OPTIGA™ Trust Charge

Product Version: V1

About this document

Scope and purpose

The scope of this document is the OPTIGA™ Trust Charge 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 Charge, 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.

Table of Contents

Table of Contents

Table of Contents

1 Introduction

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

1.1 Abbreviations

Table 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
DO	Data Object
DoS	Denial of Service
DRNG	Deterministic Random Number Generator
DTLS	Datagram Transport Layer Security
EAL	Evaluation Assurance Level
ESW	Embedded Software
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
TRNG	True Random Number Generator

Abbreviation	Term
UID	Unique Identifier
µC / MCU	Microcontroller

1.2 Naming Conventions

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

Table 2 Naming Conventions

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

1.3 References

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

Table 3 References

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

Name	Description
	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]
[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 Charge]	OPTIGA™ Trust Charge - 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
[IFX_I2C]	Infineon Technologies AG; IFX I2C Protocol Specification
[I^2C]	http://www.nxp.com/documents/user_manual/UM10204.pdf www.nxp.com/documents/user_manual/UM10204.pdf NXP; UM10204 I^2C-bus specification and user manual
[RFC2986]	https://tools.ietf.org/html/rfc2986 Certificate Request Syntax Specification Version 1.7
[RFC5116]	https://tools.ietf.org/html/rfc5116 An Interface and Algorithms for Authenticated Encryption
[RFC5280]	https://tools.ietf.org/html/rfc5280 Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile
[SP 800-38A]	https://nvlpubs.nist.gov/nistpubs/Legacy/SP/nistspecialpublication800-38a.pdf Recommendation for Block cipher modes of operation.
[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 800-90A]	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]

Name	Description
[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™ Trust Charge External Interface, OPTIGA™ Trust Charge 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 Charge External Interface.
- OPTIGA™ Trust Charge External Interface provides the necessary details of the external interface to utilize the OPTIGA™ functionality.
- OPTIGA™ Trust Charge 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

The architecture components contained in the shown solution architecture view (OPTIGA Trust IP Protection View -Toolbox-) are listed and briefly described in the table below.

Table 4 Architecture components

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]. optiga_comms_ifx_i2c usage characteristic is asynchronous, were the caller has to take care of the correct sequence of calls for a

Supported Use Cases

Name	Description
	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, etc.) 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.

The class diagram 'OPTIGA Trust IP Protection View -Toolbox-' shows the IP Protection Solution Architecture containing its main functional blocks.
This view is applied for IP protection solution kind of use cases, where the involved blocks are represented as dedicated lifelines.

Supported Use Cases

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.

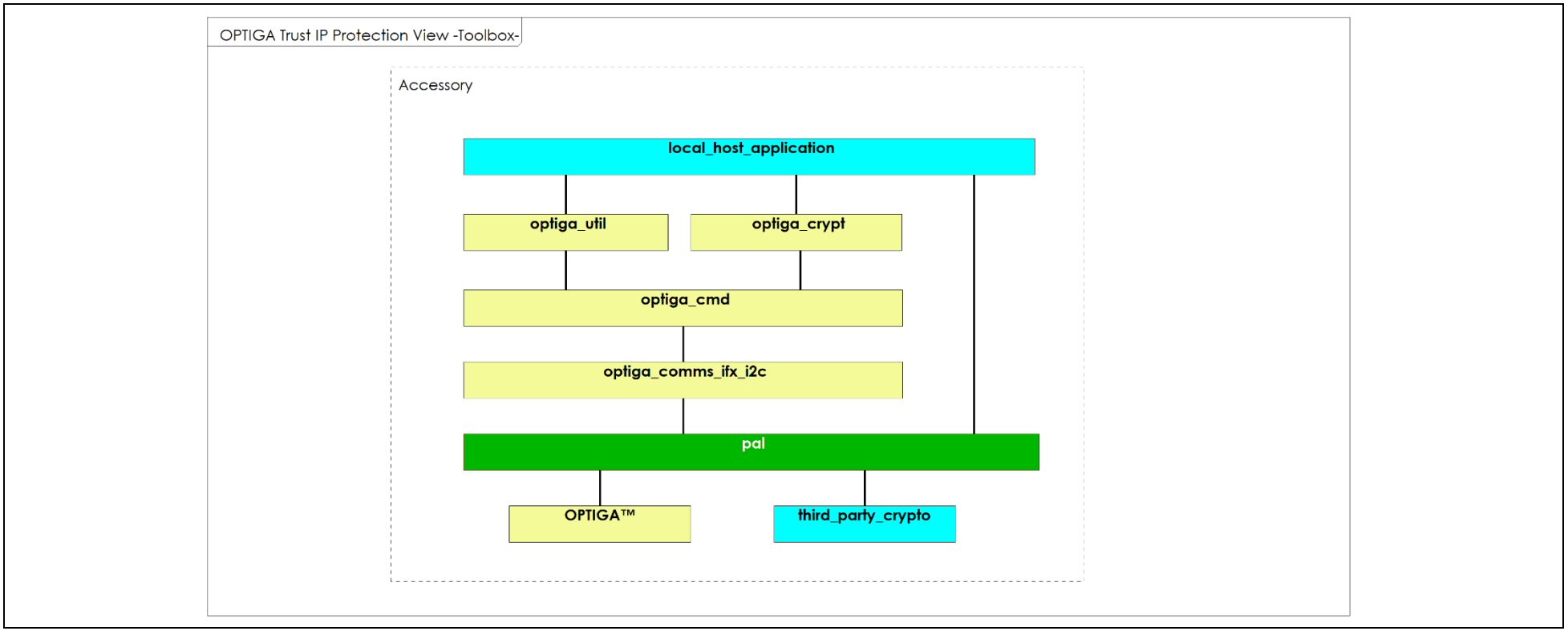

Figure 1 - OPTIGA Trust IP Protection View -Toolbox-

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.

Table 5 Host library – Code and RAM size details

Configuration	CODE	RAM
[without the Shielded Connection] The components optiga_crypt, optiga_util, optiga_cmd, optiga_comms_ifx_i2c, and pal are covered.	11.5 Kbytes	5 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).	25 Kbytes	14 Kbytes

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™.
This sequence diagram 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.

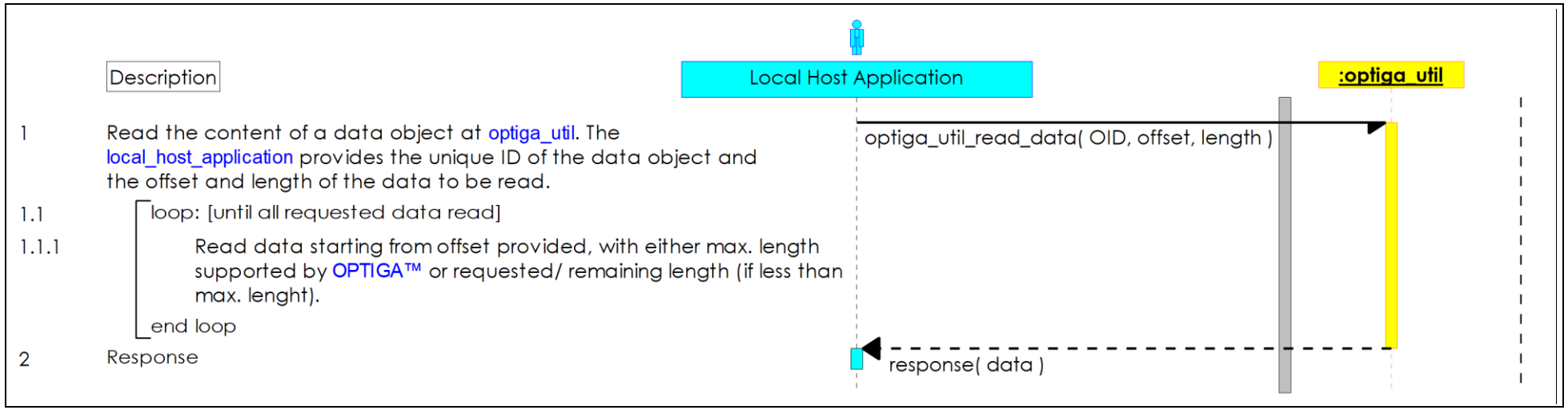

Figure 2 - Use Case: Read General Purpose Data - data object

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™.
This sequence diagram 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

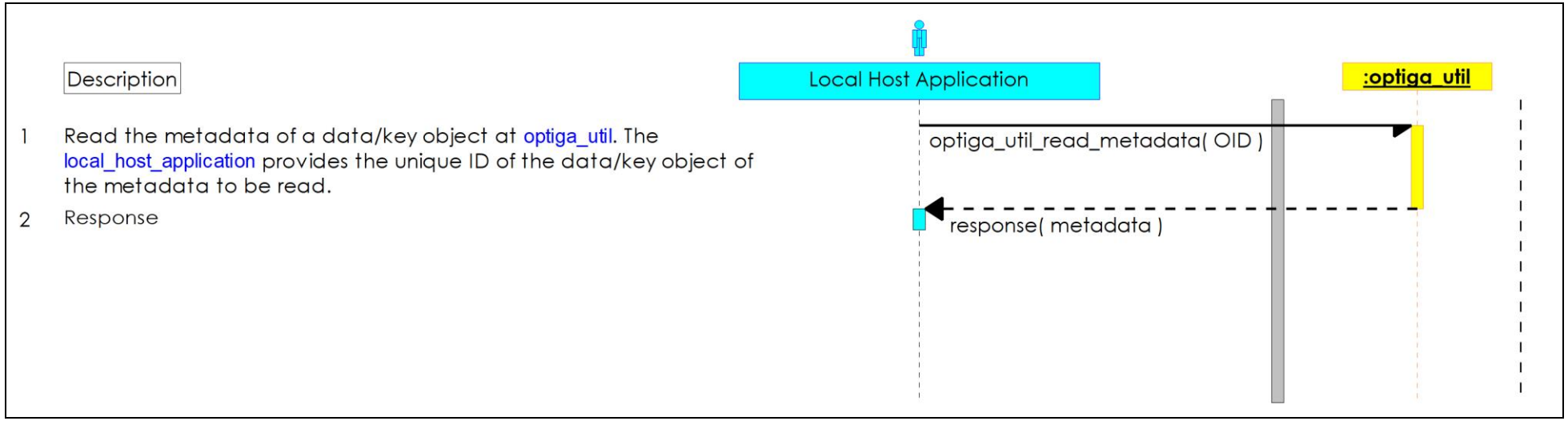

Figure 3 - Use Case: Read General Purpose Data - metadata

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™.
This sequence diagram 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

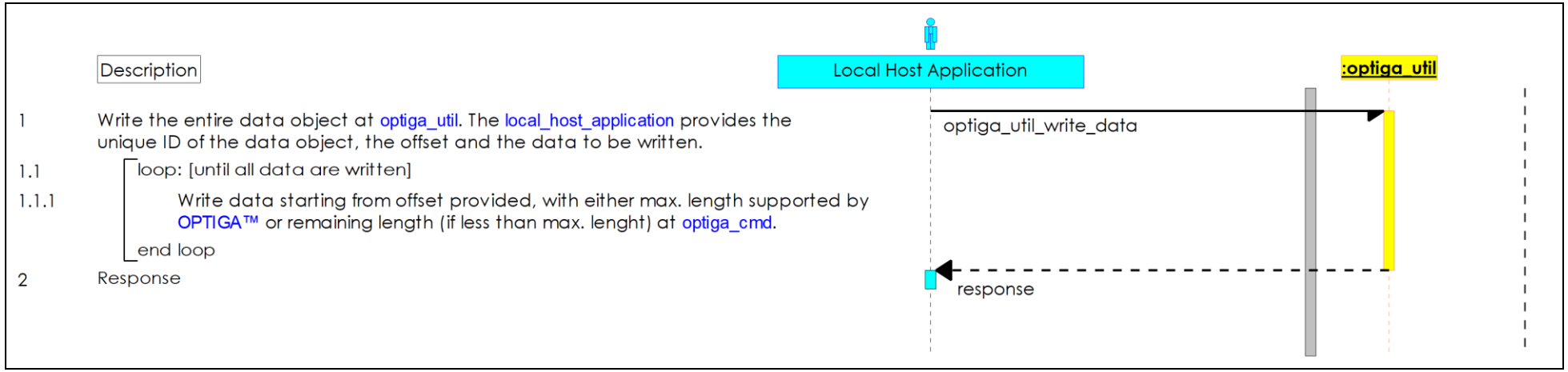

Figure 4 - Use Case: Write General Purpose Data - data object

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™.
This sequence diagram 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

