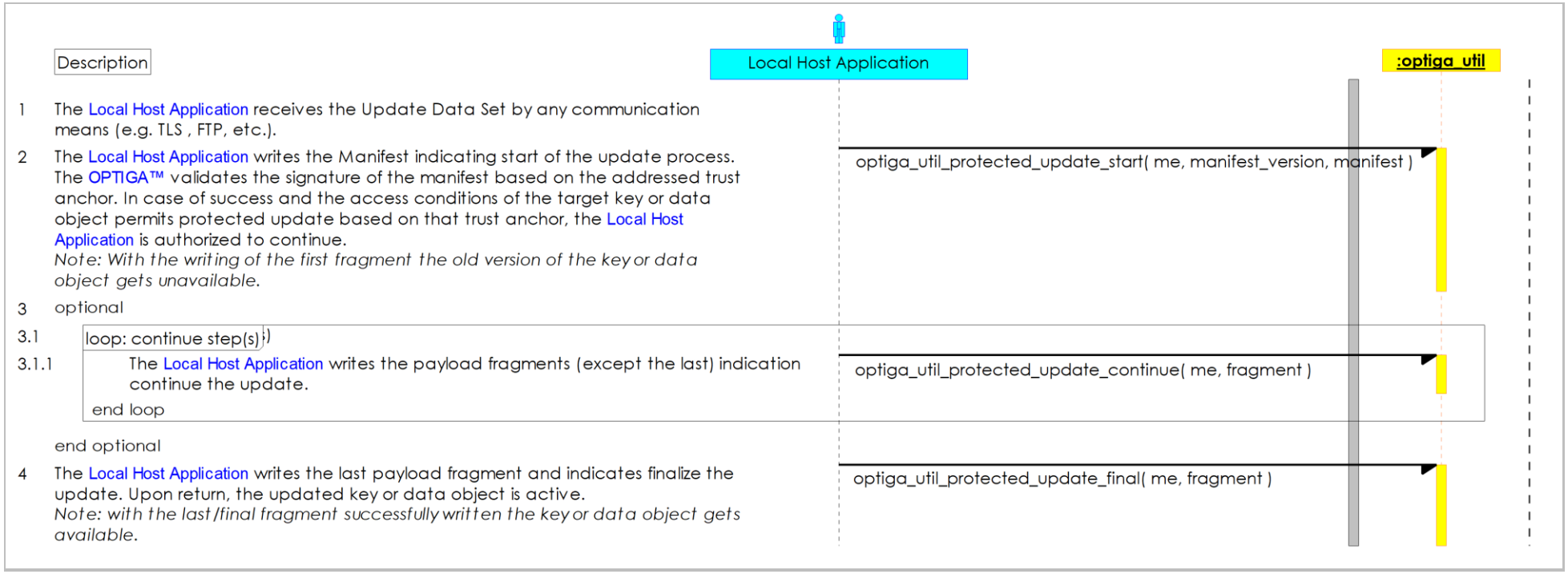


Figure 7 - Use Case: Confidentiality Protected Update of key or a data object

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

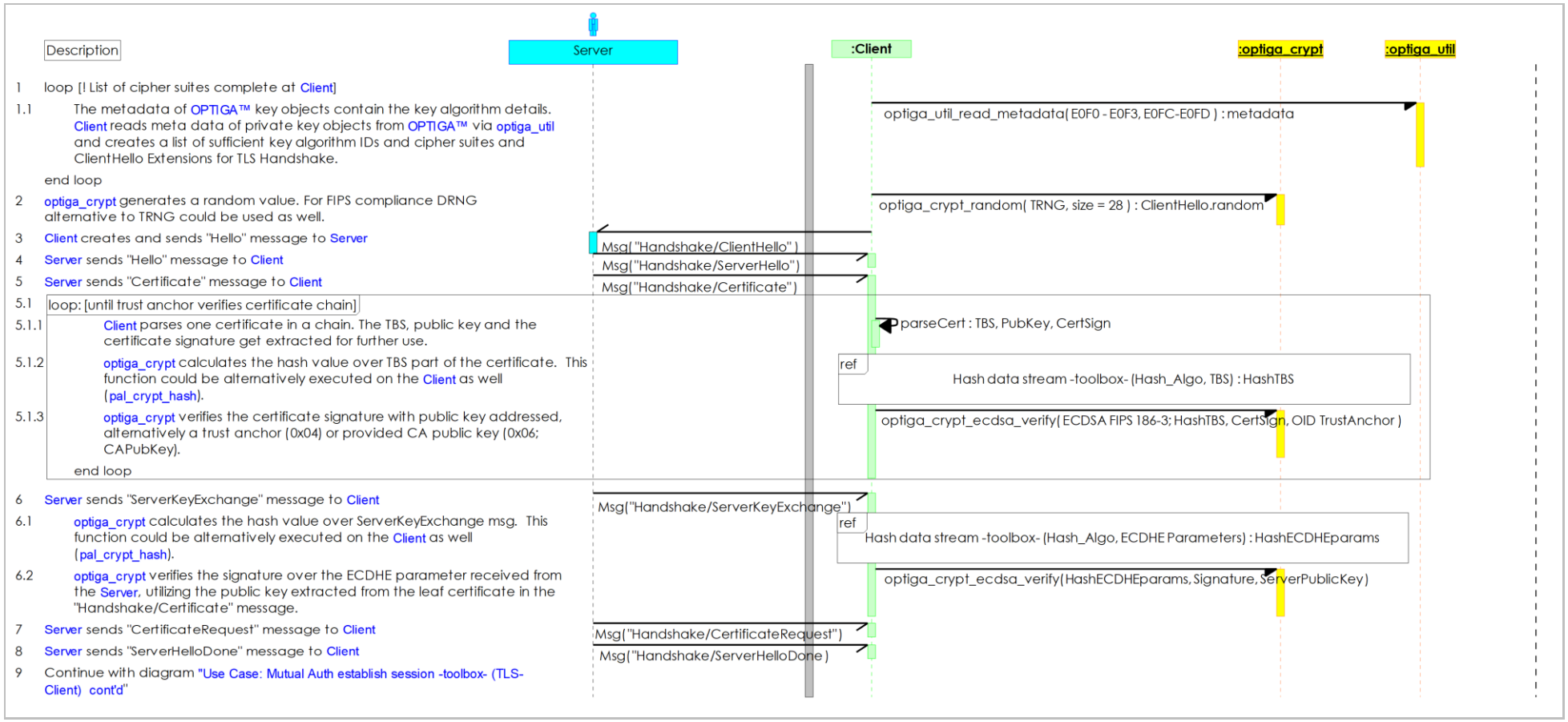


Figure 8 - Use Case: Mutual Authentication establish session -toolbox- (TLS-Client)

Supported Use Cases

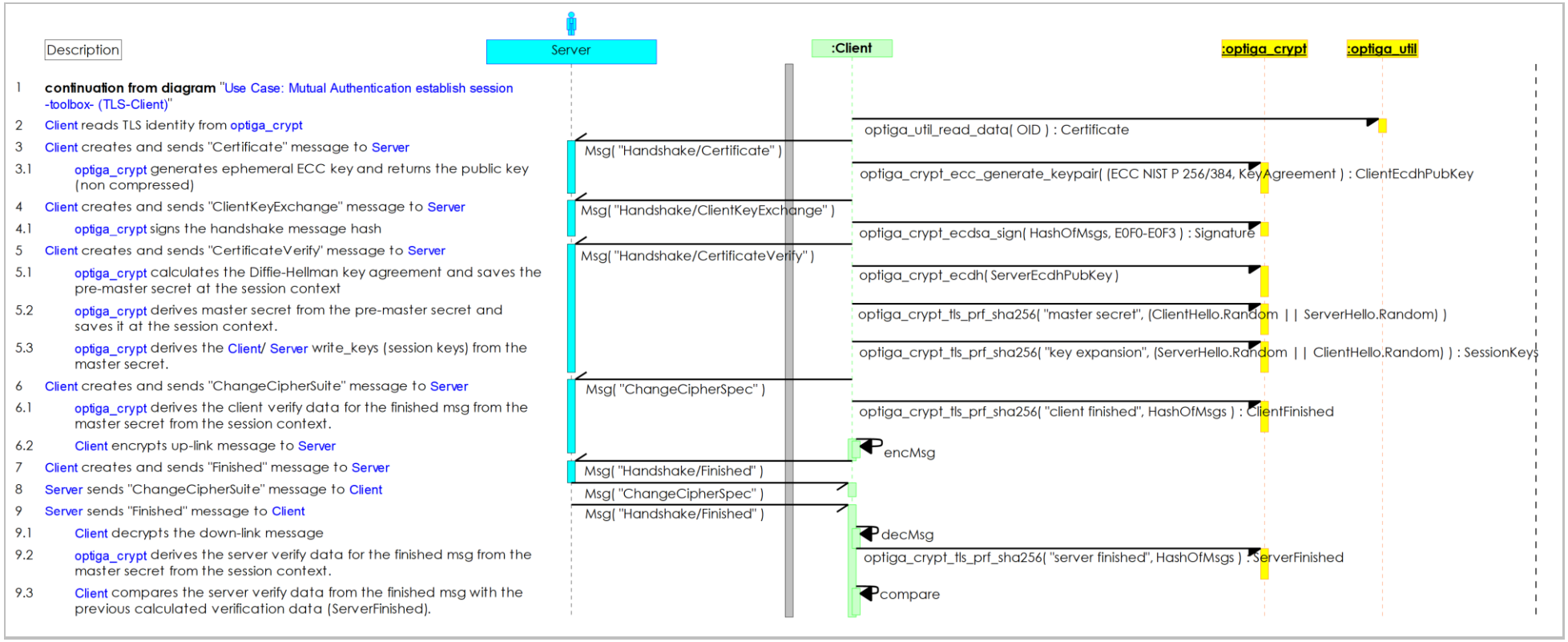


Figure 9 - Use Case: Mutual Auth establish session -toolbox- (TLS-Client) cont'd

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.

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™.

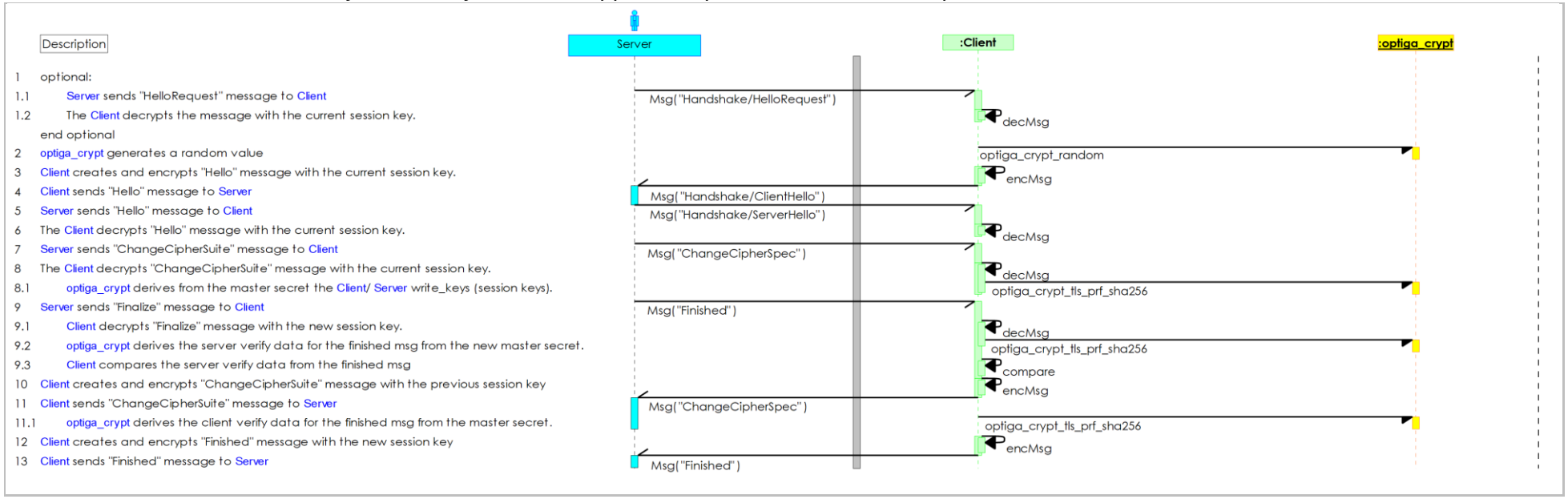


Figure 10 - Use Case: Abbreviated Handshake -toolbox- (TLS-Client)

2.3.3 Use Case: Host Firmware Update

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.
- The Trust Anchor (signer's certificate) is loaded to a data object at OPTIGA™.
- Host receives the firmware update manifest (e.g. image version, signer, hash & sign algorithms, firmware image hash, firmware image decryption key derivation information, manifest signature, etc.) and encrypted firmware image. The details to be signed (TBS) in the manifest are signed by signer and Host verifies the signature generated over TBS using the Trust Anchor installed at OPTIGA™.

Post-condition:

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

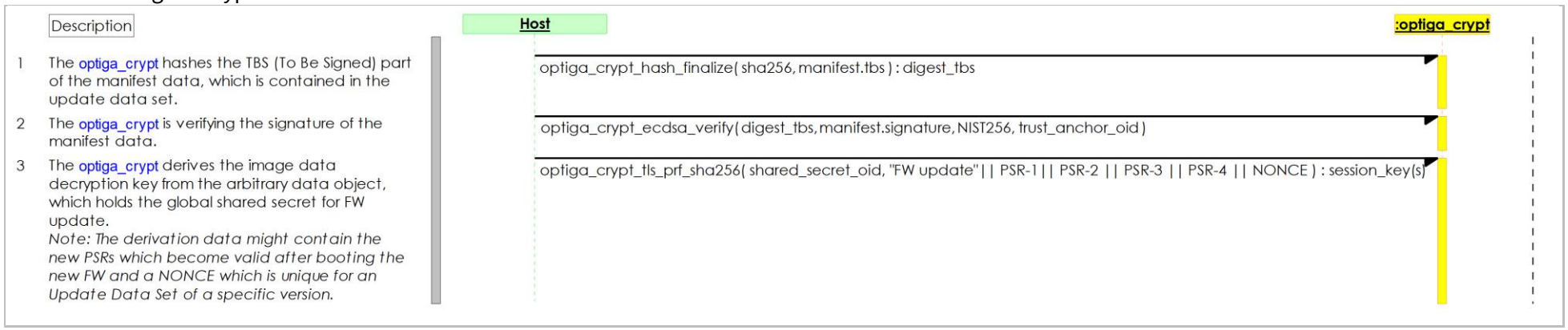


Figure 11 - Use Case: Host Firmware Update

2.3.4 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 Platform Binding Secret data object is not locked. The LcsO (Life Cycle Status of the Object) must be less than operational.

Post-conditions(s):

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

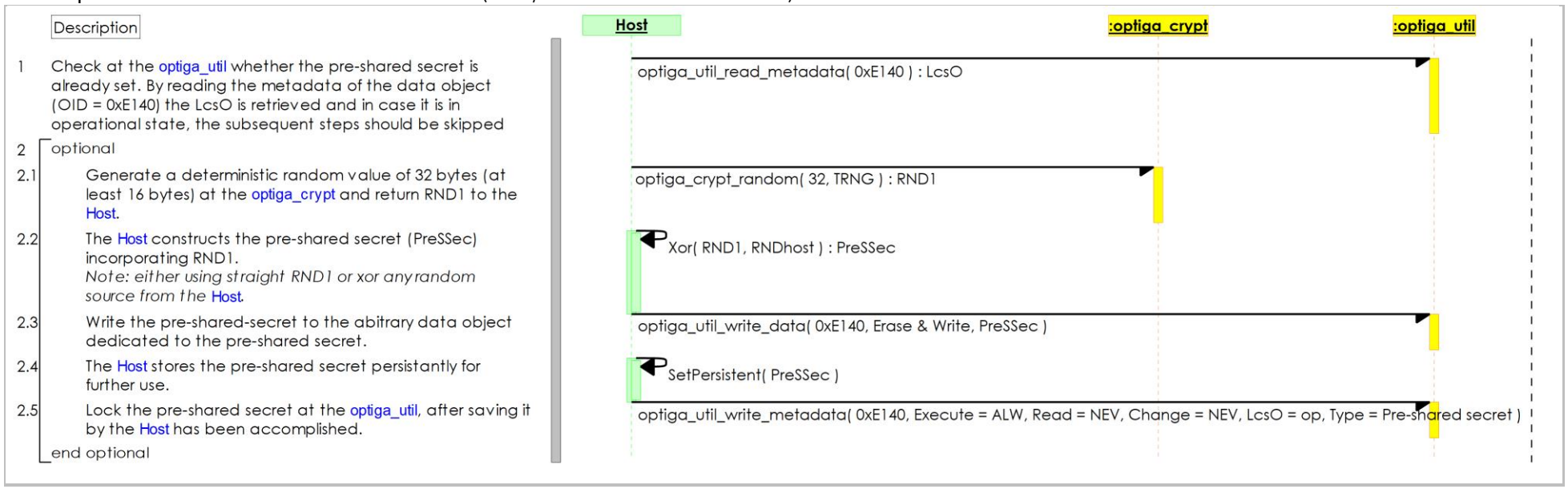


Figure 12 - Use Case: Pair OPTIGA™ with Host (Pre-Shared Secret based)

2.3.5 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.

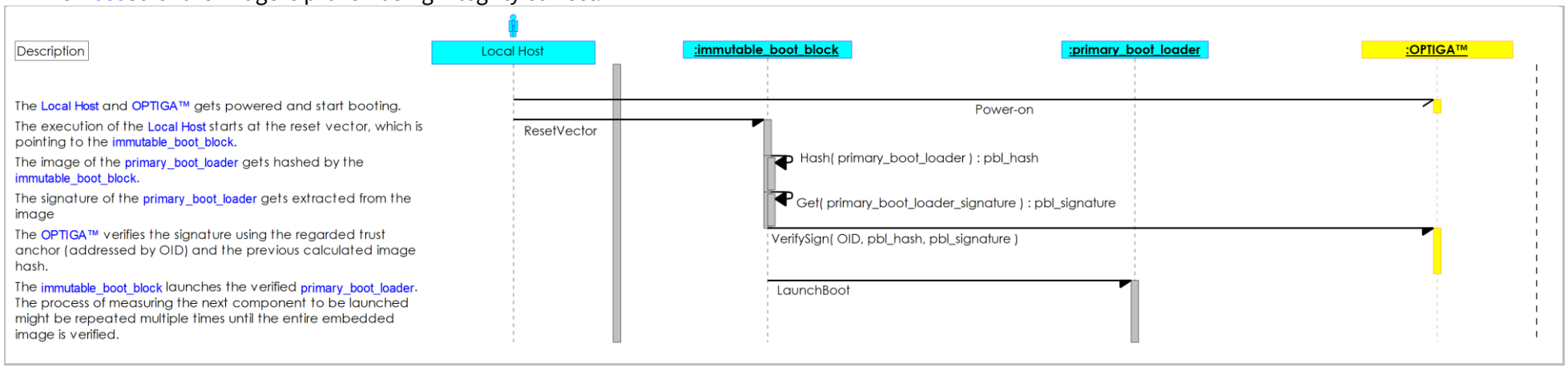


Figure 13 - Use Case: Verified Boot -toolbox-

2.3.6 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 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.

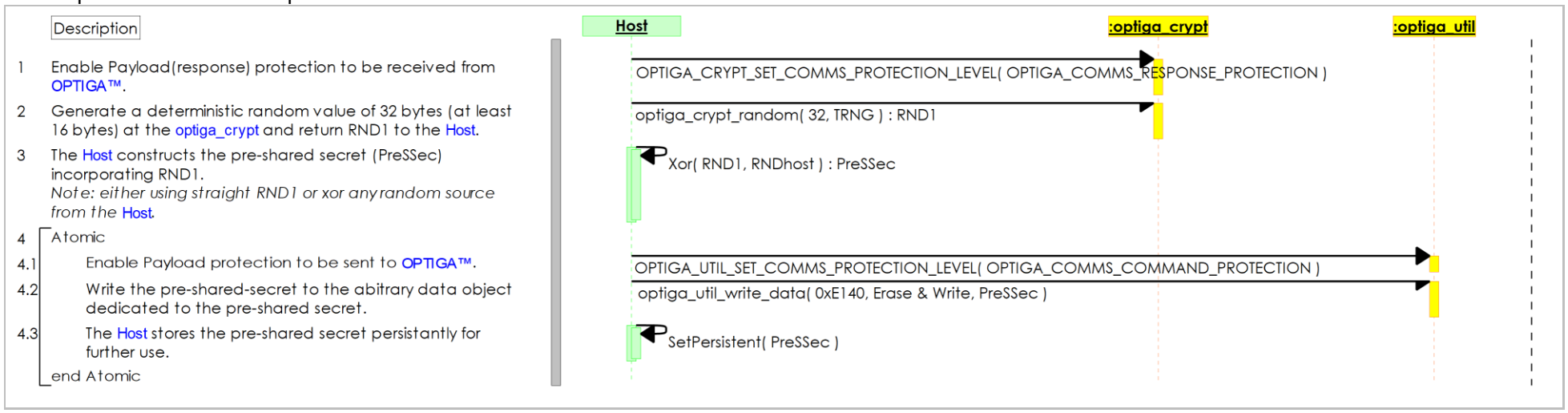


Figure 14 - Use Case: Update Platform Binding Secret during runtime (Pre-Shared Secret based)

2.3.7 Use Case: Local "data-at-rest" protection

A host needs to protect data against access by any third party. This Use Case: Local "data-at-rest" protection is about high volume data encryption at the Host. For that purpose, Host and OPTIGA™ establish a unique key for local data encryption/ decryption. Host generates a random secret once and uses 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

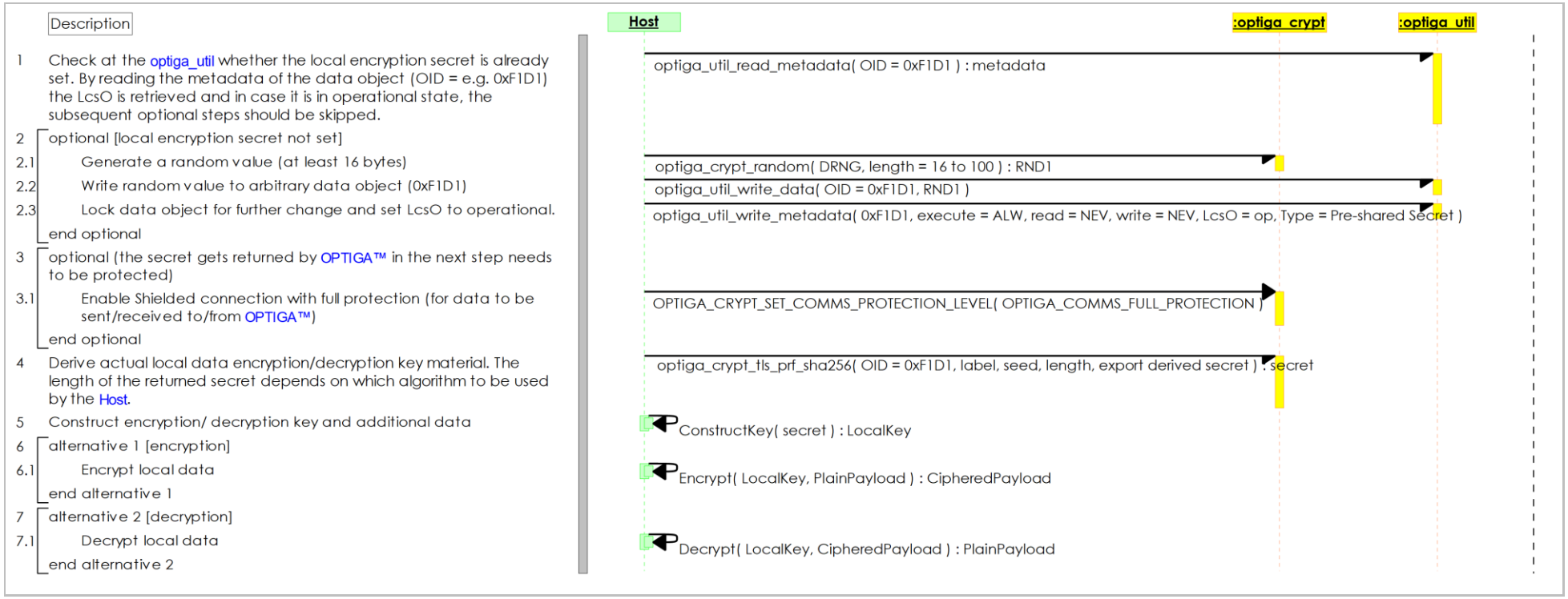


Figure 15 - Use Case: Local "data-at-rest" protection

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

A 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 needs 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.

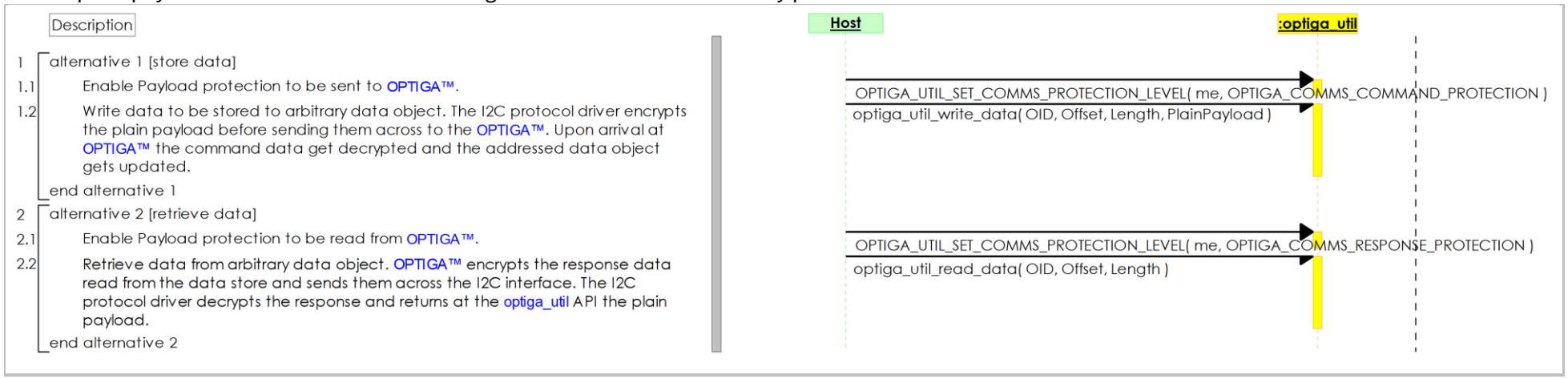


Figure 16 - Use Case: Local "data-at-rest" and "data-in-transit" protection

2.3.9 Use Case: Host "data-at-rest" and "data-in-transit" protection

A host needs to protect data against access by any third party. This Use Case: Host "data-at-rest" and "data-in-transit" protection is about protecting higher volume of data at the Host persistent storage. For that purpose OPTIGA™ encrypts (writing) or decrypts (reading) the data to be store at the host. The host has to persistently store the encrypted data objects at its NVM.

Note: OPTIGA™ Trust M Version 1 doesn't support symmetric algorithms.

Pre-condition:

- The symmetric key for local data protection is randomly generated and available at the OPTIGA™.
- The OPTIGA™ Shielded Connection is enabled.

