OPTIGA™ Trust M

Solution reference manual

About this document

Scope and purpose

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

Intended audience

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

1 All references regarding the OPTIGA™ Trust M version 1 (V1) and version 3 (V3) are given generically without indicating the dedicated version

Table of contents

Table of contents

Table of contents

Table of contents

List of tables

List of tables

List of figures

List of figures

1 Introduction

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 M external interface, OPTIGA™ Trust M data structures and Appendix.

- Supported use cases provides the main use cases in a form of sequence diagrams which explains how the host side Enabler APIs are used in the respective use cases
- Enabler APIs provides the necessary details of the host side architectural APIs which are implemented based on the OPTIGA™ Trust M external interface
- OPTIGA™ Trust M external interface provides the necessary details of the external interface to utilize the OPTIGA™ functionality
- OPTIGA™ Trust M data structures provides details of the key and data objects provided by the OPTIGA™
- Appendix provides some useful information with regards to Command coding examples, Limitations, Certificate parser details, Security guidance, Shielded connection V1 guidance, Protected update, and Terminology

2 Supported use cases

In this 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 (Figure 1) are listed and briefly described in the table below.

Table 1 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 for example: 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 (KB). 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 [2]. optiga_comms_ifx_i2c usage characteristic is asynchronous, were the caller has to take care of the correct sequence of calls for a dedicated use case.

(table continues...)

Table 1 (continued) Architecture components

Name	Description
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 platform abstraction layer, abstracting hardware and operating system functionalities for the Infineon XMC™ family of µController (MCU) 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 the third party. This module is used multifold but not limited to: • Supporting TLS/DTLS protocol either for client or server side • Supporting bulk encryption in case the record protocol is performed at the local host • Supporting cloud service specific adaptation

The class diagram Figure 1 shows the communication protection solution architecture in case the local host is invoking a third_party_crypto library (for example: WolfSSL, OpenSSL, mbedTLS, ...) containing its main functional blocks. The OPTIGA™ is integrated via its toolbox functionality. The entities communicating across a protected channel are the server and the client (host) and optionally the client and the OPTIGA™ (shielded connection). This view is applied for toolbox based solution kind of use cases, where the involved blocks are represented as dedicated lifelines.

The color coding provides information of whether the functional block is:

- **Yellow**: Platform agnostic and provided by IFX or
- **Green**: Platform ported (subject of porting to a target platform) and provided by IFX as an example ported to the evaluation board or
- **Blue**: Platform specific provided by a third party

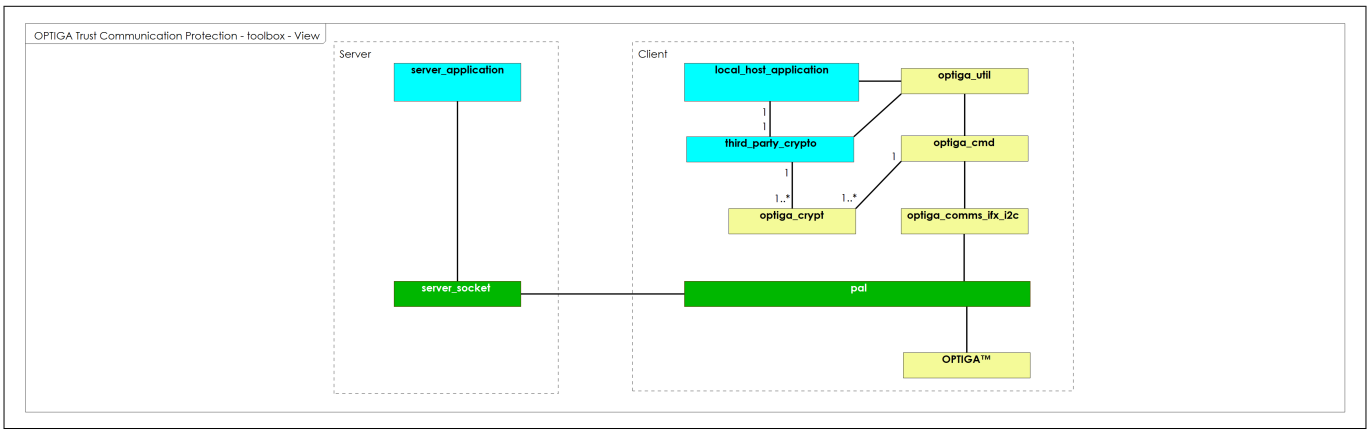

Figure 1 OPTIGA™ Trust communication protection - toolbox - view

2.1.1 Host code size

Table 2 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 2 Host library - code and RAM size details

Configuration	OPTIGA™ Trust M V1		OPTIGA™ Trust M V3	
	CODE	RAM	CODE	RAM
[without the shielded connection] The components optiga_crypt, optiga_util, optiga_cmd, optiga_comms_ifx_i2c, and pal are covered.	15 KB	5 KB	18 KB	5 KB
[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 (for example: Key derivation, encryption and decryption).	28 KB	15 KB	31 KB	15 KB

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-conditions:

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

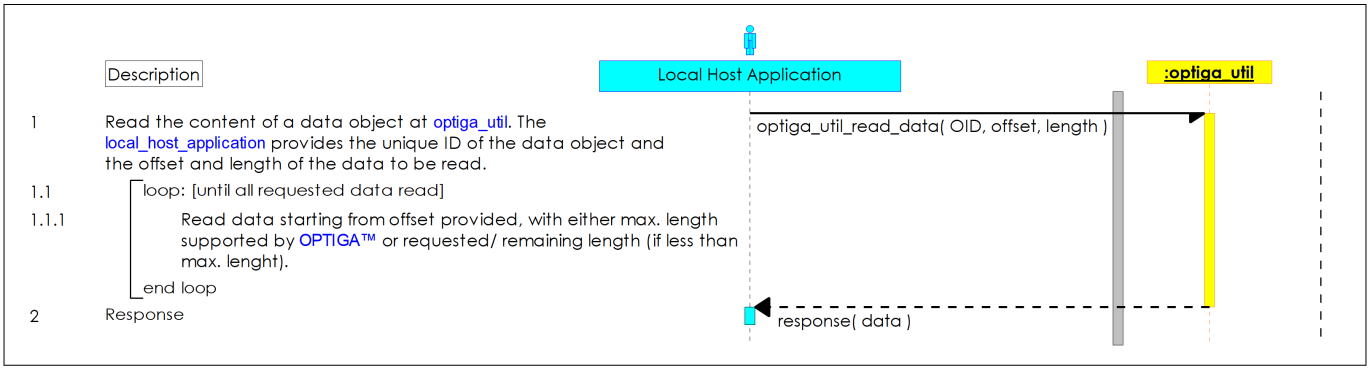

Figure 2 Use case: Read general purpose data - data object

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

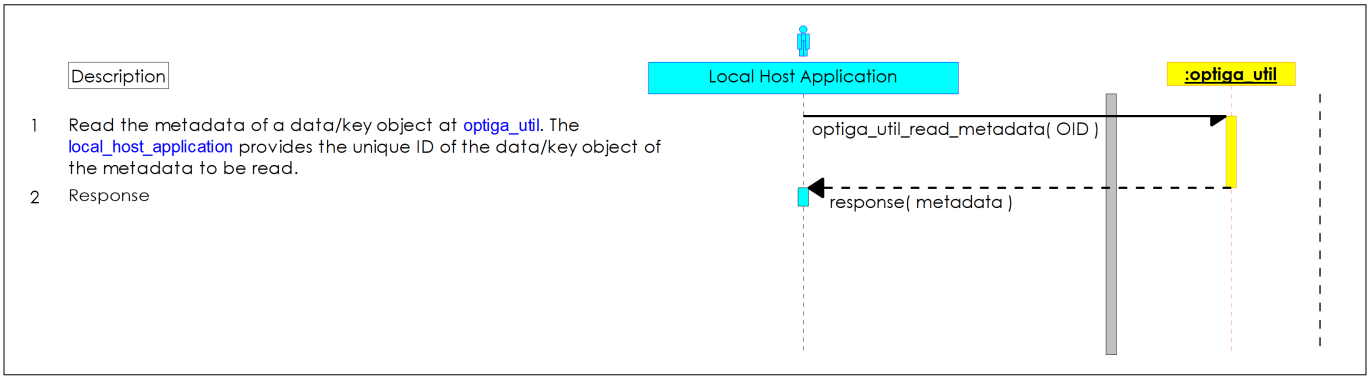

Figure 3 Use case: Read general purpose data - metadata

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-conditions:

- 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

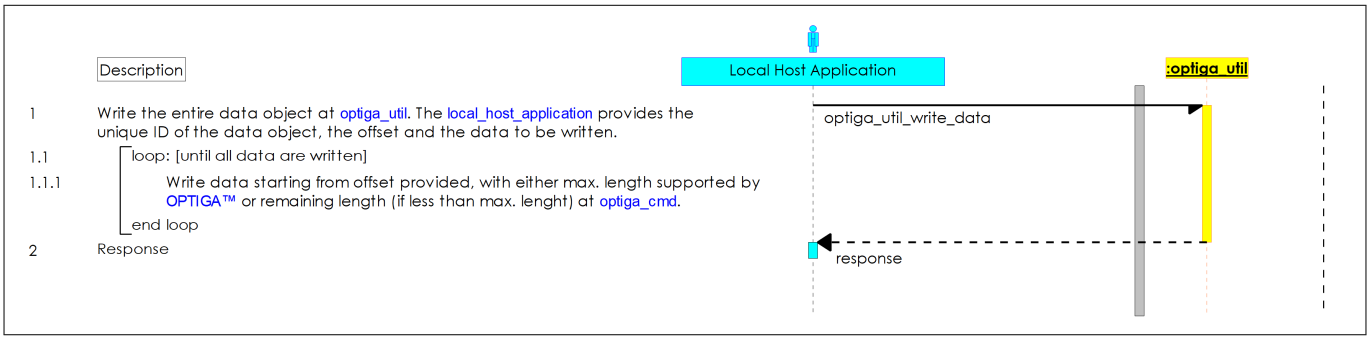

Figure 4 Use case: Write general purpose data - data object

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-conditions:

- 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

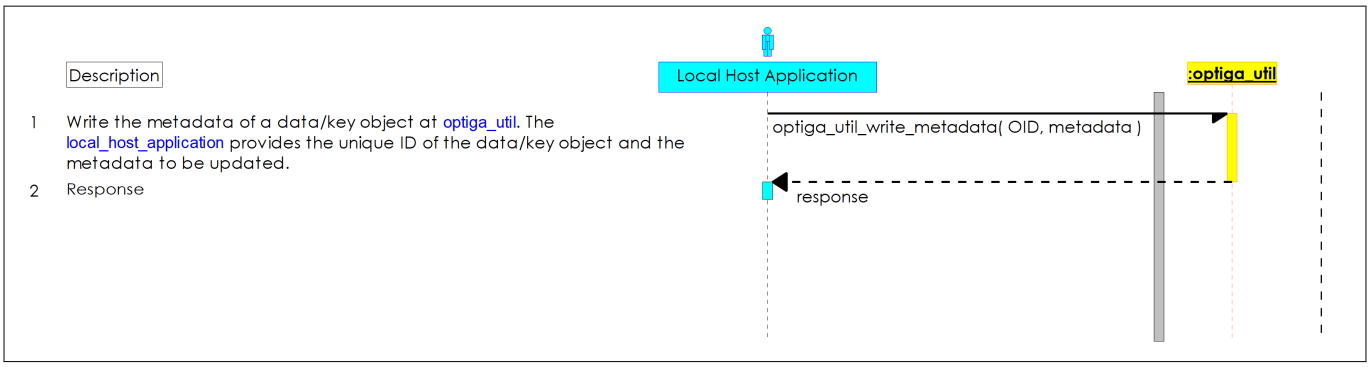

Figure 5 Use case: Write general purpose data - metadata

2.2.5 Use case: Integrity protected update of a data object

The management server intends to update a data object (for example: Trust anchor) with integrity protected. The management server provides an update data set, which is forwarded to the OPTIGA™. The OPTIGA™ checks and removes the protection and upon success updates the target data object.

Pre-conditions:

- The OPTIGA™ application is already launched

- The trust anchor for management purpose is well formatted and available at the OPTIGA™
- The access conditions of the target data object allow protected update

Post-condition:

- The target data object is updated

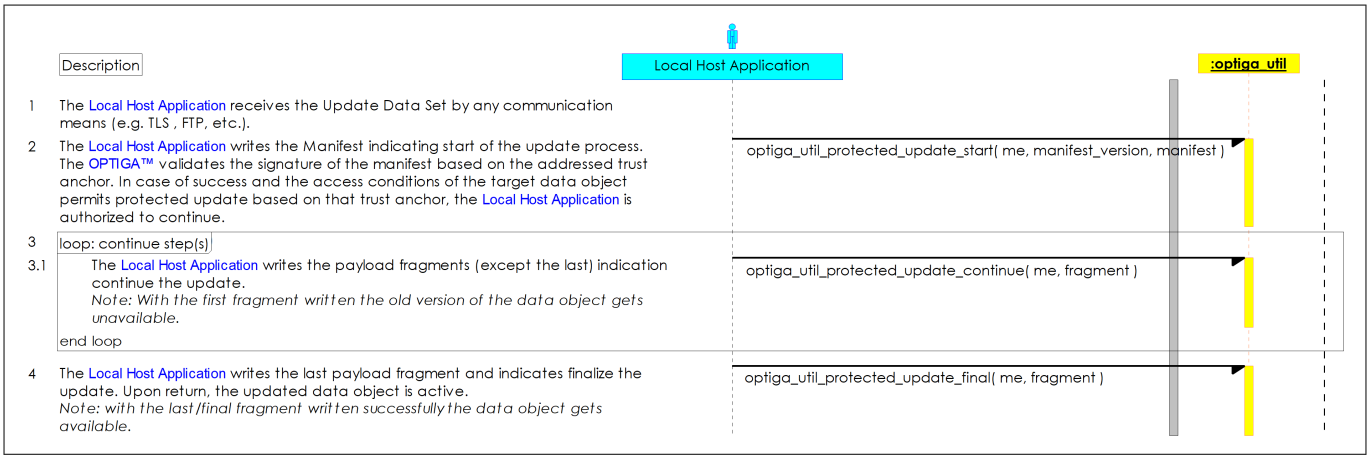

Figure 6 Use case: Integrity protected update of a data object

2.2.6 Use case: Confidentiality protected update of key or a data object

The management server intends to update a key or a data object (for example: Pre-shared secret) with integrity and confidentiality protected. The management server provides an update data set, which is forwarded to the OPTIGA™. The OPTIGA™ checks and removes the protection and upon success updates the target data/key object.