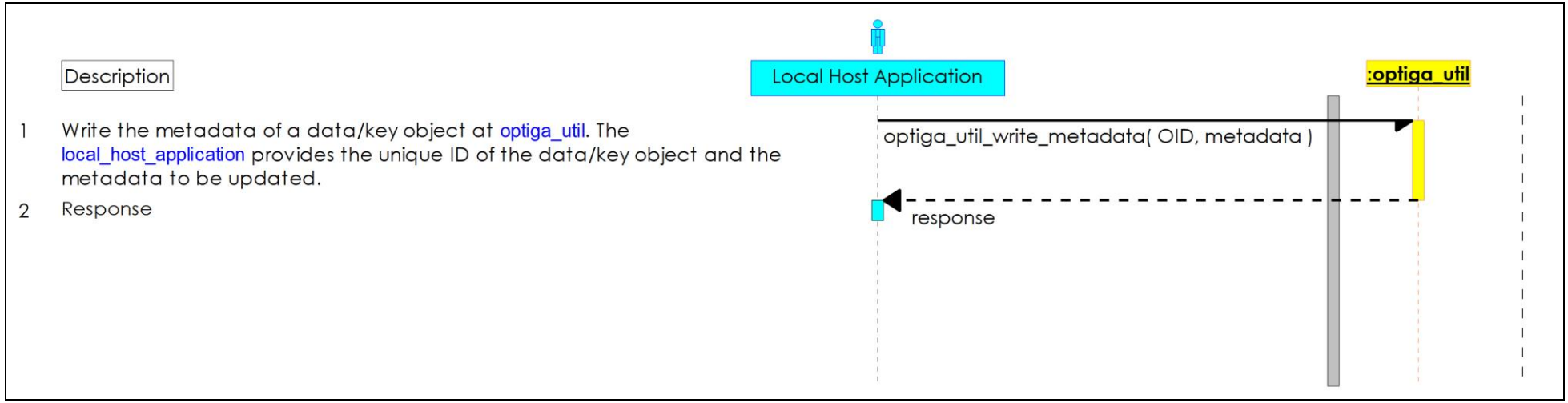

Figure 5 - Use Case: Write General Purpose Data - metadata

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 the integrity protection and upon success updates the target data object.

Pre-condition(s):

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

Post-condition:

- The target data object is updated.

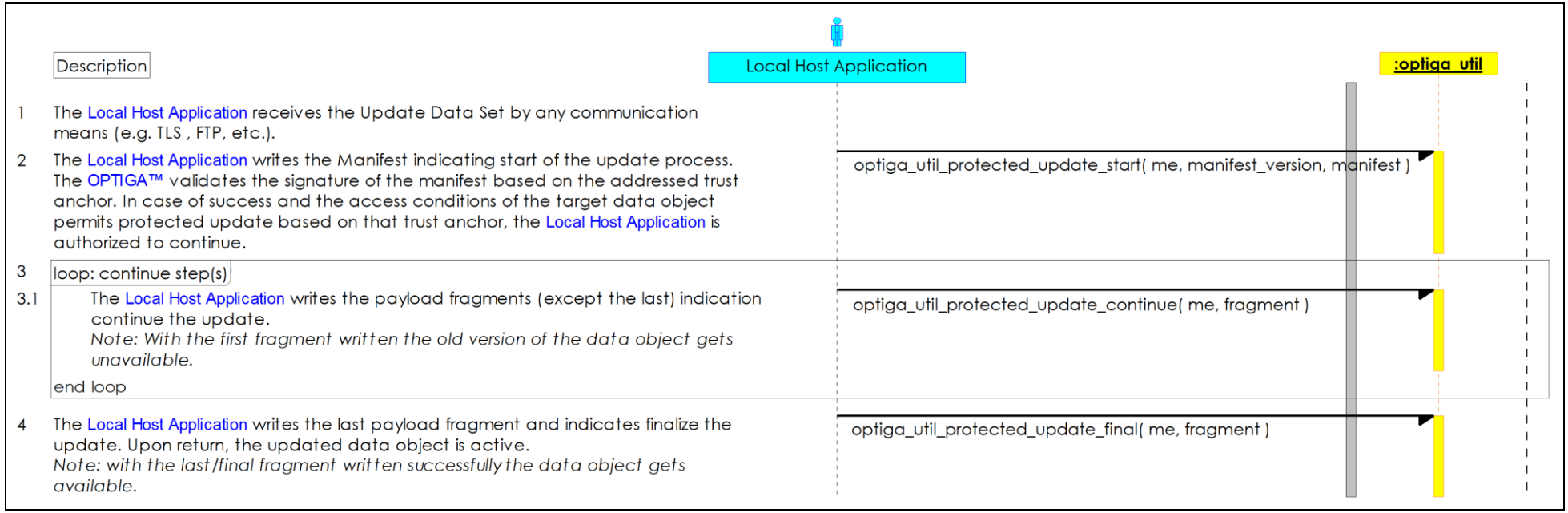

Figure 6 - Use Case: Integrity Protected Update of a data object

2.3 Sequence Diagrams utilizing cryptographic toolbox functionality

2.3.1 Use Case: One-way Authentication - IP Protection -toolbox-

The local_host_application (refer to OPTIGA Trust IP Protection View -Toolbox-) likes to verify the authenticity of the OPTIGA™. In case the authenticity verification succeeds, the local host considers itself not being cloned.
The local_host_application provides digest of a challenge (random value) and the OPTIGA™ simply applies a signature on it and returns the signature as response by which it proofs its authenticity.
This sequence diagram is provided to show the functions involved in performing a one-way authentication based on a public key signature scheme.
The pal_crypt_xyz APIs specified in the below diagram are not part of reference implementation provided, but given as a guidance.

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 (CA certificate) which signed OPTIGA™ device certificate is installed at host.

Post-condition:

- The local_host_application considers the OPTIGA™ as an authentic member of the target PKI domain.

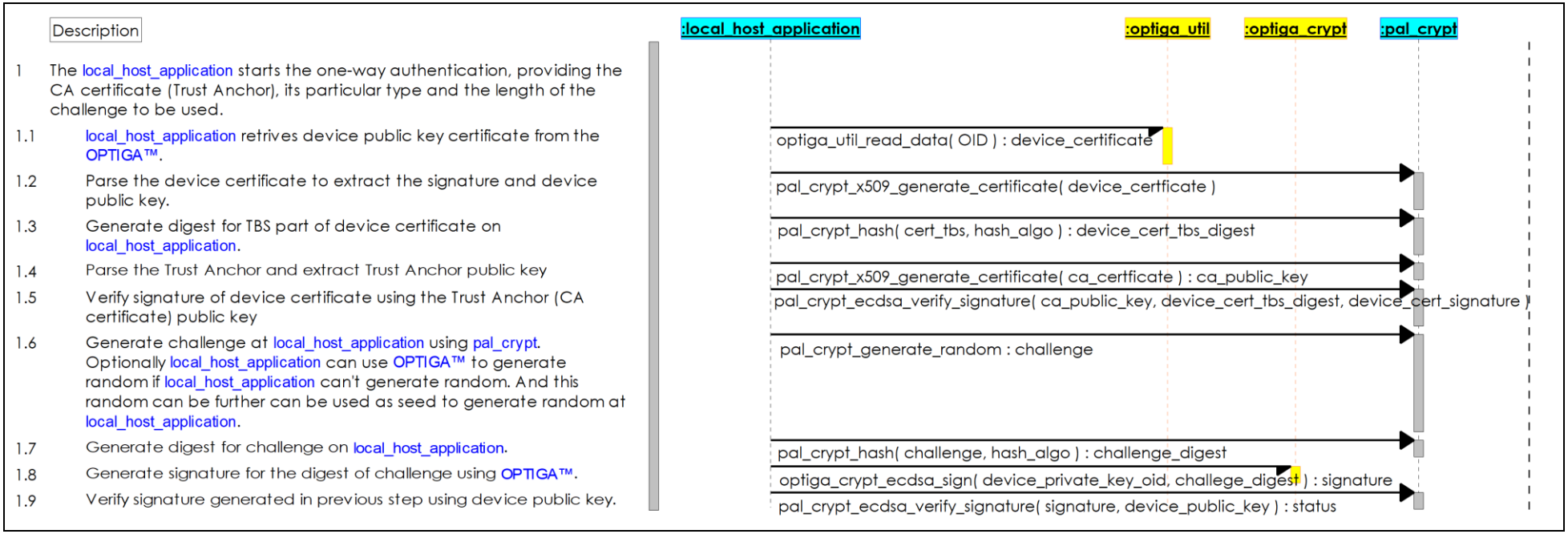

Figure 7 - Use Case: One-way Authentication -IP Protection- toolbox-

2.3.2 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 sequence diagram 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).

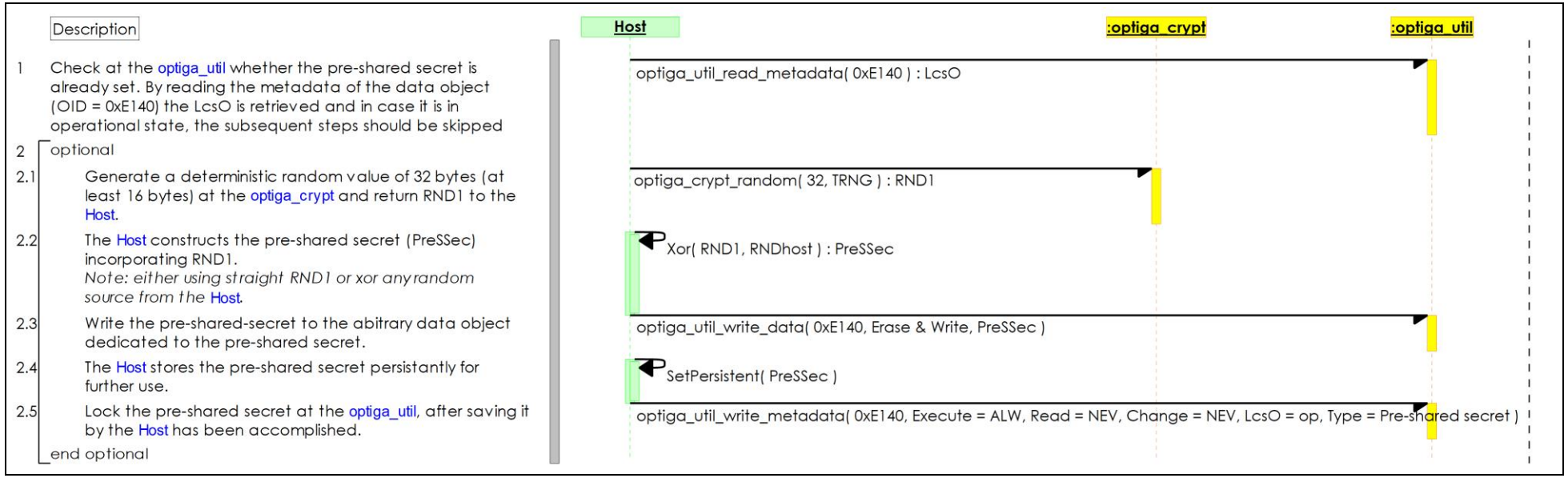

Figure 8 - Use Case: Pair OPTIGA™ with Host (Pre-Shared Secret based)

2.3.3 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.