Post-condition:

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

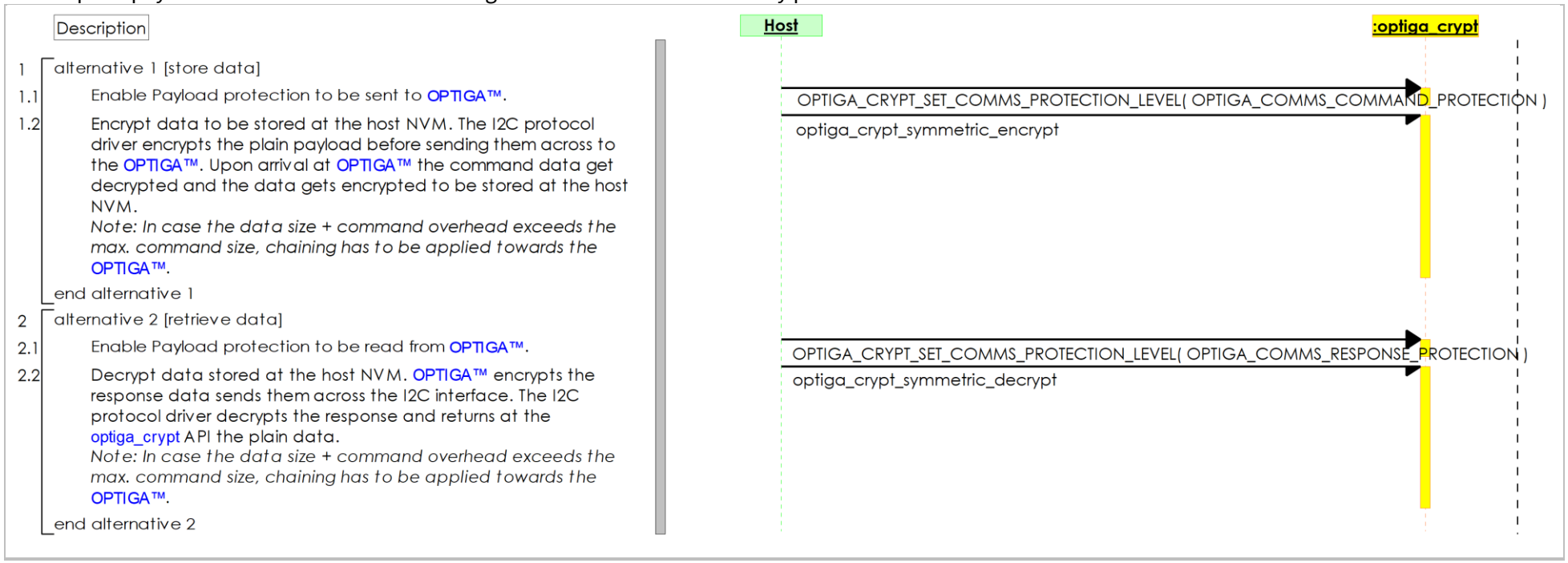


Figure 17 - Use Case: Host "data-at-rest" and "data-in-transit" protection

2.3.10 Use Case: Generate MAC (HMAC with SHA2)

This use case diagram shows the way of generating the MAC for the given input data using the secret installed at OPTIGA™.

Note: OPTIGA™ Trust M Version 1 doesn't support HMAC based operations.

Pre-condition:

- The input secret required for the hmac operation is available at OPTIGA™.

Post-condition:

- The generated MAC is available for Local Host Application for further usage.

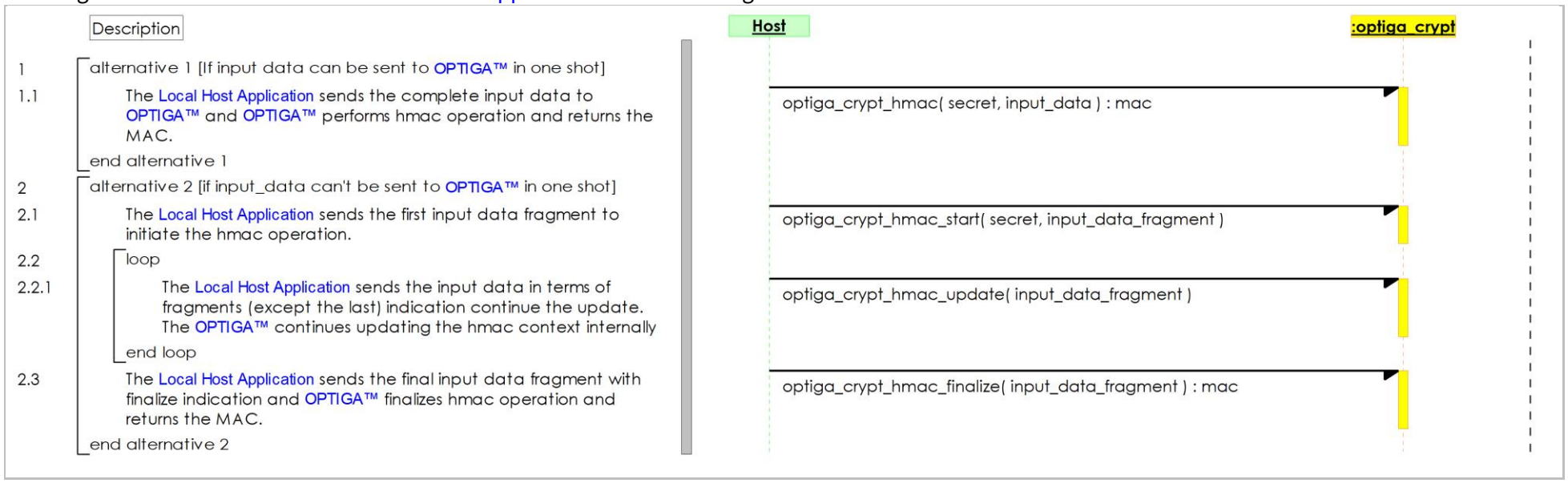


Figure 18 - Use Case: Generate MAC (HMAC with SHA2)

2.3.11 Use Case: Verify Authorization (HMAC with SHA2)

This use case diagram shows the way of verifying the MAC for the given input data using the secret installed at OPTIGA™.
Note: OPTIGA™ Trust M Version 1 doesn't support HMAC based operations.

Pre-condition:

- The input secret required for the HMAC operation is available at OPTIGA™ and its OID is known by the application.

Post-condition:

- The satisfied access condition is available at Local Host Application for further usage.

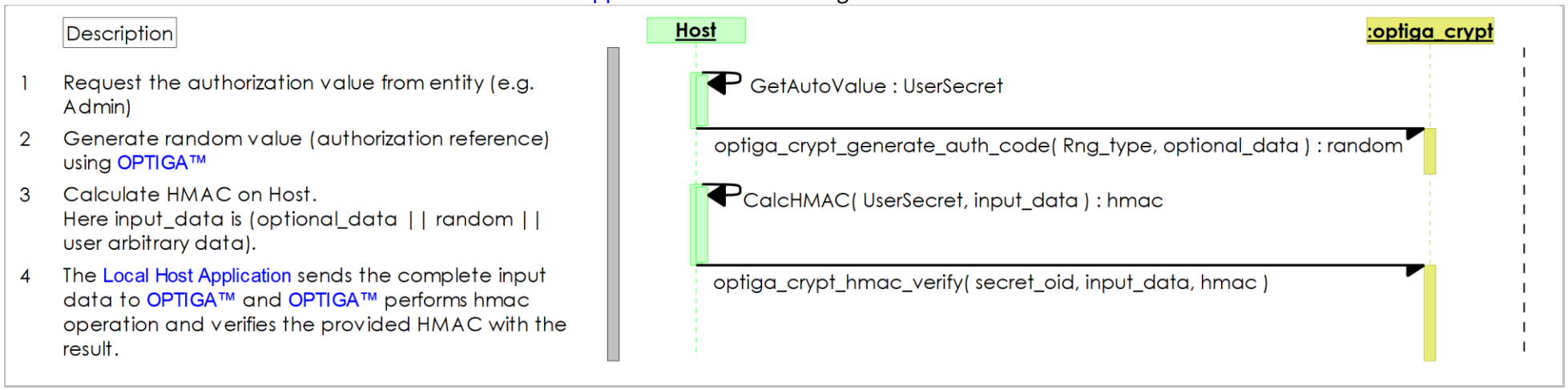


Figure 19 - Use Case: Verify Authorization (HMAC with SHA2)

2.3.12 Use Case: Generate Hash

This use case diagram shows the way of generating the Hash for the given input data using OPTIGA™.

Pre-condition:

- The OPTIGA™ is initialized.

Post-condition:

- The generated Hash is available for Local Host Application for further usage.

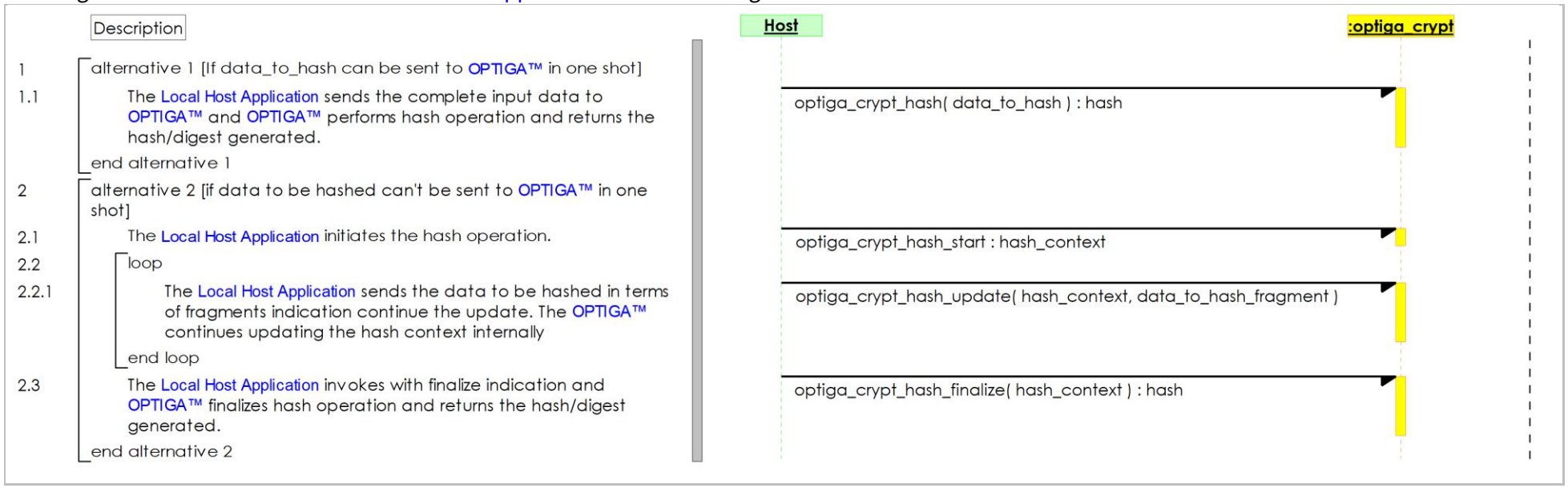


Figure 20 - Use Case: Generate Hash

3 Enabler APIs

The Enabler APIs chapter provides the specification of the host side APIs of the enabler components, which gets 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.

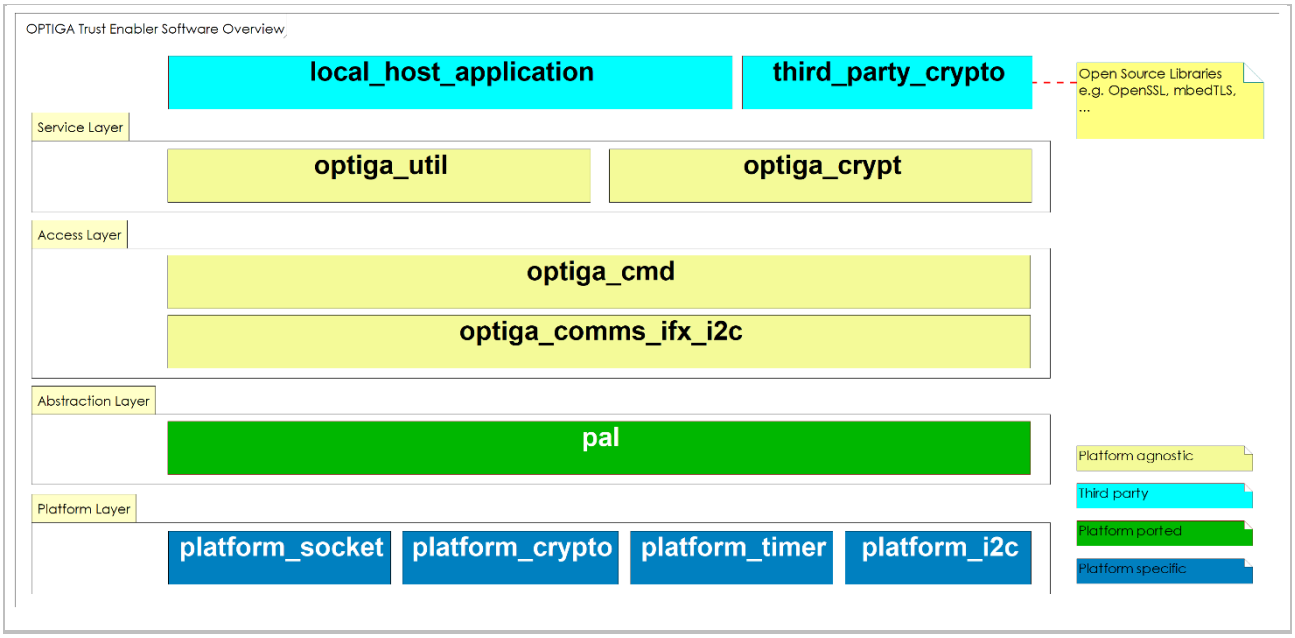


Figure 21 - OPTIGA Trust Enabler Software Overview

3.1 Service Layer Decomposition

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

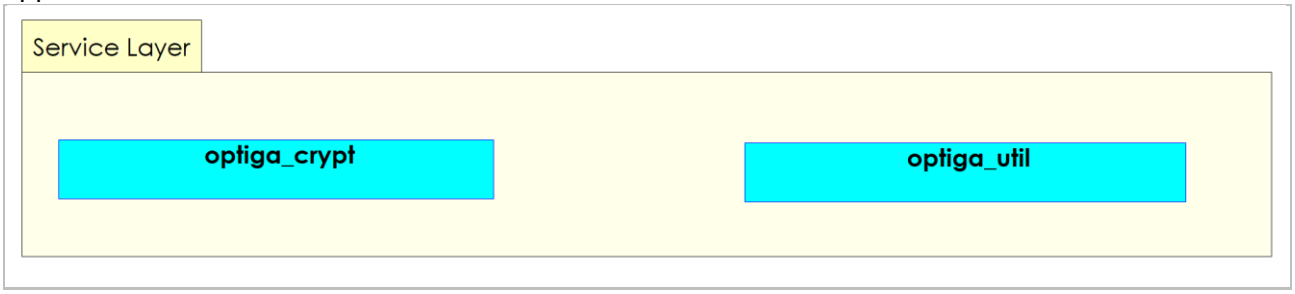


Figure 22 - Service Layer Decomposition

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 toolbox.
- Uses 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™, • The Shielded connection version and protection level gets set to the default version (OPTIGA_COMMS_PROTOCOL_VERSION_PRE_SHARED_SECRET) supported and default level (OPTIGA_COMMS_DEFAULT_PROTECTION_LEVEL) in the instance created as part of this operation. • This instance needs to be updated for the required protection level, before invoking the respective optiga_crypt 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_crypt_create	
	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 *) : optiga_crypt_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.

3.1.1.2 optiga_crypt_random

optiga_crypt_random	
Description	This operation generates random data using OPTIGA™.
Signature	optiga_crypt_random (in me : 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 : 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_generate_auth_code

optiga_crypt_generate_auth_code	
Description	This operation generates random data using OPTIGA™ which gets stored in the acquired session context at OPTIGA™. The random stored in the acquired session context gets used as authorization challenge for hmac verify (optiga_crypt_hmac_verify) kind of operations. • This operation acquires a session if not already acquired. • The minimum size of random stream is 8 bytes and the maximum size of (optional_data || generated random) is 66 bytes. • The (optional_data || generated random) gets stored in the session context

Enabler APIs

optiga_crypt_generate_auth_code	
	acquired at OPTIGA™. • The generated random at OPTIGA™ is also returned to Host.
Signature	optiga_crypt_generate_auth_code (in me : optiga_crypt_t *, in rng_type : optiga_rng_type_t, in optional_data : const uint8_t *, in optional_data_length : uint16_t, inout random_data : uint8_t *, in random_data_length : uint16_t) : optiga_lib_status_t
Parameters	`in me : 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.
	`in optional_data : const uint8_t *` Pointer to optional data which gets prepended to the generated random and gets stored in the acquired session. This is an optional parameter. If optional_data is NULL, This parameter will be ignored.
	`in optional_data_length : uint16_t` Length of optional data. optional_data_length = 0, if optional data (optional_data) is not required to be provided.
	`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.

3.1.1.4 optiga_crypt_hash

optiga_crypt_hash	
Description	This operation performs the hash operation using OPTIGA™ for the provided data (data_to_hash) and returns the digest(hash_output). If the data to be hashed (from external interface e.g. host) is not possible to be sent to OPTIGA™ in a single transaction, then optiga_cmd sends the data to OPTIGA™ automatically in fragments.
Signature	optiga_crypt_hash (in me : optiga_crypt_t *, in hash_algorithm : optiga_hash_types_t, in source_of_data_to_hash : uint8_t, in data_to_hash : const void *, inout hash_output : uint8_t *) : optiga_lib_status_t
Parameters	`in me : optiga_crypt_t *` Pointer to one instance of optiga_crypt.
	`in hash_algorithm : optiga_hash_types_t` Hash algorithm to be used as specified in optiga_hash_type_t.
	`in source_of_data_to_hash : uint8_t` Specifies the source of data to be hashed which could be either data from host or data

optiga_crypt_hash	
	from OPTIGA™ data objects.
	`in data_to_hash : const void *` Pointer to the data to hash. If source_of_data_to_hash specifies the data is from host, data_to_hash is a pointer to a structrue 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 structrue hash_data_in_optiga_t.
	`inout hash_output : uint8_t *` Pointer to a buffer to store the hash generated with a size of expected hash output length.

3.1.1.5 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 : optiga_crypt_t *, inout hash_ctx : p_optiga_hash_context_t) : optiga_lib_status_t
Parameters	`in me : 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.

3.1.1.6 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™.
Signature	optiga_crypt_hash_update (in me : 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 : 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

optiga_crypt_hash_update	
	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.

3.1.1.7 optiga_crypt_hash_finalize

optiga_crypt_hash_finalize	
Description	This operation finalizes the hash.
Signature	optiga_crypt_hash_finalize (in me : optiga_crypt_t *, in hash_ctx : p_optiga_hash_context_t, inout hash_output : uint8_t *) : optiga_lib_status_t
Parameters	`in me : 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.

3.1.1.8 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 : 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 : optiga_crypt_t *` Pointer to one instance of optiga_crypt
	`in curve_id : optiga_ecc_curve_t` Type of curve optiga_ecc_curve_t
	`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

optiga_crypt_ecc_generate_keypair	
	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.

3.1.1.9 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 : 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 : 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.
	`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.

3.1.1.10 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 : 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 : 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 parameter must be a pointer to a public key public_key_from_host_t.
	  • The examples for the format of public key (from host) are provided in ECC Public Key.
	• If the public key is from a certificate OID from OPTIGA™, this parameter must be a pointer to a uint16_t which holds a certificate OID.
	  • The data object type of OID must be set to either Trust Anchor or Device Identity.
	  • The certificate must be a single X.509 certificate (ASN.1 DER encoded) and more details are specified in Parameter Validation section.
	  • The maximum size of certificate allowed is 1300 bytes.

3.1.1.11 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 session based. In case of session based, the private key is used from the session acquired.

Enabler APIs

optiga_crypt_ecdh	
	• 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 : optiga_crypt_t *, in private_key : optiga_key_id_t, in public_key : p_public_key_from_host_t, in export_to_host : bool_t, inout shared_secret : uint8_t *) : optiga_lib_status_t
Parameters	`in me : 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 : 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 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.

3.1.1.12 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 : 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 : 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™.

3.1.1.13 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 : 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 : 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

optiga_crypt_rsa_sign	
	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.

3.1.1.14 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 : 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 : 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 parameter must be a pointer to a public key (public_key_from_host_t).
	  • The examples for the format of public key (from host) are provided in RSA Public Key.
	• If the public key is from a data object (holds a certificate) from OPTIGA™, this parameter must be a pointer to a uint16_t which holds an OID.
	  • The data object type of OID must be set to either Trust Anchor or Device Identity.
	  • The certificate must be a single X.509 certificate (ASN.1 DER encoded) and more details are specified in Parameter Validation section.
	  • The maximum size of certificate allowed is 1300 bytes.

optiga_crypt_rsa_verify	
	`in salt_length : uint16_t` Reserved for future use. The value provided is ignored.

3.1.1.15 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 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. In case of OPTIGA_RSAES_PKCS1_V15 encryption scheme, • The message_length can upto the respective RSA key modulus size - 11 bytes. • For example, message length can be upto 117 bytes if RSA 1024 key is used.
Signature	optiga_crypt_rsa_encrypt_message (in me : 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 : 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 parameter 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

optiga_crypt_rsa_encrypt_message	
	parameter 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™.

3.1.1.16 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 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 : 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 : 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 parameter 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 parameter must be a pointer to a uint16_t which holds an OID.

optiga_crypt_rsa_encrypt_session	
	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™.

3.1.1.17 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 must be the size of the key used to decrypt the message. For example, In case of RSA 2048 (Key size = 256 bytes), the encrypted_message_length is 256 bytes. The caller of this operation has to take care of chaining of encrypted message if length is more than supported in a single operation. 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 : 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 : 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

optiga_crypt_rsa_decrypt_and_export	
	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™.

3.1.1.18 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. The encrypted_message_length must be the size of the key used to decrypt the message. For example, In case of RSA 2048 (Key size = 256 bytes), the encrypted_message_length is 256 bytes.
Signature	optiga_crypt_rsa_decrypt_and_store (in me : 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 : 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.

3.1.1.19 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 66 bytes. For example, in case of RSA key exchange based TLS communication, the client generates 48 bytes of pre-master secret (version info [2] bytes || random [46] bytes).
Signature	optiga_crypt_rsa_generate_pre_master_secret (in me : 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 : 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.

3.1.1.20 optiga_crypt_symmetric_generate_key

optiga_crypt_symmetric_generate_key	
Description	This operation generates symmetric key (e.g. AES) using OPTIGA™. The generated key could be either stored at OPTIGA™ (static key from key store) or exported to host.
Signature	optiga_crypt_symmetric_generate_key (in me : optiga_crypt_t *, in key_type : optiga_symmetric_key_type_t, in key_usage : uint8_t, in export_symmetric_key : bool_t, inout symmetric_key : void *) : optiga_lib_status_t
Parameters	`in me : optiga_crypt_t *` Pointer to one instance of optiga_crypt
	`in key_type : optiga_symmetric_key_type_t` Key type as specified in optiga_symmetric_key_type_t.
	`in key_usage : uint8_t` Specifies the usage (optiga_key_usage_t) of the key generated. Typically the key usage type, OPTIGA_KEY_USAGE_ENCRYPTION is applicable for AES keys which allows to perform encrypt and decrypt operations using the stored keys at OPTIGA™. This parameter is not applicable in case of exporting the key to the host.

Enabler APIs

optiga_crypt_symmetric_generate_key	
	`in export_symmetric_key : bool_t` Specifies whether the key to be exported to the host or not. = TRUE (1) or non-zero, OPTIGA™ exports the generated key. In this case, the symmetric_key is a pointer to a buffer with at least a minimum length required based on key_type chosen. = FALSE (0), stores the generated key with in OPTIGA™. In this case, the symmetric_key is to be a pointer to the device symmetric key OID (optiga_key_id_t).
	`inout symmetric_key : void *` Pointer to the key reference (based on export_symmetric_key).

3.1.1.21 optiga_crypt_symmetric_encrypt

optiga_crypt_symmetric_encrypt	
Description	This operation encrypts (symmetric) MAC for the data provided using OPTIGA™. In case of MAC only based operations (for example, CBC-MAC, CMAC), only the MAC is returned. • Internal padding is performed by OPTIGA™ in case of OPTIGA_SYMMETRIC_CMAC, if the data provided is not block aligned while finalizing the operation. • If the length of plain_data to be encrypted can't be sent to OPTIGA™ in one transaction, then plain_data will be sent to OPTIGA™ in multiple fragments (each fragment must be block length aligned except the last) in a strict sequence which will be taken care by optiga_cmd. • If the length of plain_data can be sent to OPTIGA™ in one transaction, then plain_data will be sent to OPTIGA™ in a single fragment (with start & final option). Note: The shielded connection protection level with command protection is implicitly enabled if the Shielded connection is enabled.
Signature	optiga_crypt_symmetric_encrypt (in me : optiga_crypt_t *, in encryption_mode : optiga_symmetric_encryption_mode_t, in symmetric_key_oid : optiga_key_id_t, in plain_data : const uint8_t *, in plain_data_length : uint32_t, in iv : const uint8_t *, in iv_length : uint16_t, in associated_data : const uint8_t *, in associated_data_length : uint16_t, inout encrypted_data : uint8_t *, inout encrypted_data_length : uint32_t *) : optiga_lib_status_t
Parameters	`in me : optiga_crypt_t *` Pointer to one instance of optiga_crypt
	`in encryption_mode : optiga_symmetric_encryption_mode_t` mode to be used to encrypt the given data. Refer optiga_symmetric_encryption_mode_t for mode types. In case of OPTIGA_SYMMETRIC_CBC_MAC or OPTIGA_SYMMETRIC_CMAC, only the MAC is returned as part of encrypted_data.
	`in symmetric_key_oid : optiga_key_id_t` OID of the symmetric key object (refer optiga_key_id_t) to be used to encrypt the data.

Enabler APIs

optiga_crypt_symmetric_encrypt	
	`in plain_data : const uint8_t *` Pointer to the plain data to be encrypted.
	`in plain_data_length : uint32_t` Length of plain_data provided to be encrypted. This value must be greater than 0. This value must be according to the respective encryption_mode and symmetric key type (e.g. AES) chosen. E.g. in case of AES key and ECB mode, this value must be block length (= 16 bytes) aligned.
	`in iv : const uint8_t *` Pointer to an initialization vector (IV) or nonce to be used during encryption. • This parameter is not used in case of OPTIGA_SYMMETRIC_ECB, OPTIGA_SYMMETRIC_CBC_MAC, and OPTIGA_SYMMETRIC_CMAC encryption or mac generation schemes. The caller can set this parameter as NULL or iv_length to 0. • The length of IV must be based on respective encryption scheme (optiga_symmetric_encryption_mode_t) and key type (e.g. AES) chosen. For example, in case of OPTIGA_SYMMETRIC_CBC and AES key, the iv_length must be 16 bytes.
	`in iv_length : uint16_t` Length of iv provided.
	`in associated_data : const uint8_t *` Pointer to associated data. This is applicable for OPTIGA_SYMMETRIC_CCM scheme only.
	`in associated_data_length : uint16_t` Length of associated_data provided.
	`inout encrypted_data : uint8_t *` Pointer to the buffer to store the encrypted data and/or MAC.
	`inout encrypted_data_length : uint32_t *` Pointer to the length of the encrypted_data buffer. This will be updated with the actual length of encrypted_data returned by OPTIGA™. In case of MAC generation, if the size of MAC generated is bigger than the length of encrypted_data buffer to store/return MAC, then only the first encrypted_data_length bytes out of the data (received from OPTIGA™) are returned.