Note: The OPTIGA™ Trust M V1 does not support confidentiality and update of keys and metadata as part of protected update.

Pre-conditions:

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

Post-condition:

- The target data/key object is updated

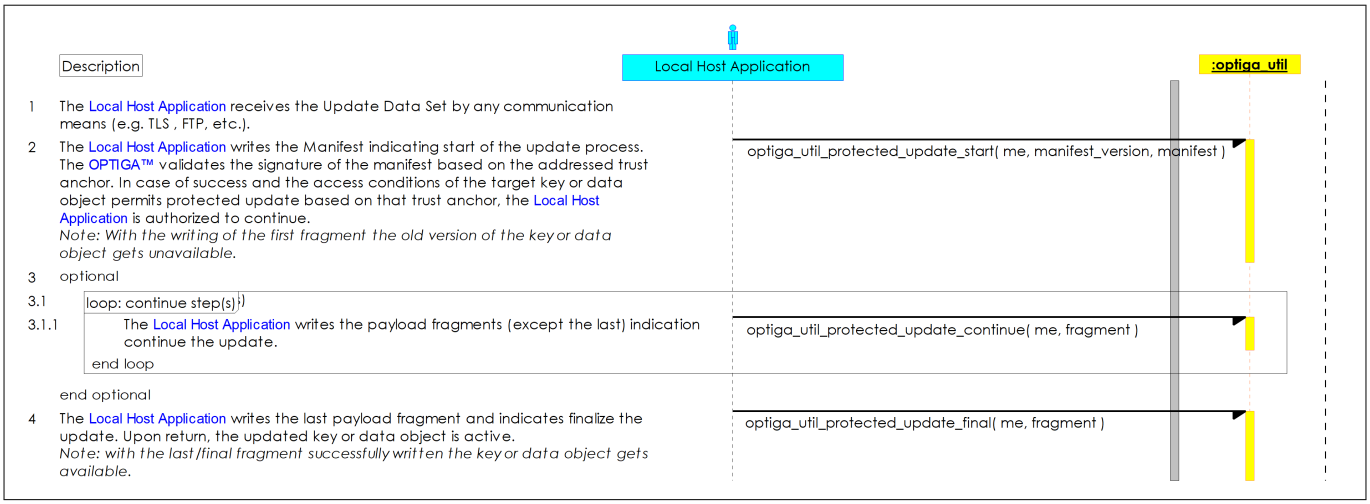

Figure 7 Use case: Confidentiality protected update of key or a data object

2.3 Sequence diagrams utilizing cryptographic toolbox functionality

2.3.1 Use case: Mutual authentication establish session -toolbox- (TLS-client)

The server and the client (on behalf of the user), which incorporates the OPTIGA™, intend to proof the authenticity of each other. Both the server and OPTIGA™ providing challenges (random value) and both entities return one or multiple cryptograms (depending on the applied authentication protocol) as response by which both parties proof their authenticity. The server and client executing ECDHE for key agreement and ECDSA FIPS 186-3 sign SHA256 hash for authentication, and the client is authenticated as well.

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

Pre-conditions:

- The OPTIGA™ application is already launched
- The public key pairs for authentication purpose and public key certificates are properly installed at the OPTIGA™
- The trust anchor for verifying the public key certificates of the authentication partner (server) is properly installed

Post-condition:

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

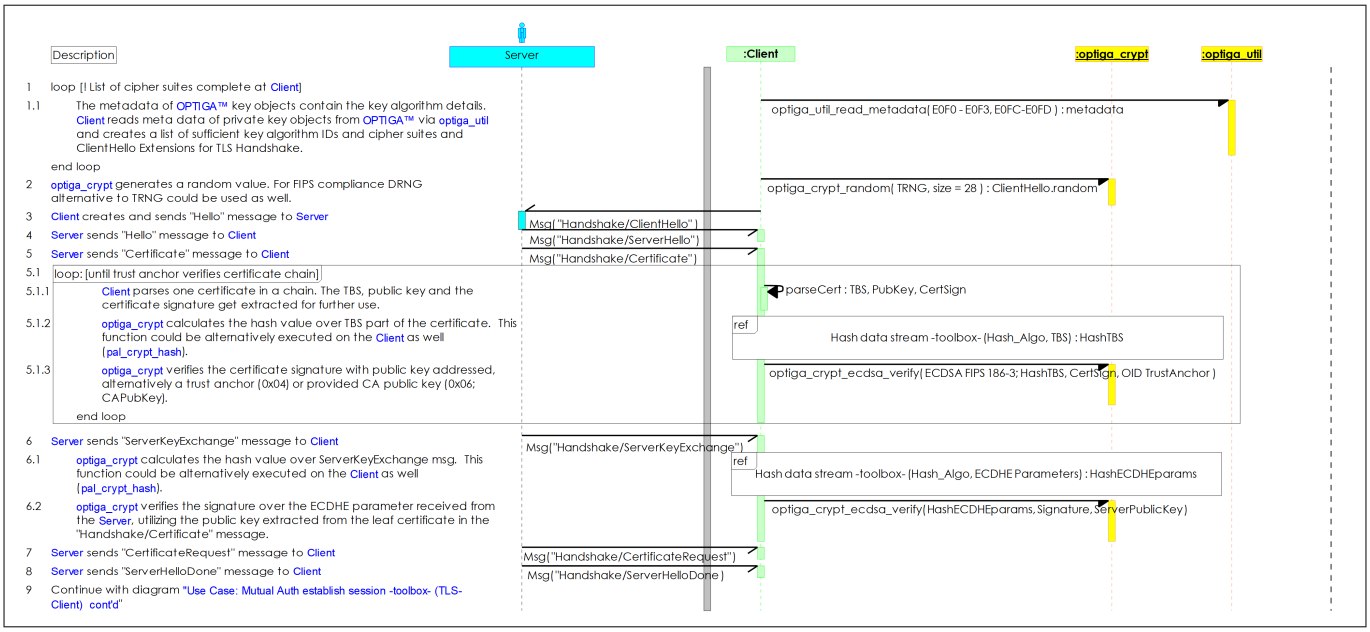

Figure 8 Use case: Mutual authentication establish session -toolbox- (TLS-client)

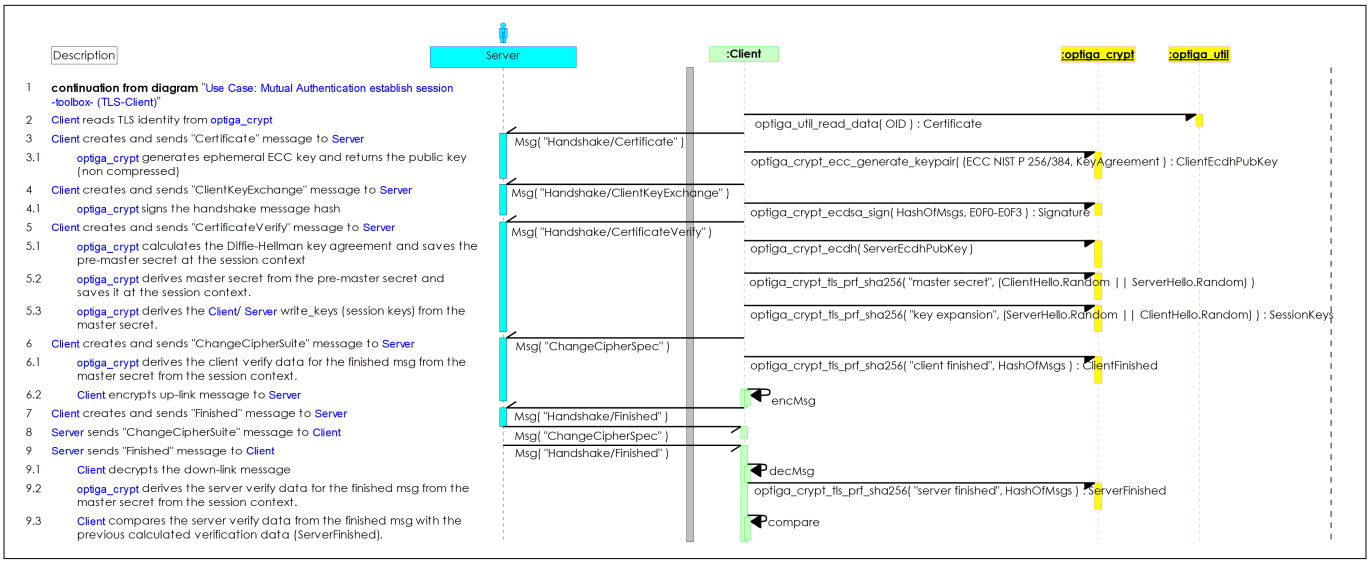

Figure 9 Use Case: Mutual Auth establish session -toolbox- (TLS-Client) cont'd

2.3.2 Use case: Abbreviated handshake -toolbox- (TLS-client)

The server and the client (on behalf of the user), which incorporates the OPTIGA™, intend to resume an established session. Both the server and OPTIGA™ providing challenges (random value via "Hello" message) and both entities providing verification data to prove the possession of the cryptographic parameters (master secret) previously negotiated.

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

Pre-conditions:

- The OPTIGA™ session master secret, which was calculated by the previous handshake - is available at the regarded session context and gets used as input by DeriveKey for the new session key(s)
- The client is able to hash all handshake messages without the help of OPTIGA™

Post-condition:

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

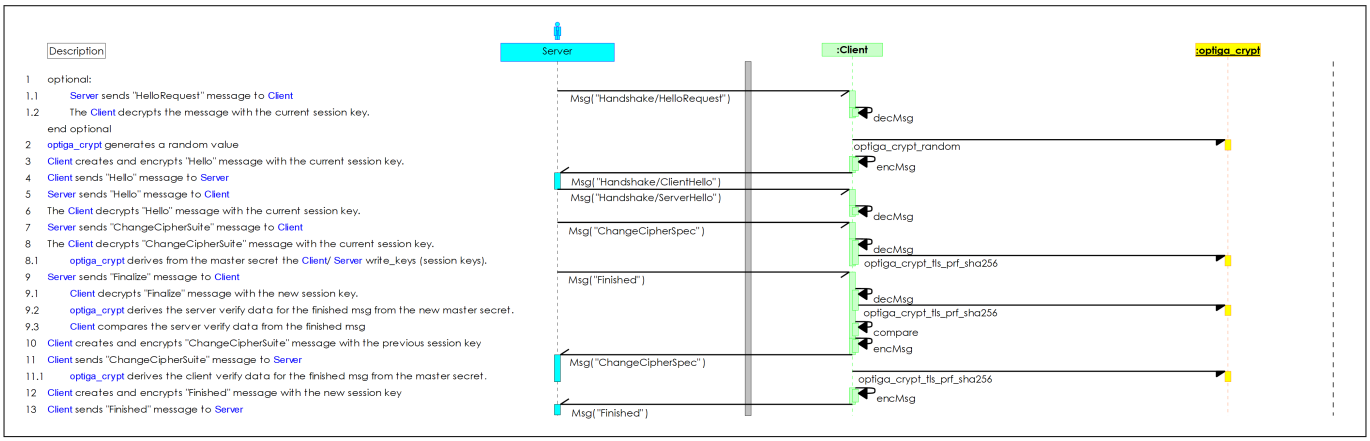

Figure 10 Use case: Abbreviated handshake -toolbox- (TLS-client)

2.3.3 Use case: Host firmware update

The host intends to update its firmware (FW) in a protected way, which prevents from installation and execution of unauthorized code. This sequence diagram is provided to show the functions involved in performing.

Pre-conditions:

- The FW-image shared secret is loaded to an arbitrary data object (for example: 0xF1D0-0xF1DF), which should be locked for read = NEV and in operational mode at least
- The trust anchor (signer's certificate) is loaded to a data object at OPTIGA™
- Host receives the firmware update manifest (example: Image version, signer, hash, and sign algorithms, firmware image hash, firmware image decryption key derivation information, manifest signature, etc..,) and encrypted firmware image. The details to be signed (TBS) in the manifest are signed by signer and host verifies the signature generated over TBS using the trust anchor installed at OPTIGA™

Post-conditions:

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

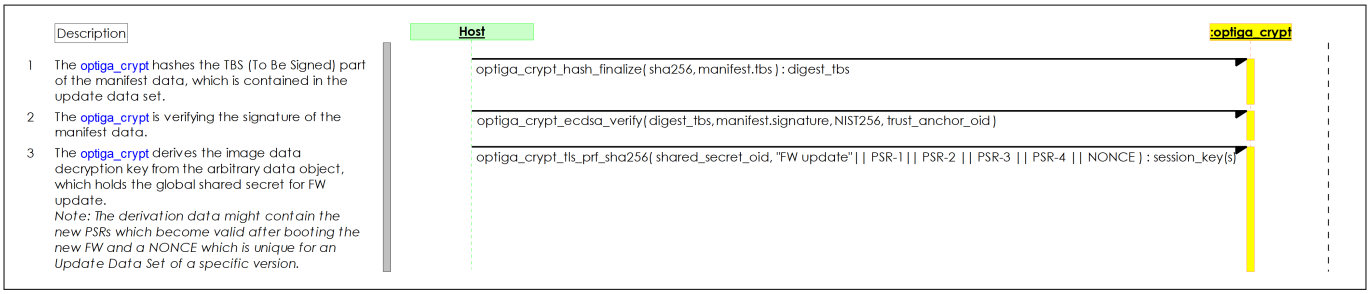

Figure 11 Use case: Host firmware update

2.3.4 Use case: Pair OPTIGA™ with host (pre-shared secret based)

The OPTIGA™ and host establishing a protected communication channel, which provides integrity and confidentiality for data exchanged between both entities. This 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:

- The platform binding secret data object is not locked. The life cycle status of the object (LcsO) must be less than operational

Post-conditions:

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

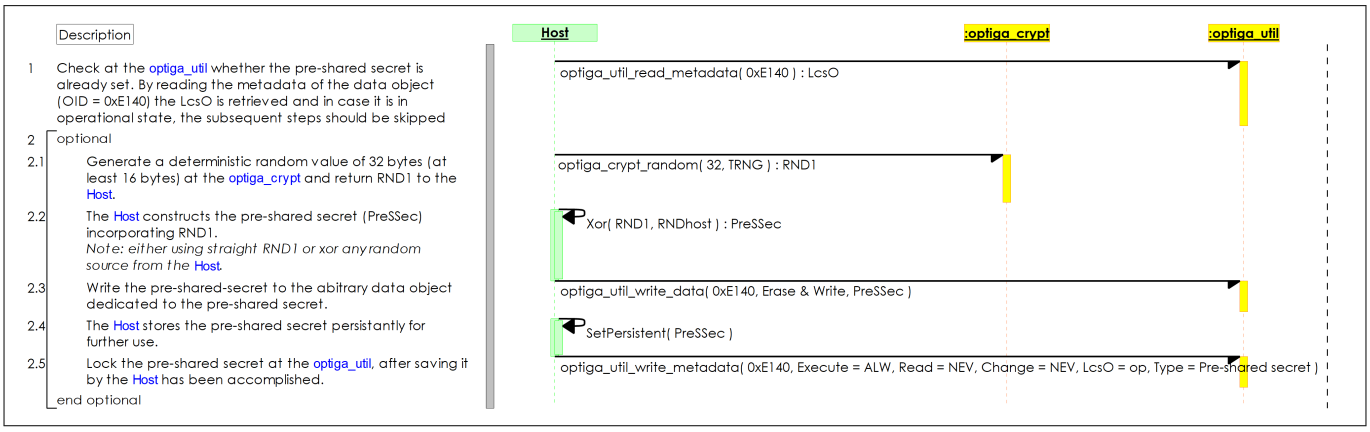

Figure 12 Use case: Pair OPTIGA™ with host (pre-shared secret based)

2.3.5 Use case: Verified boot -toolbox-

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

Pre-conditions:

- The OPTIGA™ application is already launched
- The trust anchor for verifying the image hash is properly installed at the OPTIGA™

Post-condition:

- The host software image is proven being integrity correct

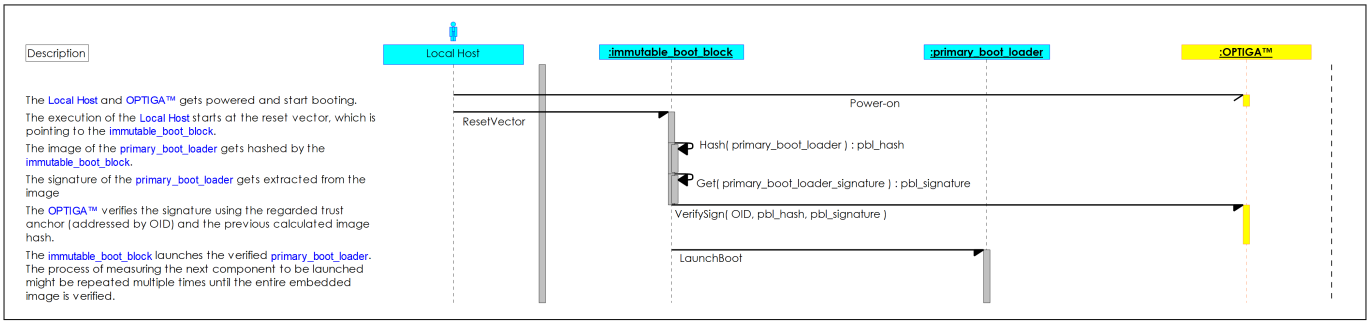

Figure 13 Use case: Verified boot -toolbox-

2.3.6 Use case: Update platform binding secret during runtime (pre-shared secret based)

This 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-conditions:

- 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 (example: CONF (E140))

Post-condition:

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

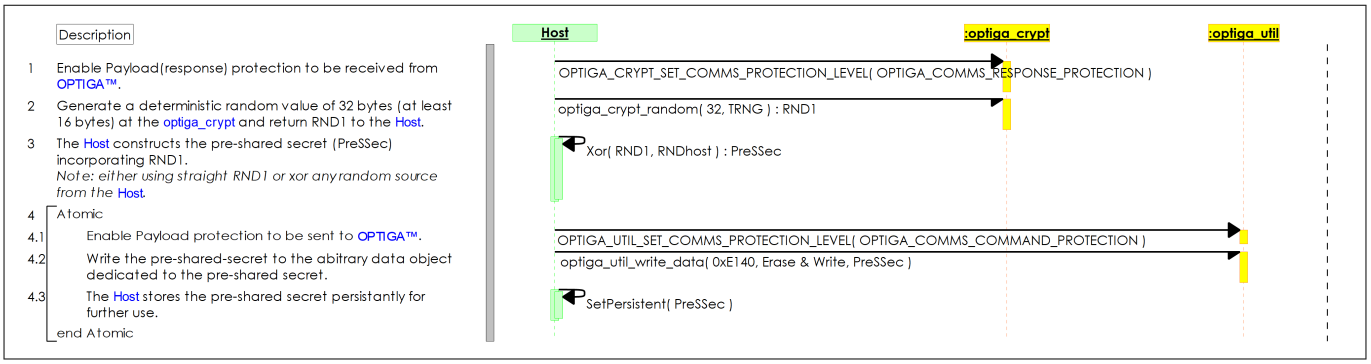

Figure 14 Use case: Update platform binding secret during runtime (pre-shared secret based)

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

The host needs to protect data against access by any third party. This sequence diagram is about high volume data encryption at the host. For that purpose, the host and OPTIGA™ establish a unique key for local data encryption/decryption. Host generates a random secret once and uses it for lifetime to derive the actual secret used for encrypt/decrypt of local data by the host.

Pre-conditions:

- Either there is at least one arbitrary data object (Table 79) of type 3 (in this example OID = 0xF1D1) available at the OPTIGA™ to save the unique secret for local encryption
- Or the unique secret for local encryption is already saved and locked
- The OPTIGA™ shielded connection is activated (presentation layer of the I2C [2] protocol is present) and it is recommended to be used for all commands and responses carrying secret data

Post-condition:

- The local secret for encryption is known by the host

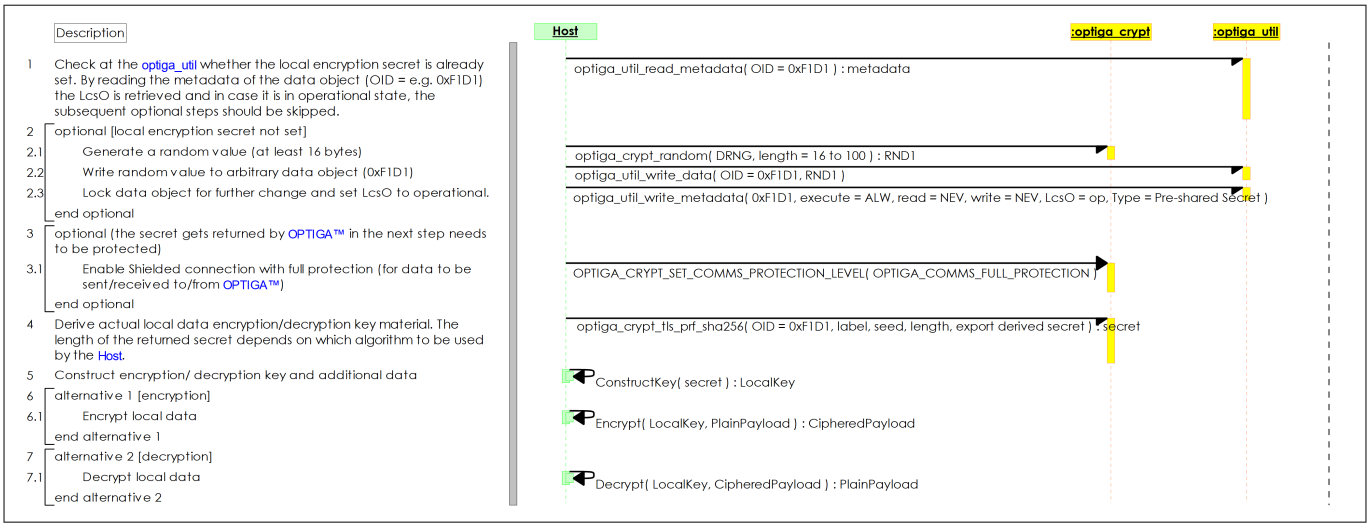

Figure 15 Use case: Local "data-at-rest" protection

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

The 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

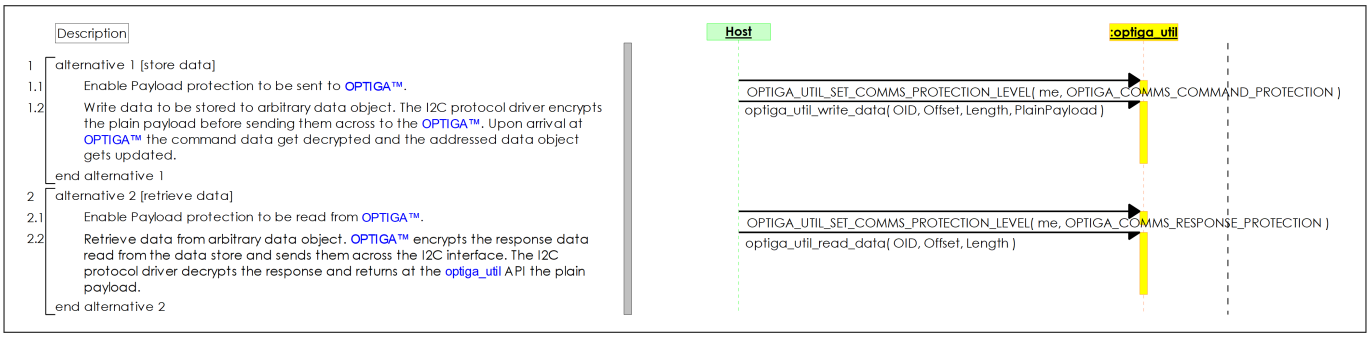

Figure 16 Use case: Local "data-at-rest" and "data-in-transit" protection

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

The host needs to protect data against access by any third party. This sequence diagram is about protecting higher volume of data at the host persistent storage. For that purpose OPTIGA™ encrypts (writing) or decrypts (reading) the data to be store at the host. The host has to persistently store the encrypted data objects at its NVM.

Note: The OPTIGA™ Trust M V1 does not support symmetric algorithms.

Pre-conditions:

- The symmetric key for local data protection is randomly generated and available at the OPTIGA™
- The OPTIGA™ shielded connection is enabled

Post-condition:

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

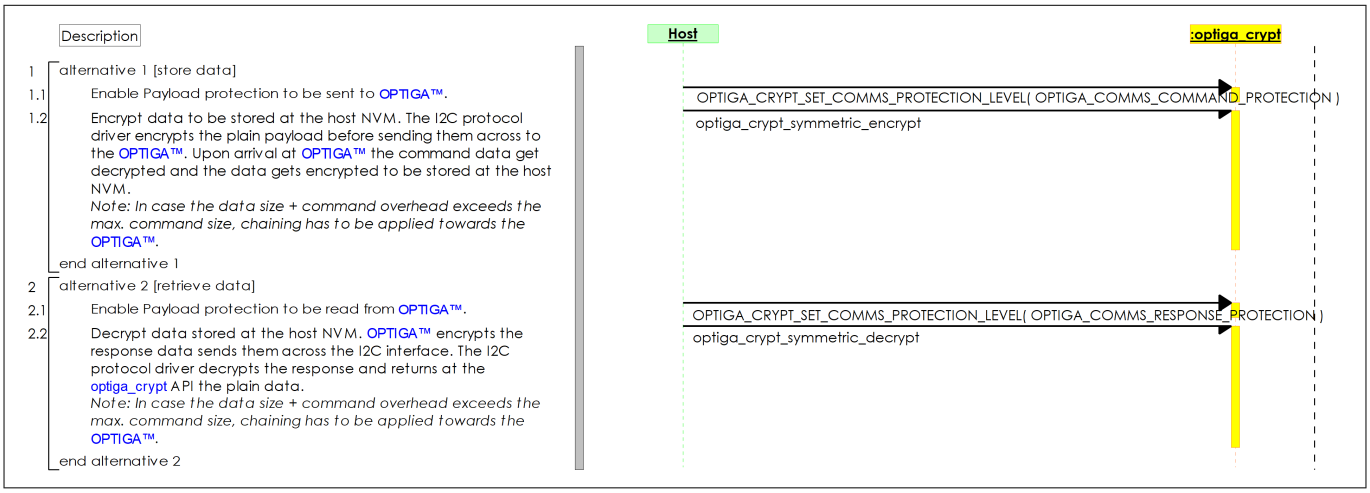

Figure 17 Use case: Host "data-at-rest" and "data-in-transit" protection

2.3.10 Use case: Generate MAC (HMAC with SHA2)

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

Note: The OPTIGA™ Trust M V1 does not support HMAC based operations.

Pre-condition:

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

Post-condition:

- The generated MAC is available for local host application for further usage

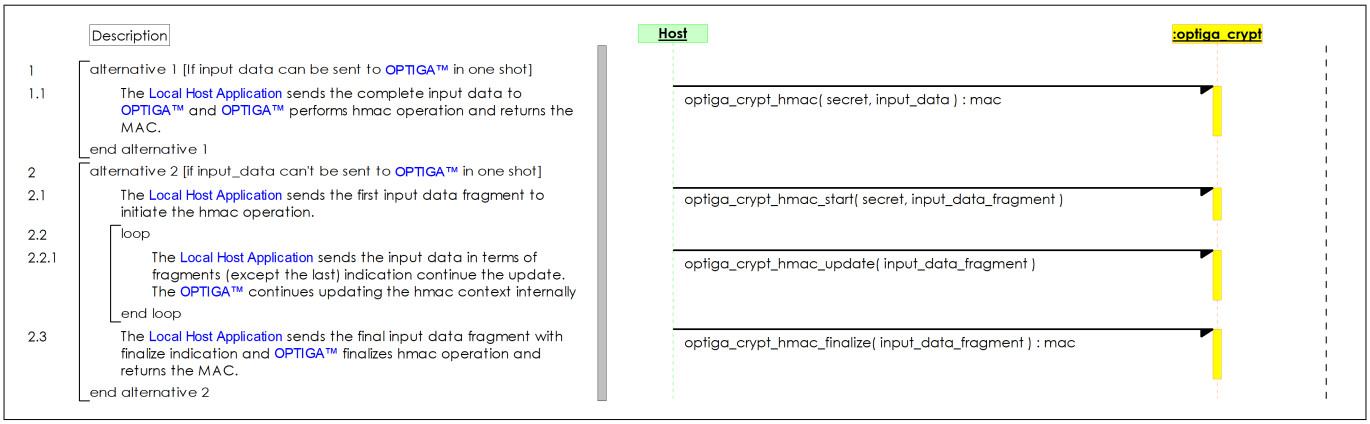

Figure 18 Use case: Generate MAC (HMAC with SHA2)

2.3.11 Use case: Verify authorization (HMAC with SHA2)

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

Note: The OPTIGA™ Trust M V1 does not support HMAC based operations.