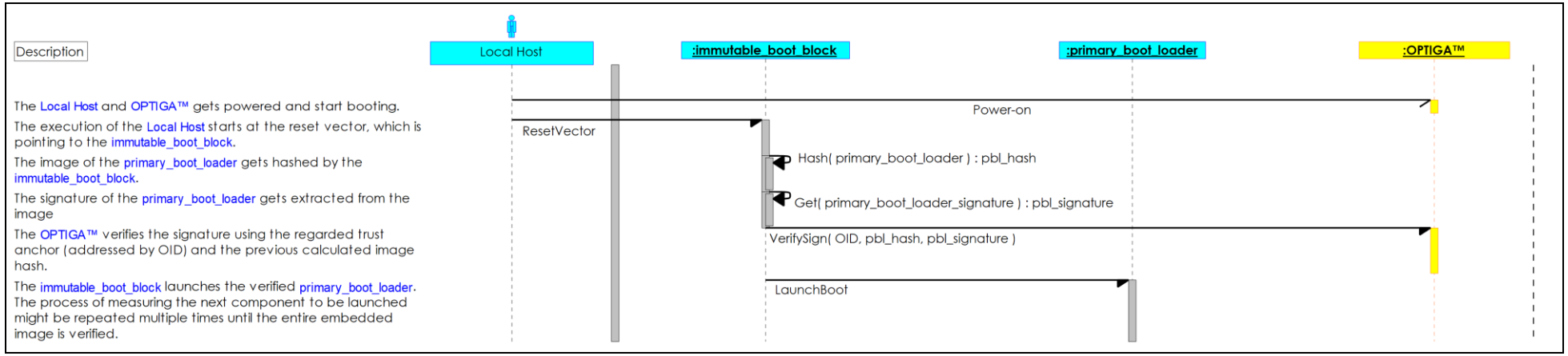

Figure 9 - Use Case: Verified Boot -toolbox-

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

This sequence diagram 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.

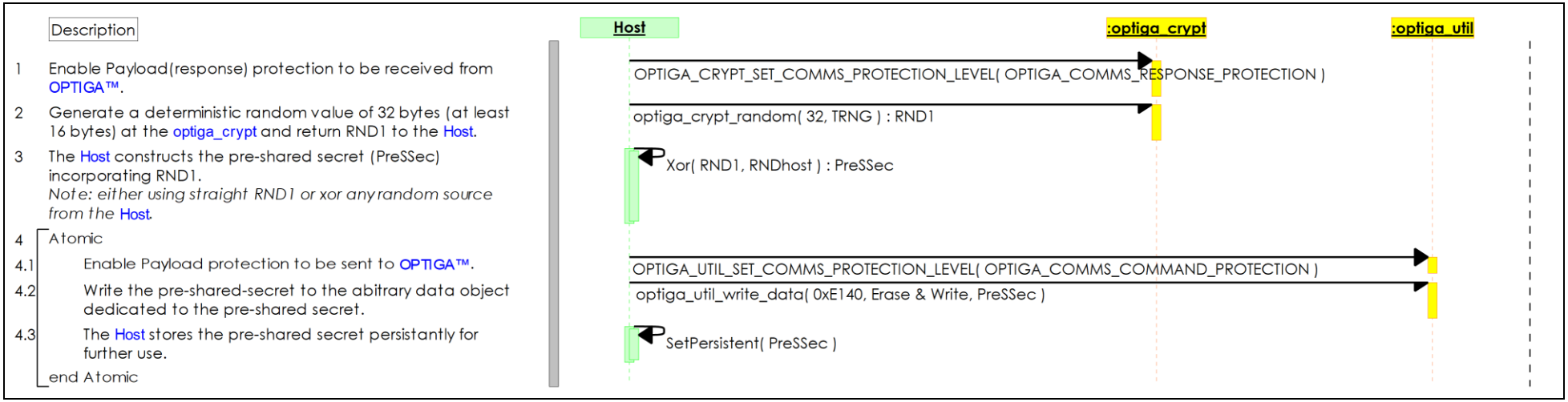

Figure 10 - Use Case: Update Platform Binding Secret during runtime (Pre-Shared Secret based)

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

A host needs to protect data against access by any third party. This sequence diagram 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.

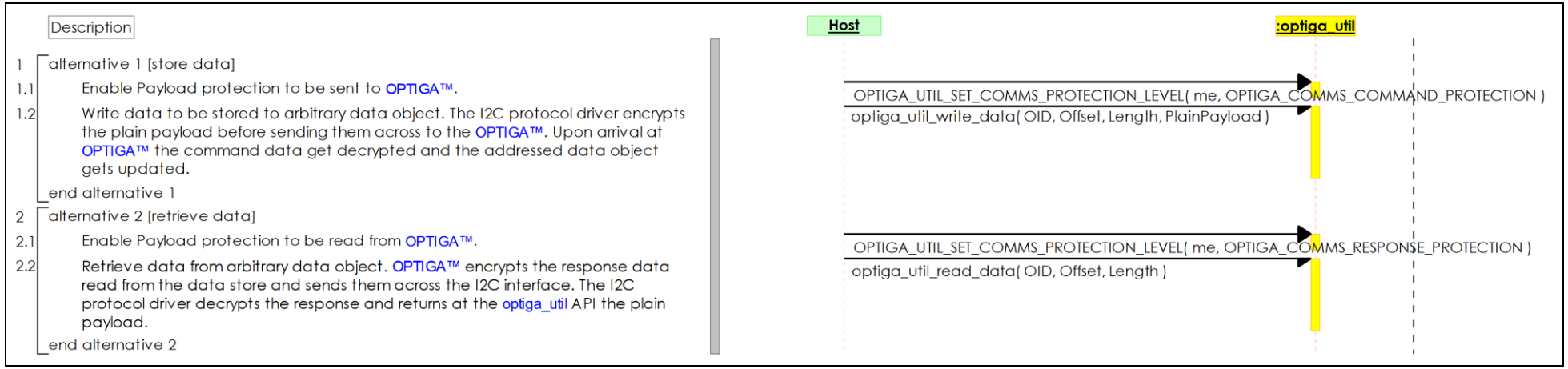

Figure 11 - Use Case: Local "data-at-rest" and "data-in-transit" protection

2.3.6 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.

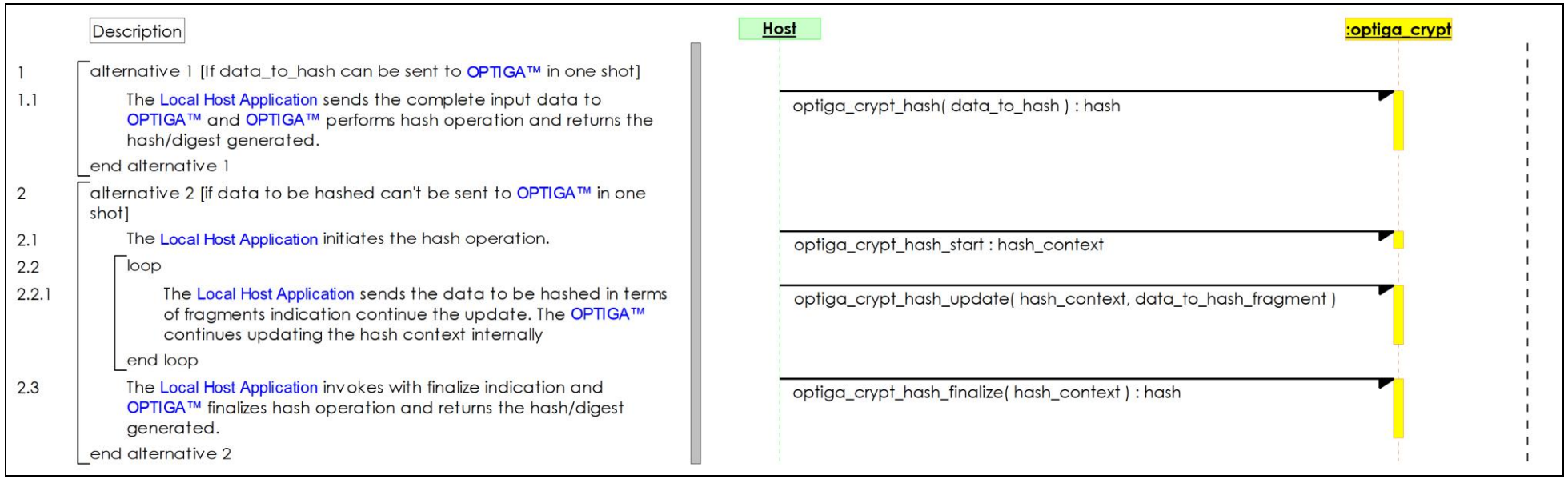

Figure 12 - Use Case: Generate Hash

3 Enabler APIs

This 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 host side library architecture and it's main functional 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.

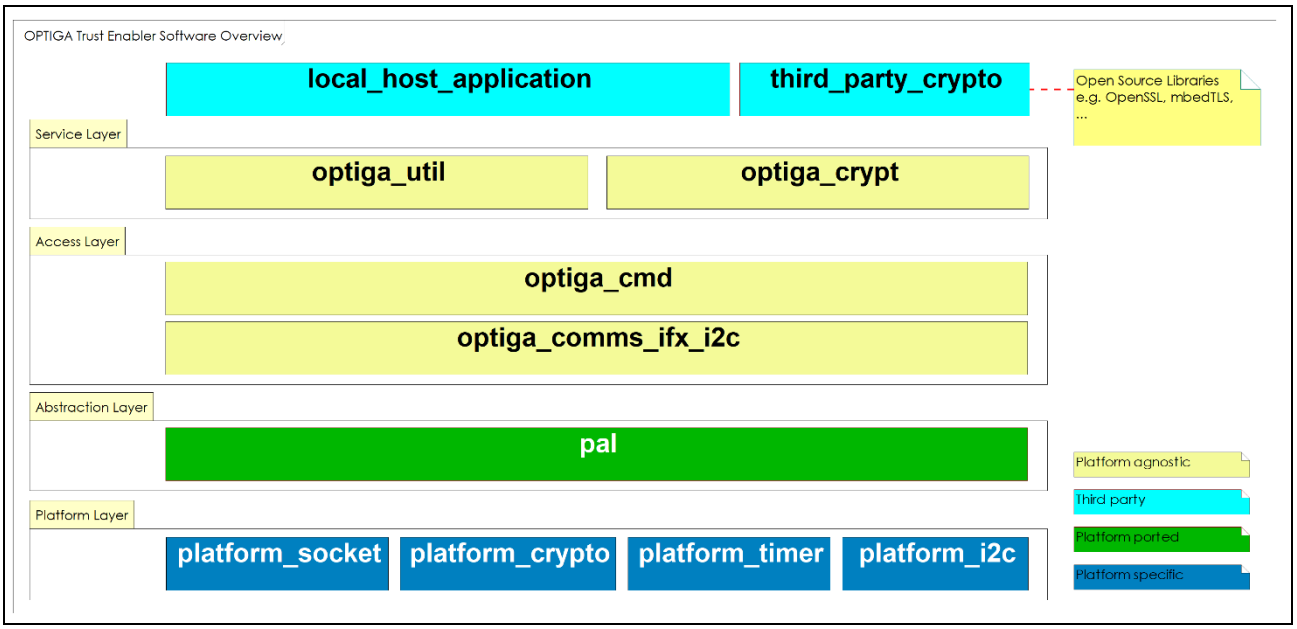

Figure 13 – OPTIGA™ Trust Enabler Software Overview

3.1 Service Layer Decomposition

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

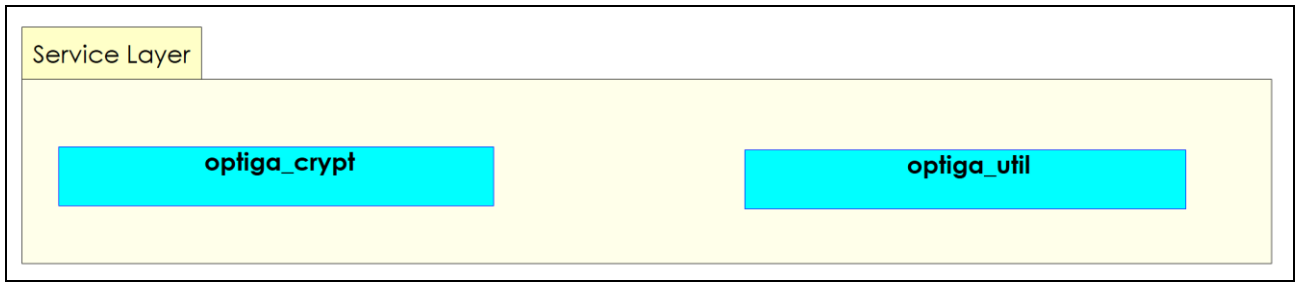

Figure 14 - Service Layer Decomposition

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 Basic (e.g. initialization, shielded connection settings) operations

Table 6 optiga_crypt - Basic (e.g. initialization, shielded connection settings) APIs

API Name	Description
optiga_crypt_create	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.
optiga_crypt_destroy	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.
optiga_crypt_set_comms_params	This operation sets the shielded connection(Encrypted communication between Host and OPTIGA™) parameters like version, protection level, etc. 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). There are macros defined based on this API to ease the usage of shielded

Enabler APIs

API Name	Description
	connection to set parameters and levels of protection. • OPTIGA_CRYPT_SET_COMMS_PROTOCOL_VERSION • OPTIGA_CRYPT_SET_COMMS_PROTECTION_ LEVEL

3.1.1.2 Random generation operations

Table 7 optiga_crypt – Random generation APIs

API Name	Description
optiga_crypt_random	This operation generates random data using OPTIGA™.