3.1.1.22 optiga_crypt_symmetric_encrypt_start

optiga_crypt_symmetric_encrypt_start	
Description	This operation initiates encryption (symmetric) sequence for the provided data using OPTIGA™. The encrypted data gets returned to the host in terms of blocks (block length is based on the encryption_mode chosen). • In case of generating MAC, the generated MAC gets returned with the successful optiga_crypt_symmetric_encrypt_final operation. In such a case, encrypted_data can be NULL. The encrypted_data_length is unchanged internally.

Enabler APIs

optiga_crypt_symmetric_encrypt_start	

	• This operation internally acquires a strict sequence (as part of optiga_cmd) before initiating the sequence and the same sequence will be used for optiga_crypt_symmetric_encrypt_continue and optiga_crypt_symmetric_encrypt_final operations.
	  • The strict sequence gets released either by successful completion of optiga_crypt_symmetric_encrypt_final or any failure until optiga_crypt_symmetric_encrypt_final is completed.
	  • Once the optiga_crypt instance is used with this operation successfully,
	  • The same instance is not supposed to be used until the optiga_crypt_symmetric_encrypt_final is completed or the encryption sequence is broken due to any other failures in between.
	  • If used for any other operation or re-initiating this operation, the strict sequence acquired will be internally released automatically and the new operation will be scheduled as usual.
	  • The other operations (optiga_util, optiga_crypt) invoked with different instances during the encryption sequence, will be scheduled automatically after the strict lock is released.
	• If the length of plain_data can't be sent to OPTIGA™ in one transaction, then plain_data will be sent to OPTIGA™ in multiple fragments (each fragment must be block length aligned) which will be taken care by optiga_cmd.
	
	Note: The shielded connection protection level with command protection is implicitly enabled if the Shielded connection is enabled.

Signature	optiga_crypt_symmetric_encrypt_start (in me : optiga_crypt_t *, in encryption_mode : optiga_symmetric_encryption_mode_t, in symmetric_key_oid : optiga_key_id_t, in plain_data : const uint8_t *, in plain_data_length : uint32_t, in iv : const uint8_t *, in iv_length : uint16_t, in associated_data : const uint8_t *, in associated_data_length : uint16_t, in total_plain_data_length : uint16_t, inout encrypted_data : uint8_t *, inout encrypted_data_length : uint32_t *) : optiga_lib_status_t
Parameters	`in me : optiga_crypt_t *` Pointer to one instance of optiga_crypt
	`in encryption_mode : optiga_symmetric_encryption_mode_t` Mode to be used to encrypt the given data. Refer optiga_symmetric_encryption_mode_t for mode types. In case of MAC based, only the MAC is returned during optiga_crypt_symmetric_encrypt_final operation.
	`in symmetric_key_oid : optiga_key_id_t` OID of the symmetric key object (refer optiga_key_id_t) to be used to encrypt the data.
	`in plain_data : const uint8_t *` Pointer to a data to be encrypted.
	`in plain_data_length : uint32_t` Length of plain_data provided. This value must be greater than 0 and block length aligned. E.g. in case of AES based operations, this value must be block length (= 16 bytes) aligned.

Enabler APIs

optiga_crypt_symmetric_encrypt_start	
	`in iv : const uint8_t *` Pointer to an initialization vector (iv) or nonce to be used during encryption. • This parameter is not used in case of OPTIGA_SYMMETRIC_ECB, OPTIGA_SYMMETRIC_CBC_MAC, and OPTIGA_SYMMETRIC_CMAC encryption or mac generation schemes. The caller can set this parameter as NULL or iv_length to 0. • The length of IV must be based on respective encryption scheme (optiga_symmetric_encryption_mode_t) and key type (e.g. AES) chosen. For example, in case of OPTIGA_SYMMETRIC_CBC and AES key, the iv_length must be 16 bytes.
	`in iv_length : uint16_t` Length of iv provided.
	`in associated_data : const uint8_t *` Pointer to associated data. This is applicable for OPTIGA_SYMMETRIC_CCM scheme only.
	`in associated_data_length : uint16_t` Length of associated_data provided.
	`in total_plain_data_length : uint16_t` Length of total data to be encrypted until optiga_crypt_symmetric_encrypt_final. This parameter is required for authenticated encryption modes e.g. CCM, GCM, etc. For other modes, this parameter is ignored internally.
	`inout encrypted_data : uint8_t *` Pointer to the buffer to store the encrypted data.
	`inout encrypted_data_length : uint32_t *` Pointer to the length of the encrypted_data buffer. This will be updated with the actual length of encrypted_data returned by OPTIGA™ in case of non MAC based operations.

3.1.1.23 optiga_crypt_symmetric_encrypt_continue

optiga_crypt_symmetric_encrypt_continue	
Description	This operation encrypts (symmetric) the data provided using OPTIGA™. • In case of generating MAC, the generated MAC gets returned with the successful optiga_crypt_symmetric_encrypt_final operation. In such a case, encrypted_data can be NULL. The encrypted_data_length is unchanged internally. • No internal padding is performed by OPTIGA™. • This operation internally uses the same strict sequence acquired internally (as part of optiga_cmd) by optiga_crypt_symmetric_encrypt_start. Hence before invoking this operation, the successful completion of optiga_crypt_symmetric_encrypt_start is must. • If the length of plain_data to be encrypted can't be sent to OPTIGA™ in one transaction, then plain_data will be sent to OPTIGA™ in multiple fragments (each fragment must be block length aligned) which will be taken care by optiga_cmd. Note: The shielded connection protection level with command protection is implicitly

optiga_crypt_symmetric_encrypt_continue	
	enabled if the Shielded connection is enabled.
Signature	optiga_crypt_symmetric_encrypt_continue (in me : optiga_crypt_t *, in plain_data : const uint8_t *, in plain_data_length : uint32_t, inout encrypted_data : uint8_t *, inout encrypted_data_length : uint32_t *) : optiga_lib_status_t
Parameters	`in me : optiga_crypt_t *` Pointer to one instance of optiga_crypt
	`in plain_data : const uint8_t *` Pointer to a data to be encrypted.
	`in plain_data_length : uint32_t` Length of plain_data provided. This value must be greater than 0 and block length aligned. E.g. in case of AES based operations, this value must be block length (= 16 bytes) aligned.
	`inout encrypted_data : uint8_t *` Pointer to the buffer to store the encrypted data.
	`inout encrypted_data_length : uint32_t *` Pointer to the length of the encrypted_data buffer. This will be updated with the actual length of encrypted_data returned by OPTIGA™ in case of non MAC based operations.

3.1.1.24 optiga_crypt_symmetric_encrypt_final

optiga_crypt_symmetric_encrypt_final	
Description	This operation encrypts (symmetric) the data provided using OPTIGA™. • In case of generating MAC, the generated MAC only gets returned upon successful completion of this operation. • Internal padding is performed by OPTIGA™ in case of OPTIGA_SYMMETRIC_CMAC, if the data provided is not block aligned while finalizing the operation. • This operation internally uses the same strict sequence acquired (as part of optiga_cmd) by optiga_crypt_symmetric_encrypt_start. Hence before invoking this operation, the successful completion of optiga_crypt_symmetric_encrypt_start is must. • If the length of plain_data to be encrypted can't be sent to OPTIGA™ in one transaction, then plain_data will be sent to OPTIGA™ in multiple fragments (each fragment must be block length aligned except the last) which will be taken care by optiga_cmd. Note: The shielded connection protection level with command protection is implicitly enabled if the Shielded connection is enabled.
Signature	optiga_crypt_symmetric_encrypt_final (in me : optiga_crypt_t *, in plain_data : const uint8_t *, in plain_data_length : uint32_t, inout encrypted_data : uint8_t *, inout encrypted_data_length : uint32_t *) : optiga_lib_status_t
Parameters	`in me : optiga_crypt_t *` Pointer to one instance of optiga_crypt
	`in plain_data : const uint8_t *`

optiga_crypt_symmetric_encrypt_final	
	Pointer to a data to be encrypted.
	`in plain_data_length : uint32_t` Length of plain_data provided. This value must be greater than 0 and must be according to the respective encryption_mode and symmetric key type (e.g. AES) chosen. E.g. in case of AES key and ECB mode, this value must be block length (= 16 bytes) aligned.
	`inout encrypted_data : uint8_t *` Pointer to the buffer to store the encrypted data or MAC (in case of MAC based algorithms e.g. CBC-MAC).
	`inout encrypted_data_length : uint32_t *` Pointer to the length of the encrypted_data buffer. This will be updated with the actual length of encrypted_data returned by OPTIGA™. In case of MAC generation, if the size of MAC generated is bigger than the length of encrypted_data buffer to store/return MAC, then only the first encrypted_data_length bytes out of the data (received from OPTIGA™) are returned.

3.1.1.25 optiga_crypt_symmetric_encrypt_ecb

optiga_crypt_symmetric_encrypt_ecb	
Description	This operation encrypts (OPTIGA_SYMMETRIC_ECB mode) the data provided using OPTIGA™. • No internal padding is performed by OPTIGA™. • If the length of plain_data to be encrypted can't be sent to OPTIGA™ in one transaction, then plain_data will be sent to OPTIGA™ in multiple fragments (each fragment is block length aligned) in a strict sequence which will be taken care by optiga_cmd. • If the length of plain_data can be sent to OPTIGA™ in one transaction, then plain_data will be sent to OPTIGA™ in a single fragment (with start & final option). Note: The shielded connection protection level with command protection is implicitly enabled if the Shielded connection is enabled.
Signature	optiga_crypt_symmetric_encrypt_ecb (in me : optiga_crypt_t *, in symmetric_key_oid : optiga_key_id_t, in plain_data : const uint8_t *, in plain_data_length : uint32_t, inout encrypted_data : uint8_t *, inout encrypted_data_length : uint32_t *) : optiga_lib_status_t
Parameters	`in me : optiga_crypt_t *` Pointer to one instance of optiga_crypt
	`in symmetric_key_oid : optiga_key_id_t` OID of the symmetric key object (refer optiga_key_id_t) to be used to encrypt the data.
	`in plain_data : const uint8_t *` Pointer to the plain data to be encrypted.

Enabler APIs

optiga_crypt_symmetric_encrypt_ecb	
	`in plain_data_length : uint32_t` Length of plain_data provided to be encrypted. This value must be greater than 0 and must be block length aligned. E.g. in case of AES based operations, this value must be block length (= 16 bytes) aligned.
	`inout encrypted_data : uint8_t *` Pointer to the buffer to store the encrypted data.
	`inout encrypted_data_length : uint32_t *` Pointer to the length of the encrypted_data buffer. This will be updated with the actual length of encrypted_data returned by OPTIGA™.

3.1.1.26 optiga_crypt_symmetric_decrypt

optiga_crypt_symmetric_decrypt	
Description	This operation decrypts the provided encrypted data using OPTIGA™ and returns the plain data to the host. • No internal padding is performed by OPTIGA™. • If the length of encrypted_data to be decrypted can't be sent to OPTIGA™ in one transaction, then encrypted_data will be sent to OPTIGA™ in multiple fragments (each fragment must be block length aligned except the last) in a strict sequence which will be taken care by optiga_cmd. • If the length of encrypted_data can be sent to OPTIGA™ in one transaction, then encrypted_data will be sent to OPTIGA™ in a single fragment (with start & final option). Note: The shielded connection protection level with response protection is implicitly enabled if the shielded connection is enabled.
Signature	optiga_crypt_symmetric_decrypt (in me : optiga_crypt_t *, in encryption_mode : optiga_symmetric_encryption_mode_t, in symmetric_key_oid : optiga_key_id_t, in encrypted_data : const uint8_t *, in encrypted_data_length : uint32_t, in iv : const uint8_t *, in iv_length : uint16_t, in associated_data : const uint8_t *, in associated_data_length : uint16_t, inout plain_data : uint8_t *, inout plain_data_length : uint32_t *) : optiga_lib_status_t
Parameters	`in me : optiga_crypt_t *` Pointer to one instance of optiga_crypt
	`in encryption_mode : optiga_symmetric_encryption_mode_t` Mode to be used to decrypt the given data. Refer optiga_symmetric_encryption_mode_t for mode types. The OPTIGA_SYMMETRIC_CBC_MAC, and OPTIGA_SYMMETRIC_CMAC are scheme is not applicable.
	`in symmetric_key_oid : optiga_key_id_t` OID of the symmetric key object (refer optiga_key_id_t) to be used to decrypt the data.
	`in encrypted_data : const uint8_t *`

optiga_crypt_symmetric_decrypt	
	Pointer to the encrypted data to be decrypted.
	`in encrypted_data_length : uint32_t` Length of encrypted_data. This value must be greater than 0 and must be according to the respective encryption_mode and symmetric key type (e.g. AES) chosen. E.g. in case of AES key and ECB mode, this value must be block length (= 16 bytes) aligned.
	`in iv : const uint8_t *` Pointer to an initialization vector (iv) or nonce to be used during decryption. This parameter is not used in case of OPTIGA_SYMMETRIC_ECB and OPTIGA_SYMMETRIC_CBC_MAC encryption schemes. The caller can set this parameter as NULL or iv_length to 0. The length of IV must be based on respective encryption scheme (optiga_symmetric_encryption_mode_t) and key type (e.g. AES) chosen. For example, in case of OPTIGA_SYMMETRIC_CBC and AES key, the iv_length must be 16 bytes.
	`in iv_length : uint16_t` Length of iv provided.
	`in associated_data : const uint8_t *` Pointer to associated data. This is applicable for OPTIGA_SYMMETRIC_CCM scheme only.
	`in associated_data_length : uint16_t` Length of associated_data provided.
	`inout plain_data : uint8_t *` Pointer to a buffer to store the decrypted data returned by OPTIGA™.
	`inout plain_data_length : uint32_t *` Pointer to the length of plain_data buffer provided. This will be updated with the actual length of plain_data returned by OPTIGA™.

3.1.1.27 optiga_crypt_symmetric_decrypt_start

optiga_crypt_symmetric_decrypt_start	

Description	This operation initiates the decrypt (symmetric) sequence for the provided encrypted data using OPTIGA™. The plain data gets exported to the host in terms of blocks (block length is based on the encryption_mode chosen)
	
	• No internal padding is performed by OPTIGA™.
	• This operation internally acquires a strict sequence (as part of optiga_cmd) before initiating the sequence and the same sequence will be used for optiga_crypt_symmetric_decrypt_continue and optiga_crypt_symmetric_decrypt_final operations.
	  • The strict sequence gets released either by successful completion of optiga_crypt_symmetric_decrypt_final or any failure until optiga_crypt_symmetric_decrypt_final is completed.
	  • Once the optiga_crypt instance is used with this operation successfully,

optiga_crypt_symmetric_decrypt_start	
	• The same instance is not supposed to be used until the optiga_crypt_symmetric_decrypt_final is completed or the decryption sequence is broken due to any other failures in between. • If used for any other operation or re-initiating this operation, the strict sequence acquired will be internally released automatically and the new operation will be scheduled as usual. • The other operations (optiga_util, optiga_crypt) invoked with different instances during the decryption sequence, will be scheduled automatically after the strict lock is released. • If the length of encrypted_data can't be sent to OPTIGA™ in one transaction, then encrypted_data will be sent to OPTIGA™ in multiple fragments (each fragment must be block length aligned) which will be taken care by optiga_cmd. Note: The shielded connection protection level with response protection is implicitly enabled if the Shielded connection is enabled.
Signature	optiga_crypt_symmetric_decrypt_start (in me : optiga_crypt_t *, in encryption_mode : optiga_symmetric_encryption_mode_t, in symmetric_key_oid : optiga_key_id_t, in encrypted_data : const uint8_t *, in encrypted_data_length : uint32_t, in iv : const uint8_t *, in iv_length : uint16_t, in associated_data : const uint8_t *, in associated_data_length : uint16_t, in total_encrypted_data_length : uint16_t, inout plain_data : uint8_t *, inout plain_data_length : uint32_t *) : optiga_lib_status_t
Parameters	`in me : optiga_crypt_t *` Pointer to one instance of optiga_crypt
	`in encryption_mode : optiga_symmetric_encryption_mode_t` Mode to be used to decrypt the given data. Refer optiga_symmetric_encryption_mode_t for mode types. The OPTIGA_SYMMETRIC_CBC_MAC, and OPTIGA_SYMMETRIC_CMAC are scheme is not applicable.
	`in symmetric_key_oid : optiga_key_id_t` OID of the symmetric key object (refer optiga_key_id_t) to be used to decrypt the data.
	`in encrypted_data : const uint8_t *` Pointer to the encrypted data to be decrypted.
	`in encrypted_data_length : uint32_t` Length of encrypted_data. This value must be greater than 0 and must be block length aligned. E.g. in case of AES based operations, this value must be block length (= 16 bytes) aligned.
	`in iv : const uint8_t *` Pointer to an initialization vector (iv) or nonce to be used during encryption. If not applicable (e.g. OPTIGA_SYMMETRIC_ECB and OPTIGA_SYMMETRIC_CBC_MAC), The caller can set this parameter as NULL or iv_length to 0. The length of IV must be 1 block length (e.g. in case of AES key usage, block length is 16 bytes)
	`in iv_length : uint16_t` Length of iv provided.

optiga_crypt_symmetric_decrypt_start	
	`in associated_data : const uint8_t *` Pointer to associated data. This is applicable for OPTIGA_SYMMETRIC_CCM scheme only.
	`in associated_data_length : uint16_t` Length of associated_data provided.
	`in total_encrypted_data_length : uint16_t` Length of total data to be decrypted (including the MAC length if applicable) until optiga_crypt_symmetric_decrypt_final. This parameter is required for authenticated encryption modes e.g. CCM, GCM, etc. For other modes, this parameter is ignored internally.
	`inout plain_data : uint8_t *` Pointer to a buffer to store the decrypted data returned by OPTIGA™.
	`inout plain_data_length : uint32_t *` Pointer to the length of plain_data buffer provided. This will be updated with the actual length of plain_data returned by OPTIGA™.

3.1.1.28 optiga_crypt_symmetric_decrypt_continue

optiga_crypt_symmetric_decrypt_continue	
Description	This operation decrypts the provided encrypted data using OPTIGA™ and returns the decrypted data to the host. • No internal padding is performed by OPTIGA™. • This operation internally uses the same strict sequence acquired internally (as part of optiga_cmd) by optiga_crypt_symmetric_decrypt_start. Hence before invoking this operation, the successful completion of optiga_crypt_symmetric_decrypt_start is must. • If the length of encrypted_data can't be sent to OPTIGA™ in one transaction, then encrypted_data will be sent to OPTIGA™ in multiple fragments (each fragment must be block length aligned) which will be taken care by optiga_cmd. Note: The shielded connection protection level with response protection is implicitly enabled if the Shielded connection is enabled.
Signature	optiga_crypt_symmetric_decrypt_continue (in me : optiga_crypt_t *, in encrypted_data : const uint8_t *, in encrypted_data_length : uint32_t, inout plain_data : uint8_t *, inout plain_data_length : uint32_t *) : optiga_lib_status_t
Parameters	`in me : optiga_crypt_t *` Pointer to one instance of optiga_crypt
	`in encrypted_data : const uint8_t *` Pointer to the encrypted data to be decrypted.
	`in encrypted_data_length : uint32_t` Length of encrypted_data. This value must be greater than 0 and must be block length aligned. E.g. in case of AES

optiga_crypt_symmetric_decrypt_continue	
	based operations, this value must be block length (= 16 bytes) aligned.
	`inout plain_data : uint8_t *` Pointer to a buffer to store the decrypted data returned by OPTIGA™.
	`inout plain_data_length : uint32_t *` Pointer to the length of plain_data buffer provided. This will be updated with the actual length of plain_data returned by OPTIGA™.

3.1.1.29 optiga_crypt_symmetric_decrypt_final

optiga_crypt_symmetric_decrypt_final	
Description	This operation decrypts the provided encrypted data using OPTIGA™ and returns the decrypted data to the host. • No internal padding is performed by OPTIGA™. • This operation internally uses the same strict sequence acquired (as part of optiga_cmd) by optiga_crypt_symmetric_decrypt_start. Hence before invoking this operation, the successful completion of optiga_crypt_symmetric_decrypt_start is must. • If the length of encrypted_data can't be sent to OPTIGA™ in one transaction, then encrypted_data will be sent to OPTIGA™ in multiple fragments (each fragment must be block length aligned except the last) which will be taken care by optiga_cmd. Note: The shielded connection protection level with response protection is implicitly enabled if the Shielded connection is enabled.
Signature	optiga_crypt_symmetric_decrypt_final (in me : optiga_crypt_t *, in encrypted_data : const uint8_t *, in encrypted_data_length : uint32_t, inout plain_data : uint8_t *, inout plain_data_length : uint32_t *) : optiga_lib_status_t
Parameters	`in me : optiga_crypt_t *` Pointer to one instance of optiga_crypt
	`in encrypted_data : const uint8_t *` Pointer to the encrypted data to be decrypted.
	`in encrypted_data_length : uint32_t` Length of encrypted_data. This value must be greater than 0 and must be according to the respective encryption_mode and symmetric key type (e.g. AES) chosen. E.g. in case of AES key and ECB mode, this must be block length (= 16 bytes) aligned.
	`inout plain_data : uint8_t *` Pointer to a buffer to store the decrypted data returned by OPTIGA™.
	`inout plain_data_length : uint32_t *` Pointer to the length of plain_data buffer provided. This will be updated with the actual length of plain_data returned by OPTIGA™.

3.1.1.30 optiga_crypt_symmetric_decrypt_ecb

optiga_crypt_symmetric_decrypt_ecb	
Description	This operation encrypts (ECB mode as specified by [SP 800-38A]) the data provided using OPTIGA™. • No internal padding is performed by OPTIGA™. • If the length of encrypted_data can't be sent to OPTIGA™ in one transaction, then encrypted_data will be sent to OPTIGA™ in multiple fragments (each fragment is block length aligned) in a strict sequence which will be taken care by optiga_cmd. • If the length of encrypted_data can be sent to OPTIGA™ in one transaction, then encrypted_data will be sent to OPTIGA™ in a single fragment (with start & final option). Note: The shielded connection protection level with command protection is implicitly enabled if the Shielded connection is enabled.
Signature	optiga_crypt_symmetric_decrypt_ecb (in me : optiga_crypt_t *, in symmetric_key_oid : optiga_key_id_t, in encrypted_data : const uint8_t *, in encrypted_data_length : uint32_t, inout plain_data : uint8_t *, inout plain_data_length : uint32_t *) : optiga_lib_status_t
Parameters	`in me : optiga_crypt_t *` Pointer to one instance of optiga_crypt
	`in symmetric_key_oid : optiga_key_id_t` OID of the symmetric key object (refer optiga_key_id_t) to be used to encrypt the data.
	`in encrypted_data : const uint8_t *` Pointer to encrypted data.
	`in encrypted_data_length : uint32_t` Length of the encrypted_data. This value must be greater than 0 and must be block length aligned. E.g. in case of AES based operations, this value must be block length (= 16 bytes) aligned.
	`inout plain_data : uint8_t *` Pointer to a buffer to store plain data returned by OPTIGA™.
	`inout plain_data_length : uint32_t *` Pointer to the length of plain_data buffer. This gets updated with the actual length of plain data returned by OPTIGA™.

3.1.1.31 optiga_crypt_hmac

optiga_crypt_hmac	
Description	This operation performs hmac operation using a shared secret at OPTIGA™. If the length of input_data, • can't be sent to OPTIGA™ in one transaction, then input_data will be sent to OPTIGA™ in multiple fragments in a strict sequence which will be taken care by

optiga_crypt_hmac	
	optiga_cmd. • can be sent to OPTIGA™ in one transaction, then input_data will be sent to OPTIGA™ in a single fragment (with start & final option). Note: The shielded connection protection level with command protection is implicitly enabled if the Shielded connection is enabled.
Signature	optiga_crypt_hmac (in me : optiga_crypt_t *, in type : optiga_hmac_type_t, in secret : uint16_t, in input_data : const uint8_t *, in input_data_length : uint32_t, inout mac : uint8_t *, inout mac_length : uint32_t *) : optiga_lib_status_t
Parameters	`in me : optiga_crypt_t *` Pointer to one instance of optiga_crypt
	`in type : optiga_hmac_type_t` Type of scheme to be used (e.g. HMAC-SHA256, HMAC-SHA384). Refer optiga_hmac_type_t for the supported schemes.
	`in secret : uint16_t` OID of input secret (shared secret from a data object or a session (OPTIGA_KEY_ID_SESSION_BASED) already acquired by the respective optiga_crypt instance).
	`in input_data : const uint8_t *` Pointer to input data.
	`in input_data_length : uint32_t` Length of input_data provided. This value must be greater than 0.
	`inout mac : uint8_t *` Pointer to the buffer to store the generated MAC.
	`inout mac_length : uint32_t *` Pointer to the length of the mac buffer. This will be updated with the actual length of mac returned by OPTIGA™. If the mac_length is lesser than the length of MAC received from OPTIGA™, then the first mac_length bytes are only returned.

3.1.1.32 optiga_crypt_hmac_start

optiga_crypt_hmac_start	

Description	This operation initiates hmac operation.
	
	• This operation internally acquires a strict sequence (as part of optiga_cmd) before initiating the sequence and the same strict sequence will be used for optiga_crypt_hmac_update and optiga_crypt_hmac_finalize operations.
	  • The strict sequence gets released either by successful completion of optiga_crypt_hmac_finalize or any failure until optiga_crypt_hmac_finalize is completed.
	  • Once the optiga_crypt instance is used with this operation successfully,
	  • The same instance is not supposed to be used until the optiga_crypt_hmac_finalize is completed or the sequence is broken due to

optiga_crypt_hmac_start	
	any other failures in between. • If used for any other operation or re-initiating this operation, the strict sequence acquired will be internally released automatically and the new operation will be scheduled as usual. • The other operations (optiga_util, optiga_crypt) invoked with different instances during the encryption sequence, will be scheduled automatically after the strict sequence is released. • If the length of input_data can't be sent to OPTIGA™ in one transaction, then input_data will be sent to OPTIGA™ in multiple fragments which will be taken care by optiga_cmd. Note: The shielded connection protection level with command protection is implicitly enabled if the Shielded connection is enabled.
Signature	optiga_crypt_hmac_start (in me : optiga_crypt_t *, in type : optiga_hmac_type_t, in secret : uint16_t, in input_data : const uint8_t *, in input_data_length : uint32_t) : optiga_lib_status_t
Parameters	`in me : optiga_crypt_t *` Pointer to one instance of optiga_crypt
	`in type : optiga_hmac_type_t` Type of scheme to be used (e.g. HMAC-SHA256, HMAC-SHA384). Refer optiga_hmac_type_t for the supported schemes.
	`in secret : uint16_t` OID of input secret (shared secret from a data object or a session (OPTIGA_KEY_ID_SESSION_BASED) already acquired by the respective optiga_crypt instance).
	`in input_data : const uint8_t *` Pointer to input data.
	`in input_data_length : uint32_t` Length of input_data provided. This value must be greater than 0.