Pre-condition:

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

Post-condition:

- The satisfied access condition is available at local host application for further usage

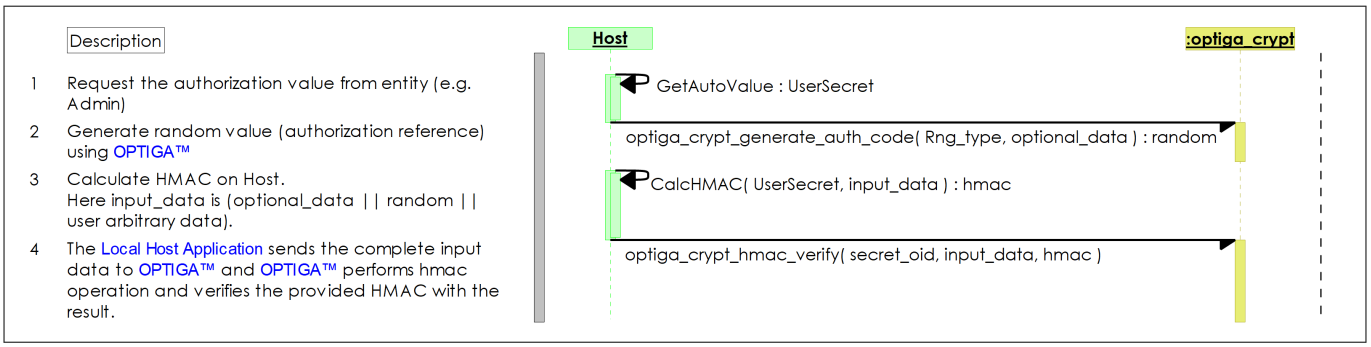

Figure 19 Use case: Verify authorization (HMAC with SHA2)

2.3.12 Use case: Generate hash

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

Pre-condition:

- The OPTIGA™ is initialized

Post-condition:

- The generated hash is available for local host application for further usage

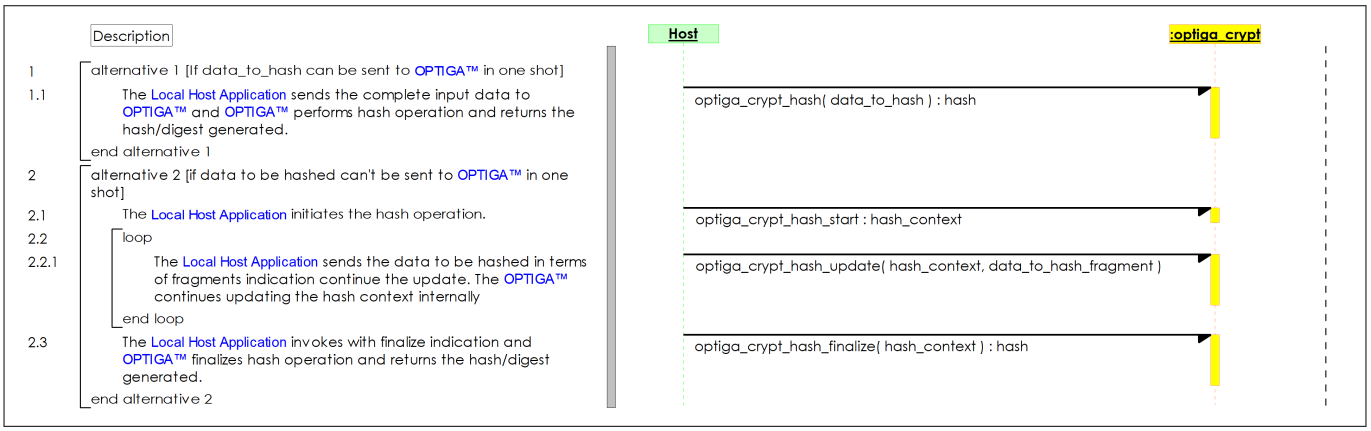

Figure 20 Use case: Generate hash

3 Enabler APIs

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 Figure 21 shows the 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 can be provided as source or in binary format
- Platform ported components (green) are used to connect platform specific and platform agnostic components. Those components exposing a platform agnostic API and calling platform specific APIs
- Third party components (light blue) need to be ported to platform agnostic APIs

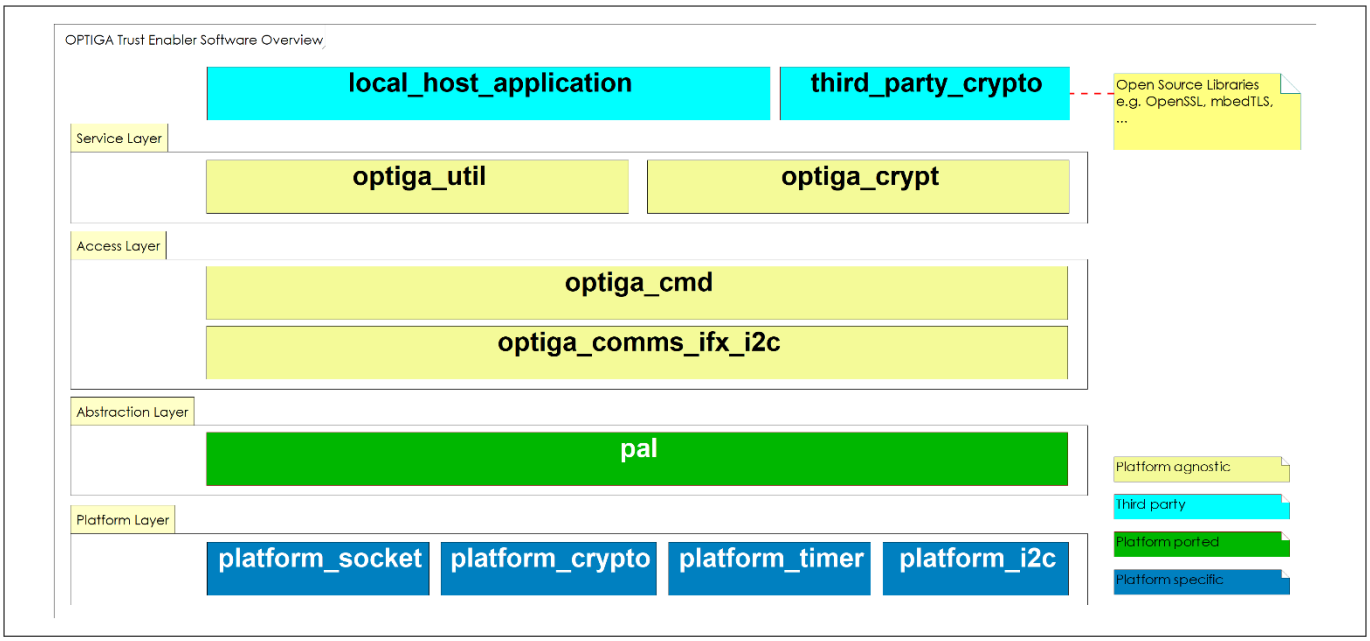

Figure 21 OPTIGA™ Trust enabler software overview

3.1 Service layer decomposition

Figure 22 shows the components providing the services at the main application interface.

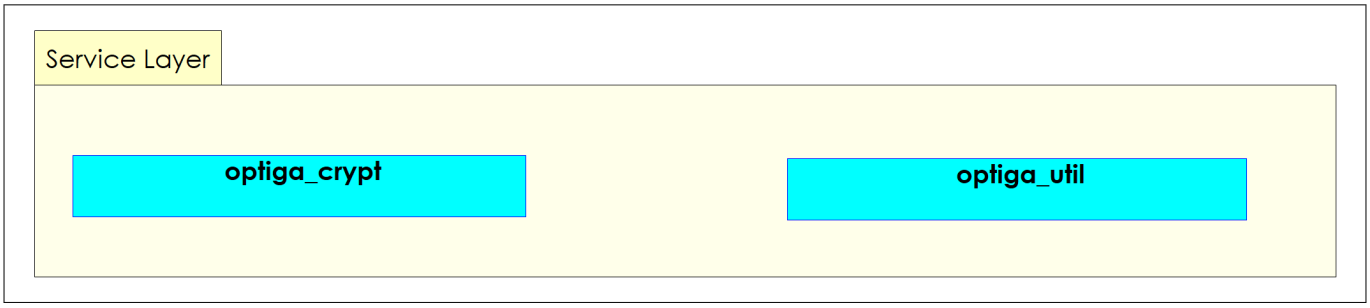

Figure 22 Service layer decomposition

3.1.1 optiga_crypt

The optiga_crypt module provides cryptographic toolbox 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 (example: Initialization, shielded connection settings) operations

Table 3 optiga_crypt - basic (example: 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	The possible shielded connection parameter types that can be set are version (example: Pre-shared secret based) and protection level (example: 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_CRYPT_SET_COMMS_PROTOCOL_VERSION • OPTIGA_CRYPT_SET_COMMS_PROTECTION_ LEVEL

3.1.1.2 Random generation operations

Table 4 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 5 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 example: 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 6 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 can be the following: • 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 maximum size of certificate allowed is 1300 bytes. More details about certificate parameters are specified in Parameter validation

(table continues...)

Table 6 (continued) optiga_crypt - ECC based APIs

API name	Description
optiga_crypt_ecdh	This operation generates shared secret. The OPTIGA™ performs ECDH operation using the referred private key and provided public key. • Here the private key is from OPTIGA™ referring to a static key store OID or session based. In case of session based, the private key is used from the session already acquired • The public key has to be sourced from host The generated shared secret can be either exported to the host or stored in OPTIGA™ session acquired by the respective optiga_crypt instance.

3.1.1.5 RSA based operations

Table 7 optiga_crypt - RSA based APIs

API name	Description
optiga_crypt_rsa_generate_keypair	This operation generates RSA key pair using OPTIGA™. The generated private key could be either stored in OPTIGA™ (static private key from key store) or exported to host
optiga_crypt_rsa_sign	This operation generates signature using RSA based static private key from key store (Table 26) in OPTIGA™
optiga_crypt_rsa_verify	This operation verifies the signature using OPTIGA™. The RSA public key could be either sourced from host or referring to OID (data object which holds a certificate) in OPTIGA™. • If the public key is from host, the format of public key (from host) is provided in RSA public key • If 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 maximum size of certificate allowed is 1300 bytes. More details about certificate parameters are specified in Parameter validation section

(table continues...)

Table 7 (continued) optiga_crypt - RSA based APIs

API name	Description
optiga_crypt_rsa_encrypt_message	This operation encrypts the message or data provided using OPTIGA™. The RSA public key could be either sourced from host or referring to a OID (data object which holds a certificate) in OPTIGA™. The message length that can be encrypted is limited as per [3]. The caller of this operation has to take care of chaining of message if message length is more than supported in a single operation. For example, in case of OPTIGA_RSAES_PKCS1_V15 and RSA 1024-bit key length, the maximum allowed message length is [128 (key length) - 11] = 117 bytes. The examples for the format of public key (from host) are provided in RSA public key.
optiga_crypt_rsa_encrypt_session	This operation encrypts the data from acquired session in OPTIGA™. The RSA public key could be either sourced from host or referring to OID (data object which holds a certificate) in OPTIGA™. If the shielded connection (OPTIGA_COMMS_SHIELDED_CONNECTION) is enabled, By default the optiga_cmd sends the command to OPTIGA™ with confidentiality protection.
optiga_crypt_rsa_decrypt_and_export	This operation decrypts the provided encrypted message using a RSA private key from OPTIGA™ and exports the decrypted message to the host. The encrypted message length must be the size of the key used to decrypt the message. For example, In case of RSA 2048 (key size = 256 bytes), the encrypted message length is 256 bytes. The caller of this operation has to take care of chaining of encrypted message if length is more than supported in a single operation. If the shielded connection is enabled, the decrypted data/ message is received with confidentiality protection from OPTIGA™.
optiga_crypt_rsa_decrypt_and_store	This operation decrypts the provided encrypted message using the referred RSA static private key from OPTIGA™ and stores message in the acquired session. The encrypted message length must be the size of the key used to decrypt the message. For example, In case of RSA 2048 (key size = 256 bytes), the encrypted message length is 256 bytes.

(table continues...)

Table 7 (continued) optiga_crypt - RSA based APIs

API name	Description
optiga_crypt_rsa_generate_pre_master_secret	This operation generates pre-master secret (optional data || random stream) for RSA key exchange and stores in the acquired session. • This operation acquires a session if not already acquired • The minimum size of random stream is 8 bytes The maximum size of pre-master secret allowed to store in the acquired session is 66 bytes (in case of OPTIGA™ Trust M V1, 48 bytes only). For example, in case of RSA key exchange based TLS communication, the client generates 48 bytes of pre-master secret (version info [2] bytes || random [46] bytes).

3.1.1.6 Symmetric based operations

Note: The OPTIGA™ Trust M V1 does not support symmetric operations.

Table 8 optiga_crypt - symmetric based APIs

API name	Description
optiga_crypt_symmetric_generate_key	This operation generates symmetric key (for example: AES) using OPTIGA™. The generated key could be either stored at OPTIGA™ (static key from key store) or exported to host.
optiga_crypt_symmetric_encrypt_ecb	This operation encrypts (OPTIGA_SYMMETRIC_ECB mode) the data provided using OPTIGA™. • No internal padding is performed by OPTIGA™ • If the length of plain data to be encrypted cannot be sent to OPTIGA™ in one transaction, then plain data will be sent to OPTIGA™ in multiple fragments (each fragment is block length aligned) internally *Note: The shielded connection protection level with command protection is implicitly enabled if the shielded connection is enabled.*

(table continues...)

Table 8 (continued) optiga_crypt - symmetric based APIs

API name	Description
optiga_crypt_symmetric_encrypt	This operation encrypts (symmetric) MAC for the data provided using OPTIGA™. In case of MAC only based operations (for example: CBC-MAC, CMAC), only the MAC is returned. • Internal padding is performed by OPTIGA™ in case of OPTIGA_SYMMETRIC_CMAC, if the data provided is not block aligned while finalizing the operation • If the length of plain data to be encrypted cannot be sent to OPTIGA™ in one transaction, then plain data will be sent to OPTIGA™ in multiple fragments internally *Note: The shielded connection protection level with command protection is implicitly enabled if the shielded connection is enabled.*
optiga_crypt_symmetric_encrypt_start	This operation initiates encryption (symmetric) sequence for the provided data using OPTIGA™. The encrypted data gets returned to the host in terms of blocks (block length is based on the selected encryption mode). • In case of generating MAC, the generated MAC gets returned with the successful optiga_crypt_symmetric_encrypt_final operation • If the length of plain data cannot be sent to OPTIGA™ in one transaction, then plain data will be sent to OPTIGA™ in multiple fragments (each fragment must be block length aligned) internally *Note: The shielded connection protection level with command protection is implicitly enabled if the shielded connection is enabled.*
optiga_crypt_symmetric_encrypt_continue	This operation encrypts (symmetric) the data provided using OPTIGA™. • In case of generating MAC, the generated MAC gets returned with the successful optiga_crypt_symmetric_encrypt_final operation • No internal padding is performed by OPTIGA™ • If the length of plain data to be encrypted cannot be sent to OPTIGA™ in one transaction, then plain data will be sent to OPTIGA™ in multiple fragments (each fragment must be block length aligned) internally *Note: The shielded connection protection level with command protection is implicitly enabled if the shielded connection is enabled.*

(table continues...)