3.1.1.3 Hash operations

Table 8 optiga_crypt - Hash APIs

API Name	Description
optiga_crypt_hash	This operation performs the hash operation using OPTIGA™ for the provided data and returns the digest. 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.
optiga_crypt_hash_start	This operation initializes OPTIGA™ to hash the data further using optiga_crypt_hash_update.
optiga_crypt_hash_update	This operation performs the hashing for the given data (could be either host or referring to a readable data object from OPTIGA™) and updates the hash context using OPTIGA™.
optiga_crypt_hash_finalize	This operation finalizes the hash.

3.1.1.4 ECC based operations

Table 9 optiga_crypt - ECC based APIs

API Name	Description
optiga_crypt_ecc_generate_keypair	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.
optiga_crypt_ecdsa_sign	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.
optiga_crypt_ecdsa_verify	This operation verifies the signature (ECDSA) using OPTIGA™. The public key could be

API Name	Description

	• Either the public key is from host, the format of public key (from host) is provided in ECC Public Key.
	• Or the public key is from a data object at OPTIGA™,
	  • The data object type of OID must be set to either Trust Anchor or Device Identity.
	  • The data object must contain only single X.509 certificate (ASN.1 DER encoded) and maxium size of certificate allowed is 1300 bytes. More details about certificate parameters are specified in Parameter Validation section.

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 Basic (e.g. initialization, shielded connection settings) operations

Table 10 optiga_util - Basic (e.g. initialization, shielded connection settings) APIs

API Name	Description
optiga_util_create	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).
optiga_util_destroy	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.
optiga_crypt_set_comms_params	This operation sets the shielded connection(Encrypted communication between Host and OPTIGA™) parameters like version, protection level, etc. 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). There are macros defined based on this API to ease the usage of shielded connection to set parameters and levels of protection. • OPTIGA_UTIL_SET_COMMS_PROTOCOL_VERSION • OPTIGA_UTIL_SET_COMMS_PROTECTION_ LEVEL

3.1.2.2 Open and Close operations

Table 11 optiga_util - Open and close APIs

API Name	Description
optiga_util_open_application	This operation initializes or restores (from a hibernate state if performed) the application on OPTIGA™.

Enabler APIs

API Name	Description
	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.
optiga_util_close_application	This operation closes the application or on OPTIGA™. With the hibernate option, the OPTIGA™ stores the current context of application and restores with next optiga_util_open_application. 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 to keep the session context intact. In this operation, after OPTIGA™ confirms the storing of state/context (command is successfully executed), the Access Layer switches off the OPTIGA™ (if 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.

3.1.2.3 Read and Write operations

Table 12 optiga_util - Read and write APIs

API Name	Description
optiga_util_read_data	This operation reads the data from the specified data object in OPTIGA™.
optiga_util_read_metadata	This operation reads the metadata from the specified data object in OPTIGA™.
optiga_util_write_data	This operation writes data to the specified data object in OPTIGA™. 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.
optiga_util_write_metadata	This operation writes metadata to the specified data object in OPTIGA™.
optiga_util_update_count	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.

API Name	Description
	Any further attempts after reaching the threshold value, the OPTIGA™ returns an error.

3.1.2.4 Protected update operations

Table 13 optiga_util - Protected update APIs

API Name	Description
optiga_util_protected_update_start	This operation initiates the protected update of data objects in OPTIGA™. The manifest provided will be validated by OPTIGA™. Upon the successful completion of this operation, The fragments (which contain the data to be updated) are to be sent using optiga_util_protected_update_continue and/or optiga_util_protected_update_final. The protected update needs to be performed in a strict sequence. The 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. 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.
optiga_util_protected_update_continue	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. 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.
optiga_util_protected_	This operation sends the last fragment and finalizes the protected update of

Enabler APIs

API Name	Description
update_final	OPTIGA™ data object and releases the strict lock acquired. The size of the fragment can be up to 640 bytes.

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.

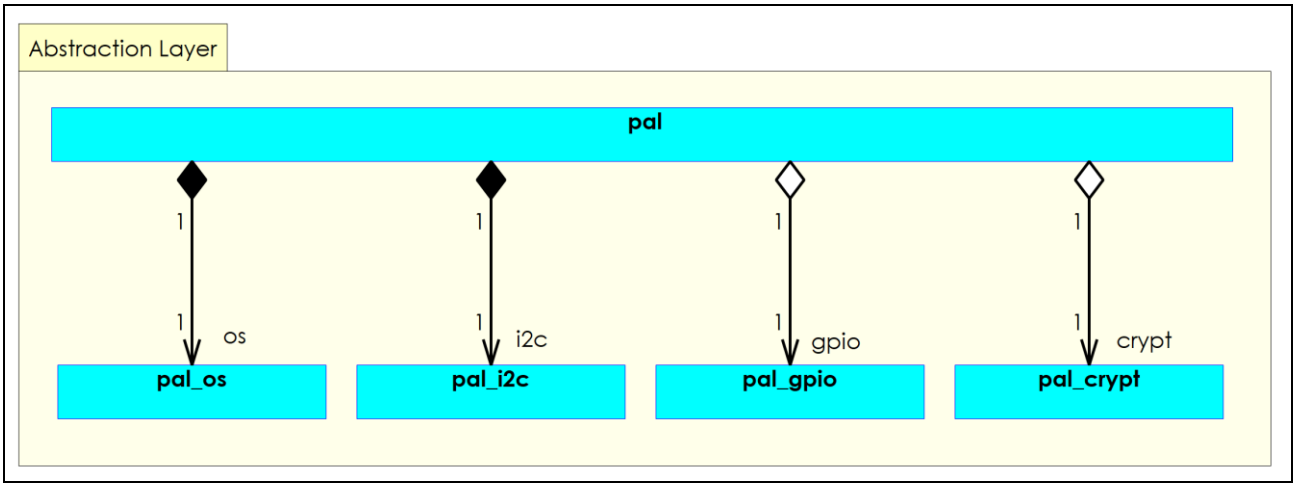

Figure 15 - Abstraction Layer Decomposition

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.

Table 14 pal APIs

API Name	Description
pal_init	This operation initializes the pal and aggregated pal modules (e.g. pal_i2c_init, pal_gpio_init, pal_os_init, etc.).
pal_deinit	This operation deinitializes the pal and aggregated pal modules (e.g. pal_i2c_deinit, pal_gpio_deinit, pal_os_deinit, etc.).

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.

Table 15 pal_crypt APIs

API Name	Description

Enabler APIs

API Name	Description
pal_crypt_tls_prf_sha256	This operation derives the secret using the TLSv1.2 PRF SHA256 for a given shared secret.
pal_crypt_encrypt_aes128_ccm	This operation encrypts the given plain text using the provided encryption key and nonce.
pal_crypt_decrypt_aes128_ccm	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.

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

Table 16 pal_gpio APIs

API Name	Description
pal_gpio_init	This operation initializes the lower level driver of gpio.
pal_gpio_deinit	This operation de-initializes the lower level driver of gpio.
pal_gpio_set_high	This operation sets the gpio pin state to high.
pal_gpio_set_low	This operation sets the gpio pin state to low.

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).

Table 17 pal_i2c APIs

API Name	Description
pal_i2c_init	This operation initializes the lower level driver of i2c.
pal_i2c_deinit	This operation de-initializes the lower level driver of i2c.
pal_i2c_read	This operation reads the data from I2C bus.
pal_i2c_write	This operation writes the data to I2C bus.
pal_i2c_set_bitrate	This operation sets the bit rate (in kHz) of I2C master.

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.

Table 18 pal_os APIs

Enabler APIs

API Name	Description
pal_os_datastore_read	This operation abstracts the reading of data from the specified location in the host platform.
pal_os_datastore_write	This operation abstracts the writing of data to the specified location in the host platform.
pal_os_event_create	This operation initializes (creates optionally) returns context to the event for the later use.
pal_os_event_register_callback_oneshot	This operation registers the callback and context. The callback will invoked pal_os_event_trigger_registered_callback with the given context after the provided time.
pal_os_event_trigger_registered_callback	This operation invokes the registered callback with the given context once the time out is triggered.
pal_os_event_start	This operation starts the event management operation.
pal_os_event_stop	This operation stops the event management operation.
pal_os_event_destroy	This operation destroys the event.
pal_os_timer_init	This operation initializes the timer on the host platform.
pal_os_timer_get_time_in_milliseconds	This operation provides the current time stamp in milliseconds.
pal_os_timer_get_time_in_microseconds	This operation provides the current time stamp in microseconds.
pal_os_timer_delay_in_milliseconds	This operation induces a delay of provided milliseconds.
pal_os_timer_deinit	This operation de-initializes the timer on the host platform.
pal_os_lock_enter_critical_section	This operation allows to enter critical section.
pal_os_lock_exit_critical_section	This operation allows to exit from critical section.
pal_os_malloc	This operation allocates memory.
pal_os_calloc	This operation allocates a clean (set to all 0's) memory.
pal_os_free	This operation frees the memory.
pal_os_memcpy	This operation copies the number of bytes (size) from p_source to p_destination.
pal_os_memset	This operation copies the first number of bytes (size) of p_buffer with the value (byte) specified.

3.3 Data Types

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

3.3.1 Enumerations

Types of ECC Curves supported by OPTIGA™

Table 19 optiga_ecc_curve_t

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

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

Table 20 optiga_hash_context_length_t

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

Types of digest/hash generation supported by OPTIGA™

Table 21 optiga_hash_type_t

Name	Description
OPTIGA_HASH_TYPE_SHA_256	Generate digest using SHA256

Key slot IDs in OPTIGA™

Table 22 optiga_key_id_t

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_SESSION_BASED	Key from session context (volatile).

Types of Key usage.

Table 23 optiga_key_usage_t

Name	Description
OPTIGA_KEY_USAGE_SIGN	Allows to use the private key for the signature generation as part of sign command

Types of random number generation supported by OPTIGA™

Table 24 optiga_rng_type_t

Name	Description
OPTIGA_RNG_TYPE_TRNG	Generate Random number using TRNG

Enabler APIs

Name	Description
OPTIGA_RNG_TYPE_DRNG	Generate Random number using DRNG

4 OPTIGA™ Trust Charge External Interface

This chapter 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.

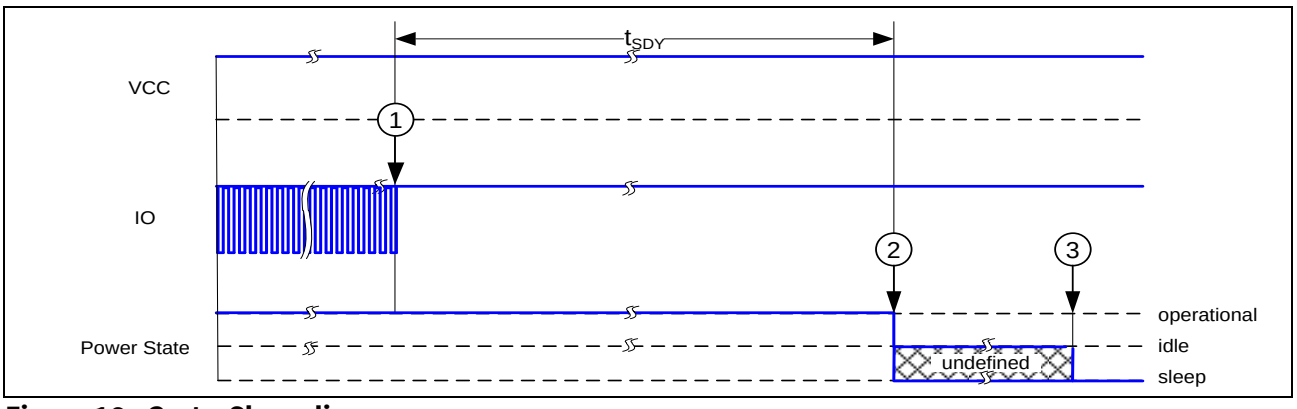

Figure 16 - Go-to-Sleep diagram

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.