3.1.1.33 optiga_crypt_hmac_update

optiga_crypt_hmac_update	
Description	This operation performs hmac operation for the data provided using OPTIGA™. • This operation internally uses the same strict sequence acquired internally (as part of optiga_cmd) by optiga_crypt_hmac_start. Hence before invoking this operation, the successful completion of optiga_crypt_hmac_start of is must. • If the length input_data can't be sent to OPTIGA™ in one transaction, then input_data will be sent to OPTIGA™ in multiple fragments which will be taken care by optiga_cmd. Note: The shielded connection protection level with command protection is implicitly

optiga_crypt_hmac_update	
	enabled if the Shielded connection is enabled.
Signature	optiga_crypt_hmac_update (in me : optiga_crypt_t *, in input_data : const uint8_t *, in input_data_length : uint32_t) : optiga_lib_status_t
Parameters	`in me : optiga_crypt_t *` Pointer to one instance of optiga_crypt
	`in input_data : const uint8_t *` Pointer to input data.
	`in input_data_length : uint32_t` Length of input_data provided. This value must be greater than 0.

3.1.1.34 optiga_crypt_hmac_finalize

optiga_crypt_hmac_finalize	
Description	This operation performs hmac operation for the data provided using OPTIGA™. • This operation internally uses the same strict sequence acquired (as part of optiga_cmd) by optiga_crypt_hmac_start. Hence before invoking this operation, the successful completion of optiga_crypt_hmac_start is must. • If the length of input_data can't be sent to OPTIGA™ in one transaction, then input_data will be sent to OPTIGA™ in multiple fragments which will be taken care by optiga_cmd. Note: The shielded connection protection level with command protection is implicitly enabled if the Shielded connection is enabled.
Signature	optiga_crypt_hmac_finalize (in me : optiga_crypt_t *, in input_data : const uint8_t *, in input_data_length : uint32_t, inout mac : uint8_t *, inout mac_length : uint32_t *) : optiga_lib_status_t
Parameters	`in me : optiga_crypt_t *` Pointer to one instance of optiga_crypt
	`in input_data : const uint8_t *` Pointer to input data.
	`in input_data_length : uint32_t` Length of input_data provided. This value must be greater than 0.
	`inout mac : uint8_t *` Pointer to the buffer to store the generated MAC.
	`inout mac_length : uint32_t *` Pointer to the length of the mac buffer. This will be updated with the actual length of mac returned by OPTIGA™. If the mac_length is lesser than the length of MAC received from OPTIGA™, then the first mac_length bytes are only returned.

3.1.1.35 optiga_crypt_hmac_verify

optiga_crypt_hmac_verify	
Description	This operation performs hmac verification for the provided authorization value using OPTIGA™. • This operation uses the session already acquired to store the authentication code (optiga_crypt_generate_auth_code). • The size of input_data is based on the respective hash algorithm used in the hmac scheme (type) (630 bytes - hmac_length). • The acquired session gets released after completion of this operation (irrespective of status of the operation once after the command is sent to OPTIGA™). Upon successful verification of provided hmac, the achieved AUTO state using the respective secret is maintained by OPTIGA™. The achieved state can be cleared by invoking optiga_crypt_clear_auto_state operation. Note: The shielded connection protection level with response protection is implicitly enabled if the shielded connection is enabled.
Signature	optiga_crypt_hmac_verify (in me : optiga_crypt_t *, in type : optiga_hmac_type_t, in secret : uint16_t, in input_data : const uint8_t *, in input_data_length : uint32_t, in hmac : const uint8_t *, in hmac_length : uint32_t)
Parameters	`in me : optiga_crypt_t *` Pointer to one instance of optiga_crypt
	`in type : optiga_hmac_type_t` Type of scheme to be used (e.g. HMAC-SHA256, HMAC-SHA384). Refer optiga_hmac_type_t for the supported schemes.
	`in secret : uint16_t` OPTIGA™ data object ID of input secret, which must be an Authorization Reference (data object type = AUTOREF).
	`in input_data : const uint8_t *` Pointer to input data. • The input data must be (optional_data || random || arbitrary data). • The optional_data is the input provided and random data which is returned in the respsective optiga_crypt_generate_auth_code operation.
	`in input_data_length : uint32_t` Length of input_data provided. This value must be greater than 0.
	`in hmac : const uint8_t *` Pointer to the buffer to provide the HMAC (generated at host) which gets verified at OPTIGA™.
	`in hmac_length : uint32_t` Length of hmac provided. This value must be the size of output defined by respective hash algorithm used in hmac scheme (type).

3.1.1.36 optiga_crypt_clear_auto_state

optiga_crypt_clear_auto_state	
Description	This operation clears the achieved authorization (AUTO) state (using optiga_crypt_hmac_verify operation) at OPTIGA™. If the session is already acquired (for example, after the generation of authentication code), • The acquired session is referred as part of APDU preparation. • The acquired session gets released after completion of this operation (irrespective of status of the operation once after the command is sent to OPTIGA™). Note: The shielded connection protection level with response protection is implicitly enabled if the shielded connection is enabled.
Signature	optiga_crypt_clear_auto_state (in me : optiga_crypt_t *, in secret : uint16_t)
Parameters	`in me : optiga_crypt_t *` Pointer to one instance of optiga_crypt
	`in secret : uint16_t` OPTIGA™ data object ID of input secret (AUTOREF type), which was used to acheive the AUTO state.

3.1.1.37 optiga_crypt_tls_prf

optiga_crypt_tls_prf	
Description	This operation derives shared secret or key using OPTIGA™. OPTIGA™ performs key derivation operation as per type chosen 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 this case, The shielded connection protection level with command protection is implicitly enabled if the Shielded connection is enabled.
Signature	optiga_crypt_tls_prf (in me : optiga_crypt_t *, in type : optiga_tls_prf_type_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 : optiga_crypt_t *` Pointer to one instance of optiga_crypt
	`in type : optiga_tls_prf_type_t` Type of scheme to be used (e.g. TLS v1.2 PRF SHA256). Refer optiga_tls_prf_type_t for the supported schemes.
	`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.

optiga_crypt_tls_prf	
	`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. = FALSE (0), stores the derived key in the acquired session by the instance with in OPTIGA™. = TRUE (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.

3.1.1.38 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 v1.2 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 this case, The shielded connection protection level with command protection is implicitly enabled if the Shielded connection is enabled.
Signature	optiga_crypt_tls_prf_sha256 (in me : 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 : 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 *`

Enabler APIs

optiga_crypt_tls_prf_sha256	
	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. = FALSE (0), stores the derived key in the acquired session by the instance with in OPTIGA™. = TRUE (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.

3.1.1.39 optiga_crypt_tls_prf_sha384

optiga_crypt_tls_prf_sha384	
Description	This operation derives shared secret or key using OPTIGA™. OPTIGA™ performs PRF SHA384 (as specified in TLS v1.2 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 this case, The shielded connection protection level with command protection is implicitly enabled if the Shielded connection is enabled.
Signature	optiga_crypt_tls_prf_sha384 (in me : 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 : 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

optiga_crypt_tls_prf_sha384	
	`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. = FALSE (0), stores the derived key in the acquired session by the instance with in OPTIGA™. = TRUE (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.

3.1.1.40 optiga_crypt_tls_prf_sha512

optiga_crypt_tls_prf_sha512	
Description	This operation derives shared secret or key using OPTIGA™. OPTIGA™ performs PRF SHA512 (as specified in TLS v1.2 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 this case, The shielded connection protection level with command protection is implicitly enabled if the Shielded connection is enabled.
Signature	optiga_crypt_tls_prf_sha512 (in me : 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 : 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

optiga_crypt_tls_prf_sha512	
	`in export_to_host : bool_t` Specifies whether the derived key/secret to be exported to the host or not. = FALSE (0), stores the derived key in the acquired session by the instance with in OPTIGA™. = TRUE (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.

3.1.1.41 optiga_crypt_hkdf

optiga_crypt_hkdf	
Description	This operation derives shared secret or key using OPTIGA™. The OPTIGA™ performs HKDF (as specified in [RFC5869]) operation using the referred data object ID or session ID, which holds a secret. Here secret is input OID, which holds the input pre-shared secret. secret = 0x0000, the secret from the acquired session of the respective optiga_crypt instance is used to derive the new shared secret. In this case, The shielded connection protection level with command protection is implicitly enabled if the Shielded connection is enabled.
Signature	optiga_crypt_hkdf (in me : optiga_crypt_t *, in type : optiga_hkdf_type_t, in secret : uint16_t, in salt : const uint8_t *, in salt_length : uint16_t, in info : const uint8_t *, in info_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 : optiga_crypt_t *` Pointer to one instance of optiga_crypt
	`in type : optiga_hkdf_type_t` Type of scheme to be used (e.g. HKDF-SHA256, HKDF-SHA384). Refer optiga_hkdf_type_t for the supported schemes.
	`in secret : uint16_t` OID of input secret. Could be a secret from a data object or acquired session.
	`in salt : const uint8_t *` Pointer to salt (non-secret random) value. This is an optional parameter. If not required to provide, either salt can be NULL or salt_length can be zero.
	`in salt_length : uint16_t` Length of salt.
	`in info : const uint8_t *` Pointer to application specific information. This is an optional parameter. If not required to provide, either info can be NULL or info_length can be zero.

Enabler APIs

optiga_crypt_hkdf	
	`in info_length : uint16_t` Length of info.
	`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. = FALSE (0), stores the derived key in the acquired session by the instance with in OPTIGA™. = TRUE (1) or non-zero, OPTIGA™ exports the derived key. In this case, the derived_key is a pointer 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.

3.1.1.42 optiga_crypt_hkdf_sha256

optiga_crypt_hkdf_sha256	
Description	This operation derives shared secret or key using OPTIGA™. The OPTIGA™ performs HKDF with SHA256 (as specified in [RFC5869]) operation using the referred data object ID or session ID, which holds a secret. Here secret is input OID, which holds the input pre-shared secret. secret = 0x0000, the secret from the acquired session of the respective optiga_crypt instance is used to derive the new shared secret. In this case, The shielded connection protection level with command protection is implicitly enabled if the Shielded connection is enabled.
Signature	optiga_crypt_hkdf_sha256 (in me : optiga_crypt_t *, in secret : uint16_t, in salt : const uint8_t *, in salt_length : uint16_t, in info : const uint8_t *, in info_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 : 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 salt : const uint8_t *` Pointer to salt (non-secret random) value. This is an optional parameter. If not required to provide, either salt can be NULL or salt_length can be zero.
	`in salt_length : uint16_t` Length of salt.
	`in info : const uint8_t *` Pointer to application specific information.

optiga_crypt_hkdf_sha256	
	This is an optional parameter. If not required to provide, either info can be NULL or info_length can be zero.
	`in info_length : uint16_t` Length of info.
	`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. = FALSE (0), stores the derived key in the acquired session by the instance with in OPTIGA™. = TRUE (1) or non-zero, OPTIGA™ exports the derived key. In this case, the derived_key is a pointer 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.

3.1.1.43 optiga_crypt_hkdf_sha384

optiga_crypt_hkdf_sha384	
Description	This operation derives shared secret or key using OPTIGA™. The OPTIGA™ performs HKDF with SHA384 (as specified in [RFC5869]) operation using the referred data object ID or session ID, which holds a secret. Here secret is input OID, which holds the input pre-shared secret. secret = 0x0000, the secret from the acquired session of the respective optiga_crypt instance is used to derive the new shared secret. In this case, The shielded connection protection level with command protection is implicitly enabled if the Shielded connection is enabled.
Signature	optiga_crypt_hkdf_sha384 (in me : optiga_crypt_t *, in secret : uint16_t, in salt : const uint8_t *, in salt_length : uint16_t, in info : const uint8_t *, in info_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 : 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 salt : const uint8_t *` Pointer to salt (non-secret random) value. This is an optional parameter. If not required to provide, either salt can be NULL or salt_length can be zero.
	`in salt_length : uint16_t` Length of salt.

Enabler APIs

optiga_crypt_hkdf_sha384	
	`in info : const uint8_t *` Pointer to application specific information. This is an optional parameter. If not required to provide, either info can be NULL or info_length can be zero.
	`in info_length : uint16_t` Length of info.
	`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. = FALSE (0), stores the derived key in the acquired session by the instance with in OPTIGA™. = TRUE (1) or non-zero, OPTIGA™ exports the derived key. In this case, the derived_key is a pointer 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.

3.1.1.44 optiga_crypt_hkdf_sha512

optiga_crypt_hkdf_sha512	
Description	This operation derives shared secret or key using OPTIGA™. The OPTIGA™ performs HKDF with SHA512 (as specified in [RFC5869]) operation using the referred data object ID or session ID, which holds a secret. Here secret is input OID, which holds the input pre-shared secret. secret = 0x0000, the secret from the acquired session of the respective optiga_crypt instance is used to derive the new shared secret. In this case, The shielded connection protection level with command protection is implicitly enabled if the Shielded connection is enabled.
Signature	optiga_crypt_hkdf_sha512 (in me : optiga_crypt_t *, in secret : uint16_t, in salt : const uint8_t *, in salt_length : uint16_t, in info : const uint8_t *, in info_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 : 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 salt : const uint8_t *` Pointer to salt (non-secret random) value. This is an optional parameter. If not required to provide, either salt can be NULL or salt_length can be zero.

Enabler APIs

optiga_crypt_hkdf_sha512	
	`in salt_length : uint16_t` Length of salt.
	`in info : const uint8_t *` Pointer to application specific information. This is an optional parameter. If not required to provide, either info can be NULL or info_length can be zero.
	`in info_length : uint16_t` Length of info.
	`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. = FALSE (0), stores the derived key in the acquired session by the instance with in OPTIGA™. = TRUE (1) or non-zero, OPTIGA™ exports the derived key. In this case, the derived_key is a pointer 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.

3.1.1.45 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, etc.
Signature	optiga_crypt_set_comms_params (in me : optiga_crypt_t *, in parameter_type : uint8_t, in value : uint8_t) : void
Parameters	`in me : 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_COMMS_PROTECTION_LEVEL
	`in value : uint8_t` Value to be configured to the respective parameter.

3.1.1.46 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 : optiga_crypt_t *, in protocol_version) : void
Parameters	`in me : optiga_crypt_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

3.1.1.47 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 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 : optiga_crypt_t *, in protection_level) : void
Parameters	`in me : optiga_crypt_t *` Pointer to one instance of optiga_crypt.

OPTIGA_CRYPT_SET_COMMS_PROTECTION_LEVEL	
	`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.

3.1.1.48 optiga_crypt_destroy

optiga_crypt_destroy	
Description	This operation destructs an instance of optiga_crypt. The volatile memory is 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 : optiga_crypt_t *) : optiga_lib_status_t
Parameters	`in me : 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 OPTIGA™ (open/close) and data/key objects with the following characteristics:

- Multiple instances could be created to allow concurrent access to other services.
- Uses 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™, • The Shielded connection version and protection level gets set to the default version (OPTIGA_COMMS_PROTOCOL_VERSION_PRE_SHARED_SECRET) supported and default level (OPTIGA_COMMS_DEFAULT_PROTECTION_LEVEL) in the instance created as part of this operation. • This instance needs to be updated for the required protection level, before invoking

optiga_util_create	
	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 are 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 *) : 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 : optiga_util_t *, in perform_restore : bool_t) : optiga_lib_status_t
Parameters	`in me : 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 (command 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 : optiga_util_t *, in perform_hibernate : bool_t) : optiga_lib_status_t
Parameters	`in me : 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 : 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 : 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 : optiga_util_t *, in optiga_oid : uint16_t, inout buffer : uint8_t *, inout length : uint16_t *) : optiga_lib_status_t
Parameters	`in me : 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 : 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 : 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 : optiga_util_t *, in optiga_oid : uint16_t, in buffer : const uint8_t *, in length : uint8_t) : optiga_lib_status_t
Parameters	`in me : 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 the metadata to be written
	`in length : uint8_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 : optiga_util_t *, in optiga_counter_oid : uint16_t, in count : uint8_t) : optiga_lib_status_t
Parameters	`in me : 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.

optiga_util_protected_update_start	
	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 strict 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_final completed or the protected update sequence failed with other operations. • If used for any other operation, the strict sequence acquired will be internally released automatically and the new operation will be scheduled as usual. The protected update operation has to be restarted from beginning to complete the protected update. • The other operations (optiga_util, optiga_crypt) invoked with different instances during the protected update sequence, 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 : 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 : 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 a 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_continue	
	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 : optiga_util_t *, in fragment : const uint8_t *, in fragment_length : uint16_t) : optiga_lib_status_t
Parameters	`in me : optiga_util_t *` Pointer to one instance of optiga_util.
	`in fragment : const uint8_t *` Pointer to a 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 : optiga_util_t *, in fragment : const uint8_t *, in fragment_length : uint16_t) : optiga_lib_status_t
Parameters	`in me : optiga_util_t *` Pointer to one instance of optiga_util.
	`in fragment : const uint8_t *` Pointer to a 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, etc.

Enabler APIs

optiga_util_set_comms_params	
Signature	optiga_util_set_comms_params (in me : optiga_util_t *, in parameter_type : uint8_t, in value : uint8_t) : void
Parameters	`in me : 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 : optiga_util_t *, in protocol_version) : void
Parameters	`in me : 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_UTIL_SET_COMMS_PROTECTION_LEVEL	
	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 : optiga_util_t *, in protection_level) : void
Parameters	`in me : optiga_util_t *` Pointer to one instance of optiga_util.
	`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.

3.1.2.15 optiga_util_destroy

optiga_util_destroy	
Description	This operation destructs an instance of optiga_util. The volatile memory is 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 : optiga_util_t *) : optiga_lib_status_t
Parameters	`in me : optiga_util_t *` Pointer to one instance of optiga_util.

3.2 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.

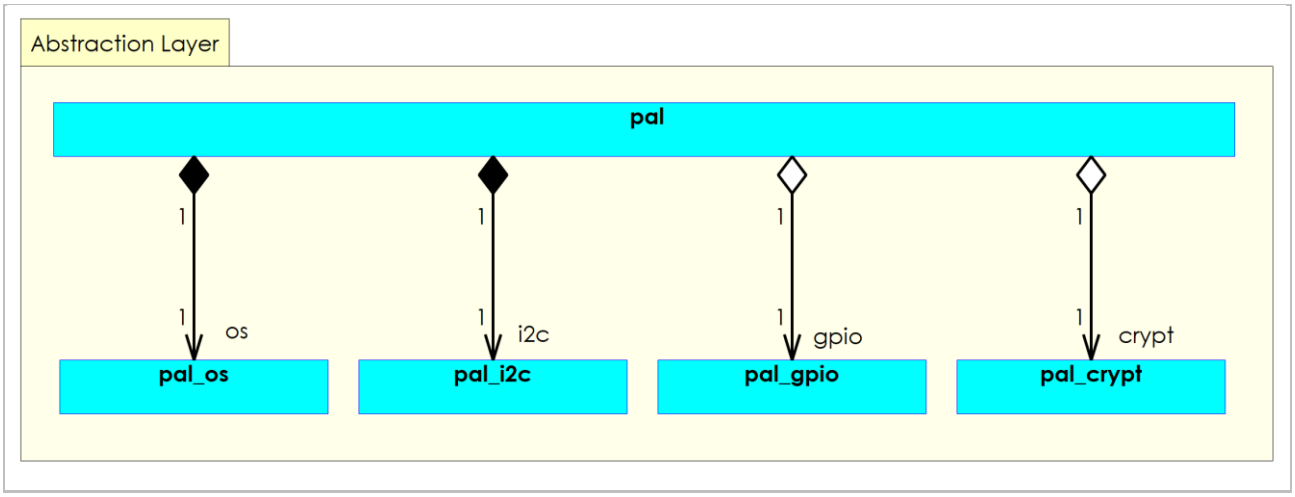


Figure 23 - Abstraction Layer Decomposition

3.2.1 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.

3.2.1.1 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

3.2.1.2 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

3.2.2 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.

3.2.2.1 pal_crypt_tls_prf_sha256

pal_crypt_tls_prf_sha256	
Description	This operation derives the secret using the TLSv1.2 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.
	`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.

3.2.2.2 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

pal_crypt_encrypt_aes128_ccm	
	`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

3.2.2.3 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`

pal_crypt_decrypt_aes128_ccm	
	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

3.2.3 pal_gpio

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

- Power on/off
- HW Reset on/off

3.2.3.1 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.

3.2.3.2 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

3.2.3.3 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

3.2.3.4 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

3.2.4 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 is 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).

3.2.4.1 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

3.2.4.2 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

3.2.4.3 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.

pal_i2c_read	
	`inout p_data : uint8_t *` Pointer to a buffer to store the read data.
	`in length : uint16_t` Length of data to be read.

3.2.4.4 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.

3.2.4.5 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` Bitrate to be used by I2C master in KHz.

3.2.5 pal_os

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

3.2.5.1 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

pal_os_datastore_read	
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.

3.2.5.2 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

3.2.5.3 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

3.2.5.4 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`

pal_os_event_register_callback_oneshot	
	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.

3.2.5.5 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

3.2.5.6 pal_os_event_start

pal_os_event_start	
Description	This operation starts the event management operation.
Signature	pal_os_event_start (in p_pal_os_event : pal_os_event_t *, in callback : register_callback, in callback_args : void *) : void
Parameters	`in p_pal_os_event : pal_os_event_t *`
	`in callback : register_callback`
	`in callback_args : void *`

3.2.5.7 pal_os_event_stop

pal_os_event_stop	
Description	This operation stops the event management operation.
Signature	pal_os_event_stop (in p_pal_os_event : pal_os_event_t *) : void
Parameters	`in p_pal_os_event : pal_os_event_t *`

3.2.5.8 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)

3.2.5.9 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

3.2.5.10 pal_os_timer_get_time_in_milliseconds

pal_os_timer_get_time_in_milliseconds	
Description	This operation provides the current time stamp in milliseconds.
Signature	pal_os_timer_get_time_in_milliseconds () : uint32_t

3.2.5.11 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

3.2.5.12 pal_os_timer_delay_in_milliseconds

pal_os_timer_delay_in_milliseconds	
Description	This operation induces a delay of provided milliseconds.
Signature	pal_os_timer_delay_in_milliseconds (in milliseconds : uint16_t) : void
Parameters	`in milliseconds : uint16_t`

3.2.5.13 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

3.2.5.14 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

3.2.5.15 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

3.2.5.16 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

3.2.5.17 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

3.2.5.18 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

3.2.5.19 pal_os_memcpy

pal_os_memcpy	
Description	This operation copies the number of bytes (size) from p_source to p_destination.
Signature	pal_os_memcpy (inout p_destination : void *, in p_source : const void *, in size : uint32_t) : void *
Parameters	`inout p_destination : void *`

Enabler APIs

pal_os_memcpy	
	Pointer to the destination buffer.
	`in p_source : const void *` Pointer to the source buffer.
	`in size : uint32_t` Number of bytes to be copied.

3.2.5.20 pal_os_memset

pal_os_memset	
Description	This operation copies the first number of bytes (size) of p_buffer with the value (byte) specified.
Signature	pal_os_memset (inout p_buffer : void *, in value : uint32_t, in size : uint32_t) : void *
Parameters	`inout p_buffer : void *`
	`in value : uint32_t` Value (byte) to be replaced in the memory.
	`in size : uint32_t` Number of bytes of memory to be updated with value.

3.3 Data Types

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

3.3.1 Enumerations

3.3.1.1 optiga_ecc_curve_t

Types of ECC Curves supported by OPTIGA™

Name	Description
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_ECC_CURVE_NIST_P_521	Curve type - ECC NIST P-521
OPTIGA_ECC_CURVE_BRAIN_POOL_P_256R1	Curve type - ECC Brainpool P256r1
OPTIGA_ECC_CURVE_BRAIN_POOL_P_384R1	Curve type - ECC Brainpool P384r1
OPTIGA_ECC_CURVE_BRAIN_POOL_P_512R1	Curve type - ECC Brainpool P512r1

Note: OPTIGA™ Trust M Version 1 doesn't support Brainpool and ECC NIST P 521 curves.

3.3.1.2 optiga_hash_context_length_t

Hash context length/size while using OPTIGA™ for digest generation.

Name	Description
OPTIGA_HASH_CONTEXT_LENGTH_SHA_256	Hash context length (in bytes) in case of SHA256.

3.3.1.3 optiga_hash_type_t

Types of digest/hash generation supported by OPTIGA™

Name	Description
OPTIGA_HASH_TYPE_SHA_256	Generate digest using SHA256

3.3.1.4 optiga_hkdf_type_t

Types of key derivation based on HKDF supported by OPTIGA™.

Name	Description
OPTIGA_HKDF_SHA_256	Key derivation using HKDF-SHA256 [RFC5869].
OPTIGA_HKDF_SHA_384	Key derivation using HKDF-SHA384 [RFC5869].
OPTIGA_HKDF_SHA_512	Key derivation using HKDF-SHA384 [RFC5869].

Note: OPTIGA™ Trust M Version 1 doesn't support HKDF.

3.3.1.5 optiga_hmac_type_t

Types of hmac generation supported by OPTIGA™.

Name	Description
OPTIGA_HMAC_SHA_256	Generate MAC using HMAC-SHA256 [RFC2104].
OPTIGA_HMAC_SHA_384	Generate MAC using HMAC-SHA384 [RFC2104].
OPTIGA_HMAC_SHA_512	Generate MAC using HMAC-SHA512 [RFC2104].

Note: OPTIGA™ Trust M Version 1 doesn't support HMAC.

3.3.1.6 optiga_key_id_t

Key slot IDs in OPTIGA™

Name	Description
OPTIGA_KEY_ID_E0F0	Key from key store (non-volatile). Supports only ECC (optiga_ecc_curve_t).
OPTIGA_KEY_ID_E0F1	Key from key store (non-volatile). Supports only ECC (optiga_ecc_curve_t).
OPTIGA_KEY_ID_E0F2	Key from key store (non-volatile). Supports only ECC

	(optiga_ecc_curve_t).
OPTIGA_KEY_ID_E0F3	Key from key store (non-volatile). Supports only ECC (optiga_ecc_curve_t).
OPTIGA_KEY_ID_E0FC	Key from key store (non-volatile). Supports only RSA (optiga_rsa_key_type_t).
OPTIGA_KEY_ID_E0FD	Key from key store (non-volatile). Supports only RSA (optiga_rsa_key_type_t).
OPTIGA_KEY_ID_E200	Key from key store (non-volatile). Supports AES 128/192/256 key types (optiga_symmetric_key_type_t).
OPTIGA_KEY_ID_SESSION_BASED	Key from session context (volatile).

Note: OPTIGA™ Trust M Version 1 doesn't support Symmetric Key (OPTIGA_KEY_ID_E200).

3.3.1.7 optiga_key_usage_t

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).

Name	Description
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 encrypt and decrypt operations. This type is applicable for RSA or AES 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 (for example, ECDH operations)

3.3.1.8 optiga_rng_type_t

Types of random number generation supported by OPTIGA™

Name	Description
OPTIGA_RNG_TYPE_TRNG	Generate Random number using TRNG
OPTIGA_RNG_TYPE_DRNG	Generate Random number using DRNG

3.3.1.9 optiga_rsa_encryption_scheme_t

RSA Encryption schemes supported by OPTIGA™ for encryption and decryption.