Table 8 (continued) optiga_crypt - symmetric based APIs

API name	Description
optiga_crypt_symmetric_encrypt_final	This operation encrypts (symmetric) the data provided using OPTIGA™. • In case of generating MAC, the generated MAC only gets returned upon successful completion of this operation • Internal padding is performed by OPTIGA™ in case of OPTIGA_SYMMETRIC_CMAC, if the data provided is not block aligned while finalizing the operation • If the length of plain_data to be encrypted cannot be sent to OPTIGA™ in one transaction, then plain_data will be sent to OPTIGA™ in multiple fragments (each fragment must be block length aligned except the last) internally *Note: The shielded connection protection level with command protection is implicitly enabled if the shielded connection is enabled.*
optiga_crypt_symmetric_decrypt_ecb	This operation encrypts (ECB mode as specified in [4]) the data provided using OPTIGA™. • No internal padding is performed by OPTIGA™ • If the length of encrypted data cannot be sent to OPTIGA™ in one transaction, then encrypted data will be sent to OPTIGA™ in multiple fragments (each fragment is block length aligned) internally *Note: The shielded connection protection level with command protection is implicitly enabled if the shielded connection is enabled.*
optiga_crypt_symmetric_decrypt	This operation decrypts the provided encrypted data using OPTIGA™ and returns the plain data to the host. • No internal padding is performed by OPTIGA™ • If the length of encrypted data to be decrypted cannot be sent to OPTIGA™ in one transaction, then encrypted data will be sent to OPTIGA™ in multiple fragments (each fragment must be block length aligned except the last) internally *Note: The shielded connection protection level with response protection is implicitly enabled if the shielded connection is enabled.*

(table continues...)

Table 8 (continued) optiga_crypt - symmetric based APIs

API name	Description
optiga_crypt_symmetric_decrypt_start	This operation initiates the decrypt (symmetric) sequence for the provided encrypted data using OPTIGA™. The plain data gets exported to the host in terms of blocks (block length is based on the encrypted data chosen) • No internal padding is performed by OPTIGA™ • If the length of encrypted data cannot be sent to OPTIGA™ in one transaction, then encrypted data will be sent to OPTIGA™ in multiple fragments (each fragment must be block length aligned) *Note: The shielded connection protection level with response protection is implicitly enabled if the shielded connection is enabled.*
optiga_crypt_symmetric_decrypt_continue	This operation decrypts the provided encrypted data using OPTIGA™ and returns the decrypted data to the host. • No internal padding is performed by OPTIGA™ • If the length of encrypted data cannot be sent to OPTIGA™ in one transaction, then encrypted data will be sent to OPTIGA™ in multiple fragments (each fragment must be block length aligned) internally *Note: The shielded connection protection level with response protection is implicitly enabled if the shielded connection is enabled.*
optiga_crypt_symmetric_decrypt_final	This operation decrypts the provided encrypted data using OPTIGA™ and returns the decrypted data to the host. • No internal padding is performed by OPTIGA™ • If the length of encrypted data cannot be sent to OPTIGA™ in one transaction, then encrypted data will be sent to OPTIGA™ in multiple fragments (each fragment must be block length aligned except the last) internally *Note: The shielded connection protection level with response protection is implicitly enabled if the shielded connection is enabled.*

3.1.1.7 HMAC, key derivation based operations

Note: The OPTIGA™ Trust M V1 does not support HMAC based operations.

Table 9 optiga_crypt - HMAC generation APIs

API name	Description
optiga_crypt_hmac	This operation performs HMAC operation using a shared secret at OPTIGA™. If the length of input data cannot be sent to OPTIGA™ in one transaction, then input data will be sent to OPTIGA™ in multiple fragments internally. *Note: The shielded connection protection level with command protection is implicitly enabled if the shielded connection is enabled.*
optiga_crypt_hmac_start	This operation initiates HMAC operation. • If the length of input data cannot be sent to OPTIGA™ in one transaction, then input data will be sent to OPTIGA™ in multiple fragments internally *Note: The shielded connection protection level with command protection is implicitly enabled if the shielded connection is enabled.*
optiga_crypt_hmac_update	This operation performs HMAC operation for the data provided using OPTIGA™. • If the length input data cannot be sent to OPTIGA™ in one transaction, then input data will be sent to OPTIGA™ in multiple fragments internally *Note: The shielded connection protection level with command protection is implicitly enabled if the shielded connection is enabled.*
optiga_crypt_hmac_finalize	This operation performs HMAC operation for the data provided using OPTIGA™. • If the length of input data cannot be sent to OPTIGA™ in one transaction, then input data will be sent to OPTIGA™ in multiple fragments internally *Note: The shielded connection protection level with command protection is implicitly enabled if the shielded connection is enabled.*

Note: The OPTIGA™ Trust M V1 does not support HMAC authorization operations.

Table 10 optiga_crypt - HMAC based authorization APIs

API name	Description
optiga_crypt_generate_auth_code	This operation generates random data using OPTIGA™ which gets stored in the acquired session context at OPTIGA™. The random stored in the acquired session context gets used as authorization challenge for HMAC verify (optiga_crypt_hmac_verify) kind of operations.
optiga_crypt_hmac_verify	This operation performs HMAC verification for the provided authorization value using OPTIGA™. • This operation uses the session already acquired to store the authentication code (generated using optiga_crypt_generate_auth_code) • The size of input data is based on the respective hash algorithm used in the HMAC scheme (630 bytes - hmac_length) Upon successful verification of provided HMAC, the achieved AUTO state using the respective secret is maintained by OPTIGA™. The achieved state can be cleared by invoking optiga_crypt_clear_auto_state operation. *Note: The shielded connection protection level with response protection is implicitly enabled if the shielded connection is enabled.*
optiga_crypt_clear_auto_state	This operation clears the achieved authorization (AUTO) state (using optiga_crypt_hmac_verify operation) at OPTIGA™. The acquired session gets released after completion of this operation (irrespective of status of the operation once after the command is sent to OPTIGA™). *Note: The shielded connection protection level with response protection is implicitly enabled if the shielded connection is enabled.*

Table 11 optiga_crypt - key derivation APIs

API name	Description
optiga_crypt_tls_prf	This operation derives shared secret or key using OPTIGA™. OPTIGA™ performs PRF (as specified in TLS v1.2) operation as per type chosen using the referred data object ID or session ID holding a secret. There are dedicated APIs (macro based) for the respective hash algorithm used as part of PRF. • optiga_crypt_tls_prf_sha256 • optiga_crypt_ tls_prf_sha384 • optiga_crypt_ tls_prf_sha512 *Note: The OPTIGA™ Trust M V1 does not support PRF (with SHA384/512) algorithms.*

(table continues...)

Table 11 (continued) optiga_crypt - key derivation APIs

API name	Description
optiga_crypt_hkdf	This operation derives shared secret or key using OPTIGA™. The OPTIGA™ performs HKDF (as specified in [5]) operation using the referred data object ID or session ID, which holds a secret. There are dedicated APIs (macro based) for the respective hash algorithm used as part of HKDF. • optiga_crypt_hkdf_sha256 • optiga_crypt_hkdf_sha384 • optiga_crypt_hkdf_sha512 *Note: The OPTIGA™ Trust M V1 does not support HKDF algorithm.*

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 can be created to allow concurrent access to other services
- Uses optiga_cmd module to interact with the OPTIGA™

3.1.2.1 Basic (example: Initialization, shielded connection settings) operations

Table 12 optiga_util - basic (example: 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 (example: Pre-shared secret based) and protection level (example: 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 13 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™. 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 V_{cc} 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™ does not allow the hibernate operation. Therefore, this operation leads to failure.*

3.1.2.3 Read and write operations

Table 14 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 and write) or Write. OPTIGA_UTIL_ERASE_AND_WRITE (erase and 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™

(table continues...)

Table 14 (continued) optiga_util - read and write APIs

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

3.1.2.4 Protected update operations

Table 15 optiga_util - protected update APIs

API Name	Description
optiga_util_protected_update_start	This operation initiates the protected update of data/key objects in OPTIGA™. *Note: The OPTIGA™ Trust M V1 does not support confidentiality update and key and metadata update.* 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.

(table continues...)

Table 15 (continued) optiga_util - protected update APIs

API Name	Description
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. Example: 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. *Note: 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_update_final	This operation sends the last fragment and finalizes the protected update of 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

Figure 23 shows the components providing an agnostic interface to the underlying hardware and software platform functionality used by the higher-level components of the architecture.

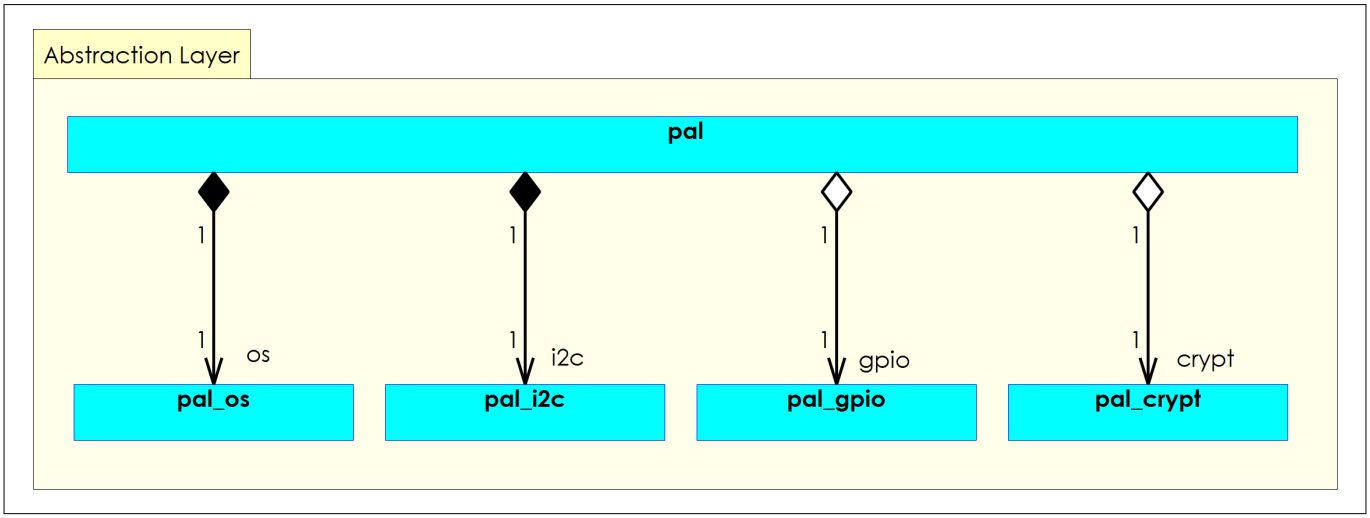

Figure 23 Abstraction layer decomposition

3.2.1 pal

The platform abstraction layer (PAL) is a, abstracting hardware and operating system functionalities for the Infineon XMC™ family of µController (MCU) or upon porting to any other MCU. It abstracts away the low level

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

Table 16 pal APIs

API name	Description
pal_init	This operation initializes the pal and aggregated pal modules (example: pal_i2c_init, pal_gpio_init, pal_os_init, etc..,)
pal_deinit	This operation deinitializes the pal and aggregated pal modules (example: 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 software libraries or HW) and is exposing cryptographic primitives invoked by platform agnostic modules.

Table 17 pal_crypt APIs

API name	Description
pal_crypt_tls_prf_sha256	This operation derives the secret using the TLS v1.2 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

The pal_gpio module provides APIs to set GPIO high/low to perform below operations.

- Power on/off
- Hardware reset on/off

Table 18 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 hardware 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 19 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 (example: RTOS) based functionality, which is invoked by platform agnostic modules.

Table 20 pal_os 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

(table continues...)

Table 20 (continued) pal_os APIs

API name	Description
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 21 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
OPTIGA_ECC_CURVE_NIST_P_521	Curve type - ECC NIST P-521
OPTIGA_ECC_CURVE_BRAIN_POOL_P_256R1	Curve type - ECC Brainpool P256r1
OPTIGA_ECC_CURVE_BRAIN_POOL_P_384R1	Curve type - ECC Brainpool P384r1
OPTIGA_ECC_CURVE_BRAIN_POOL_P_512R1	Curve type - ECC Brainpool P512r1

Note: The OPTIGA™ Trust M V1 does not support Brainpool and ECC NIST P 521 curves.

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

Table 22 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 23 optiga_hash_type_t

Name	Description
OPTIGA_HASH_TYPE_SHA_256	Generate digest using SHA256

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

Table 24 optiga_hkdf_type_t

Name	Description
OPTIGA_HKDF_SHA_256	Key derivation using HKDF-SHA256 [6]
OPTIGA_HKDF_SHA_384	Key derivation using HKDF-SHA384 [6]

(table continues...)

Table 24 (continued) optiga_hkdf_type_t

Name	Description
OPTIGA_HKDF_SHA_512	Key derivation using HKDF-SHA384 [6]

Note: The OPTIGA™ Trust M V1 does not support HKDF.

Types of HMAC generation supported by OPTIGA™.

Table 25 optiga_hmac_type_t

Name	Description
OPTIGA_HMAC_SHA_256	Generate MAC using HMAC-SHA256 [7]
OPTIGA_HMAC_SHA_384	Generate MAC using HMAC-SHA384 [7]
OPTIGA_HMAC_SHA_512	Generate MAC using HMAC-SHA512 [7]

Note: The OPTIGA™ Trust M V1 does not support HMAC.

Key slot IDs in OPTIGA™.

Table 26 optiga_key_id_t

Name	Description
OPTIGA_KEY_ID_E0F0	Key from key store (non-volatile) Supports only ECC (refer Table 21)
OPTIGA_KEY_ID_E0F1	Key from key store (non-volatile) Supports only ECC (refer Table 21)
OPTIGA_KEY_ID_E0F2	Key from key store (non-volatile) Supports only ECC (refer Table 21)
OPTIGA_KEY_ID_E0F3	Key from key store (non-volatile) Supports only ECC (refer Table 21)
OPTIGA_KEY_ID_E0FC	Key from key store (non-volatile) Supports only RSA (refer Table 30)
OPTIGA_KEY_ID_E0FD	Key from key store (non-volatile) Supports only RSA (refer Table 30)
OPTIGA_KEY_ID_E200 *Note: The OPTIGA™ Trust M V1 does not support Symmetric Key (OPTIGA_KEY_ID_E200).*	Key from key store (non-volatile) Supports AES 128/192/256 key types (refer Table 30)
OPTIGA_KEY_ID_SESSION_BASED	Key from session context (volatile)

Types of key usage.
The multiple key usage types can be selected based on the requirement and key type.