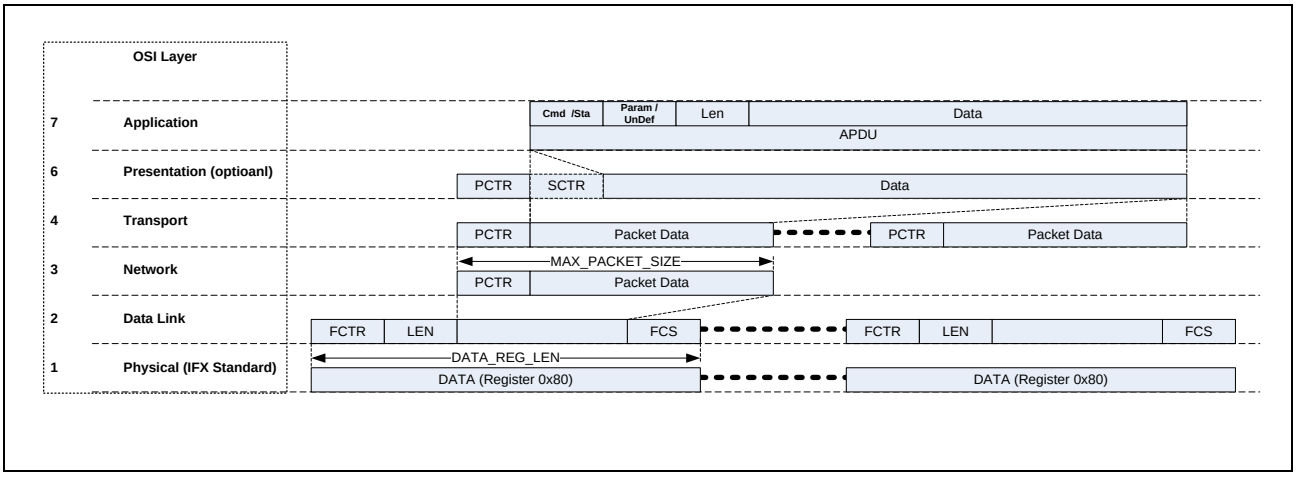

Figure 17 - Overview protocol stack used

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".

Table 25 Protocol stack variation

Property	Value	Notes
MAX_PACKET_SIZE	0x110	
WIN_SIZE	1	
MAX_NET_CHAN	1	
CHAINING	TRUE	
TRANS_TIMEOUT	10	ms
TRANS_REPEAT	3	

Property	Value	Notes
PWR_SAVE_TIMEOUT		Not implemented
BASE_ADDR	0x30	I2C base address default
MAX_SCL_FREQU	1000[1]	KHz
GUARD_TIME	50	µs
I2C_STATE		SOFT_RESET = 1; CONT_READ = 0; REP_START = 0; CLK_STRETCHING = 0; PRESENT_LAYER = 1;

4.4 Commands

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

4.4.1 Command definitions

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

Table 26 Command codes

Command code	Command	Short description
0x01 or 0x81	GetDataObject	Command to get (read) a data object
0x02 or 0x82	SetDataObject	Command to set (write) a data object
0x03 or 0x83	SetObjectProtected	Command to set (write) a data object protected.
0x0C or 0x8C	GetRandom	Command to generate a random stream
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
0x38 or 0xB8	GenKeyPair	Command to generate public key pairs
0x70 or 0xF0	OpenApplication	Command to launch an application
0x71 or 0xF1	CloseApplication	Command to close/terminate an application

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

Table 27 APDU Fields

Name	Description
Cmd	Command code[2] as defined in Table "Command Codes"
Param	Parameter to control major variants of a command. For details, refer to the particular command definition.

[1] The default setting is 400 KHz

[2] 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

Name	Description
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

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.
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.

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

Table 28 Response Status Codes

Response Status Code	Offset [direction]	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.

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.

Table 29 Error Codes

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

Field	Code	Description
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 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.
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

4.4.1.1 OpenApplication

This command 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.

Table 30 OpenApplication

Field	Offset [direction]	Description
Cmd	0 [in]	**0x70 | 0xF0**[3] 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

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

4.4.1.2 CloseApplication

This command 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.

Table 31 CloseApplication

Field	Offset [direction]	Description
Cmd	0 [in]	**0x71 | 0xF1**[4] 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.[5]
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).

[4] In case of 0xF1 the Last Error Code gets flushed

[5] 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

This command 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.

Table 32 GetDataObject

Field	Offset [direction]	Description
Cmd	0 [in]	**0x01 | 0x81**[6] 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[7] 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

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

4.4.1.4 SetDataObject

This command 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.

Table 33 SetDataObject

Field	Offset [direction]	Description
Cmd	0 [in]	**0x02 | 0x82**[8] Command Code
Param	1 [in]	• **0x00** Write data • **0x01** Write metadata[9] • **0x02** Count data object [10] [11] [12] • **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

[8] In case of 0x82 the Last Error Code gets flushed
[9] 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
[10] The offset given in InData must be ignored
[11] 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.
[12] The execute (EXE) access condition is considered for counting

4.4.1.5 SetObjectProtected

This command command is used to write data objects protected (integrity) 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.

Notes:

- *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 and*
  - *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.*

Table 34 SetObjectProtected

Field	Offset [direction]	Description
Cmd	0 [in]	**0x03 | 0x83**[13] 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[14] continue => first to n-1th fragment of update data set[15] final => last fragment of update data set[16]
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

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

4.4.1.6 GetRandom

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

Table 35 GetRandom

Field	Offset [direction]	Description
Cmd	0 [in]	**0x0C | 0x8C**[17] Command Code
Param	1 [in]	• **0x00** Random number from TRNG (according [AIS-31]) • **0x01** Random number from DRNG (according [SP 800-90A])
InLen	2 [in]	**0x0002** Length of InData
InData	4 [in]	**0x0008-0x0100** length of random stream to be returned
Sta	0 [out]	**0x00 | 0xFF** Response Status Code
UnDef	1 [out]	**0x00-0xFF** Undefined Value
OutLen	2 [out]	**0x0000-0xFFFF** Length of OutData
OutData	4 [out]	**0x00-0xFF** Random stream.

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

4.4.1.7 CalcHash

This command 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.

Table 36 CalcHash

Field	Offset [direction]	Description
Cmd	0 [in]	**0x30 | 0xB0**[18] 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[19], Message data (start y = 0, start&final y = 1, continue y = 2, final y=3, final and keep intermediate hash y=5[20]) • **0x04**, 0x0000 - To terminate the hash sequence in case initialized already. • **0x1y**, 0x0006, OID[21], Offset, Length[22] (start y = 0, start&final y = 1, continue y = 2, final y=3, final and keep intermediate hash y=5) (optional one or multiple) (only allowed in conjunction with continue (y=2) or final (y=3) or final and keep intermediate hash (y=5) indication) **0x06**, Length, Intermediate hash context data (only allowed in conjunction with start (y=0) or continue (y=2) indication • **0x07**, 0x0000 indicate exporting the Intermediate hash context via the external interface) *Note: allowed sequences are "start-(zero to n-times continue)-final" or "start&final" (atomic) or "start-(zero to n-times continue)-terminate"*
Sta	0 [out]	**0x00 | 0xFF** Response Status Code
UnDef	1 [out]	**0x00-0xFF** Undefined Value

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

Field	Offset [direction]	Description
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.

4.4.1.8 CalcSign

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

Table 37 CalcSign

Field	Offset [direction]	Description
Cmd	0 [in]	**0x31 | 0xB1**[23] Command Code
Param	1 [in]	**0xXX** Signature Scheme (refer to *Signature Schemes*)
InLen	2 [in]	**0xXXXX** Length of InData
InData	4 [in]	Signature Input InData [InLen] • **0x01**, Length[24], Digest to be signed • **0x03**, 0x0002, OID of signature key[25] 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[26] Note: The length of the signature is derived from the applied key and signature scheme.

[23] In case of 0xB1 the Last Error Code gets flushed
[24] For ECC shall be 10 bytes up to the length of the addressed signature key
[25] The addressed signing key shall be a private key
[26] ECC case: The signature pair (r,s) is encoded as two DER "INTEGER

4.4.1.9 VerifySign

This command is used to verify a signature over a given digest provided with the InData.

Table 38 VerifySign

Field	Offset [direction]	Description
Cmd	0 [in]	**0x32 | 0xB2**[27] 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[28], Digest • **0x02**, Length[29], Signature over Digest[30] • alternate one :**{** (**0x04**, 0x0002, OID of Public Key Certificate[31]), (**0x05**, 0x0001, Algorithm Identifier (of the Public Key), **0x06**, Length, Public Key[32])**}**
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

[27] In case of 0xB2 the Last Error Code gets flushed
[28] 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
[29] The length is limited to max. 520 bytes
[30] ECC case: The signature pair (r,s) is encoded as two DER "INTEGER"
[31] 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)
[32] PubKey is encoded as DER "BIT STRING"

4.4.1.10 GenKeyPair

This command 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.

Table 39 GenKeyPair

Field	Offset [direction]	Description
Cmd	0 [in]	**0x38 | 0xB8**[33] 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[34] 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]	• ECC key 0x01, Len, PrivKey[35] 0x02, Len, PubKey[36]

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

[34] 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.

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

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

4.4.2 Command Parameter Identifier

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

Table 40 Algorithm Identifier

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

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

Table 41 Key Usage Identifier

Value	Description
0x01	Auth (Authentication)
0x10	Sign (Signature Calculation / Verification)

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

Table 42 Signature Schemes

Value	Description
0x11	ECDSA w/o hash

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.

Table 43 Command Performance Metrics

Operation	Command	Scheme	Execution Time[37]	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
Verify signature	VerifySign	ECDSA	~ 85ms	• ECC NIST P 256 • No data hashing • Public Key from external interface
Key pair generation	GenKeyPair	ECC	~ 55 ms	• ECC NIST P 256 • Ephemeral key
Hash calculation	CalcHash	SHA 256	~ 12 Kbyte/s	In blocks of 0x500 bytes

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

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.

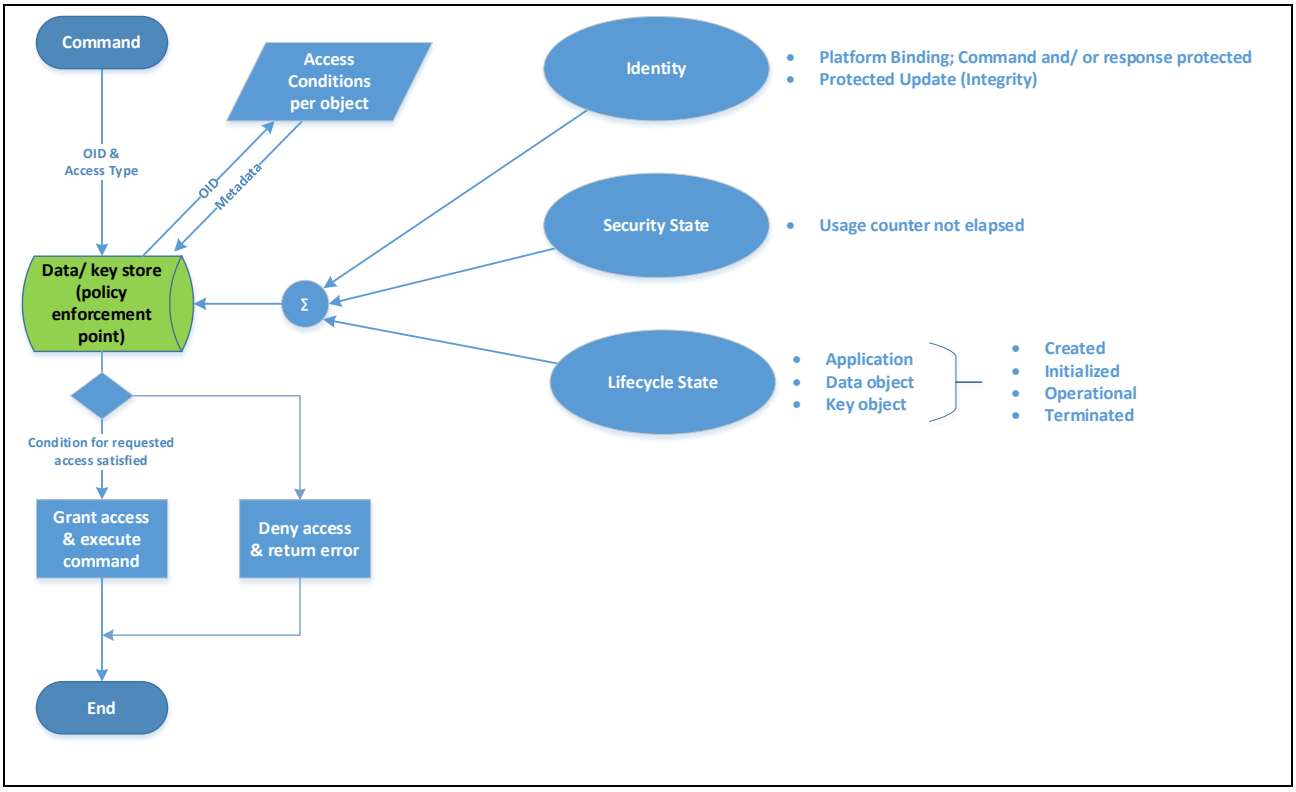

Figure 18 - Security Policy Architecture

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.