Enabler APIs

Name	Description
OPTIGA_RSAES_PKCS1_V15	Encryption scheme - RSAES PKCS1-v1_5

3.3.1.10 optiga_rsa_key_type_t

Types of RSA keys supported by OPTIGA™

Name	Description
OPTIGA_RSA_KEY_1024_BIT_EXPONENTIAL	RSA Key type - 1024 Bit exponential
OPTIGA_RSA_KEY_2048_BIT_EXPONENTIAL	RSA Key type - 2048 Bit exponential

3.3.1.11 optiga_rsa_signature_scheme_t

RSA Signature schemes supported by OPTIGA™ for sign and verify

Name	Description
OPTIGA_RSASSA_PKCS1_V15_SHA256	Signature scheme - RSA SSA PKCS1-v1_5 with SHA256 digest [w/o hash operation]
OPTIGA_RSASSA_PKCS1_V15_SHA384	Signature scheme - RSA SSA PKCS1-v1_5 with SHA384 digest [w/o hash operation]
OPTIGA_RSASSA_PKCS1_V15_SHA512	Signature scheme - RSA SSA PKCS1-v1_5 with SHA512 digest [w/o hash operation]

Note: OPTIGA™ Trust M Version 1 doesn't support RSA SSA PKCS#1 v1.5 SHA512.

3.3.1.12 optiga_symmetric_encryption_mode_t

Symmetric Encryption schemes supported by OPTIGA™ for encryption and decryption.

Name	Description
OPTIGA_SYMMETRIC_ECB	Symmetric Encryption mode - ECB mode as specified by [SP 800-38A].
OPTIGA_SYMMETRIC_CBC	Symmetric Encryption mode - CBC as specified by [SP 800-38A].
OPTIGA_SYMMETRIC_CBC_MAC	Symmetric Encryption mode - CBC MAC (MAC generation) as specified by [ISO 9797-1] [MAC Algorithm 1]
OPTIGA_SYMMETRIC_CMAC	Symmetric Encryption mode - CMAC (MAC generation) as specified by [SP 800-38B]

Note: OPTIGA™ Trust M Version 1 doesn't support above specified symmetric encryption and MAC algorithms.

3.3.1.13 optiga_symmetric_key_type_t

Types of symmetric keys supported by OPTIGA™

Name	Description
OPTIGA_SYMMETRIC_AES_128	AES 128
OPTIGA_SYMMETRIC_AES_192	AES 192
OPTIGA_SYMMETRIC_AES_256	AES 256

Note: OPTIGA™ Trust M Version 1 doesn't support above specified symmetric keys.

3.3.1.14 optiga_tls_prf_type_t

Types of key derivation based on TLSv1.2 PRF supported by OPTIGA™.

Name	Description
OPTIGA_TLS12_PRF_SHA_256	Key derivation using TLSv1.2 PRF-SHA256 [RFC5246].
OPTIGA_TLS12_PRF_SHA_384	Key derivation using TLSv1.2 [RFC5246] PRF-SHA384.
OPTIGA_TLS12_PRF_SHA_512	Key derivation using TLSv1.2 [RFC5246] PRF-SHA512.

Note: OPTIGA™ Trust M Version 1 doesn't support PRF with SHA384 and SHA512.

3.3.2 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 : uint16_t [1]` Length of context buffer. Refer optiga_hash_context_length_t for the respective hash_algo.
	`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

Enabler APIs

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

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 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.

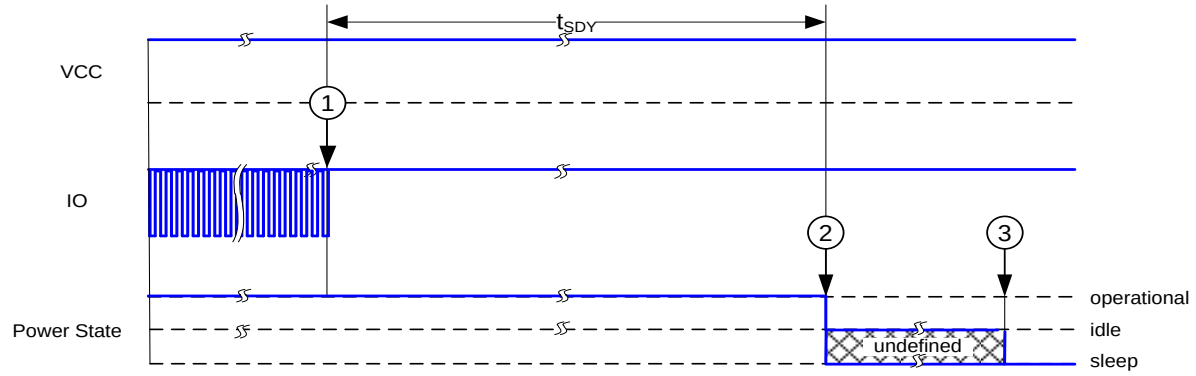


Figure 24 - Go-to-Sleep diagram

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.
The used ISO/OSI layers are Physical, Data Link, Network, Transport, Presentation and Application layer

as the figure below depicts.

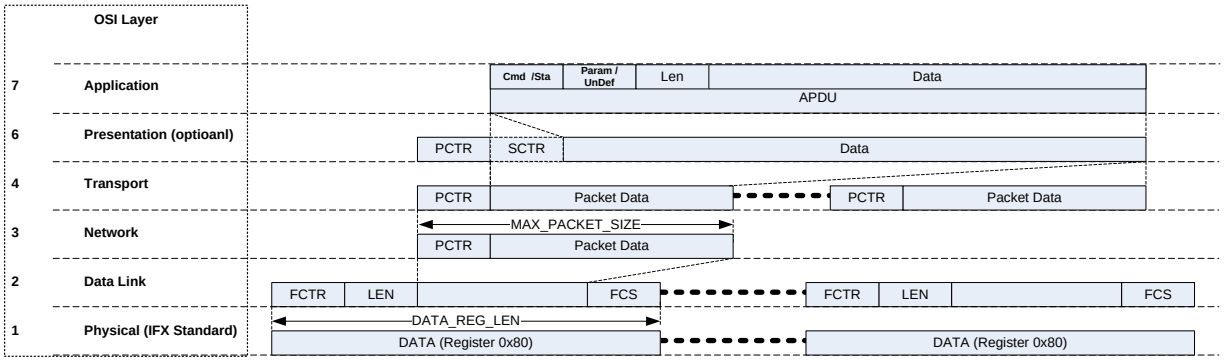


Figure 25 - Overview protocol stack used

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 4 - 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 key or data object or metadata of a key or data object protected.
0x0C or 0x8C	GetRandom	Command to generate a random stream
0x14 or 0x94	EncryptSym	Command to encrypt data based on a symmetric key scheme
0x15 or 0x95	DecryptSym	Command to decrypt data based on a symmetric key scheme
0x1E or 0x9E	EncryptAsym	Command to encrypt data based on an asymmetric key scheme
0x1F or 0x9F	DecryptAsym	Command to decrypt data 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
0x39 or 0xB9	GenSymKey	Command to generate symmetric (secret) keys

[2] The default setting is 400 KHz

OPTIGA™ Trust M External Interface

Command Code	Name	Short description
0x70 or 0xF0	OpenApplication	Command to launch an application
0x71 or 0xF1	CloseApplication	Command to close/terminate an application

Table 5 - Command Codes

Note: OPTIGA™ Trust M Version 1 doesn't support EncryptSym, DecryptSym, and GenSymKey commands.

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

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 6 - 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.

[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

OPTIGA™ Trust M External Interface

Response Status Codes

Table 'Response Status Codes' lists the status codes provided by a 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 7 - 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 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.
Invalid/Wrong Payload Version	0x10	The Payload Version provided in the Manifest is

Field	Code	Description
		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 Metadata of the Key/Data object	0x11	A command is acting on metadata for key or data objects and the current metadata are invalid.
Unsupported extension/ identifier	0x24	• An unsupported extension found in the message • Unsupported key usage / Algorithm extension/identifier for the usage of Private key
Unsupported Parameters	0x25	• At least one parameter received in the handshake message is not supported. • Unsupported Parameter in the command APDU InData.
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	0x2A	• The size of the certificate is more than the 1300 bytes where OPTIGA™ can't parse the certificate internally due to insufficient memory. (or) • At least one cryptographic algorithm specified in the certificate is not supported (e.g. hash or sign algorithms).
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
Authorization Failure	0x2F	Session random comparison failure or HMAC verification failure.

Table 8 - 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 is 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 9 - 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.
The CloseApplication also writes the current SEC maintained in RAM (SEC_{CURR}) to Security Event Counter (SEC), if SEC_{CURR} is not same as in Security Event Counter (SEC).

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, closes 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 10 - CloseApplication Coding

[5] In case of 0xF1 the Last Error Code gets flushed
[6] Saving the context is only possible when the current Security Event Counter (SEC) value is zero, otherwise it returns an error "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 11 - 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 12 - 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). 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 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 or metadata objects protected (integrity and optionally confidentiality) 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:

- *OPTIGA™ Trust M Version 1 doesn't support confidentiality protection and as well doesn't support updating keys and metadata.*
- *This command supports chaining (start, continue, finalize) through partial write.*
- *This command does not support the data objects specified below.*
  - *Life Cycle Status (Global/Application)*
  - *Security Status (Global/Application)*
  - *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 fragment of update data set[16] final => last fragment 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

[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

OPTIGA™ Trust M External Interface

Field	Offset [direct]	Description
OutData	4 [out]	Absent

Table 13 - SetObjectProtected Coding

4.4.1.6 GetRandom

The GetRandom command is used to generate a random stream to be used by various security schemes. The generated random stream is either returned and/or gets stored in the addressed session context (OID) if specified. The field "Param" contains the type of random stream (0x00 or 0x01).
Note: OPTIGA™ Trust M Version 1 doesn't support storing random in session when Param is TRNG or DRNG.

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 **0xE100-0xE103** OID of session Context • [optional] In case of Param **= 0x00-0x01** • [mandatory] In case of Param **= 0x04** **00x41**, Length, prepending optional data[20] [21] • [mandatory] in case session OID is present
Sta	0 [out]	**0x00 | 0xFF** Response Status Code
UnDef	1 [out]	**0x00-0xFF** Undefined Value
OutLen	2 [out]	**0x0000-0xFFFF** Length of OutData
OutData	4 [out]	**0x00-0xFF** Random stream. • In case of Param = **0x00-0x01**, the random stream gets returned. *Note: absent in case of Param = 0x04 (OutLen = 0x0000).*

Table 14 - 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 length plus the requested length of the random value shall not exceed 66 bytes
[21] Length could be 0x0000

4.4.1.7 EncryptSym

The EncryptSym is used to protect data by the OPTIGA™, based on a secret key scheme or to calculate a MAC over provided data. Those data and their sequence of exchange between the OPTIGA™ and the connected host are defined in detail in Chapter "Use Cases Analysis". The data padding must be provided for the last block of data in case the used algorithm (refer to Symmetric Modes of Operation) doesn't define default padding.

Notes:

1. *OPTIGA™ Trust M Version 1 doesn't support this command.*
2. *In case the presentation layer protection is enabled the command must be protected. The response protection is up to the caller.*
3. *In case of applied chaining, the sequence from start to final must be strict (no other command in between).*

Field	Offset [direct]	Description
Cmd	0 [in]	**0x14 | 0x94**[22] Command Code
Param	1 [in]	**0xXX** cipher suite as specified by table Symmetric Modes of Operation, which define whether algorithm block aligned and padding must be provided.
InLen	2 [in]	**0xXXXX** Length of InData [23]
InData	4 [in]	• Secret Key OID[24] *Note: The Secret Key OID can be any of the below:* *1. Key object OID in case of AES based operations,* *2. In case of hash based operations, any pre-shared secret,* *i. Data object of type PRESSEC* *ii. Session OID (post ECDH or derive key command)* • 0x0y, Length[25] [26] [27] [28], data (start y = 0, start&final y = 1, continue y = 2 and final y=3), Data to be encrypted or hashed. (below is optional depending on the applied cipher suite) • 0x40, 0xXXXX, 0x00-0xFF[29], Additional Data • 0x41, 0xXXXX, 0x00-0xFF[30], IV • 0x42, 0x0002, 0x00-0xFF[31], over all payload length without padding

[22] In case of 0x94 the Last Error Code gets flushed
[23] Maximum length of InData is 640 bytes.
[24] [for continue and final] The provided OID gets ignored
[25] Length must be > 0
[26] [for start and continue] For block ciphers (e.g. AES) length must be algorithm block size aligned
[27] [for start&final and final] For block ciphers (e.g. AES) length must be algorithm block size aligned and padded in case the "Modes of operation" doesn't define default padding
[28] [for start, start&final, continue, final] For hash based modes (e.g. HMAC) length need not be algorithm block size aligned. Just byte alignment is sufficient
[29] might be present at start and start&final with CCM mode
[30] might be present at start and start&final with CBC/CCM mode

OPTIGA™ Trust M External Interface

Field	Offset [direct]	Description
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 Encrypted data

Table 15 - EncryptSym Coding

[31] present at start and start&final with CCM mode

4.4.1.8 DecryptSym

The DecryptSym is used to unprotect data by the OPTIGA™ or to verify a provided verification value, based on a secret key scheme. Those data and their sequence of exchange between the OPTIGA™ and the connected host are defined in detail in Chapter "Use Cases Analysis".

Notes:

1. *OPTIGA™ Trust M Version 1 doesn't support this command.*
2. *In case the presentation layer protection is enabled the response must be protected. The command protection is up to the caller.*
3. *In case of applied chaining, the sequence from start to final must be strict (no other command in between).*
4. *In case of verification failure, at any execution state of the command, the Auto(X) state for the respective data object gets cleared.*

Field	Offset [direct]	Description
Cmd	0 [in]	**0x15 | 0x95**[32] Command Code
Param	1 [in]	**0xXX** cipher suite as specified by table Symmetric Modes of Operation[33].
InLen	2 [in]	**0xXXXX** Length of InData [34]
InData	4 [in]	• Secret Key OID[35] *Note: The Secret Key OID can be any of the below:* *1. Key object OID in case of AES based operations,* *2. In case of hash based operations (hmac verification),* data object of type authorization reference (AUTOREF) • 0x0y, Length[36] [37] [38] [39], data (start y = 0, start&final y = 1, continue y = 2 and final y=3), Data to be decrypted/verified[40]. (below are optional depending on the applied cypher suite) • 0x40, 0xXXXX, 0x00-0xFF[41], Additional Data • 0x41, 0xXXXX, 0x00-0xFF[42], IV • 0x42, 0x0002, 0x00-0xFF[43], over all payload length

[32] In case of 0x95 the Last Error Code gets flushed
[33] Not applicable for MAC modes except HMAC
[34] Maximum length of InData is 640 bytes.
[35] [for continue and final] The provided OID gets ignored
[36] Length must be > 0
[37] [for start and continue] For block ciphers (e.g. AES) length must be algorithm block size aligned
[38] [for start&final and final] For block ciphers (e.g. AES) length must be algorithm block size aligned and padded in case the "Modes of operation" doesn't define default padding
[39] [for start&final] For hash based modes (e.g. HMAC) length need not be algorithm block size aligned. Just byte alignment is sufficient
[40] In case of verification the structure is (session OID || (optional data || random) stored in the respective session || arbitrary data) and only Start&Final is applicable. The random expires once used or by any verification failure avoiding replay attacks.
[41] might be present at start and start&final with CCM mode
[42] might be present at start and start&final with CBC/CCM mode

Field	Offset [direct]	Description
		• 0x43, 0xXXXX[44], 0x00-0xFF[45], verification value
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]	(absent in case of verification) • 0x61, 0xXXXX, 0x00-0xFF decrypted (plain) data[46]

Table 16 - DecryptSym Coding

[43] present at start and start&final with CCM mode
[44] The length is defined by the output size of the used hash algorithm
[45] present at start&final with HMAC mode
[46] including padding in case the "Modes of operation" doesn't define default padding.

4.4.1.9 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**[47] 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[48], 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[49] [50], **(0x05**, 0x0001, Algorithm Identifier (of the Public Key), **0x06**, Length, Public Key[51]**)}**
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 17 - EncryptAsym Coding

[47] In case of 0x9E the Last Error Code gets flushed
[48] The length of the message must be up to the key size minus the minimum size of the applied padding scheme
[49] 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)
[50] The key usage in certificate must be either KeyEncipherment (encrypting data from session or data from externa interface) or DataEncipherment(encrypting data from external interface)
[51] Public Key is encoded as two DER INTEGER (Modulus || Public Exponent) contained in a DER "BIT STRING"

4.4.1.10 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**[52] 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[53], Protected message • **0x03**, 0x0002, OID of decryption key[54] *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[55].
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 18 - DecryptAsym Coding

[52] In case of 0x9F the Last Error Code gets flushed

[53] The length of the message must match the key size

[54] The addressed decryption key shall be a RSA private key

[55] the length of the decrypted data shall not exceed 66 bytes and the usage is limited as input shared secret in DeriveKey command

4.4.1.11 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**[56] 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[57], Message data (start y = 0, start&final y = 1, continue y = 2, final y=3, final and keep intermediate hash y=5[58]) • **0x04**, 0x0000 - To terminate the hash sequence in case initialized already. • **0x1y**, 0x0006, OID[59], Offset, Length[60] (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

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

OPTIGA™ Trust M External Interface

Field	Offset [direct]	Description
UnDef	1 [out]	**0x00-0xFF** Undefined Value
OutLen	2 [out]	**0xXXXX** Length of OutData
OutData	4 [out]	Digest or intermediate hash context data 0x01, Length, Hash/Digest 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 19 - CalcHash Coding

4.4.1.12 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**[61] 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[62], Digest to be signed • **0x03**, 0x0002, OID of signature key[63] 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[64] Note: The length of the signature is derived from the applied key and signature scheme.

Table 20 - CalcSign Coding

[61] In case of 0xB1 the Last Error Code gets flushed

[62] For ECC shall be 10 bytes up to the length of the addressed signature key; RSA case: must be exactly equal to the output length of the hash algorithm used

[63] The addressed signing key shall be a private key

[64] ECC case: The signature pair (r,s) is encoded as two DER "INTEGER"; RSA case: The signature encoding is OCTET STRING

4.4.1.13 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**[65] 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[66], Digest • **0x02**, Length[67], Signature over Digest[68] • alternate one :**{** (**0x04**, 0x0002, OID of Public Key Certificate[69]), (**0x05**, 0x0001, Algorithm Identifier (of the Public Key), **0x06**, Length, Public Key[70])**}**
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 21 - VerifySign Coding

[65] In case of 0xB2 the Last Error Code gets flushed

[66] 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

[67] The length is limited to max. 520 bytes

[68] ECC case: The signature pair (r,s) is encoded as two DER "INTEGER"; RSA case: The signature encoding is OCTET STRING

[69] 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)

[70] PubKey is encoded as DER "BIT STRING"

4.4.1.14 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**[71] 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[72] 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[73] 0x02, Len, PubKey[74]**}** **{**RSA Key 0x01, Len, PrivKey[75] 0x02, Len, PubKey[76]**}]**

Table 22 - GenKeyPair Coding

[71] In case of 0xB8 the Last Error Code gets flushed

[72] Private Key can either be a non-volatile Device Private Key OR a Session Context (volatile), in which case the generated Key has to be stored in the respective Session Context and can be addressed later.

[73] PrivKey is encoded as DER "OCTET STRING"

[74] PubKey is encoded as DER "BIT STRING"

[75] PrivKey is encoded as DER "OCTET STRING"

[76] PubKey is encoded as two DER INTEGER (Modulus || Public Exponent) contained in a DER "BIT STRING"

4.4.1.15 GenSymKey

The GenSymKey is used to generate a symmetric key. The key gets stored at the provided OID of a Key or it gets returned to the caller in case no OID is provided.
Note: OPTIGA™ Trust M Version 1 doesn't support this command.

Field	Offset [direct]	Description
Cmd	0 [in]	**0x39 | 0xB9**[77] 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 Secret Key Input InData[InLen]
		• Alternative one **{**
		  • 0x01, 0x0002, OID of Secret Key[78] to be generated and stored as indicated by the OID (no Secret Key export!).
		  • 0x02, 0x0001, key usage (ref to Key Usage Identifier)
		  • 0x07, 0x0000 (export Secret Key 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]	• {Key 0x01, Len, SecKey[79]}

Table 23 - GenSymKey Coding

[77] In case of 0xB8 the Last Error Code gets flushed
[78] Secret Key can be a non-volatile Device Symmetric Key
[79] SecKey is encoded as octet string. The length is determined by the regarded algorithm (see Param XE "Param")

4.4.1.16 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**[80] 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 *Note: the key usage of the addressed key must be set to* KeyAgree*; Refer to* Key Usage Identifier. • 0x05, 0x0001, Algorithm Identifier, 0x06, Length, Public Key[81] (alternative one) • 0x07, 0x0000[82] • 0x08, 0x0002, OID of Shared Secret[83]
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 [84] Note 1: Shared secret is only returned in case it is requested by InData (0x07, 0x0000)

Table 24 - CalcSSec Coding

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

4.4.1.17 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**[85] 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(*Data object of type* PRESSEC) to derive the new secret from[86] • 0x02, Len, Secret derivation data/Salt[87] • 0x03, 0x0002, Length of the key to be derived[88] • (optional) 0x04, Len, Info[89] (alternative one) • 0x07, 0x0000[90] • 0x08, 0x0002, OID of derived key[91]
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 25 - DeriveKey Coding

[85] In case of 0xB4 the Last Error Code gets flushed
[86] The source of the shared secret could be a session context or data object. The used size of the data object must be maximum 64 bytes.
[87] In case of HKDF this tag is optional and thus the length could be 0 to 1024 byte or in all other cases 8 to 1024 byte
[88] Minimum Length = 16 byte; maximum length = 66 byte in case of session reference; maximum length = 256 byte in case of returned secret
[89] Applicable only for HKDF, max. length = 256 bytes
[90] Indicates exporting the derived key via the external interface
[91] 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™.
Note: OPTIGA™ Trust M Version 1 doesn't support ECC Brainpool, ECC NIST P 521 curves and symmetric (AES).

Value	Description
0x03	Elliptic Curve Key on NIST P256 curve.
0x04	Elliptic Curve Key on NIST P384 curve
0x05	Elliptic Curve Key on NIST P521 curve.
0x13	Elliptic Curve Key on Brainpool P256 r1 curve.
0x15	Elliptic Curve Key on Brainpool P384 r1 curve.
0x16	Elliptic Curve Key on Brainpool P512 r1 curve.
0x41	RSA Key 1024 bit exponential format
0x42	RSA Key 2048 bit exponential format
0x81	AES key with 128 bit
0x82	AES key with 192 bit
0x83	AES key with 256 bit
0xE2	SHA 256

Table 26 - 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 (Signature Calculation / Verification)
0x20	KeyAgree (Key Agreement)

Table 27 - 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 28 - 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 29 - Key Agreement Schemes

Table 'Key Derivation Method' lists the key derivation method supported by the OPTIGA™.
Note: OPTIGA™ Trust M Version 1 doesn't support HKDF and PRF with SHA384 and SHA512.

Value	Description
0x01	TLS PRF SHA256 according to [RFC5246].
0x02	TLS PRF SHA384 according to [RFC5246].
0x03	TLS PRF SHA512 according to [RFC5246].
0x08	HKDF-SHA256 according to *[RFC5869]*
0x09	HKDF-SHA384 according to *[RFC5869]*
0x0A	HKDF-SHA512 according to *[RFC5869]*

Table 30 - 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
0x03	PKCS#1v2.2 RSASSA-PKCS1-v1.5_SHA512 according to [RFC8017] w/o hash
0x11	ECDSA w/o hash

Table 31 - Signature Schemes

Table 'Symmetric Modes of Operation' lists the supported symmetric cipher modes of operation used in secret key schemes and their coding.
Note: OPTIGA™ Trust M Version 1 doesn't support symmetric operations specified in the below table.

Value	Description
0x08	ECB for block ciphers as specified by [SP 800-38A]. *Note: the last block of the provided data for encryption must be block aligned and padded.*
0x09	CBC for block ciphers as specified by [SP 800-38A]. *Note: the last block of the provided data for encryption must be block aligned and padded.*
0x0A	CBC_MAC for block ciphers as specified by [ISO 9797-1].MAC Algorithm 1. *Note: the last block of the provided data for encryption must be block aligned and padded.*
0x0B	CMAC for block ciphers as specified by [SP 800-38B].
0x20	HMAC-SHA256 for SHA 256 as specified by [RFC2104].

Value	Description
0x21	HMAC-SHA384 for SHA 384 as specified by [RFC2104].
0x22	HMAC-SHA512 for SHA 512 as specified by [RFC2104].

Table 32 - Symmetric Modes of Operation

4.4.3 Command Performance

The performance metrics for various schemes are provided by Table 'Command 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

The performance of the commands, which use the secrets (e.g. private keys, shared secrets, etc.) at OPTIGA™ would get influenced by Security Monitor behavior.
The values specified in the below table are without shielded connection.

Operation	Command	Scheme	.Execution Time[92]	Additional Info
Read data	GetDataObject		~30 ms	Data size = 256 Bytes
Write data	SetDataObject		~ 55 ms	Data size = 256 Bytes
Calculate signature	CalcSign	ECDSA	~ 65 ms	• ECC NIST P 256 • no data hashing
Calculate signature	CalcSign	PKCS#1v2.2 RSASSA-PKCS1-v1.5_SHA256	~ 310 ms	• RSA 2048 Exponential • No data hashing
Verify signature	VerifySign	ECDSA	~ 85ms	• ECC NIST P 256 • No data hashing • Public Key from external interface
Verify signature	VerifySign	PKCS#1v2.2 RSASSA-PKCS1-v1.5_SHA256	~ 40 ms	• RSA 2048 Exponential • No data hashing • Public Key from external interface
Diffie Hellman key agreement	CalcSSec	ECDH SP-800 56A	~ 60ms	• Based on ephemeral key pair • ECC NIST P 256
Key pair generation	GenKeyPair	ECC	~ 55 ms	• ECC NIST P 256 • Ephemeral key

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

OPTIGA™ Trust M External Interface

Operation	Command	Scheme	.Execution Time[92]	Additional Info
Key pair generation	GenKeyPair	RSA	minimum 2900 ms	RSA 2048
Key derivation	DeriveKey	TLS v1.2 PRF SHA256	~ 50 ms	• To derive a key of 40 bytes • Shared secret (32 bytes) from session context and • The input key derivation data size is 48 bytes.
Key derivation	DeriveKey	HKDF SHA256	~ 130 ms	• To derive a key of 40 bytes • Shared secret (48 bytes) from a data object and the key derivation data is 48 bytes
Hash calculation	CalcHash	SHA 256	~ 15 Kbyte/s	In blocks of 0x500 bytes
RSA Encryption	EncryptAsym	PKCS#1v2.2 RSAES-PKCS1-v1.5	~ 40 ms	RSA 2048 Public key from external interface
RSA Decryption	DecryptAsym	PKCS#1v2.2 RSAES-PKCS1-v1.5	~ 315 ms	RSA 2048 Exponential
Symmetric encryption	EncryptSym	AES 128 ECB	~ 28 ms	• 128 bytes of data • No chaining
Symmetric decryption	DecryptSym	AES 128 ECB	~ 35 ms	• 128 bytes of data • No chaining
hmac generation	EncryptSym	HMAC-SHA256	~ 90 ms	• using a pre-shared secret (64 bytes) from a data object • 128 bytes of data from host
CMAC generation	EncryptSym	AES 128 CMAC	~ 28 ms	• 128 bytes of data • No chaining

Table 33 - Command 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.