For example, if the private key from OPTIGA™ to be used for key agreement (Diffie-Hellmann) and signature generation purpose, then the key usage can be selected as (OPTIGA_KEY_USAGE_SIGN | OPTIGA_KEY_USAGE_KEY_AGREEMENT).

Table 27 optiga_key_usage_t

Name	Description
OPTIGA_KEY_USAGE_AUTHENTICATION	Allows to use the private key for the signature generation as part of authentication and sign commands
OPTIGA_KEY_USAGE_ENCRYPTION	Allows to use the (private) key for encrypt and decrypt operations. This type is applicable for RSA or AES key type only
OPTIGA_KEY_USAGE_SIGN	Allows to use the private key for the signature generation as part of sign command
OPTIGA_KEY_USAGE_KEY_AGREEMENT	Allows to use the private key for key agreement (for example: ECDH operations)

Types of random number generation supported by OPTIGA™.

Table 28 optiga_rng_type_t

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

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

Table 29 optiga_rsa_encryption_scheme_t

Name	Description
OPTIGA_RSAES_PKCS1_V15	Encryption scheme - RSAES PKCS1-v1_5

Types of RSA keys supported by OPTIGA™.

Table 30 optiga_rsa_key_type_t

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

The RSA signature schemes supported by OPTIGA™ for sign and verify.

Table 31 optiga_rsa_signature_scheme_t

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

Note: The OPTIGA™ Trust M V1 does not support RSA SSA PKCS#1 v1.5 SHA512.

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

Table 32 optiga_symmetric_encryption_mode_t

Name	Description
OPTIGA_SYMMETRIC_ECB	Symmetric encryption mode - ECB mode as specified by #unique_17/unique_17_Connect_42_li_mf1_rzy_x5b
OPTIGA_SYMMETRIC_CBC	Symmetric encryption mode - CBC as specified by #unique_17/unique_17_Connect_42_li_mf1_rzy_x5b
OPTIGA_SYMMETRIC_CBC_MAC	Symmetric encryption mode - CBC MAC (MAC generation) as specified by [8] [MAC Algorithm 1]
OPTIGA_SYMMETRIC_CMAC	Symmetric encryption mode - CMAC (MAC generation) as specified by [9]

Note: The OPTIGA™ Trust M V1 does not support above specified symmetric encryption and MAC algorithms:

Types of symmetric keys supported by OPTIGA™.

Table 33 optiga_symmetric_key_type_t

Name	Description
OPTIGA_SYMMETRIC_AES_128	AES 128
OPTIGA_SYMMETRIC_AES_192	AES 192
OPTIGA_SYMMETRIC_AES_256	AES 256

Note: The OPTIGA™ Trust M V1 does not support above specified symmetric keys.

Types of key derivation based on TLS v1.2 PRF supported by OPTIGA™.

Table 34 optiga_tls_prf_type_t

Name	Description
OPTIGA_TLS12_PRF_SHA_256	Key derivation using TLS v1.2 PRF-SHA256 [10]
OPTIGA_TLS12_PRF_SHA_384	Key derivation using TLS v1.2 PRF-SHA384 [10]
OPTIGA_TLS12_PRF_SHA_512	Key derivation using TLS v1.2 PRF-SHA512 [10]

Note: The OPTIGA™ Trust M V1 does not support PRF with SHA384 and SHA512.

4 OPTIGA™ Trust M external interface

This chapter provides the detailed definition of the OPTIGA™ device commands and responses available at its I2C [1] interface.

4.1 Warm reset

The warm reset (reset w/o power off/on cycle) of the OPTIGA™ might be triggered either by hardware signal or by software. In case of a hardware triggered warm reset the RST pin must be set to low (for more details refer to [11]). In case of a software triggered Warm Reset the I2C master must write to the SOFT_RESET register (for more details refer to [2]).

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 hardware device in steps of 1 mA from 6 mA to 15 mA with a precision of ±5% (refer to Table 68).

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 24.

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 (for example: Counting down the SEC). In case all idle tasks are finished and no command is pending, the OPTIGA™ enters sleep mode

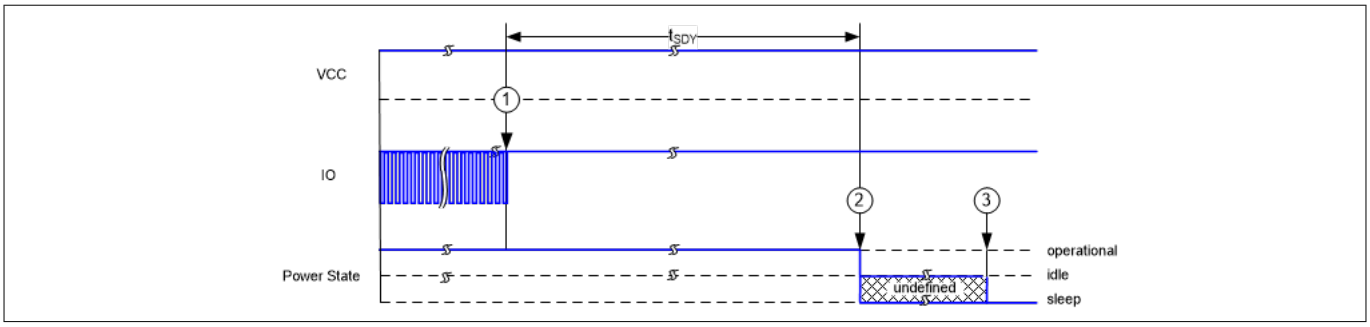

Figure 24 Go-to-sleep diagram

4.3 Protocol stack

The OPTIGA™ is an I2C slave device. The protocol stack from the physical up to the application layer is specified in [2]. The protocol is defined for point-to-point connection and a multilayer 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 shown below.

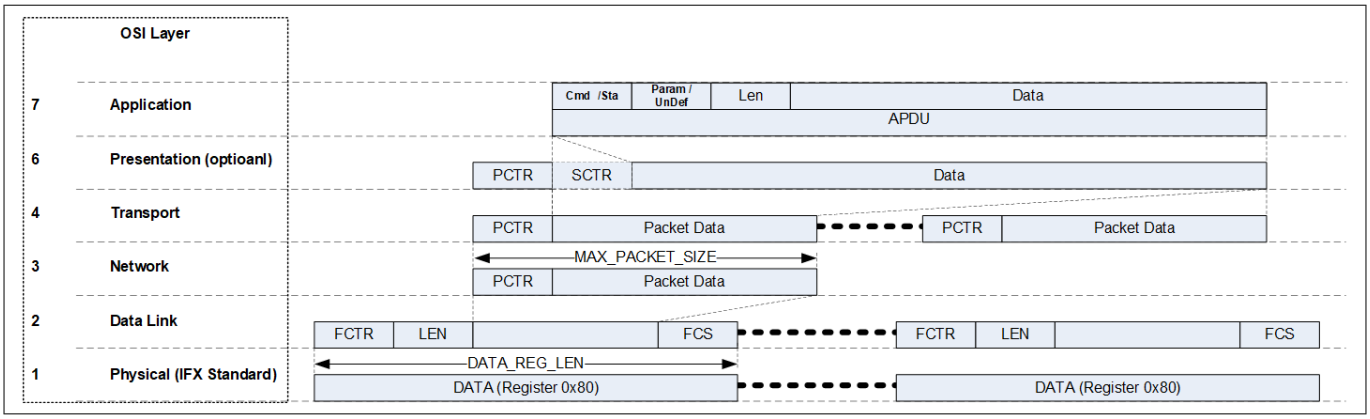

Figure 25 Overview protocol stack used

The physical layer is entirely defined in [12]. 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 [2]. 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 Commands of this document.

The protocol variation for the OPTIGA™ is defined by Table 35.

Table 35 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	-
PWR_SAVE_TIMEOUT	-	Not implemented
BASE_ADDR	0x30	I2C base address default
MAX_SCL_FREQU	1000[2)]	KHz
GUARD_TIME	50	µs
I2C_STATE	-	SOFT_RESET = 1; CONT_READ = 0; REP_START = 0; CLK_STRETCHING = 0; PRESENT_LAYER = 1;

2 The default setting is 400 KHz

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 36 lists the command codes for the functionality provided by the OPTIGA™.

Table 36 Command codes

Command code	Command	Short description
0x01 or 0x81	GetDataObject	Command to get (read) an data object
0x02 or 0x82	SetDataObject	Command to set (write) an data object
0x03 or 0x83	SetObjectProtected	Command to set (write) a key or data object or metadata of a key or data object protected
0x0C or 0x8C	GetRandom	Command to generate a random stream
0x14 or 0x94	EncryptSym	Command to encrypt data based on a symmetric key scheme
0x15 or 0x95	DecryptSym	Command to decrypt data based on a symmetric key scheme
0x1E or 0x9E	EncryptAsym	Command to encrypt data based on an asymmetric key scheme
0x1F or 0x9F	DecryptAsym	Command to decrypt data based on an asymmetric key scheme
0x30 or 0xB0	CalcHash	Command to calculate a hash
0x31 or 0xB1	CalcSign	Command to calculate a signature
0x32 or 0xB2	VerifySign	Command to verify a signature
0x33 or 0xB3	CalcSSec	Command to execute a Diffie-Hellmann key agreement
0x34 or 0xB4	DeriveKey	Command to derive keys
0x38 or 0xB8	GenKeyPair	Command to generate public key pairs
0x39 or 0xB9	GenSymKey	Command to generate symmetric (secret) keys
0x70 or 0xF0	OpenApplication	Command to launch an application
0x71 or 0xF1	CloseApplication	Command to close/terminate an application

Note: The OPTIGA™ Trust M V1 does not support EncryptSym, DecryptSym, and GenSymKey commands.

Table 37 lists the fields contained in a command and response APDU.

Table 37 APDU fields

Name	Description
Cmd	Command code[3)] as defined in Table 36
Param	Parameter to control major variants of a command. For more details, refer to the particular command definition.
InLen	Length of the command data section

(table continues...)

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

Table 37 (continued) APDU fields

Name	Description
InData	Command data section
Sta	Response status code as defined in Table 38
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 can be:

- An ephemeral key
- A shared session secret, etc..,

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

Table 38 lists the status codes provided by a response APDU.

Table 38 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 39). In this case, the OutData field is absent

The possible error codes are list in Table 39. If multiple commands happen to produce subsequent errors then only the highest-numbered error code is stored.

Table 39 Error codes

Field	Code	Description
No error	0x00	No error
Invalid OID	0x01	Invalid OID
Invalid parameter field	0x03	Invalid parameter 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

(table continues...)

Table 39 (continued) Error codes

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 maximum 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 unsupported 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
Invalid metadata of the key/data object	0x11	A command is acting on metadata for key or data objects and the current metadata are invalid
Unsupported extension/identifier	0x24	• An unsupported extension found in the message • Unsupported key usage/algorithm extension/identifier for the usage of private key
Unsupported parameters	0x25	• At least one parameter received in the handshake message is not supported • Unsupported parameter in the command APDU InData
Invalid certificate format	0x29	Invalid certificate(s) in certificate message with the following reasons: • Invalid format • Invalid chain of certificates • Signature verification failure
Unsupported certificate	0x2A	• The size of the certificate is more than the 1300 bytes where OPTIGA™ cannot parse the certificate internally due to insufficient memory (or) • At least one cryptographic algorithm specified in the certificate is not supported (example: Hash or sign algorithms)
Signature verification failure	0x2C	Signature verification failure

(table continues...)

Table 39 (continued) Error codes

Field	Code	Description
Integrity validation failure	0x2D	Message integrity validation failure (example: During CCM decryption)
Decryption failure	0x2E	Decryption failure
Authorization failure	0x2F	Session random comparison failure or HMAC verification 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 40 OpenApplication

Field	Offset [direction]	Description
Cmd	0 [in]	0x70 | 0xF0[4] command code
Param	1 [in]	0x00 initialize a clean application context 0x01 restore the application context from the previously saved context
InLen	2 [in]	0xXXXX length of InData
InData	4 [in]	Param = 0x00: • 0x00-0xFF unique application identifier (refer to Table 78) Param = 0x01: • 0x00-0xFF unique application identifier (refer to Table 78) • 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
OuData	4 [out]	Absent

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

[4] In case of 0xF0 the last error code gets flushed

the OS for further reuse. After the CloseApplication command is successful executed no further commands specified for the closed application, except OpenApplication, is available. Optionally, this command might save the application context persistently. This allows surviving power-lost by keeping the achieved security state and session contexts of the application. This application context could be restored once by the next OpenApplication command.

The CloseApplication also writes the current SEC maintained in RAM (SEC_{CURR}) to security event counter (SEC), if SEC_{CURR} is not same as in security event counter (SEC).

Table 41 CloseApplication

Field	Offset [direction]	Description
Cmd	0 [in]	0x71 | 0xF1[5] command code
Param	1 [in]	0x00 close the application instance without saving the application context. 0x01 saving the application context, closes the application instance, and return the random (TRNG) context handle[6].
InLen	2 [in]	0x0000 length of InData
InData	4 [in]	Absent
Sta	0 [out]	0x00 | 0xFF response status code
UnDef	1 [out]	0x00-0xFF undefined value
OutLen	2 [out]	0x0000 or 0x0008 length of OutData
OutData	4 [out]	Param = 0x00 Absent Param = 0x01: • 0x00-0xFF (8 bytes) context handle to be used by the OpenApplication command as reference for restoring the context (Param = 0x01)

4.4.1.3 GetDataObject

This 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 and length as appropriate.

Table 42 GetDataObject

Field	Offset [direction]	Description
Cmd	0 [in]	0x01 | 0x81[7] command code
Param	1 [in]	• 0x00 read data • 0x01 read metadata

(table continues...)

5 In case of 0xF1 the last error code gets flushed
6 Saving the context is only possible when the current security event counter (SEC) value is zero, otherwise it returns an error "command out of sequence" to the application
7 In case of 0x81 the last error code gets flushed

Table 42 (continued) GetDataObject

Field	Offset [direction]	Description
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 OPTIGA™ Trust M data structures) 0x0000-0xLLLL offset within the data object (0xLLLL denotes the length of the data object - 1) 0x0001-0xFFFF number of data bytes to be read. In case the length is longer than the available data the length will be adapted to the maximum possible length[8] and returned with the response APDU. (example: 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

4.4.1.4 SetDataObject

This 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 and length as appropriate.