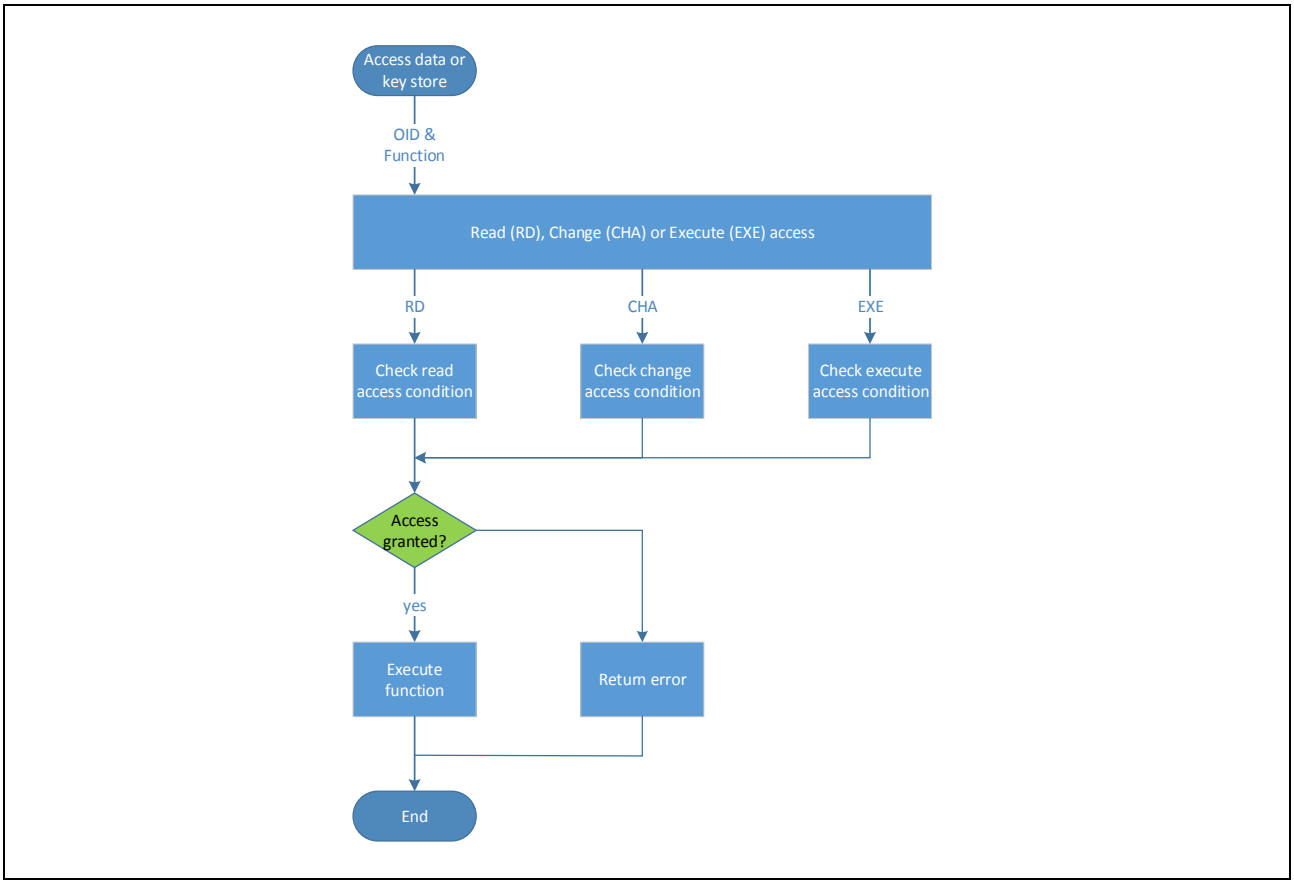

Figure 19 - Policy Enforcement Flow

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.

Table 44 Security Events

Name	Description
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.
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.

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 set to 5 seconds (± 5%).

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

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

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.

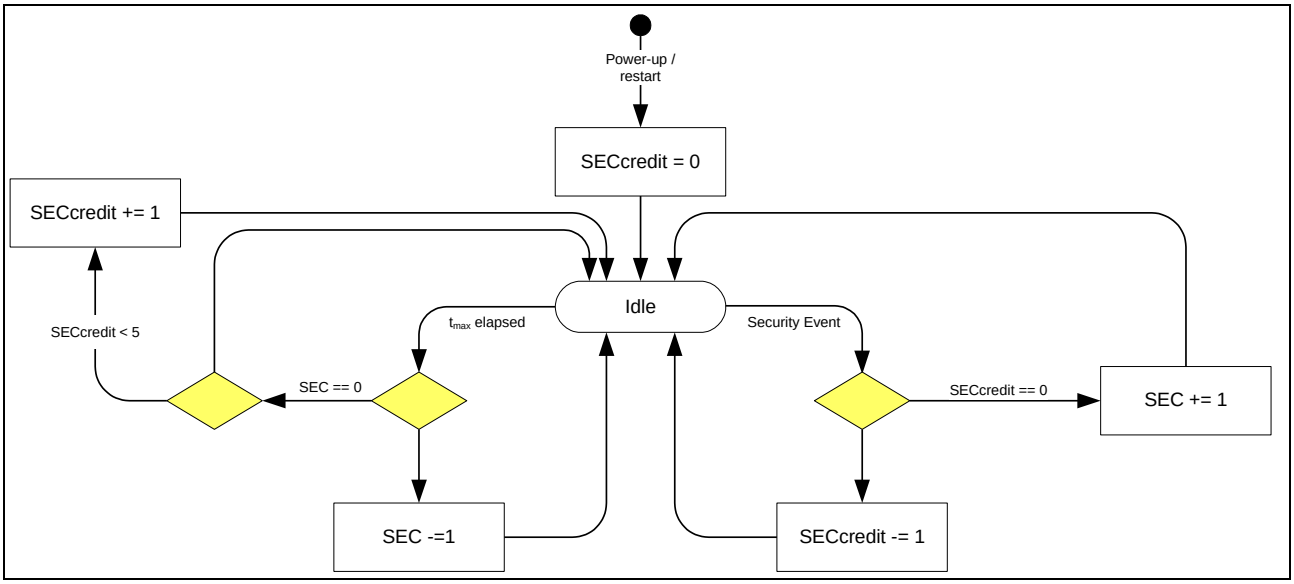

Figure 20 - Security Monitor flow diagram

4.6.3 Security Monitor Characteristics

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

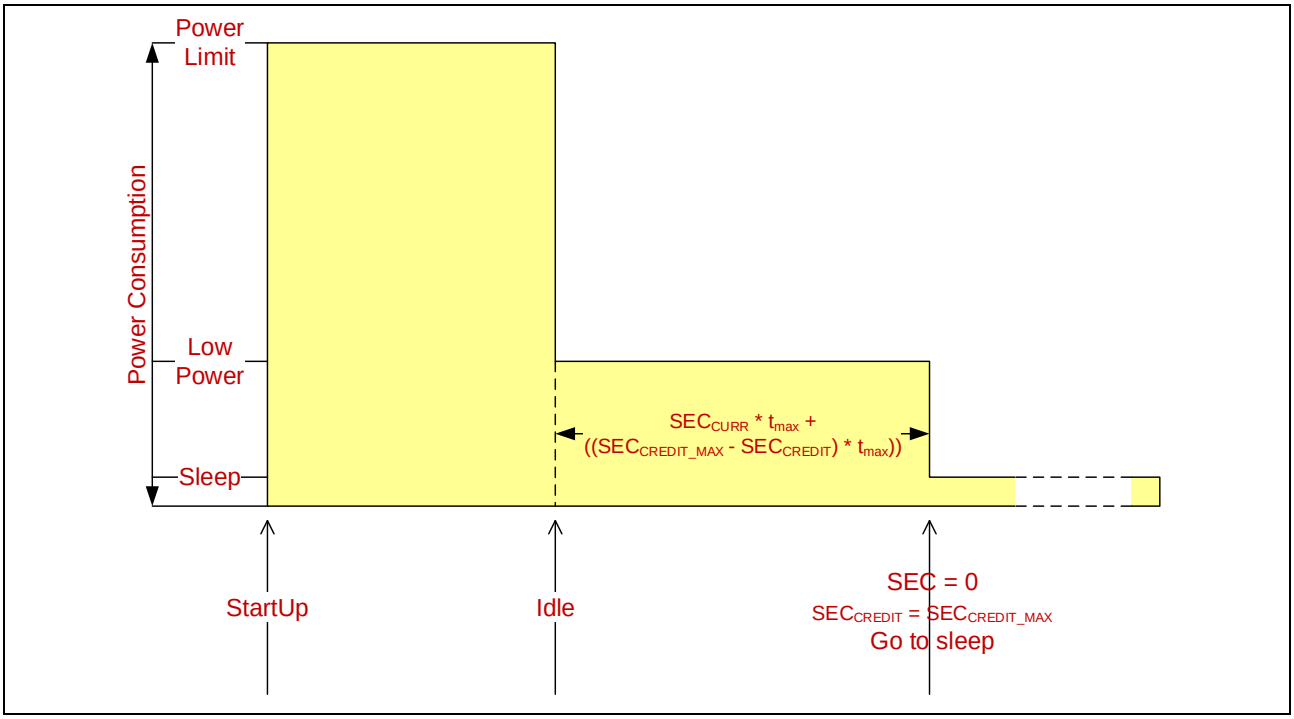

Figure 21 - Power profile

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.

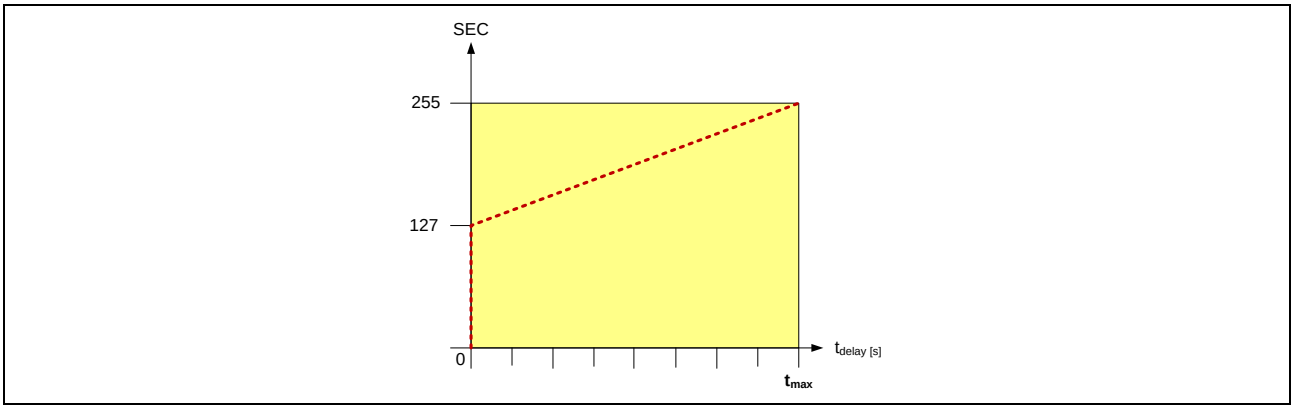

Figure 22 - Throttling down profile

5 OPTIGA™ Trust Charge 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 Charge 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 three tearing-safe-programming-cycles
- 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

- 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.