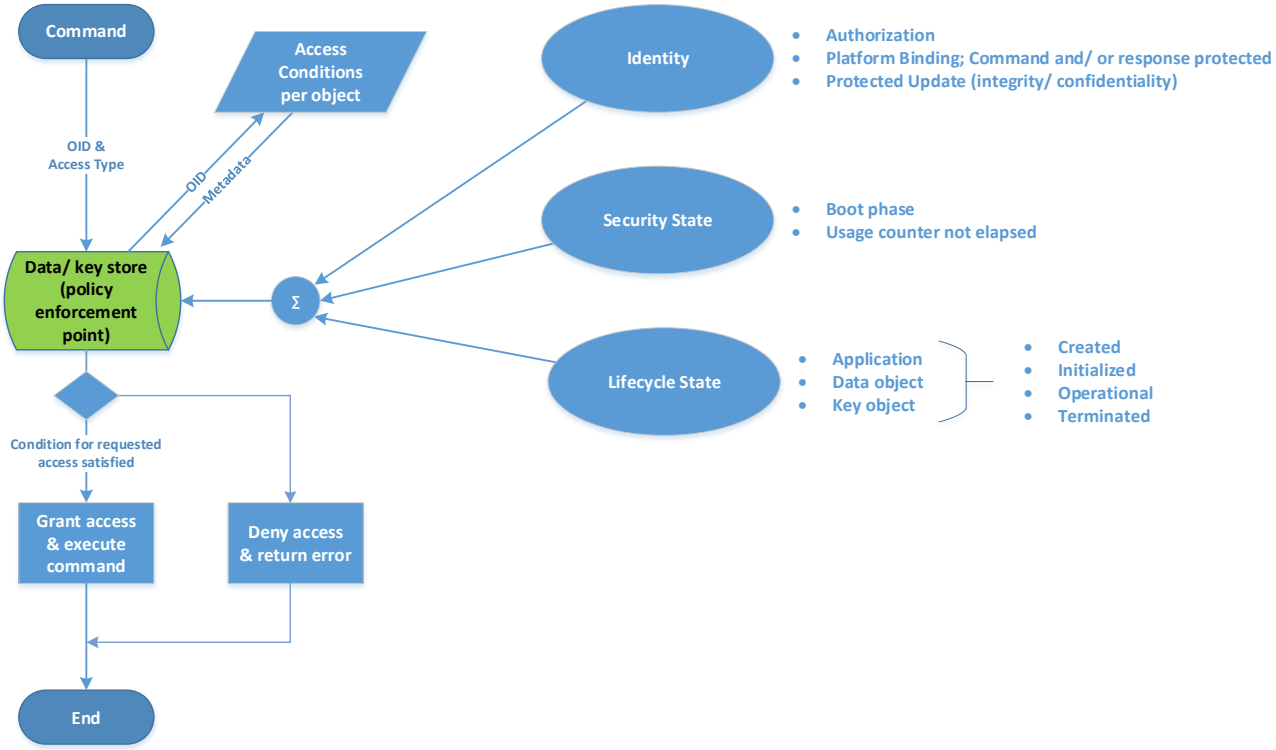


Figure 26 - Security Policy Architecture

Note: OPTIGA™ Trust M Version 1 doesn't support security policies like boot phase, authorization, confidentiality (for protected update).

4.5.2 Policy Attributes

The Policy Attributes are grouped in Identity, Security 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 or key object 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.

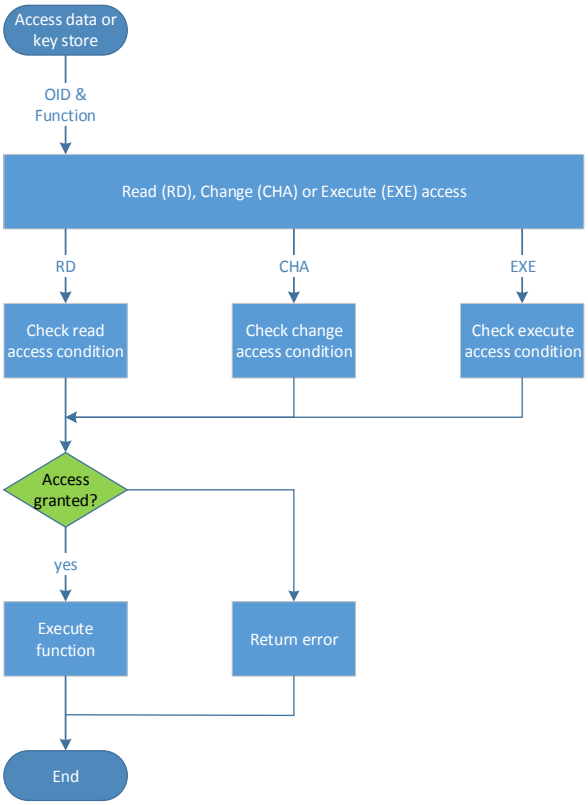


Figure 27 - Policy Enforcement Flow

The access conditions and the policy attribute details are provided in Access Conditions (ACs) section. The Security Guidance section provides the recommendations with respect to security policy enforcements.

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
Decryption Failure	The *Decryption Failure* event occurs in case a decryption and/ or integrity check of provided data lead to an integrity failure.
Key Derivation	The Key Derivation event occurs in case an operation, which executes a key derivation gets applied on a persistent data object which contains a 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 temporary keys from session context are used for key agreement like ECDH.
Secret Key Use	The Secret Key Use event occurs in case the internal services are going to use a OPTIGA™ hosted secret (symmetric) key (once per respective command), except temporary keys from session context are used.
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 34 - 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 shared secrets, 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 configurable, and default value is 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.

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

- After power-up or restart the SEC_{CREDIT} 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 SEC_{CREDIT} gets increased by one to a maximum limit configured.
- In case a Security Event occurs and the SEC_{CREDIT} is > 0, the SEC_{CREDIT} gets decreased by one.
- In case the SEC_{CREDIT} is = 0 and a Security Event occurs, the SEC increased.

4.6.3 Security Monitor Configurations

The possible security monitor configurations (available in Security Monitor Configurations data object) are,

- **t_{max}**
  The default value of t_{max} is 5 seconds. Due to use case demands, the t_{max} can be set to a different value. If t_{max} is set to 0, the security monitor gets disabled. In general, higher t_{max} value lowers the possible frequency of attacks.
  The maximum value of t_{max} that will be applied internally is 5 seconds even if t_{max} is configured to a higher value than 5 seconds.
  In case, the current SEC is > 0 and t_{max} is configured to 0, the SEC gets set to 0.

- **Maximum SEC_{CREDIT}** (SEC_{CREDIT_MAX})
  The maximum SEC_{CREDIT} that can be achieved is configurable. The default value is 5.
  If this value is set to 0, the SEC_{CREDIT} will be set to 0 and will not be incremented.

  For example, if t_{max} = 4000 milliseconds, if there are 720 sign operations distributed across in an hour (3600 seconds) (on average, for every 5 seconds, there is one sign operation), Then there is a possibility of SEC_{CREDIT} gets incremented about 180 times.

- **Delayed SEC decrement synchronization count**
  The SEC is as well maintained in RAM (SEC_{CURR}) in addition to the NVM (Security Event Counter (SEC))(SEC_{NVM}) and the synchronization (writing to SEC data object in NVM) of decrement events (e.g. t_{max} elapsed or Decryption Failure) can be delayed by configuring this value (> 1). If there are multiple security events with in t_{max} due to use case demand, the number of NVM write operations can be avoided by configuring this count appropriately.
  The default and minimum value is 1. In case, this value is configured to 0, minimum 1 will be applied.

  This option reduces SEC NVM write operations reasonably across the life time, if there are too frequent security events.
  For example, if this configuration is set to 4, and there are 4 security events (e.g. sign operations) immediate after reset which led to SEC increment, then the total number of SEC NVM write operations are 5 instead of 8 (in case of default (= 1) configuration).

The offset details of above specified configurations in Security Monitor Configurations data object are specified in Security Monitor Configurations table.

Note: OPTIGA™ Trust M Version 1 doesn't support this security monitor configurations.

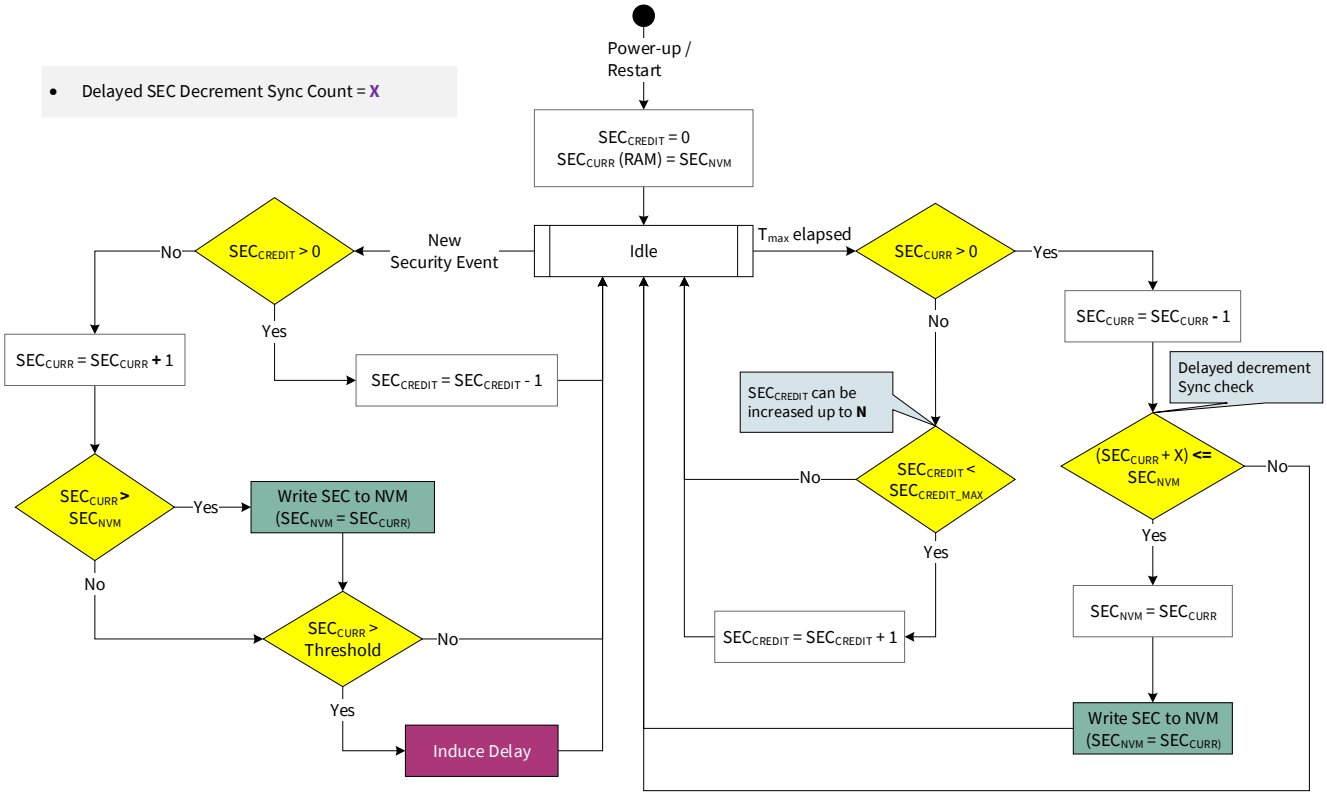


Figure 28 - Security Monitor flow diagram

The security monitor configurations are adopted during the power-up sequence and if there is change in the configuration (e.g. updated using SetDataObject, or SetObjectProtected). In case of SetObjectProtected - metadata update, these values get reset to default values if reset type (random/zeroes) is defined.

The MSB (invalid flag) of version tag and EXE access conditions are ignored for Security Monitor Configurations data object while using internally.

4.6.4 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.

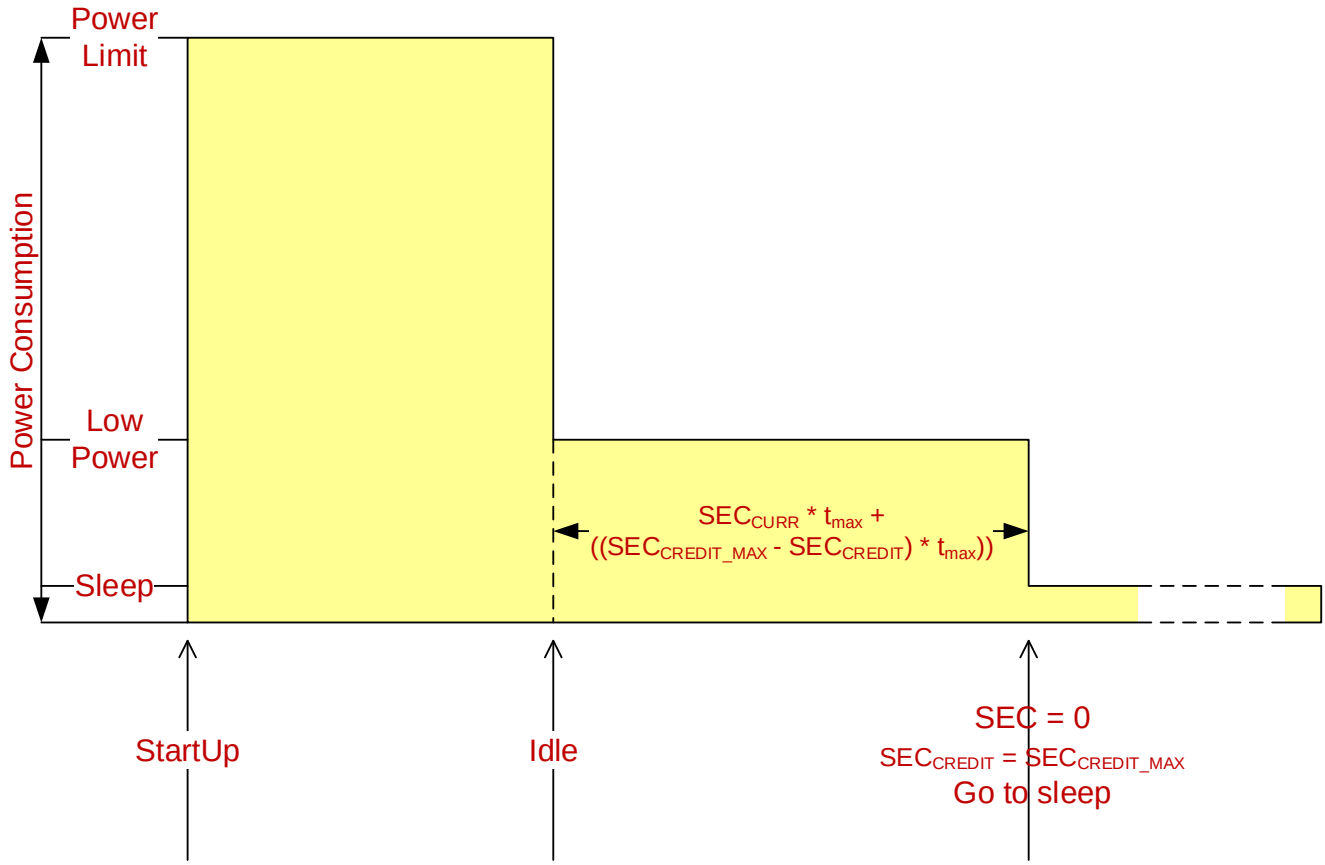


Figure 29 - Power profile

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 [Data Sheet M]). 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 SEC_{CREDIT} 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.

OPTIGA™ Trust M External Interface

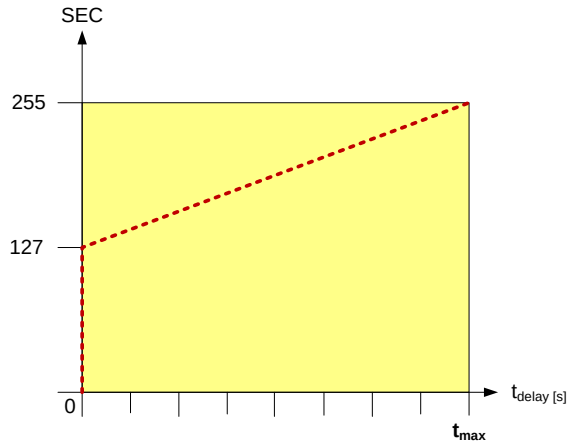


Figure 30 - Throttling down profile

5 OPTIGA™ Trust M Data Structures

5.1 Overview Data and Key Store

The data and key store has some important parameters to be considered while utilizing it. Those parameters are the data-retention-after-testing, the data-retention-after-cycling, hardening, the maximum endurance and the supported number of tearing-safe-programming-cycles.

- **data-retention-after-testing** defines how long NVM programmed during production (at wafer tester) is preserved, in case the NVM experiences no additional programming attempts, this time is the same as the device lifetime defined in the data sheet.
- **data-retention-after-cycling** defines how long NVM data, programmed throughout lifetime, are preserved. The number of experienced programming cycles is important for how long the data are preserved. For maximum 100 times the same as **data-retention-after-testing** applies. After e.g. 20 000 times the NVM data retention declines linearly down to 2 years and even more down to ½ year after about 40 000 NVM programming cycles. ½ year **data-retention-after-cycling** is the worst case. With other words, if NVM data get often cycled the time between programming attempts should not exceed half a year. If erases (sub-function of a programming cycle) are frequently done, than regular hardening is performed, in this case the **data-retention-after-cycling** worst case improves from ½ year to 3 years. In case of high cycling, beyond 20 000 times, the cycles shall be homogeneous distributed across lifetime.
- **hardening** is a process which is embedded in any erase cycle (each NVM programming cycle causes one or multiple erase cycles) and refreshes a randomly chosen NVM page. After a certain number of erase cycles, which is dependent on the physical NVM size (For OPTIGA™ Trust M about 5 000), all NVM is “refreshed”. Hardening is performed without erasing or physically moving the hardened data, i.e. hardening is neither susceptible to tearing nor does it count as a programming cycle.
- **maximum endurance** defines how often a NVM data could be programmed until it wears out.
- **number of tearing-safe-programming-cycles** defines how many tearing-safe-programming-cycles could be performed for the data and key store. It is worth to mention, that each data or key store programming attempt is performed in a tearing safe manner. The maximum number of **tearing-safe-programming-cycles** is **2 million**.

The following list provides the cases when a **tearing-safe-programming-cycle** takes place:

- Update of a data object causes one tearing-safe-programming-cycle
- Update of a key object causes one to six tearing-safe-programming-cycles (average for ECC is one and for RSA is five)
- Usage (EXE or CHA or RD) of a data or key object which is linked to a usage counter causes one (additional) tearing-safe-programming-cycle
- The Security Event Counter (SEC) gets increased and subsequently decreased (refer to list of Security Events) and causes two tearing-safe-programming-cycle

OPTIGA™ Trust M Data Structures

- One hibernate cycle causes five tearing-safe-programming-cycles

The figure "Overview Data and Key Store" below provides an overview of all data and key objects hosted by the OPTIGA™ and the recommended maximum cycling (color coding) per object. In case those recommendations are meet and for higher cycled data objects the homogeneous distributed of applied programming cycles across lifetime are respected, the data integrity over lifetime could be considered being safe.

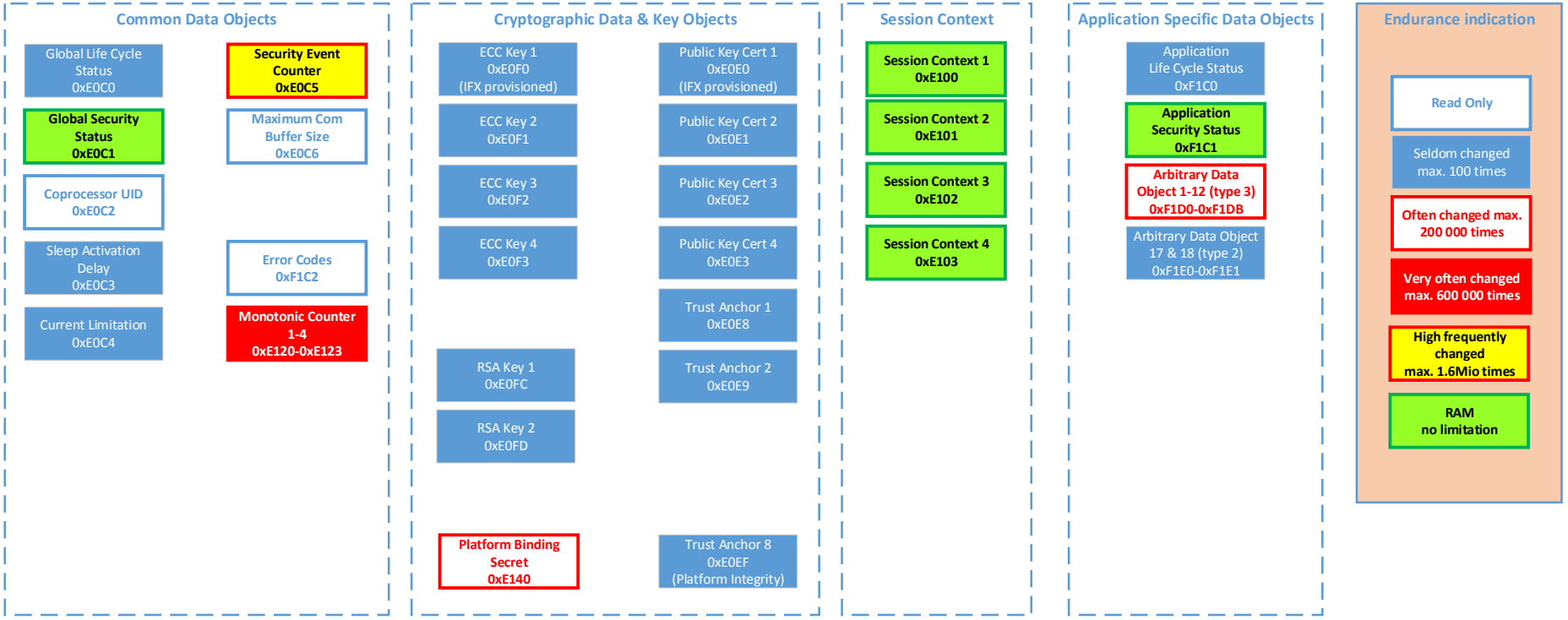


Figure 31 – OPTIGA™ Trust M (Version 1) Overview Data and Key Store

OPTIGA™ Trust M Data Structures

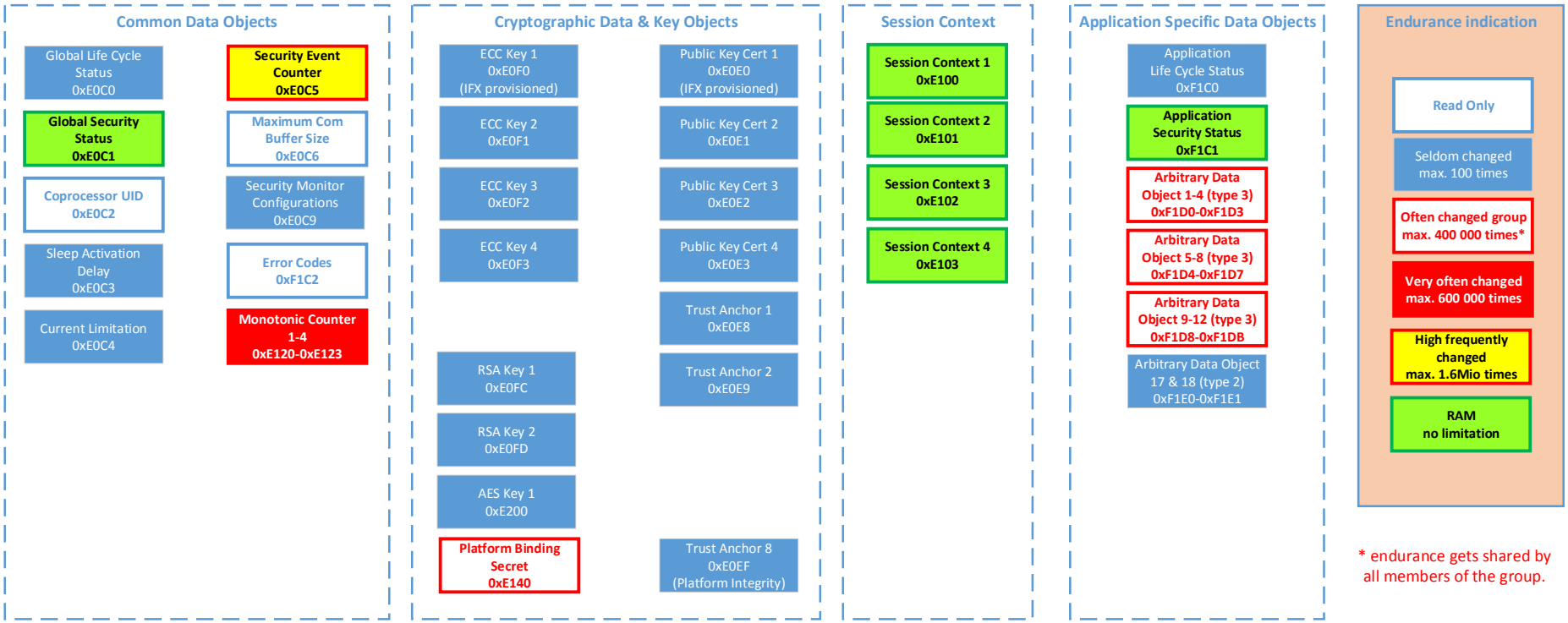


Figure 32 – OPTIGA™ Trust M (Version 3) Overview Data and Key Store

Examples:

- Each monotonic counter can be updated up to a maximum of 600 000 times.
- The maximum number of updates across all objects (key/data) is allowed up to 2 million times either due to external interface requests or due to internal operations (the list is given above). This means the maximum updates per object and the overall update limit across all objects must be respected to prevent reliability issues.

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

OPTIGA™ Trust M Data Structures

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 command (e.g. CalcSign, GenKeyPair, ...)

MUPD Updating metadata[93] by an external command (e.g. SetObjectProtected).

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.

Auto(X) the action is only possible in case the authorization of the external world was successful performed, providing the Authorization Value which matches the Data Object Types. Authorization Reference given by X.
Note: Up to 4 independent **Auto(X)** states at a time are supported.

SecStaG(X) the action is only possible in case the global security status ANDed with X is equal X.

SecStaA(X) the action is only possible in case the application specific security status ANDed with X is equal X.

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

[93] OPTIGA™ Trust M Version 1 doesn't support updating metadata.

OPTIGA™ Trust M Data Structures

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. However, the implementation evaluates from left to right and acts accordingly.