Table 43 SetDataObject

Field	Offset [direction]	Description
Cmd	0 [in]	0x02 | 0x82[9] command code
Param	1 [in]	• 0x00 write data • 0x01 write metadata[10] • 0x02 count data object[11][12][13] • 0x40 erase and write data
InLen	2 [in]	0xXXXX length of InData

(table continues...)

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

Table 43 (continued) SetDataObject

Field	Offset [direction]	Description
InData	4 [in]	0x0000-0xFFFF OID of data object to be written (refer to OPTIGA™ Trust M data structures) 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

4.4.1.5 SetObjectProtected

This command is used to write data or metadata objects protected (integrity and optionally confidentiality) to the OPTIGA™. The field “Param” contains the manifest version of the update data set. The field “InData” contains the protected update data set to be written. The contained manifest addresses the protection keys and the target object.

Note:

- *The OPTIGA™ Trust M V1 does not support confidentiality protection and as well does not support updating keys and metadata*
- *This command supports chaining (start, continue, finalize) through partial write*
- *This command does not support the data objects specified below:*
  - *Life cycle status (global/application)*
  - *Security status (global/application)*
  - *Coprocessor UID*
  - *Sleep mode activation delay*
  - *Current limitation*
  - *Security event counter*
  - *Last error code 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 [13] encoding as specified in manifest and signature format (refer to Appendix)*
- *For “WriteType = Write" (refer to manifest CDDL format [14]), 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 44 SetObjectProtected

Field	Offset [direction]	Description
Cmd	0 [in]	0x03 | 0x83[14] command code
Param	1 [in]	• 0x01 manifest format (CDDL CBOR)
InLen	2 [in]	0xXXXX length of InData
InData	4 [in]	0x3y, 0xXXXX, 0x00-0xFF (start y = 0, final y = 1, continue y = 2) Start => manifest of update data set[15] Continue => first to n-1th fragment of update data set[16] Final => last fragment of update data set[17]
Sta	0 [out]	0x00 | 0xFF response status code
UnDef	1 [out]	0x00-0xFF undefined value
OutLen	2 [out]	0x0000 length of data
OutData	4 [out]	Absent

4.4.1.6 GetRandom

This command is used to generate a random stream to be used by various security schemes. The generated random stream is either returned and/or gets stored in the addressed session context (OID) if specified. The field “Param” contains the type of random stream (0x00 or 0x01).

Note: The OPTIGA™ Trust M V1 does not support storing random in session when Param is TRNG or DRNG.

Table 45 GetRandom

Field	Offset [direction]	Description
Cmd	0 [in]	0x0C | 0x8C[18] command code
Param	1 [in]	• 0x00 random number from TRNG (according to [15]) • 0x01 random number from DRNG (according to [16]) • 0x04 pre-master secret to be temporarily stored in the addressed session context[19]
InLen	2 [in]	0xXXXX length of InData
InData	4 [in]	0x0008-0x0100 length of random stream to be generated 0xE100-0xE103 OID of session context • [optional] in case of Param = 0x00-0x01 • [mandatory] in case of Param = 0x04 00x41, length, pre-pending optional data[20][21] • [mandatory] in case session OID is present

(table continues...)

14 In case of 0x83 the last error code gets flushed
15 Start will terminate and clear any not completed sequence (final not executed)
16 The length must be 640 bytes
17 The length must be in a range of 1 to 640 bytes
18 In case of 0x8C the last error code gets flushed
19 The DRNG mode gets used to generate the random value

Table 45 (continued) GetRandom

Field	Offset [direction]	Description
Sta	0 [out]	0x00 | 0xFF response status code
UnDef	1 [out]	0x00-0xFF undefined value
OutLen	2 [out]	0x0000-0xFFFF length of OutData
OutData	4 [out]	0x00-0xFF random stream • In case of Param = 0x00-0x01, the random stream gets returned *Note: Absent in case of Param = 0x04 (OutLen = 0x0000)*

4.4.1.7 EncryptSym

This command is used to protect data by the OPTIGA™, based on a secret key scheme or to calculate a MAC over provided data. Those data and their sequence of exchange between the OPTIGA™ and the connected host are defined in detail in Supported use cases. The data padding must be provided for the last block of data in case the used algorithm (refer to Table 63) does not define default padding.

Notes:

1. *The OPTIGA™ Trust M V1 does not support this command*
2. *In case the presentation layer protection is enabled the command must be protected. The response protection is up to the caller*
3. *In case of applied chaining, the sequence from start to final must be strict (no other command in between)*

Table 46 EncryptSym

Field	Offset [direction]	Description
Cmd	0 [in]	0x14 | 0x94[22] command code
Param	1 [in]	0xXX cipher suite as specified by Table 63, which define whether algorithm block aligned and padding must be provided
InLen	2 [in]	0xXXXX length of InData[23]

(table continues...)

20 The pre-pending optional data length plus the requested length of the random value shall not exceed 66 bytes (in case of OPTIGA™ Trust M V1, shall not exceed 48 bytes)
21 Length could be 0x0000
22 In case of 0x94 the last error code gets flushed
23 Maximum length of InData is 640 bytes

Table 46 (continued) EncryptSym

Field	Offset [direction]	Description
InData	4 [in]	• Secret key OID[24] *Notes:* *The secret key OID can be any of the below:* ***1.*** *Key object OID in case of AES based operations* ***2.*** *In case of hash based operations, any pre-shared secret* ***a.*** *Data object of type PRESSEC* ***b.*** *Session OID (post ECDH or derive key command)* • 0x0y, length[25][26][27][28], data (start y = 0, start and final y = 1, continue y = 2 and final y=3), data to be encrypted or hashed Below is optional depending on the applied cipher suite: • 0x40, 0xXXXX, 0x00-0xFF[29], Additional Data • 0x41, 0xXXXX, 0x00-0xFF[30], IV • 0x42, 0x0002, 0x00-0xFF[31], over all payload length without padding
Sta	0 [out]	0x00 | 0xFF response status code
UnDef	1 [out]	0x00-0xFF undefined value
OutLen	2 [out]	0xXXXX length of OutData
OutData	4 [out]	• 0x61, 0xXXXX, 0x00-0xFF Encrypted data

4.4.1.8 DecryptSym

This command is used to unprotected data by the OPTIGA™ or to verify a provided verification value, based on a secret key scheme. Those data and their sequence of exchange between the OPTIGA™ and the connected host are defined in detail in “Supported use cases”.

Notes:

1. *The OPTIGA™ Trust M V1 does not support this command*

2. *In case the presentation layer protection is enabled the response must be protected. The command protection is up to the caller*

24 [For continue and final] the provided OID gets ignored
25 [1] length must be > 0
26 [For start and continue] for block ciphers (example: AES) length must be algorithm block size aligned
27 [For start and final and final] for block ciphers (example: AES) length must be algorithm block size aligned and padded in case the "modes of operation" does not define default padding
28 [For start, start and final, continue, final] for hash-based modes (example: HMAC) length need not be algorithm block size aligned. Just byte alignment is sufficient
29 Might be present at start and start and final with CCM mode
30 Might be present at start and start and final with CBC/CCM mode
31 Present at start and start and final with CCM mode

3. *In case of applied chaining, the sequence from start to final must be strict (no other command in between)*
4. *In case of verification failure, at any execution state of the command, the auto(X) state for the respective data object gets cleared*

Table 47 DecryptSym

Field	Offset [direction]	Description
Cmd	0 [in]	0x15 | 0x95[32] command code
Param	1 [in]	0xXX cipher suite as specified by Table 63[33]
InLen	2 [in]	0xXXXX length of InData[34]
InData	4 [in]	• Secret key OID[35] *Notes:* *The secret key OID can be any of the below:* ***1.*** *Key object OID in case of AES based operations* ***2.*** *In case of hash based operations (HMAC verification), data object of type authorization reference (AUTOREF)* • 0x0y, length[36][37][38][39], data (start y = 0, start and final y = 1, continue y = 2 and final y=3), data to be decrypted/verified[40] Below are optional depending on the applied cypher suite: • 0x40, 0xXXXX, 0x00-0xFF[41], Additional Data • 0x41, 0xXXXX, 0x00-0xFF[42], IV • 0x42, 0x0002, 0x00-0xFF[43], over all payload length • 0x43, 0xXXXX[44], 0x00-0xFF[45], verification value
Sta	0 [out]	0x00 | 0xFF response status code
UnDef	1 [out]	0x00-0xFF undefined value
OutLen	2 [out]	0xXXXX length of OutData
OutData	4 [out]	Absent in case of verification: • 0x61, 0xXXXX, 0x00-0xFF Decrypted (plain) data[46]

32 In case of 0x95 the last error code gets flushed
33 Not applicable for MAC modes except HMAC
34 Maximum length of InData is 640 bytes
35 [For continue and final] the provided OID gets ignored
36 Length must be > 0
37 [For start and continue] for block ciphers (example: AES) length must be algorithm block size aligned
38 [For start and final and final] for block ciphers (example: AES) length must be algorithm block size aligned and padded in case the "modes of operation" does not define default padding
39 [For start and final] for hash-based modes (example: HMAC) length need not be algorithm block size aligned. Just byte alignment is sufficient
40 In case of verification the structure is (session OID || (optional data || random) stored in the respective session || arbitrary data) and only start and final is applicable. The random expires once used or by any verification failure avoiding replay attacks
41 Might be present at start and start and final with CCM mode
42 Might be present at start and start and final with CBC/CCM mode
43 Present at start and start and final with CCM mode
44 The length is defined by the output size of the used hash algorithm
45 Present at start and final with HMAC mode
46 Including padding in case the "modes of operation" does not define default padding

4.4.1.9 EncryptAsym

This command is used to protect an arbitrary message by the OPTIGA™, based on a public key scheme.

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

Table 48 EncryptAsym

Field	Offset [direction]	Description
Cmd	0 [in]	0x1E | 0x9E[47] command code
Param	1 [in]	0xXX cipher suite as specified by Table 59
InLen	2 [in]	0xXXXX length of InData
InData	4 [in]	Encryption input InData [InLen] • Alternate one: {(0x61, length[48], message)(0x02, 0x0002, OID of session context to be used in case a pre-master secret (from GetRandom) gets encrypted)} • Alternate one: {(0x04, 0x0002, OID of public key certificate[49][50] , (0x05, 0x0001, Algorithm Identifier (of the public key), 0x06, length, public key[51])}
Sta	0 [out]	0x00 | 0xFF response status code
UnDef	1 [out]	0x00-0xFF undefined value
OutLen	2 [out]	0xXXXX length of OutData
OutData	4 [out]	• 0x61, 0xXXXX, 0x00-0xFF Message data protected

4.4.1.10 DecryptAsym

This command is used to unprotect an arbitrary message by the OPTIGA™, based on a public key scheme.

Note: In case the presentation layer protection is enabled the response must be protected in case the unprotected data get exported. The command protection is up to the caller.

Table 49 DecryptAsym

Field	Offset [direction]	Description
Cmd	0 [in]	0x1F | 0x9F[52] command code
Param	1 [in]	0xXX cipher suite as specified by Table 59
InLen	2 [in]	0xXXXX length of InData

(table continues...)

47 In case of 0x9E the last error code gets flushed
48 The length of the message must be up to the key size minus the minimum size of the applied padding scheme
49 Must be a single certificate (DER coded) with the key usage as encryption according to [5]. The first byte of the object must be 0x30 (which is the start byte of a DER encoded certificate)
50 The key usage in certificate must be either KeyEncipherment (encrypting data from session or data from external interface) or DataEncipherment (encrypting data from external interface)
51 Public key is encoded as two DER INTEGER (modulus || public exponent) contained in a DER "BIT STRING"
52 In case of 0x9F the last error code gets flushed

Table 49 (continued) DecryptAsym

Field	Offset [direction]	Description
InData	4 [in]	Decryption input InData [InLen] • 0x61, length[53], protected message • 0x03, 0x0002, OID of decryption key[54] *Note: The key usage of the addressed key must be set to Enc; refer to Table 58.* Optional: • 0x02, 0x0002, OID of session context to store the decrypted data[55]
Sta	0 [out]	0x00 | 0xFF response status code
UnDef	1 [out]	0x00-0xFF undefined value
OutLen	2 [out]	0xXXXX length of OutData
OutData	4 [out]	• 0x61, 0xXXXX, 0x00-0xFF Message data unprotected *Note: Absent in case targeting the session context (OID of session context provided in InData).*

4.4.1.11 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 (example: CalcHash with start hashing) will terminate the currently running hash calculation and drop the result.

Table 50 CalcHash

Field	Offset [direction]	Description
Cmd	0 [in]	0x30 | 0xB0[56] command code
Param	1 [in]	0xXX hash algorithm identifier (refer to Table 57)
InLen	2 [in]	0xXXXX length of InData

(table continues...)

53 The length of the message must match the key size
54 The addressed decryption key shall be RSA private key
55 The length of the decrypted data shall not exceed 66 bytes (in case of OPTIGA™ Trust M V1, shall not exceed 48 bytes) and the usage is limited as input shared secret in DeriveKey command
56 In case of 0xB0 the last error code gets flushed

Table 50 (continued) CalcHash

Field	Offset [direction]	Description
InData	4 [in]	Hash input InData [InLen] (alternative one) • 0x0y, length[57], message data (start y = 0, start and final y = 1, continue y = 2, final y = 3, final and keep intermediate hash y = 5[58]) • 0x04, 0x0000 - To terminate the hash sequence in case initialized already • 0x1y, 0x0006, OID[59], offset, length[60] (start y = 0, start and 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 and final" (atomic) or "start-(zero to n-times continue)-terminate".*
Sta	0 [out]	0x00 | 0xFF response status code
UnDef	1 [out]	0x00-0xFF undefined value
OutLen	2 [out]	0xXXXX length of OutData
OutData	4 [out]	Digest or intermediate hash context data 0x01, length, hash/digest 0x06, length, intermediate hash context data *Note: 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: Intermediate hash context is only returned if indicated by InData.*

4.4.1.12 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.

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

Table 51 CalcSign

Field	Offset [direction]	Description
Cmd	0 [in]	0x31 | 0xB1[61] command code
Param	1 [in]	0xXX signature scheme (refer to Table 62)
InLen	2 [in]	0xXXXX length of InData
InData	4 [in]	Signature input InData [InLen] • 0x01, length[62], digest to be signed • 0x03, 0x0002, OID of signature key[63] *Note: The key usage of the addressed key must be set to sign or auth; refer to Table 58*
Sta	0 [out]	0x00 | 0xFF response status code
UnDef	1 [out]	0x00-0xFF undefined value
OutLen	2 [out]	0xXXXX length of OutData
OutData	4 [out]	0x00-0xFF signature[64] *Note: The length of the signature is derived from the applied key and signature scheme*

4.4.1.13 VerifySign

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

Table 52 VerifySign

Field	Offset [direction]	Description
Cmd	0 [in]	0x32 | 0xB2[65] command code
Param	1 [in]	0xXX signature scheme (refer to Table 62)
InLen	2 [in]	0xXXXX length of InData

(table continues...)