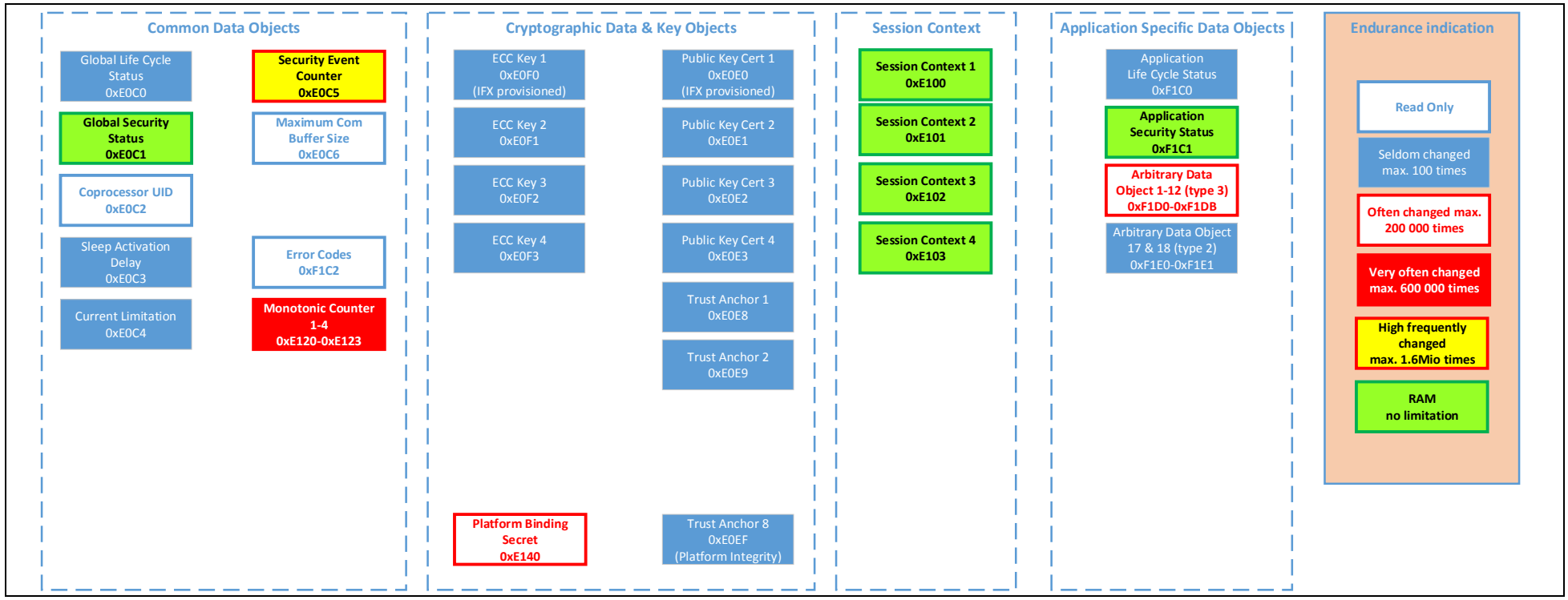

Figure 23 – OPTIGA™ Trust Charge: Overview Data and Key Store

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 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, ...)

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.

OPTIGA™ Trust Charge Data Structures

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

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

Table 'Access Condition Identifier and Operators' defines the Access Condition Identifier and Operands to be used, to define ACs associated with data objects. Access Condition Identifier must be used with the commands trying to achieve associated ACs.
There are ***simple*** and ***complex*** Access Condition expressions defined (Examples how to code are given in chapter Metadata expression).

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

Notes:

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

Remark: With the rules given above it doesn't matter whether starting the evaluation of a complex expression from the beginning or the end. 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'.

Table 45 Access Condition Identifier and Operators

AC ID	Operator	Value	Description
ALW	-	0x00	1 byte; Value
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)

AC ID	Operator	Value	Description
			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 private key object with an internal operation (e.g. performing signature generation using a private key at OPTIGA™), the shielded connection must be established already using the specified platform binding secret (0xE140) and the command is sent with protection (encrypted).
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 private key object with an internal operation (e.g. Performing signature generation using a private key at OPTIGA™), 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 respective Trust Anchor.
Luc	-	0x40	3 byte; Value, Counter Reference

AC ID	Operator	Value	Description
			(e.g. Linked Counter 1 → 0x40, 0xE1, 0x20) For example, The private key is to be restricted to be used for 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 respective key object as shown below. (e.g. EXE, Luc, Counter Object → 0xD3, 0x03, 0x40, 0xE1, 0x20) The counter data object gets updated (counter value gets incremented by 1 up to maximum limit) automatically when the respective key based operation is performed.
LcsG	-	0x70	3 byte; Value, Qualifier, Reference (e.g. LcsG < op → 0x70, 0xFC, 0x07)
LcsA	-	0xE0	3 byte; Value, Qualifier, Reference (e.g. LcsA > in → 0xE0, 0xFB, 0x03)
LcsO	-	0xE1	3 byte; Value, Qualifier, Reference (e.g. LcsO < op → 0xE1, 0xFC, 0x07)
-	==	0xFA	equal
-	>	0xFB	greater than
-	<	0xFC	less than
-	&&	0xFD	logical AND
-	||	0xFE	logical OR
NEV	-	0xFF	1 byte; Value

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

Table 46 Data Object Types

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.

Name	Value	Description
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 chain of certificates (WPC compressed, WPC X.509, USB-Type C, etc.) which was issued to vouch for the cryptographic identity of the end-device.
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).

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.

Table 47 Common data objects with TAG's and AC's

OPTIGA™ Trust Charge Data Structures

Tag	Structure definition	Default Value	EXE	Change	Read	Note
0xE0C0	Global Life Cycle Status (LcsG)	0x07	NEV	ALW	ALW	default LcsO = op
0xE0C1	Global Security Status	0x00	NEV	ALW[38]	ALW	default LcsO = op
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
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"[39]
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"[40]
0xE0E8-0xE0E9	Root CA Public Key Certificate 1-2[41] (refer to 'Common data structures')	0x00	ALW	LcsO < op	ALW	default LcsO = cr; default Data Type is "Trust Anchor"

[38] It is only possible to reset an achieved security status
[39] 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
[40] 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
[41] 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 Charge Data Structures

Tag	Structure definition	Default Value	EXE	Change	Read	Note
0xE0EF	Root CA Public Key Certificate 8[42]. 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 'Common key objects with TAG's and AC's' lists all common Keys defined for the OPTIGA™ with its TAG's and AC's.

Table 48 Common key objects with TAG's and AC's

Tag	Structure definition	EXE	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	ALW	LcsO < op	NEV	default LcsO = cr

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

Tag	Structure definition	EXE	Change	Read	Note
	metadata states the access rights differently.				
0xE100-0xE103	Session context 1-4 (OID to address one of the four session contexts). These OIDs are not applicable for the *GetDataObject*, SetDataObject commands. The session context holds a private key generated using *GenKeyPair* command.	ALW	NEV	NEV	default LcsO = op

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.

Table 49 Authentication application-specific data objects with TAG's and AC's

Tag	Structure definition	Default Value	EXE	Change	Read	Note
0xF1C0	Data Structure application '*Life Cycle Status*' (LcsA)	0x01	NEV	ALW	ALW	default LcsO = op
0xF1C1	Data Structure application '*Security Status*'	0x00	NEV	ALW[43]	ALW	default LcsO = op
0xF1C2	Error codes (refer to Table '*Error Codes*')		NEV	NEV[44]	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

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

[43] It is only possible to reset an achieved security status
[44] cleared on read

…) 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.

Table 50 Metadata associated with data and key objects

Tag	Structure definition	EXE	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	

Tag	Structure definition	EXE	Change	Read	Note
0xD3	Execute Access Condition descriptor	n.a.	LcsO < op	ALW	
0xE0	Algorithm associated with key container	n.a.	auto[45]	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).