The access conditions which could be associated to OPTIGA™ data and key objects are defined by Table 'Access Condition Identifier and Operators'.
Note: OPTIGA™ Trust M Version 1 doesn't support SecStaG(X), SecStaG(X), and Auto (X) Identifiers.

AC ID	Operator	Value	Description
ALW	-	0x00	1 byte; Value
SecStaG	-	0x10	2 byte; Value (e.g. Enable access if boot phase flag in Security Status application is set → 0x10, 0x20) Note: SetDataObject with Param = erase&write clears all bits and with Param = write clears all corresponding bits not set to 1 in data to be written

OPTIGA™ Trust M Data Structures

AC ID	Operator	Value	Description
Conf	-	0x20	3 byte; Value, Key Reference (OID) (e.g. Conf first Session Key → 0x20, 0xE1, 0x40) • Read, Conf, Binding Secret (e.g. 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 Binding secret (e.g. 0xE140) and the response is requested with protection (encrypted). • Change, Conf, Binding Secret (e.g. 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). • Execute, Conf, Binding Secret (e.g. 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 binding secret (0xE140) and the command is sent protection (encrypted). • Change, Conf, Protected Update Secret → (e.g. 0xD0, 0x03, 0x20, 0xF1, 0xD0) In case of writing a data object (using SetObjectProtected), the manifest must specify the same Protected Update Secret (e.g. 0xF1, 0xD0) which is specified in the object metadata. This enforces to use the defined Protected Update Secret to decrypt the object data in fragments. Notes: • Conf (Protected Update Secret) must be used in association(Operator AND) with Integrity (Trust Anchor), to enforce the right Protected Update Secret to be used to decrypt the object data as part of SetObjectProtected. If Conf (Protected Update Secret) not specified

AC ID	Operator	Value	Description
			in the metadata access conditions, SetObjectProtected uses Protected Update Secret specified in the manifest, to decrypt the object data as part of fragments. The usage of this identifier is to enforce the right secret used (Integrity Trust Anchor, Operator AND, Confidentiality Protected Update Secret OID). The Protected Update Secret must not same as the target data object to be updated.
Int	-	0x21	3 byte; Value, Key Reference (e.g. Int first Session Key → 0x21, 0xF1, 0xF0) • Read, Int, Binding Secret (e.g. 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 protection (MAC). • Change, Int, Binding Secret (e.g. 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). • Execute, Int, Binding Secret (e.g. 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). • Change, Int, Trust Anchor (e.g. 0xD0, 0x03, 0x21, 0xE0, 0xEF) In case of writing a data object (e.g. using SetObjectProtected), the signature associated with the meta data in the manifest must be verified with the addressed trust anchor (e.g. 0xE0EF) in the access conditions. In case of SetObjectProtected command, the change access conditions of target OID must have Integrity access condition identifier with the

OPTIGA™ Trust M Data Structures

AC ID	Operator	Value	Description
			respective Trust Anchor.
Auto	-	0x23	3 byte; Value, Reference (Authorization Reference OID) (e.g. Auto → 0x23, 0xF1, 0xD0)
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.
LcsG	-	0x70	3 byte; Value, Qualifier, Reference (e.g. LcsG < op → 0x70, 0xFC, 0x07)
SecStaA	-	0x90	2 byte; Value (e.g. Enable access if boot phase flag in Security Status application is set → 0x90, 0x20) Note: SetDataObject with Param = erase&write clears all bits and with Param = write clears all corresponding bits not set to 1 in data to be written
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

OPTIGA™ Trust M Data Structures

AC ID	Operator	Value	Description
-	&&	0xFD	logical AND
-	||	0xFE	logical OR
NEV	-	0xFF	1 byte; Value

Table 35 - Access Condition Identifier and Operators

Table 'Data Object Types' lists the various types of 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 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. The maximum length is limited to 64 bytes, even if the hosting data object has a higher maximum length.
AUTOREF	0x31	The Authorization Reference contains a binary data string which makes up a reference value for verifying an external entity (admin, user, etc.) authorization.

Table 36 - 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.

Tag	Structure definition	Default Value	Execute	Change	Read	Note
0xE0C0	Global Life Cycle Status (LcsG)	0x07	NEV	ALW	ALW	default LcsO = op
0xE0C1	Global Security Status	0x20	NEV	ALW[94]	ALW	default LcsO = op

[94] It is only possible to reset an achieved security status

OPTIGA™ Trust M Data Structures

Tag	Structure definition	Default Value	Execute	Change	Read	Note
0xE0C2	Coprocessor UID OPTIGA™ Trust Family		NEV	NEV	ALW	default LcsO = op
0xE0C3	Sleep Mode Activation Delay (refer to 'Common data structures')	0x14	NEV	ALW	ALW	default LcsO = op
0xE0C4	Current limitation (refer to 'Common data structures')	0x06	NEV	ALW	ALW	default LcsO = op
0xE0C5	Security Event Counter (SEC) (refer to 'Common data structures')		NEV	NEV	ALW	default LcsO = op
0xE0C6	Maximum Com Buffer Size (refer to' Common data structures')	0x0615	NEV	NEV	ALW	default LcsO = op
0xE0C9	Security Monitor Configurations	50 00 05 01 00 00 00 00	NEV	LcsO < 7	ALW	Default LcsO = op; This data object holds the security monitor configurations (e.g. maximum SEC_{CREDIT}, t_{max}, etc.). The Data Object Types are not allowed to be configured with this data object. For more details, refer Security Monitor Configurations section.
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"[95]
0xE0E1-0xE0E3	Project-specific device Public Key Certificate 1-3. (refer to 'Common data structures')	0x00	ALW	LcsO < op	ALW	default LcsO = cr; default Data Type is "Device Identity"[96]

[95] 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

[96] 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

OPTIGA™ Trust M Data Structures

Tag	Structure definition	Default Value	Execute	Change	Read	Note
0xE0E8-0xE0E9	Root CA Public Key Certificate 1-2[97] (refer to 'Common data structures')	0x00	ALW	LcsO < op	ALW	default LcsO = cr; default Data Type is "Trust Anchor"
0xE0EF	Root CA Public Key Certificate 8[98]. This trust anchor is assigned to platform integrity use cases (refer to 'Common data structures').	0x00	ALW	LcsO < op	ALW	default LcsO = cr; default Data Type is "Trust Anchor"
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 37 - 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.

[97] 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.

[98] due to its size the public key or certificate is not written in an atomic way. With other words in case the write gets terminated by a tearing event like power lost or reset, the write might be incomplete - the certificate is just partly written.

OPTIGA™ Trust M Data Structures

Tag	Structure definition	Execute	Change	Read	Note
0xE0F0	Device Private ECC Key 1; The GetDataObject, SetDataObject commands are not allowed for the data part of the key object even if the metadata states the access rights differently.	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 states the access rights differently.	ALW	LcsO < op	NEV	default LcsO = cr
0xE0FC-0xE0FD	Device Private RSA Key 1-2; The GetDataObject, SetDataObject commands are not allowed for the data part of the key object even if the metadata states the access rights differently.	ALW	LcsO < op	NEV	default LcsO = cr
0xE100-0xE103	Session context 1-4 (OID to address one of the four session contexts e.g. (D)TLS connection state). These OIDs are not applicable for the *GetDataObject*, SetDataObject commands. The session context holds either Private key or Shared secret or is target for toolbox commands like (*GenKeyPair*, *CalcSSec*, *DeriveKey*).	ALW	NEV	NEV	default LcsO = op
0xE200	Device Symmetric Key 1; 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	NEV	NEV	default LcsO = cr

Table 38 - 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

Tag	Structure definition	Default Value	Execute	Change	Read	Note
0xF1C1	Data Structure application '*Security Status*'	0x20	NEV	ALW[99]	ALW	default LcsO = op
0xF1C2	Error codes (refer to Table '*Error Codes*')		NEV	NEV[100]	ALW	default LcsO = op
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 39 - 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.

[99] It is only possible to reset an achieved security status

[100] cleared on read

OPTIGA™ Trust M Data Structures

Tag	Structure definition	Execute	Change	Read	Note
0x20	Metadata constructed TLV-Object	n.a.	n.a.	ALW	
0xC0	Life Cycle State of the data/key object (LcsO)	n.a.	ALW	ALW	refer to Table 'Life Cycle Status'
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 the SetObjectProtected and the SetDataObject(metadata)command.	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.	NEV	ALW	
0xC5	Used size of the data object	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.	LcsO < op	ALW	
0xD1	Read Access Condition descriptor	n.a.	LcsO < op	ALW	
0xD3	Execute Access Condition descriptor	n.a.	LcsO < op	ALW	
0xD8	Metadata Update descriptor	n.a.	LcsO < op	ALW	This tag defines the condition under which the metadata update is permitted. (e.g. Metadata Update Descriptor, Int, Trust Anchor → 0xD8, 0x03, 0x21, 0xE0, 0xEF) In case of updating object metadata (e.g. using SetObjectProtected), the signature associated with the metadata update must be verified with the addressed

OPTIGA™ Trust M Data Structures

Tag	Structure definition	Execute	Change	Read	Note
					trust anchor (e.g. 0xE0EF) and the associated (Factory) Reset Type (0xF0) gets applied before the metadata are newly set. In case no (Factory) Reset Type is given by the current metadata, the update must fail.
0xE0	Algorithm associated with key container	n.a.	auto[101]	ALW	refer to Table 'Algorithm Identifier'
0xE1	Key usage associated with key container	n.a.	LcsO < op	ALW	refer to Table 'Key Usage Identifier'
0xE8	Data object Type	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 and 0xC5).
0xF0	(Factory) Reset Type	n.a.	LcsO < op	ALW	It defines what happens with the object data in case of updating the metadata, refer to table Metadata Update Identifier for the values. Will be applied as part of SetObjectProtected implementation

Table 40 - Metadata associated with data and key objects

Table 'Metadata Update Identifier' lists the metadata update identifier supported by the OPTIGA™. The identifier could be combined by ORing them (e.g. 0x11 => LcsO = cr & flush object with zero). However, the bits in the upper nibble are not allowed being combined. In case of protected metadata update the new LcsO gets provided, this overrules the setting given by the Metadata Update Identifier.

Value	Description
0x0X	Setting the LcsO of either a key or data object. The definition of X is given by Life Cycle Status (lower nibble)

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

Value	Description
0x10	• Flushing of either a key or data object with zero and set the used length of data objects, if present, to 0x0000. • If none of the flushing options is set in metadata, then the SetObjectProtected Manifest setting (if present) gets used. • In case of a key object the algorithm associated gets cleared and sets again with successful generation or writing (protected update) a new key.
0x20	• Overwriting either a key or data object with random values and set the used length of data objects, if present, to 0x0000. • If none of the flushing options is set in metadata, then the SetObjectProtected Manifest setting (if present) gets used. • In case of a key object the algorithm associated gets cleared and sets again with successful generation or writing (protected update) a new key.

Table 41 - Metadata Update Identifier

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
```

OPTIGA™ Trust M Data Structures

Note: there is no ordering rule for metadata tags

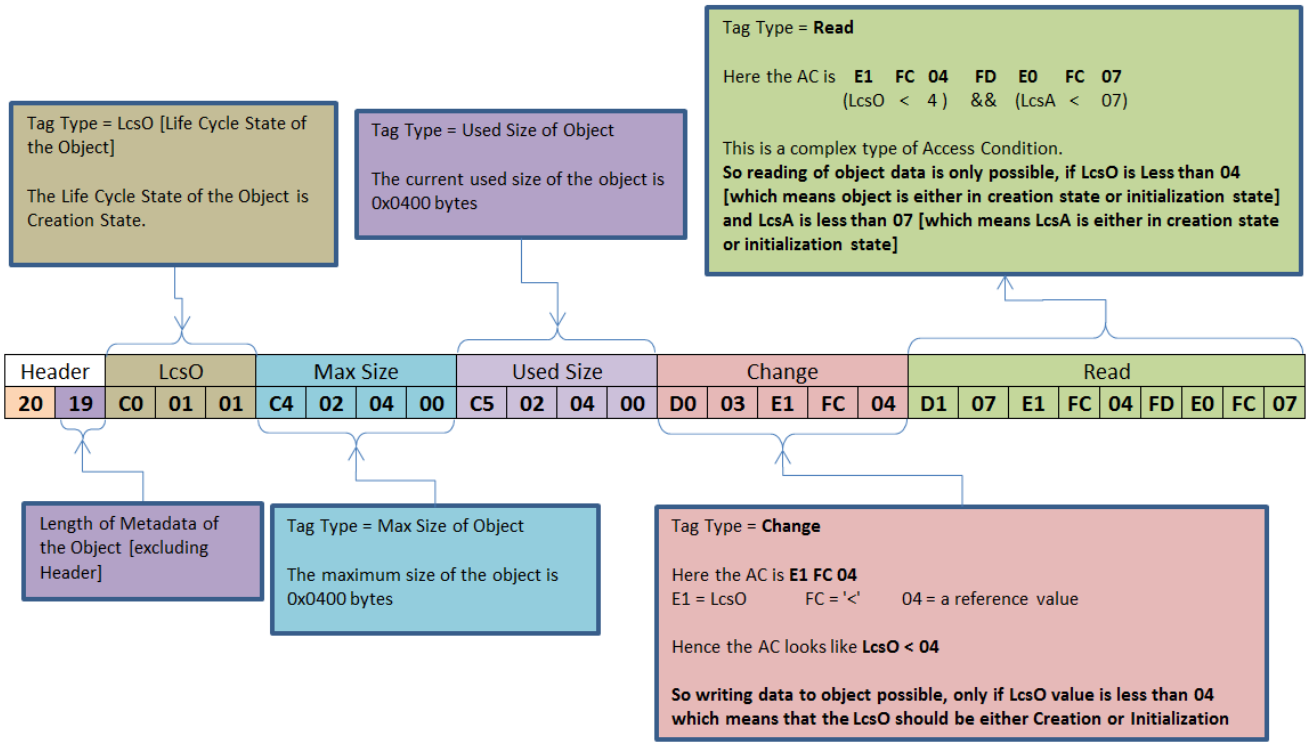


Figure 33 - Metadata sample

OPTIGA™ Trust M Data Structures

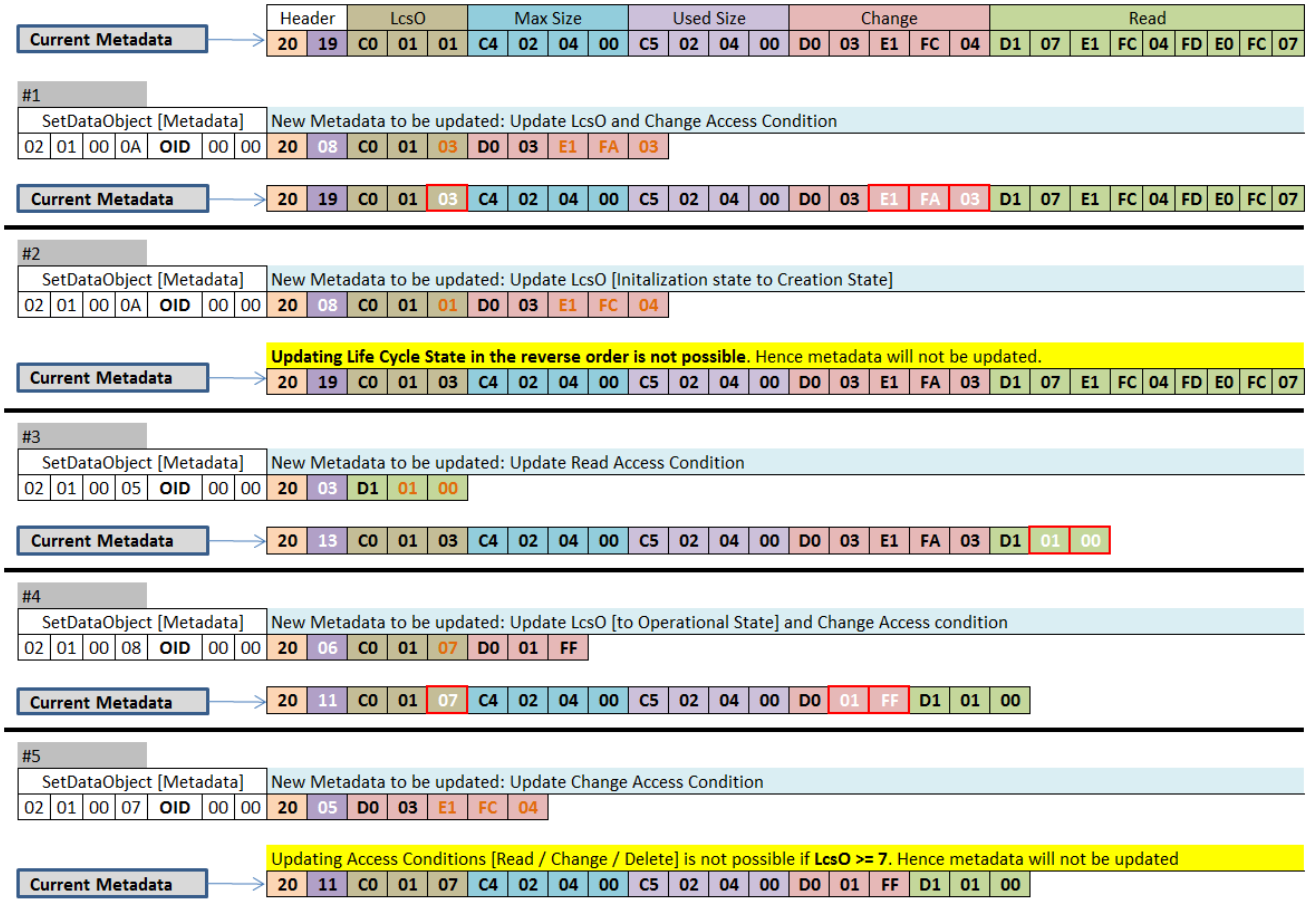


Figure 34 - SetDataObject (Metadata) examples

Note: The values specified in Figure 33 and Figure 34 are in HEX format.

5.6 Common data structures

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

Data element	Coding	Length (Bytes)	Description
Coprocessor UID OPTIGA™ Trust Family		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.
Sleep Mode Activation Delay in ms	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	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.

OPTIGA™ Trust M Data Structures

Data element	Coding	Length (Bytes)	Description
			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	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	1	The SEC holds the current value of the Security Event Counter as described in chapter Security Monitor.
Public Key Certificate	x.509	1728 (max.)	The Public Key Certificate data object holds one or multiple of X.509 Certificate (refer to [RFC5280]). The certificate was issued by IFX or a Customer. 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[102] • **USB Type-C Identity:** **Tag** = 0xC2 **Length** = Value length (2 Bytes) **Value** = USB Type-C Certificate Chain[USB Auth][103]

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

OPTIGA™ Trust M Data Structures

Data element	Coding	Length (Bytes)	Description
			• **WPC Compressed Identity:** **Tag** = 0xC4 **Length** = Value length (2 Bytes) **Value** = WPC compressed Certificate Chain [WPC Auth][104] • **WPC X.509 Identity:** **Tag** = 0xC6 **Length** = Value length (2 Bytes) **Value** = WPC X.509 Certificate Chain
Root CA Public Key Certificate aka "Trust Anchor"	x.509 (maybe self-signed certificate)	1200 (max.)	The Root CA Public Key Certificate data object holds the X.509 Public Key Certificate of the IFX or Customer Root or Intermediate Certification Authority. It is used as "Trust Anchor" to authenticate external entities (e.g. verify server signature).
Platform Binding Secret	BinaryNumber8	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.
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 42 - Common data structures

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

[104] Format as defined in the WPC Authentication Specification.

Bit 8-5	Bit 4-1	Description
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)[105]

Table 43 - 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[106] [107].

Bit 8-7	Bit 6-1	Description
	_ x x x x x	The *Security Status* flag Boot is set=1 by default after reset for both global and application specific. As soon as the boot phase specific permissions should be terminated the boot flag could be reset by writing 0b1101 1111 to the regarded *Security Status*.
x x	_ x x x x x	RFU

Table 44 - Security Status

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

[105] this state is not applicable for the LcsA
[106] bit = 0 status not satisfied
[107] bit = 1 status satisfied

OPTIGA™ Trust M Data Structures

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

Table 45 - Coprocessor UID OPTIGA™ Trust Family

Table 'Security Monitor Configurations' shows the possible configurations w.r.t. Security Monitor.

Offset	Name	Data Type	Notes
0	t_{max} configuration	uint8_t	= Required Tmax in milliseconds / 100. E.g. if the required t_{max} is 4 seconds (4000 milliseconds), this will be configured as 40. Default value = 50.
1	RFU[1]	uint8_t	Default is 0x00.
2	SEC_{CREDIT} Maximum value	uint8_t	The maximum value of SEC_{CREDIT} that can be achieved.

Offset	Name	Data Type	Notes
	(SEC_{CREDIT_MAX})		Default is 5.
3	Delayed SEC Decrement Sync count	uint8_t	The SEC is as well maintained in RAM in addition to the NVM (Security Event Counter (SEC)) and the synchronization (writing to data object in NVM) of decrement event (e.g. Tmax elapsed) can be delayed by configuring this value greater than 1. Default is 1.
4	RFU[4]	uint8_t	Default is 00 00 00 00.

Table 46 - Security Monitor Configurations

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[108]	HexNumber5	5
PIX	'GenAuthAppl' 0x47656E4175746841 70706C	Ascii11	11

Table 47 - Data Structure Unique Application Identifier

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

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

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

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

Table 48 - 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 and key objects which are used to update those in a secure/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 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:

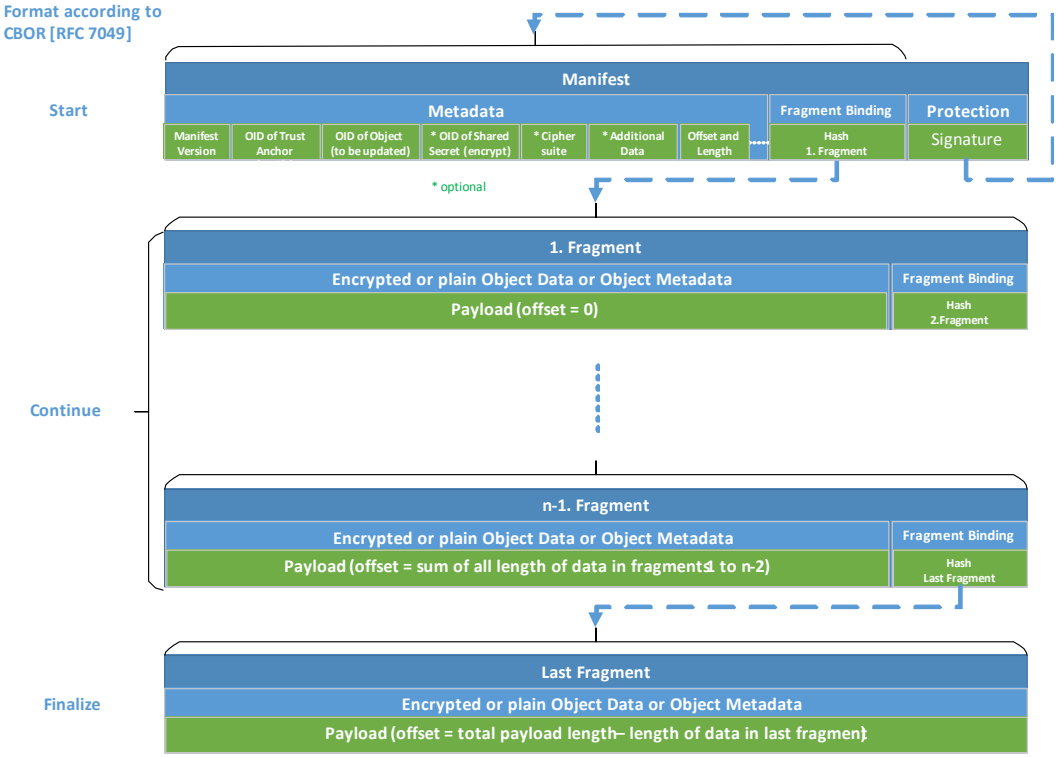
- Version number
- Unique identifier (OID) of the trust anchor to be used for verifying the metadata signature.
- Unique identifier (OID) of the object to be updated
- Cipher suite specifying all cryptographic algorithms (signature, encryption, hash, key derivation, ...) used during executing the update.
- In case of encrypted object data, OID of the shared secret to be used for derivation of the decryption key.
- In case of encrypted object data, Additional data used in key derivation
- 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.

The **Confidentiality Protection** is based on a shared secret installed at OPTIGA™, and additional data used to derive the object data decryption key. The

session key gets applied as soon as the integrity of the current block of the object data is successfully verified.

Chapter Protected Update provides more detailed information regarding the format of manifest and data fragments consumed.



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 35 - GetDataObject [Read data] example

Note: The values specified in Figure 35 *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 36 - SetDataObject [Write data] example

Note: The values specified in Figure 36 *are in HEX format.*

6.2 Data encoding format examples

This section provides the examples for the encoding format of Asymmetric key pairs and Signature used across the Enabler APIs.

6.2.1 ECC Private Key

The examples for format of ECC Private Key exported as part of Generate Key Pair are given below.

- Example for ECC NIST P-256 Private Key [Values are in hex]

```
// DER OCTET String Tag (Private Key)
04
    // Length of Tag
    20
        // Private Key
        20 DC 58 98 CF 51 31 44 22 EA 01 D4 0B 23 B2 45
        7C 42 DF 3C FB 0D 33 10 B8 49 B7 AA 0A 85 DE E7
```

- Example for ECC NIST P-384 Private Key [Values are in hex]

```
// DER OCTET String Tag (Private Key)
04
    // Length of Tag
    30
        // Private Key
        53 94 F7 97 3E A8 68 C5 2B F3 FF 8D 8C EE B4 DB
        90 A6 83 65 3B 12 48 5D 5F 62 7C 3C E5 AB D8 97
        8F C9 67 3D 14 A7 1D 92 57 47 93 16 62 49 3C 37
```

6.2.2 ECC Public Key

The examples for the format of ECC Public Key (referred by Generate Key Pair, Verify signature and ECDH operations) are given below.

- Example for ECC NIST P-256 Public Key [Values are in hex]

```
// Bit String tag
03
    // Length of Bit string tag
    42
        // Unused bits
        00
        // Compression format. Supports only 04 [uncompressed]
        04
        // Public Key (e.g. ECC NIST P-256, 0x40 Bytes)
        // Qx - 0x20 Bytes
            8B 88 9C 1D D6 07 58 2E D6 F8 2C C2 D9 BE D0 FE
            6D F3 24 5E 94 7D 54 CD 20 DC 58 98 CF 51 31 44
        // Qy - 0x20 Bytes
            22 EA 01 D4 0B 23 B2 45 7C 42 DF 3C FB 0D 33 10
            B8 49 B7 AA 0A 85 DE E7 6A F1 AC 31 31 1E 8C 4B
```

- Example for ECC NIST P-384 Public Key [Values are in hex]

```
// Bit String tag
03
    // Length of Bit string tag
```

```
62
    // Unused bits
    00
    // Compression format. Supports only 04 [uncompressed]
    04
    // Public Key (e.g. ECC NIST P-384, 0x60 Bytes)
    // Qx - 0x30 Bytes
        1F 94 EB 6F 43 9A 38 06 F8 05 4D D7 91 24 84 7D
        13 8D 14 D4 F5 2B AC 93 B0 42 F2 EE 3C DB 7D C9
        E0 99 25 C2 A5 FE E7 0D 4C E0 8C 61 E3 B1 91 60
    // Qy - 0x30 Bytes
        1C 4F D1 11 F6 E3 33 03 06 94 21 DE B3 1E 87 31
        26 BE 35 EE B4 36 FE 20 34 85 6A 3E D1 E8 97 F2
        6C 84 6E E3 23 3C D1 62 40 98 9A 79 90 C1 9D 8C
```

6.2.3 ECDSA Signature

The examples for format of ECDSA Signature (referred by Signature generation and verification operations) are given below.

- Example for signature in case of ECC NIST P-256 key [Values are in hex]

```
// Integer tag for R component
02
    // Length
    20
        // R Component
        2B 82 6F 5D 44 E2 D0 B6 DE 53 1A D9 6B 51 E8 F0
        C5 6F DF EA D3 C2 36 89 2E 4D 84 EA CF C3 B7 5C

// Integer tag for S component
02
    // Length
    21
        // S Component
        00
        A2 24 8B 62 C0 3D B3 5A 7C D6 3E 8A 12 0A 35 21
        A8 9D 3D 2F 61 FF 99 03 5A 21 48 AE 32 E3 A2 48
```

- Example for signature in case of ECC NIST P-384 key [Values are in hex]

```
//Integer tag for R component
02
    // Length
    31
        //R Component
        00
        C3 6E 5F 0D 3D E7 14 11 E6 E5 19 F6 3E 0F 56 CF
        F4 32 33 0A 04 FE FE F2 99 3F DB 56 34 3E 49 F2
        F7 DB 5F CA B7 72 8A CC 1E 33 D4 69 25 53 C0 2E

//Integer tag for S component
02
    // Length
    30
        //S Component
        0D 40 64 39 9D 58 CD 77 1A B9 42 0D 43 87 57 F5
        93 6C 38 08 E9 70 81 E4 57 BC 86 2A 0C 90 52 95
        DC A6 0E E9 4F 45 37 59 1C 6C 7D 21 74 53 90 9B
```

Notes:

- The size of R and S components is based on the key size chosen. (e.g. in case of ECC NIST P256, size is 32 bytes and for ECC NIST P384, size is 48 bytes)

Appendix

- If the first byte of R or S component is greater than 0x7F (negative integer), then the respective component gets prepended with 0x00.
- The caller must consider the length of buffer for signature accordingly considering the additional encoding information.

6.2.4 RSA Private Key

The examples for the format of RSA Private key (Private Exponent) exported as part of optiga_crypt_rsa_generate_keypair (export private key = True) are given below.

For RSA 1024, the length of the Private exponent is 128 bytes.
For RSA 2048, the length of the Private exponent is 256 bytes.

- Example for RSA 1024 exponential - Private Key (Private Exponent) [Values are in hex]

```
// DER OCTET String Tag (Private Exponent)
04
    // Length of Tag
    81 80
        // Private Exponent (0x80 Bytes)
        53 33 9C FD B7 9F C8 46 6A 65 5C 73 16 AC A8 5C
        55 FD 8F 6D D8 98 FD AF 11 95 17 EF 4F 52 E8 FD
        8E 25 8D F9 3F EE 18 0F A0 E4 AB 29 69 3C D8 3B
        15 2A 55 3D 4A C4 D1 81 2B 8B 9F A5 AF 0E 7F 55
        FE 73 04 DF 41 57 09 26 F3 31 1F 15 C4 D6 5A 73
        2C 48 31 16 EE 3D 3D 2D 0A F3 54 9A D9 BF 7C BF
        B7 8A D8 84 F8 4D 5B EB 04 72 4D C7 36 9B 31 DE
        F3 7D 0C F5 39 E9 CF CD D3 DE 65 37 29 EA D5 D1
```

- Example for RSA 2048 exponential - Private key (Private Exponent) [Values are in hex]

```
// DER OCTET String Tag (Private Exponent)
04
    // Length of Tag
    82 01 00
        // Private Exponent (0x100 Bytes)
        21 95 08 51 CD F2 53 20 31 8B 30 5A FA 0F 37 1F
        07 AE 5A 44 B3 14 EB D7 29 F5 DC B1 5D A7 FA 39
        47 AC DD 91 5D AE D5 74 BD 16 DF 88 BF 85 F6 10
        60 B3 87 17 2F AE 6E 01 26 2B 38 64 C2 D3 C2 2F
        94 E0 4A 81 59 42 2B 4E D2 79 C4 8A 4C 9D 76 7D
        49 66 07 1A 5B BF 5D 04 3E 16 FF 46 EC 1B A0 71
        6F 00 BB C9 7B FF 5D 56 93 E2 14 E9 9C 97 21 F1
        2B 3E C6 28 2A E2 A4 85 72 1B 96 DD CF 74 03 FA
        03 7D 0C 57 AB 46 3C 44 8D E5 CC 12 26 5A DD 88
        6D 31 1E A8 D8 A5 90 3F A5 6C 5F 1C 9C F2 EB 11
        CB 65 7A 1A 7D 3E 41 35 2D C3 E6 86 89 8C 4C E4
        30 5E 8B 63 8E 1B 08 A2 A8 6C C9 EB 98 66 F3 49
        9A C7 7B 61 36 B8 1C B2 76 D6 14 CF EB 7B 6E D3
        F3 BC 77 5E 46 C0 00 66 EB EE E2 CF F7 16 6B 57
        52 05 98 94 7F F6 21 03 20 B2 88 FB 4F 2C 3F 8F
        E9 7B 27 94 14 EB F7 20 30 00 A1 9F C0 42 48 75
```

6.2.5 RSA Public Key

The examples for the format of RSA Public Key (referred by Generate Key Pair, Verify signature and Asymmetric Encryption operations) are given below.

- Example for RSA 1024 Exponential Public Key (modulus and public exponent) [Values are in hex]

Appendix

```
// Bit string tag
03
    // Bit String tag Length
    81 8E
        // Unused Bits
        00
        // SEQUENCE
        30
            // Length
            81 8A
                // Integer tag (Modulus)
                02
                    // Length of Modulus
                    81 81
                        // Modulus
                        00
                        A1 D4 6F BA 23 18 F8 DC EF 16 C2 80 94 8B 1C F2
                        79 66 B9 B4 72 25 ED 29 89 F8 D7 4B 45 BD 36 04
                        9C 0A AB 5A D0 FF 00 35 53 BA 84 3C 8E 12 78 2F
                        C5 87 3B B8 9A 3D C8 4B 88 3D 25 66 6C D2 2B F3
                        AC D5 B6 75 96 9F 8B EB FB CA C9 3F DD 92 7C 74
                        42 B1 78 B1 0D 1D FF 93 98 E5 23 16 AA E0 AF 74
                        E5 94 65 0B DC 3C 67 02 41 D4 18 68 45 93 CD A1
                        A7 B9 DC 4F 20 D2 FD C6 F6 63 44 07 40 03 E2 11
                // Integer tag for Public Exponent
                02
                    // Length of Public Exponent
                    04
                        // Public Exponent
                        00 01 00 01
```

Notes:

- The size of Modulus component is based on the key size chosen. (e.g. in case of RSA 1024, size is 128 bytes and for RSA 2048, size is 256 bytes)
- If the first byte of Modulus is greater than 0x7F (negative integer), then 0x00 gets prepended while coding as it is in Integer format.

6.2.6 RSA Signature

The example for format of RSA Signature (referred by RSA Signature generation and verification operations) is given below.

There is no additional encoding (format) considered for RSA signature. The length of the signature is based on the key length chosen.
For RSA 1024, the length of the signature is 128 bytes.
For RSA 2048, the length of the signature is 256 bytes.

- Example for RSA signature in case of RSA 1024-Exp key [Values are in hex]

```
5B DE 46 E4 35 48 F4 81 45 7C 72 31 54 55 E8 9F
1D D0 5D 9D EC 40 E6 6B 89 F3 BC 52 68 B1 D8 70
35 05 FC 98 F6 36 99 24 53 F0 17 B8 9B D4 A0 5F
12 04 8A A1 A7 96 E6 33 CA 48 84 D9 00 E4 A3 8E
2F 6F 3F 6D E0 1D F8 EA E0 95 BA 63 15 ED 7B 6A
B6 6E 20 17 B5 64 DE 49 64 97 CA 5E 4D 84 63 A0
F1 00 6C EE 70 89 D5 6E C5 05 31 0D AA B7 BA A0
AA BF 98 E8 39 93 70 07 2D FF 42 F9 A4 6F 1B 00
```

6.3 Limitations

6.3.1 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.

6.4 Certificate Parser Details

6.4.1 Parameter Validation

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

- X.509 DER coding and length checks
- Must be v3 only
- Serial number
  - The length must be 1 to 20 byes
- Public Key
  - Only uncompressed Format is supported
  - ECC Curves supported
    - ECC NIST P 256/384/521
    - ECC Brainpool P256r1/ P384r1 / P512r1
      Note: OPTIGA™ Trust M Version 1 doesn't support ECC Brainpool and ECC NIST P 521.
  - RSA 1024/2048 Exponential
  - The public key must be aligned to the key length. In case the public key size is less than the respective key size, then the respective component must be prepended with zeroes before generating the certificate.
- The signature algorithm identifier must not contain the parameters (not even as NULL) in case of ECDSA. (section 2.2.3 in [RFC3279])

```
AlgorithmIdentifier ::= SEQUENCE {
   algorithm OBJECT IDENTIFIER,
   parameters ANY DEFINED BY algorithm OPTIONAL }
```

- Issuer DN must not be Empty.
- Basic Constraints
  - CA - Flag to indicate CA certificate (TRUE/asserted for CA)
  - Path Length Constraint
    - Exists only for CA certificate (if CA flag is TRUE)
- The signature algorithm identifier in tbscertificate and signature tags must be same.

[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.

6.5 Security Guidance

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

6.5.1 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.

6.5.2 Use Case: Host Firmware Update -toolbox-

The shared secret size shall be at least 32 byte to render a brute force useless.

Firmware update 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 respective **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 respective **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

6.5.3 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 respective **key object usage** shall be configured with the **least usage profile** (in most cases just one) as required by the target host application
- The shielded connection shall be enforced in the respective key object EXE access condition to enable

the restricted usage (only with the respective host).

- 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.

6.5.4 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 respective **key object and public key certificate access conditions CHA** shall be configured with **never allowed** (NEV) or allow just protected update with **integrity and confidentiality protection** and the **lifecycle state** of the key and data object (LcsO) shall be **set to operational** (op) to prevent changes to the access conditions.

6.5.5 Static key generation associated to toolbox functionality

The generated static keys are stored in key objects. The attacker attempts to re-generate a key. Doing that the attacker is dropping the static key for encryption / decryption and could be considered as DoS attack.

- After generating the key, the regarded **key object access conditions CHA** shall be configured with **never allowed** (NEV) or allow just protected update with **integrity and confidentiality protection** and the **lifecycle state** of the key object (LcsO) shall be **set to operational** (op) to prevent changes to the access conditions.

6.5.6 Shared secret for key derivation or MAC generation associated to toolbox and protected update functionalities

- A shared secret (could be pre-shared secret or Authorization reference which gets fed in a key derivation or MAC generation and/or verification (e.g. hmac with sha256) operations, 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
- 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 respective critical operations (e.g. key derivation or MAC generation) which doesn't allow the usage with unknown Hosts.
- If the shared secret gets updated on field either by protected update (integrity and confidentiality) or shielded connection, the regarded usage counter must be updated accordingly to allow the usage of shared secret further for the required number of times.

6.5.7 Auto states

- The AUTO state achieved using the respective pre-shared secret (Authorization reference) data object need to be cleared manually as when the respective use case sequence is complete. This can be done using optiga_crypt_clear_auto_state.

6.5.8 Shielded Connection

- The security level of the Shielded connection is as high as typical microcontroller/Host side hardware security level.

Appendix

- The recommended length of Platform binding shared secret is 32 bytes or more.
- 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, AES 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.

6.5.9 Algorithm usage

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

6.6 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.

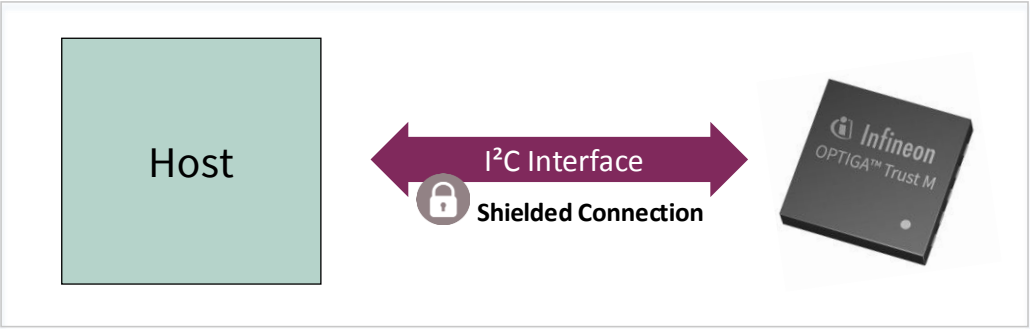


Figure 37 - Overview OPTIGA™ Shielded Connection

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.

6.6.1 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_read and pal_os_datastore_write are the abstracted APIs for reading and

Appendix
writing the platform binding secret at host platform. These APIs are 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.

6.6.2 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.

6.7 Protected Update

The [CDDL] coding for the manifest and signature structures are provided in the package.
This section explains the key derivation and encryption schemes and the pattern of input data to achieve the confidentiality, format of keys & metadata and etc.
Note: OPTIGA™ Trust M Version 1 doesn't support confidentiality and update of keys & metadata.

6.7.1 Payload Confidentiality

As part of protected update, the confidentiality is optional which can be enabled or disabled based on necessity of payload confidentiality requirement. The confidentiality is achieved using the pre-shared secret (protected update secret). The payload encryption key is derived for the respective protected update data set using the "protected update secret".

Appendix

Note: The reference details required for the key derivation and encryption algorithm details are specified in the manifest.

Key Derivation

The below figure depicts the derivation of encryption key.

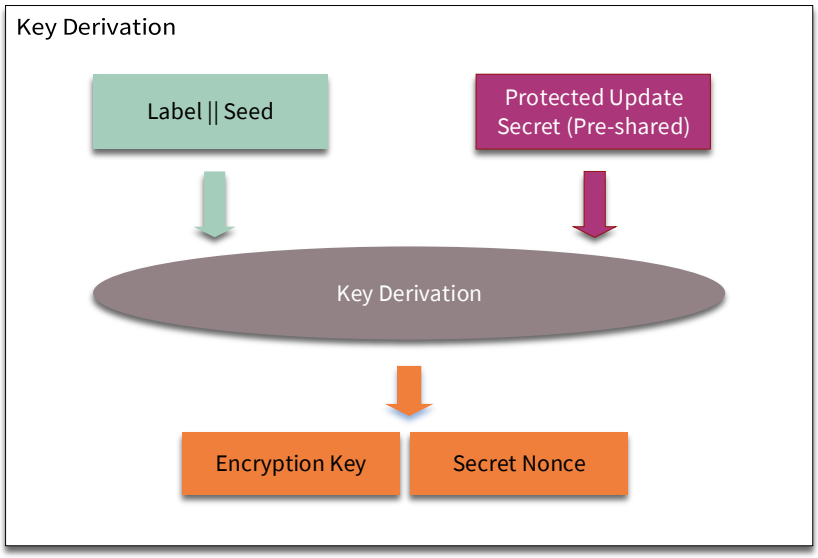


Figure 38 – Protected Update - Derivation for Payload Encryption Key

The key derivation uses the “protected update secret” as input secret and derives the payload encryption key and nonce. The size of the derived encryption key and secret nonce is based on the encryption algorithm chosen.

Encryption & Decryption

The below figure depicts the components (input and output) involved as part of encryption

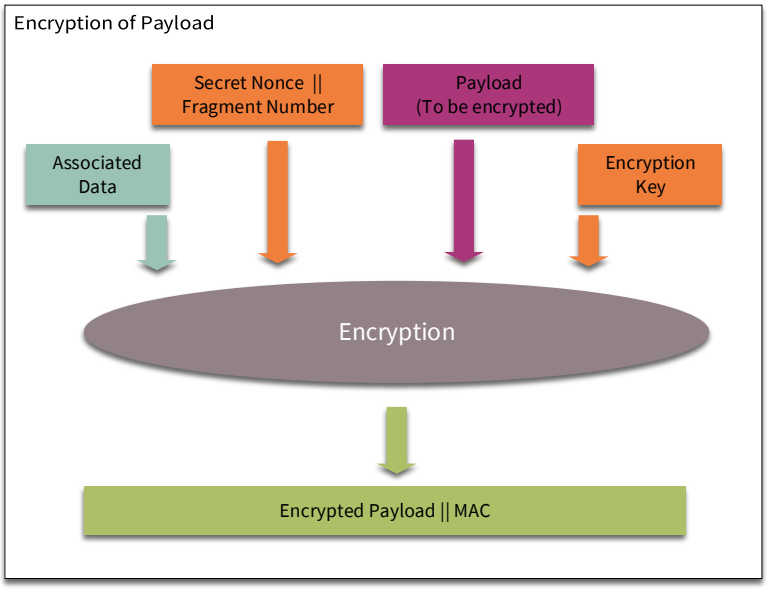


Figure 39 – Protected Update - Payload Encryption

Appendix

The below figure depicts the components (input and output) involved as part of decryption

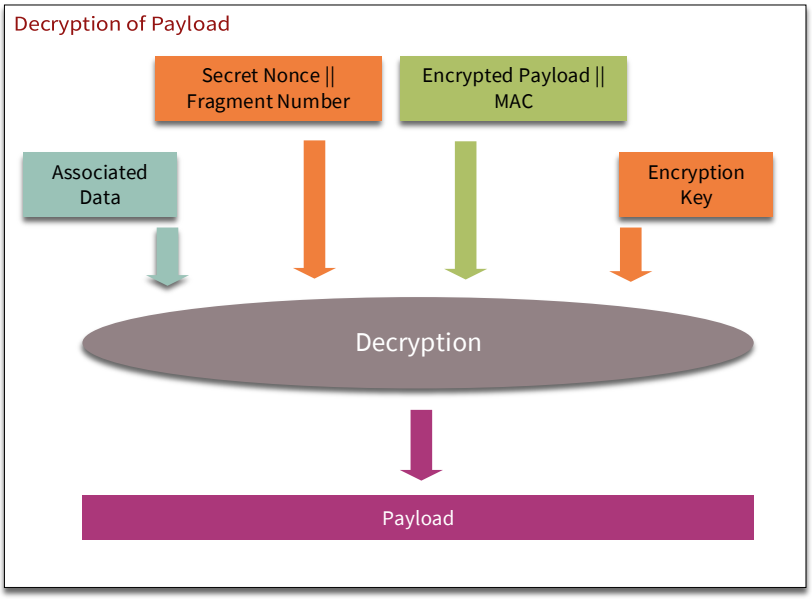


Figure 40 – Protected Update - Payload Decryption

- The fragment number is represented in two bytes in octet string format (e.g. 0x02 is represented as “0x00 0x02”). This is used as part of nonce which gets concatenated with the secret nonce derived using key derivation algorithm.
    - AES-CCM-16-64-128 from [COSE], the nonce length to be used is 13 bytes. Hence we derive 11 bytes secret nonce and concatenate with 2 bytes of fragment number.
- The size of the MAC is based on the encryption algorithm.
- The format of association data is as shown below.

  Associated data = (Payload Version [2] ||
  Offset of payload in fragment [3] ||
  Total payload length [3])

  The above data is represented in octet string with the respective specified lengths. E.g. the total payload length of 0x0568 bytes are represented as “0x00 0x05 0x68”.

  Here the offset of payload is based on the size of the payload considered in the respective fragment. For example, if the total payload to be encrypted is 1500 bytes and encryption is based on AES-CCM-16-64-128 and fragment digest algorithm is SHA256 then the offset values are 0, 600, 1200 in the fragments 1, 2 and 3 respectively as the size of payload in each fragment is about 600 bytes.

The size of the payload (in the respective fragment) to be encrypted based on the encryption (which defines the size of MAC) and digest algorithms chosen.

6.7.2 Format of keys in Payload

As part of protected update data set, the keys (e.g. ECC, RSA, AES, etc.) are provided in specific format as part of payload based on the key type as given below.

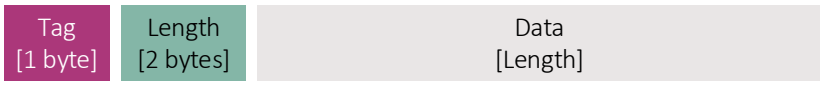


Figure 41 – Protected Update – Encoding of keys in Payload

Appendix

Here Length = 0x120, this will be represented as “01 20”.

6.7.2.1 ECC

- The private key and public key are provided as part of payload with the respective tag.
  - If the target OPTIGA™ doesn’t store the public key, then the public key is optional. If provided, the target OPTIGA™ ignores the provided public key.
  - If the target OPTIGA™ stores the public key, then the public key is must.
- As part of the payload, the length of the key is based on curve type specified in the manifest. For example, in case of ECC NIST P 256, the private key size must be 32 bytes and public key size must be 64 bytes (public key).
  - If the private key or components of public key (x or y) are less than key length of the respective curve type, then the respective component must be prepended with 0x00’s to make it aligned with respective key length.
- The encoding format of keys in the payload is given below. The tags could be in any order in the payload.

  The value for private key tag is 0x01.

  The value for the public key tag is 0x02.

  E.g. The ECC NIST P 256 keys are as given below [value shown below are in hex format].

  <01> < *00 20*> <*29 2E FD 39 …. 4C 74 BC C8*>

  <02> < *00 40*> <*91 8A 12 34 …. 8A 98 1A 52* || 1A *1B 1C 1D …. 2A 2B 2C 2D*>

  E.g. The ECC NIST P 256 keys are as given below with padding if the respective component size less than key size [value shown below are in hex format].

  <01> < *00 20*> <*00 00 FD 39 …. 4C 74 BC C8*>

  <02> < *00 40*> <*00 8A 12 34 …. 8A 98 1A 52* || 1A *1B 1C 1D …. 2A 2B 2C 2D*>

6.7.2.2 RSA

- In case of RSA, the private exponent, modulus and public exponent must be provided as part of payload with the respective tag.
- The length of the private exponent and modulus must be strictly aligned to the respective key type/algorithm specified in the manifest and the public exponent is always 4 bytes.
  - If any of these components is of lesser size, then the component must be prepended with 0x00’s to align it with the respective length.
- The tag value for the components as specified below. The tags could be in any order in the payload.

  The tag value for private exponent is 0x01.

  The tag value for the modulus is 0x02.

  The tag value for the public exponent is 0x03.

  E.g. RSA 1024 keys are as given below [value shown below are in hex format].

  Private exponent: <01> <*00 80*> <data = 1A 2F 3D 37 …. 5A 4B E5 F0>

  Modulus: <02> <*00 80*> <data = 29 2E FD 39 …. 4C 74 BC C8>

  Public exponent: <03> <*00 04*> <data = 00 10 00 01 >

6.7.2.3 AES

- Only the symmetric key is provided as part of payload.

Appendix

- The length of the symmetric key must be strictly aligned to the respective key type/algorithm specified in the manifest.
- The tag value for the components as specified below.

  The tag value for symmetric key is 0x01.

  E.g., the AES 128 key is as specified below [value shown below are in hex format].

  Symmetric Key: <01> <*00 10*> <data = 79 28 49 62 …. 12 58 37 61>

6.7.3 Metadata update

- The payload contains the metadata to be updated in target OPTIGA™ OID metadata.

  The following metadata tags must not be part of payload.

  - Version [0xC1]
  - Maximum size of the data object [0xC4]
  - Used size of the data object [0xC5]
  - Algorithm associated with key container [0xE0]

- The format of metadata in the payload is as same as in GetDataObject command.

```
0x20, 0x0B,                     // TL metadata TLV-Object
   0xC0, 0x01, 0x03,            // TLV LcsO = in
   0xD1, 0x01, 0x00,            // TLV Read = ALW
   0xD0, 0x03, 0xE1, 0xFC, 0x07 // TLV Change = LcsO < op
```

- At OPTIGA™,
  - The reset type (F0) tag must be available in metadata of the Target OPTIGA™ OID to be updated and the metadata update descriptor are verified whether to allow the protected update or not.
  - If the new metadata (in payload) contains the LcsO tag, then the LcsO in the target OPTIGA™ OID metadata gets with this value and the LcsO value specified in the metadata update Identifier is ignored.
  - If the new metadata (in payload) does not contain the LcsO tag, then the LcsO in the target OPTIGA™ OID current metadata gets updated with the value specified in the metadata update Identifier of the current metadata.
  - The payload version specified in the manifest gets updated in the target OPTIGA™ OID metadata.
  - The tags which are specified in the new metadata (in payload), gets replaced in the target OPTIGA™ OID metadata and the remaining tags in target OPTIGA™ OID metadata still remain unchanged.
  - If the new metadata (in payload) does not contain any tags (e.g. 0x20 0x00), then the rules as per the metadata update identifier in the current metadata are applied.
  - The protected metadata update allows setting the LcsO of respective data/key object to Termination state. Once the data/key object is set to Termination state, the target OID is not allowed to be used in any read / execute operations.

6.7.4 CDDL Tool

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

Additionally, the protected update data set generation tool is available as part of package.

6.8 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,

Appendix

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
Class	is a fundamental concept of object-oriented programming. It contains basically the implementation of methods. Upon instantiation the particular context is created and makes up an object of that type of class.	
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
deadline	In Real-Time concepts a deadline is the time after action initiation by which the action must be completed.	
Denial of Service	In computing, a denial-of-service attack (DoS attack) is a cyber-attack in which the perpetrator seeks to make a machine or network resource unavailable to its intended users by temporarily or indefinitely disrupting service of a host or network.	DoS
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	
hot spot	in *Non-Volatile Memory* technologies a *hot spot* is a very often written data object	
latency	In Real-Time concepts *latency* is a time interval between the stimulation (interrupt, event ...) and response, or, from a more general point of view, as a time delay between the cause and the effect of some physical change in the system being observed.	
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,	NVM

Appendix

Term	Description	Abbreviation
	optical discs, and early computer storage methods such as paper tape and punched cards.	
object	in object oriented programming an *object* is an instance of a *Class*.	
programming NVM	*programming NVM* comprises of erase followed by write to the *Non-Volatile Memory*	
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
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
Trust Anchor	A Trust Anchor represents an authoritative entity via a public key and associated data. The public key is used to verify digital signatures, and the associated data is used to constrain the types of information for which the trust anchor is authoritative.	
Trust Anchor Store	A Trust Anchor Store is a set of one or more trust anchors stored in a device. A device may have more than one trust anchor in the store, each of which may be used by one or more applications.	

Table 49 - Terms of OPTIGA™ Vocabulary

6.9 Change History

Version	Date	Description
3.00	2020-06-01	• Release version
3.05	2020-06-16	• Additional information provided w.r.t OPTIGA™ Trust M Version 1 and Version 3 compatibility •

Edition 2020-06-01

Published by

Infineon Technologies AG

81726 Munich, Germany

Do you have a question about this document?

Email:
DSSCustomerService@infineon.com

Document reference

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.