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
```

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

Note: there is no ordering rule for metadata tags

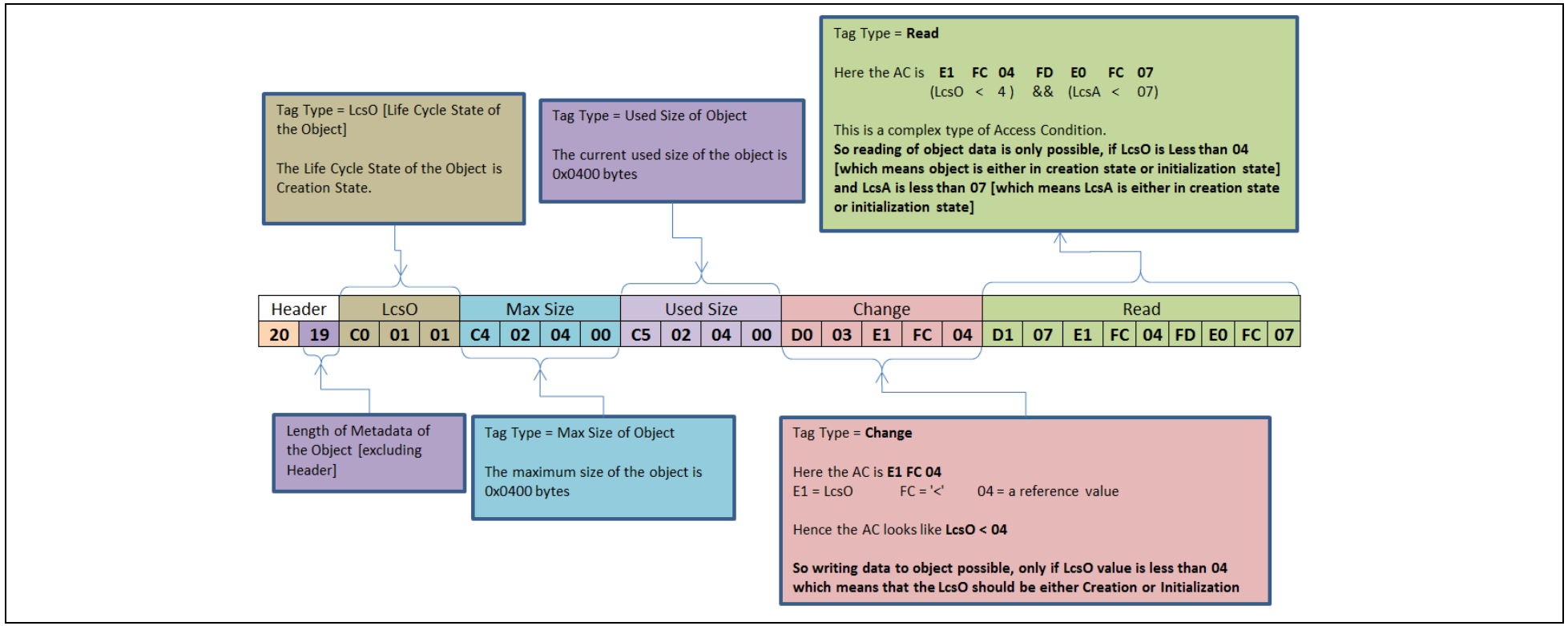

Figure 24 - Metadata sample

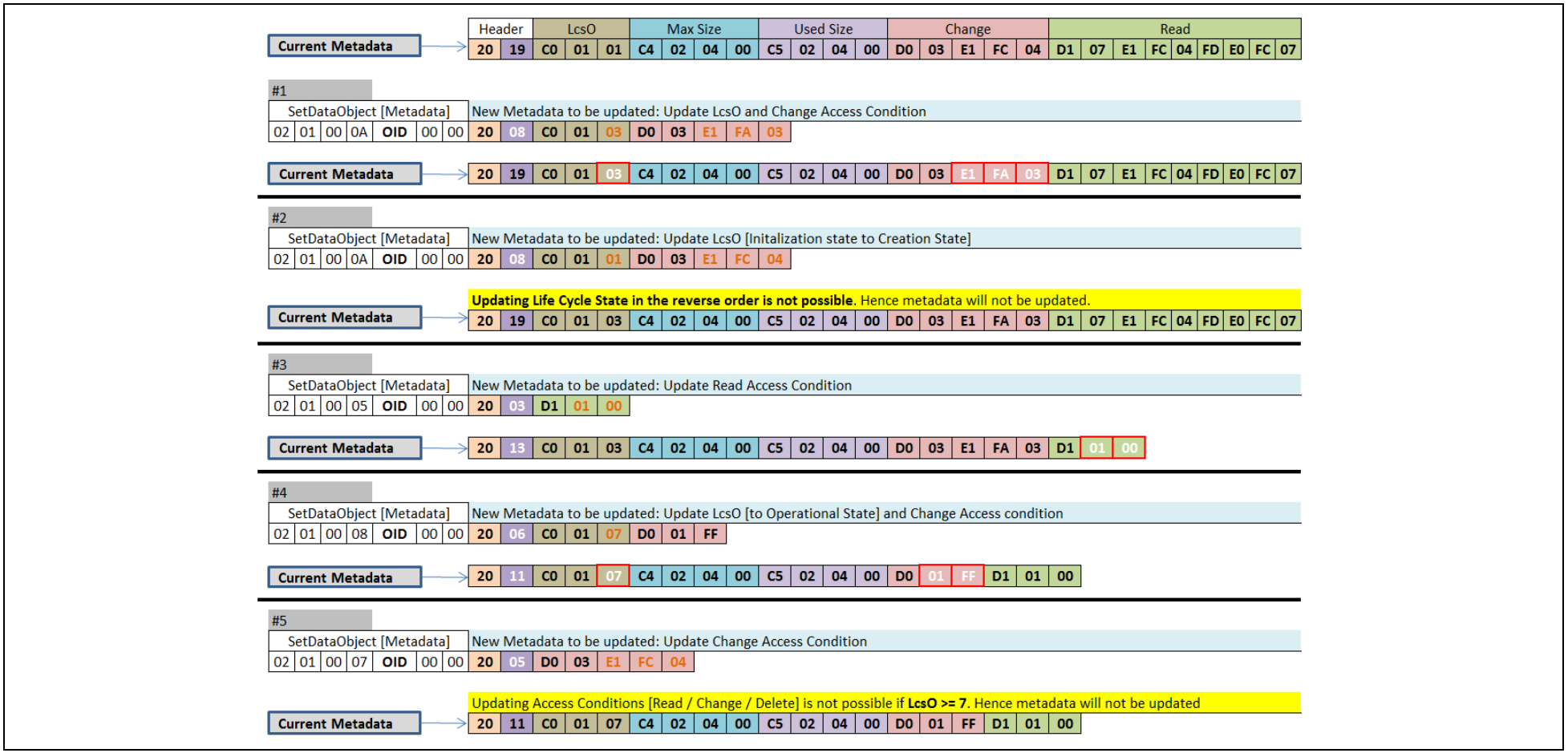

Figure 25 - SetDataObject (Metadata) examples

Note: The values specified in above example figures are in hex

5.6 Common data structures

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

Table 51 Common data structures

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*')	BinaryNumber 8	1	The LCS is implemented in a way that the four primary states only progress in one direction from a lower value to a higher value (e.g. initialization (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*')	BinaryNumber 8	1	The device and each application may have a security status associated. The device security status is further referenced to by “Global security status” and the application specific status by “Application-specific security status”.
Last Error Code (refer to Table '*Error Codes*')	BinaryNumber 8	1	The Last Error Code stores the most recent error code generated since the data object was last cleared. The 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	BinaryNumber 8	1	The Sleep Mode Activation Delay holds the delay time in milliseconds starting from the last communication until the device enters its power saving sleep mode. The allowed values are 20-255 (ms). Its default content is 20.
Current limitation in mA	BinaryNumber 8	1	The Current limitation holds the maximum value of current allowed to be consumed by the OPTIGA™ across all operating conditions. The allowed values are 6-15 (mA). This register resides in Non-Volatile Memory (NVM) and will be restored upon power up or reset. Its default content is 6mA. **Note:** *15mA will cause best case performance. 9 mA will cause roughly 60% of the best case*

OPTIGA™ Trust Charge Data Structures

Data element	Coding	Length (Bytes)	Description
			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	BinaryNumber 16	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 chain of X.509 Certificates (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). • **USB Type-C Identity:** **Tag** = 0xC2 **Length** = Value length (2 Bytes) **Value** = USB Type-C Certificate Chain[USB Auth][46] • **WPC Compressed Identity:** **Tag** = 0xC4 **Length** = Value length (2 Bytes) **Value** = WPC compressed Certificate Chain [WPC Auth][47] • **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	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

[46] Format as defined in Section 3.2 of the USB Type-C Authentication Specification.
[47] Format as defined in the WPC Authentication Specification.

OPTIGA™ Trust Charge Data Structures

Data element	Coding	Length (Bytes)	Description
	certificate)		authenticate external entities (e.g. verify server signature).
Platform Binding Secret	BinaryNumber 8	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	BinaryNumber 64	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 'Life Cycle Status' lists all coding of the Life Cycle Status defined for the OPTIGA™.

Table 52 Life Cycle Status

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)[48]

Table 'Security Status' shows the security status defined for the device either global or application-specific.

Table 53 Security Status

[48] this state is not applicable for the LcsA

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

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

Table 54 Coprocessor UID OPTIGA™ Trust Family

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

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.

Table 55 Data Structure Unique Application Identifier

Data Element	Value	Coding	Length

OPTIGA™ Trust Charge Data Structures

Data Element	Value	Coding	Length
RID	0xD276000004[49]	HexNumber5	5
PIX	'GenAuthAppl' 0x47656E417574684170706C	Ascii11	11

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

Table 56 Data Structure Arbitrary data object

Data Element	Value	Coding	Length
Arbitrary Data object Type 2	0x00 - 0xFF	application-specific	1500
Arbitrary Data object Type 3	0x00 - 0xFF	application-specific	140

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

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 26 - GetDataObject [Read data] example

Note: The values specified in above figure 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 27 - GetDataObject [Read data] example

Note: The values specified in above figure 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]

Appendix

```
// 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 and Verify signature) 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
```

Appendix

```
        // 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)
- 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.

6.3 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,

Appendix

- 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
  - 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 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.

6.5.2 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 object access**

conditions CHA shall be configured with **never allowed** (NEV). The public key certificate object can be allowed with just protected update (**integrity 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.

6.5.3 Shielded Connection

- The security level of the Shielded connection is as high as typical microcontroller/Host side hardware security level.
- The recommended length of Platform binding shared secret is 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. Private keys) if assets are not intended to be used extremely.

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.

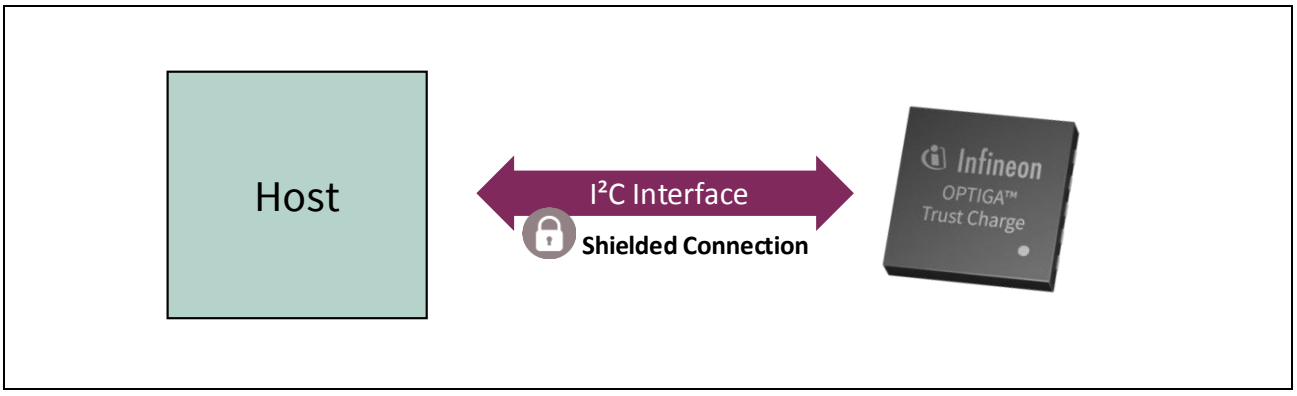

Figure 28 - Overview OPTIGA™ Shielded Connection

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 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 data read from OPTIGA™ (command protection for authentication and response protection for data encryption).

- Invoke OPTIGA_UTIL_SET_COMMS_PROTECTION_LEVEL (me_util, OPTIGA_COMMS_FULL_PROTECTION)
- Invoke optiga_util_read_data (me_util, 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

This section provides the definition and some useful information of update data sets for data objects which are used to update those in a 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.

Appendix

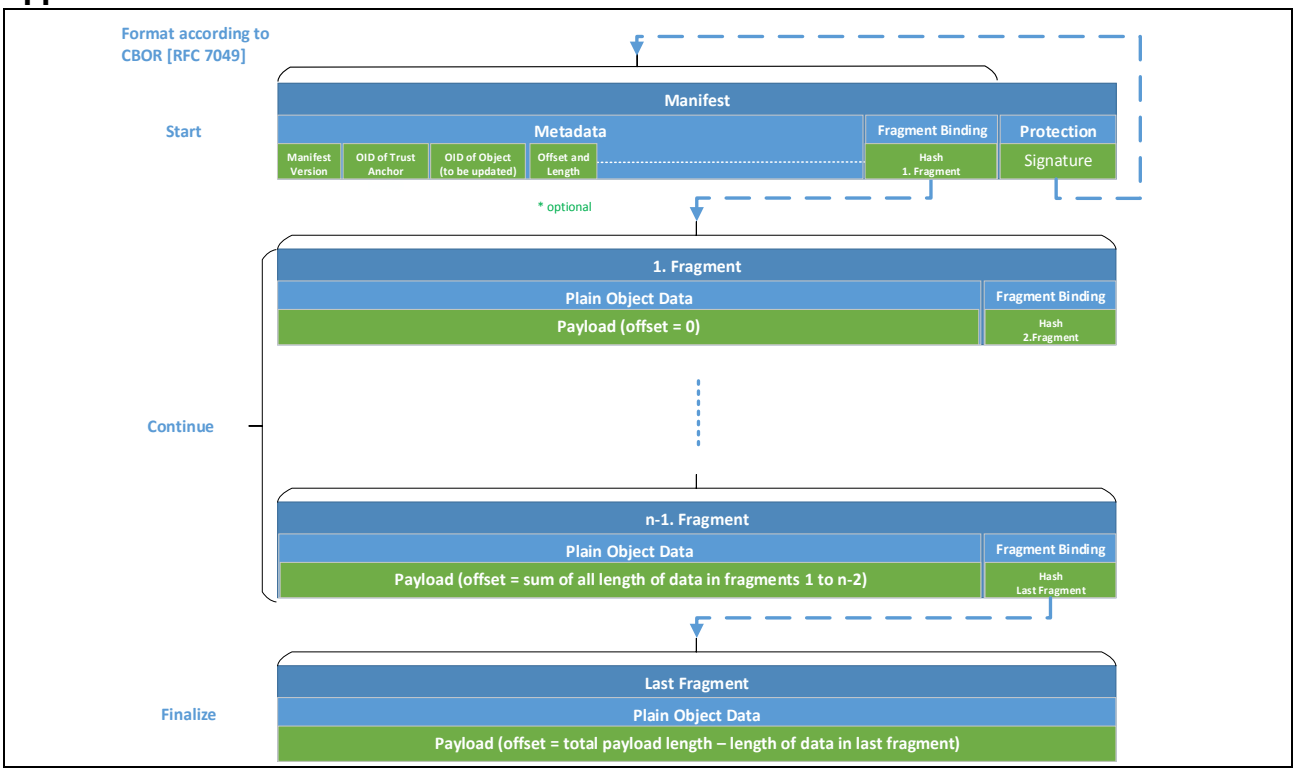

Figure 29 – Protected update – high level structure

The coding of the structure is according to [CBOR]. The [CDDL] format for the manifest and signature structures are provided in the package.

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** (in manifest) provide information enabling interpretation and manage the update data set by the OPTIGA™. It contains:

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

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

6.7.1 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.

Appendix

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, 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**.**

Table 57 Terms of OPTIGA™ Vocabulary

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.	
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, optical discs, and early computer storage methods such as paper tape and punched cards.	NVM
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	*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	RAM

Appendix

Term	Description	Abbreviation
Memory	roughly the same amount of time regardless of the order in which data items are accessed.	
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.	
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.	

Appendix
Revision history

Version	Date	Description
1.00	2020-07-30	• Release version

Edition 2020-07-30

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.