61 In case of 0xB1 the last error code gets flushed
62 For ECC shall be 10 bytes up to the length of the addressed signature key; RSA case: Must be exactly equal to the output length of the hash algorithm used
63 The addressed signing key shall be a private key
64 ECC case: The signature pair (r,s) is encoded as two DER "INTEGER"; RSA case: The signature encoding is OCTET STRING
65 In case of 0xB2 the last error code gets flushed

Table 52 (continued) VerifySign

Field	Offset [direction]	Description

InData	4 [in]	Signature input InData [InLen]
		• 0x01, length[66], digest
		• 0x02, length[67], signature over digest[68]
		• Alternate one: {
		(0x04, 0x0002, OID of public key certificate[69]),
		(0x05, 0x0001, refer to Table 57 (of the public key), 0x06, length, public key[70])
		}

Sta	0 [out]	0x00 | 0xFF response status code
UnDef	1 [out]	0x00-0xFF undefined value
OutLen	2 [out]	0x0000 length of OutData
OutData	4 [out]	Absent

4.4.1.14 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 53 GenKeyPair

Field	Offset [direction]	Description
Cmd	0 [in]	0x38 | 0xB8[71] command code
Param	1 [in]	0xXX algorithm identifier (refer to Table 57) of the key to be generated
InLen	2 [in]	0xXXXX length of InData
InData	4 [in]	Generate key pair input InData [InLen] • Alternative one: { [0x01, 0x0002, OID of private key[72] to be generated and stored as indicated by the OID (no private key export!). The public key gets exported in plain, 0x02, 0x0001, key usage (refer to Table 58)] 0x07, 0x0000 (export key pair in plain) }
Sta	0 [out]	0x00 | 0xFF response status code
UnDef	1 [out]	0x00-0xFF undefined value

(table continues...)

66 ECC case: The length of the digest must be up to the key size used for the signature (example: ECC256 = 32) and its maximum length is 64 bytes; RSA case: must be exactly equal to the output length of the hash algorithm used
67 The length is limited to maximum 520 bytes
68 ECC case: The signature pair (r,s) is encoded as two DER "INTEGER"; RSA case: The signature encoding is OCTET STRING
69 Must be a single certificate (DER coded) with the key usage either digitalSignature or keyCertSign according to [5]. The first byte of the object must be 0x30 (which is the start byte of a DER encoded certificate)
70 PubKey is encoded as DER "BIT STRING"
71 In case of 0xB8 the last error code gets flushed
72 Private key can either be a non-volatile device private key OR a session context (volatile), in which case the generated key has to be stored in the respective session context and can be addressed later

Table 53 (continued) GenKeyPair

Field	Offset [direction]	Description
OutLen	2 [out]	0xXXXX length of OutData
OutData	4 [out]	• Alternative one: [{ECC key 0x01, len, PrivKey[73] 0x02, len, PubKey[74]} {RSA key 0x01, len, PrivKey[75] 0x02, len, PubKey[76]}]

4.4.1.15 GenSymKey

This command is used to generate a symmetric key. The key gets stored at the provided OID of a Key or it gets returned to the caller in case no OID is provided.

Note: The OPTIGA™ Trust M V1 does not support this command.

Table 54 GenSymKey

Field	Offset [direction]	Description
Cmd	0 [in]	0x39 | 0xB9[77] command code
Param	1 [in]	0xXX algorithm identifier (refer to Table 57) of the key to be generated
InLen	2 [in]	0xXXXX length of InData
InData	4 [in]	Generate secret key input InData [InLen] • Alternative one: { - 0x02, 0x0001, key usage (refer to Table 58) - 0x07, 0x0000 (export secret key in plain) }
Sta	0 [out]	0x00 | 0xFF response status code
UnDef	1 [out]	0x00-0xFF undefined value
OutLen	2 [out]	0xXXXX length of OutData
OutData	4 [out]	• {Key 0x01, len, SecKey[78]}

73 PrivKey is encoded as DER "OCTET STRING"
74 PubKey is encoded as DER "BIT STRING"
75 PrivKey is encoded as DER "OCTET STRING"
76 PubKey is encoded as two DER INTEGER (modulus || public exponent) contained in a DER "BIT STRING"
77 In case of 0xB8 the last error code gets flushed
78 SecKey is encoded as octet string. The length is determined by the regarded algorithm (see Param)

4.4.1.16 CalcSSec

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

Table 55 CalcSSec

Field	Offset[direction]	Description
Cmd	0 [in]	0x33 | 0xB3[79] command code
Param	1 [in]	0xXX key agreement primitive (refer to Table 60)
InLen	2 [in]	0xXXXX length of InData
InData	4 [in]	• 0x01, 0x0002, OID of private key *Note: The key usage of the addressed key must be set to KeyAgree (refer to Table 58)* • 0x05, 0x0001, refer to Table 57, 0x06, length, public key[80] (alternative one) • 0x07, 0x0000[81] • 0x08, 0x0002, OID of shared secret[82]
Sta	0 [out]	0x00 | 0xFF response status code
UnDef	1 [out]	0x00-0xFF undefined value
OutLen	2 [out]	0xXXXX length of OutData
OutData	4 [out]	0x00-0xFF shared secret[83] *Note: Shared secret is only returned in case it is requested by InData (0x07, 0x0000)*

4.4.1.17 DeriveKey

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

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

Table 56 DeriveKey

Field	Offset [direction]	Description
Cmd	0 [in]	0x34 | 0xB4[84] command code
Param	1 [in]	0xXX key derivation method (refer to Table 61)

(table continues...)

79 In case of 0xB3 the last error code gets flushed
80 Public key is encoded as DER "BIT STRING"
81 Indicates exporting the shared secret via the external interface
82 The shared secret becomes part of the session context and can be addressed until the session context gets flushed
83 The shared secret is encoded as OCTET STRING

Table 56 (continued) DeriveKey

Field	Offset [direction]	Description
InLen	2 [in]	0xXXXX length of InData
InData	4 [in]	Key derivation parameter InData [InLen] • 0x01, 0x0002, OID of shared secret(data object of type PRESSEC) to derive the new secret from[85)] • 0x02, len, Secret derivation data/salt[86)] • 0x03, 0x0002, length of the key to be derived[87)] • (optional)0x04, len, info[88)] (alternative one) • 0x07, 0x0000[89)] • 0x08, 0x0002, OID of derived key[90)]
Sta	0 [out]	0x00 | 0xFF response status code
UnDef	1 [out]	0x00-0xFF undefined value
OutLen	2 [out]	0xXXXX length of OutData
OutData	4 [out]	0x00-0xFF derived data *Note: Derived data is only returned in case it is requested by InData (0x07, 0x0000)*

4.4.2 Command parameter identifier

Table 57 lists the algorithm identifier supported by the OPTIGA™.

Note: The OPTIGA™ Trust M V1 does not support ECC Brainpool, ECC NIST P 521 curves, and symmetric (AES).

Table 57 Algorithm identifier

Value	Description
0x03	Elliptic curve key on NIST P256 curve
0x04	Elliptic curve key on NIST P384 curve
0x05	Elliptic curve key on NIST P521 curve
0x13	Elliptic curve key on Brainpool P256 r1 curve
0x15	Elliptic curve key on Brainpool P384 r1 curve
0x16	Elliptic curve key on Brainpool P512 r1 curve

(table continues...)

84 In case of 0xB4 the last error code gets flushed
85 The source of the shared secret could be a session context or data object. The used size of the data object must be maximum 64 bytes
86 In case of HKDF this tag is optional and so the length could be 0 to 1024 byte or in all other cases 8 to 1024 byte
87 Minimum length = 16-byte; maximum length = 66 bytes (in case of OPTIGA™ Trust M V1, = 48 bytes) in case of session reference; maximum length = 256-byte in case of returned secret
88 Applicable only for HKDF, maximum length = 256 bytes
89 Indicates exporting the derived key via the external interface
90 The key becomes part of the session context and can be addressed as shared secret until the session context gets flushed

Table 57 (continued) Algorithm identifier

Value	Description
0x41	RSA key 1024-bit exponential format
0x42	RSA key 2048-bit exponential format
0x81	AES key with 128-bit
0x82	AES key with 192-bit
0x83	AES key with 256-bit
0xE2	SHA 256

Table 58 lists the key usage identifier supported by the OPTIGA™.

Table 58 Key usage identifier

Value	Description
0x01	Auth (authentication)
0x02	Enc (encryption, decryption, key transport)
0x10	Sign (signature calculation/verification)
0x20	KeyAgree (key agreement)

Table 59 lists the supported asymmetric cipher suites used in public key schemes and their coding.

Table 59 Asymmetric cipher suite identifier

Value	Description
0x11	Cipher suite PKCS#1v2.2 RSAES-PKCS1-v1.5 (RSA key pair with PKCS1 v1.5 padding) according to [3]

Table 60 lists the key agreement schemes supported by the OPTIGA™.

Table 60 Key agreement schemes

Value	Description
0x01	Elliptic curve Diffie-Hellman shared secret agreement according to [17]

Table 61 lists the key derivation method supported by the OPTIGA™.

Note: The OPTIGA™ Trust M V1 does not support HKDF (SHA256/384/512) and PRF (SHA384/512).

Table 61 Key derivation method

Value	Description
0x01	TLS PRF SHA256 according to [10]
0x02	TLS PRF SHA384 according to [10]
0x03	TLS PRF SHA512 according to [10]
0x08	HKDF-SHA256 according to [6]
0x09	HKDF-SHA384 according to [6]

(table continues...)

Table 61 (continued) Key derivation method

Value	Description
0x0A	HKDF-SHA512 according to [6]

Table 62 lists the signature schemes supported by the OPTIGA™.

Table 62 Signature schemes

Value	Description
0x01	PKCS#1v2.2 RSASSA-PKCS1-v1.5_SHA256 according to [3] with respect to hash
0x02	PKCS#1v2.2 RSASSA-PKCS1-v1.5_SHA384 according to [3] with respect to hash
0x03	PKCS#1v2.2 RSASSA-PKCS1-v1.5_SHA512 according to [3] with respect to hash
0x11	ECDSA with respect to hash

Table 63 lists the supported symmetric cipher modes of operation used in secret key schemes and their coding.

Note: The OPTIGA™ Trust M V1 does not support symmetric operations specified in the below table.

Table 63 Symmetric modes of operation

Value	Description
0x08	ECB for block ciphers as specified by [4] *Note: The last block of the provided data for encryption must be block aligned and padded.*
0x09	CBC for block ciphers as specified by [4] *Note: The last block of the provided data for encryption must be block aligned and padded.*
0x0A	CBC_MAC for block ciphers as specified by [8].MAC Algorithm 1 *Note: The last block of the provided data for encryption must be block aligned and padded.*
0x0B	CMAC for block ciphers as specified by [9]
0x20	HMAC-SHA256 for SHA 256 as specified by [7]
0x21	HMAC-SHA384 for SHA 384 as specified by [7]
0x22	HMAC-SHA512 for SHA 512 as specified by [7]

4.4.3 Command performance

The performance metrics for various schemes are listed in Table 64.

If not particularly mentioned the performance is measured at OPTIGA™ I/O interface including data transmission with:

- I2C FM mode (400 KHz)
- Without power limitation

- At 25°C
- VCC = 3.3 V

The performance of the commands, which use the secrets (for example: Private keys, shared secrets, etc.) at OPTIGA™ would get influenced by Security monitor behavior.

The values specified in Table 64 are without shielded connection.

Table 64 Command performance metrics

Operation	Command	Scheme	Execution time[91]	Additional info
Read data	GetDataObject	-	~30 ms	Data size = 256 bytes
Write data	SetDataObject	-	~55 ms	Data size = 256 bytes
Calculate signature	CalcSign	ECDSA	~65 ms	• ECC NIST P 256 • No data hashing
Calculate signature	CalcSign	PKCS#1v2.2 RSASSA-PKCS1-v1.5_SHA256	~310 ms	• RSA 2048 exponential • No data hashing
Verify signature	VerifySign	ECDSA	~85 ms	• ECC NIST P 256 • No data hashing • Public key from the external interface
Verify signature	VerifySign	PKCS#1v2.2 RSASSA-PKCS1-v1.5_SHA256	~40 ms	• RSA 2048 exponential • No data hashing • Public key from the external interface
Diffie-Hellman key agreement	CalcSSec	ECDH SP-800 56A	~60 ms	• Based on ephemeral key pair • ECC NIST P 256
Key pair generation	GenKeyPair	ECC	~55 ms	• ECC NIST P 256 • Ephemeral key
Key pair generation	GenKeyPair	RSA	Minimum 2900 ms	RSA 2048
Key derivation	DeriveKey	TLS v1.2 PRF SHA256	~50 ms	• To derive a key of 40 bytes • Shared secret (32 bytes) from session context and the input key derivation data size is 48 bytes
Key derivation	DeriveKey	HKDF SHA256	~130 ms	• To derive a key of 40 bytes • Shared secret (48 bytes) from a data object and the key derivation data is 48 bytes
Hash calculation	CalcHash[92]	SHA 256	~15 KB/s	In blocks of 0x500 bytes

(table continues...)

91 Execution of the entire sequence, except the external world timings, with I2C at 400 KHz and current limitation maximum value

92 In case of OPTIGA™ Trust M V1, The CalcHash performance is ~12 KB

Table 64 (continued) Command performance metrics

Operation	Command	Scheme	Execution time[91]	Additional info
RSA encryption	EncryptAsym	PKCS#1v2.2 RSAES-PKCS1-v1.5	~40 ms	RSA 2048 public key from the external interface
RSA decryption	DecryptAsym	PKCS#1v2.2 RSAES-PKCS1-v1.5	~315 ms	RSA 2048 exponential
Symmetric encryption	EncryptSym	AES 128 ECB	~28 ms	• 128 bytes of data • No chaining
Symmetric decryption	DecryptSym	AES 128 ECB	~35 ms	• 128 bytes of data • No chaining
HMAC generation	EncryptSym	HMAC-SHA256	~90 ms	• Using a pre-shared secret (64 bytes) from a data object • 128 bytes of data from the host
CMAC generation	EncryptSym	AES 128 CMAC	~28 ms	• 128 bytes of data • No chaining

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

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.

[91] Execution of the entire sequence, except the external world timings, with I2C at 400 KHz and current limitation maximum value

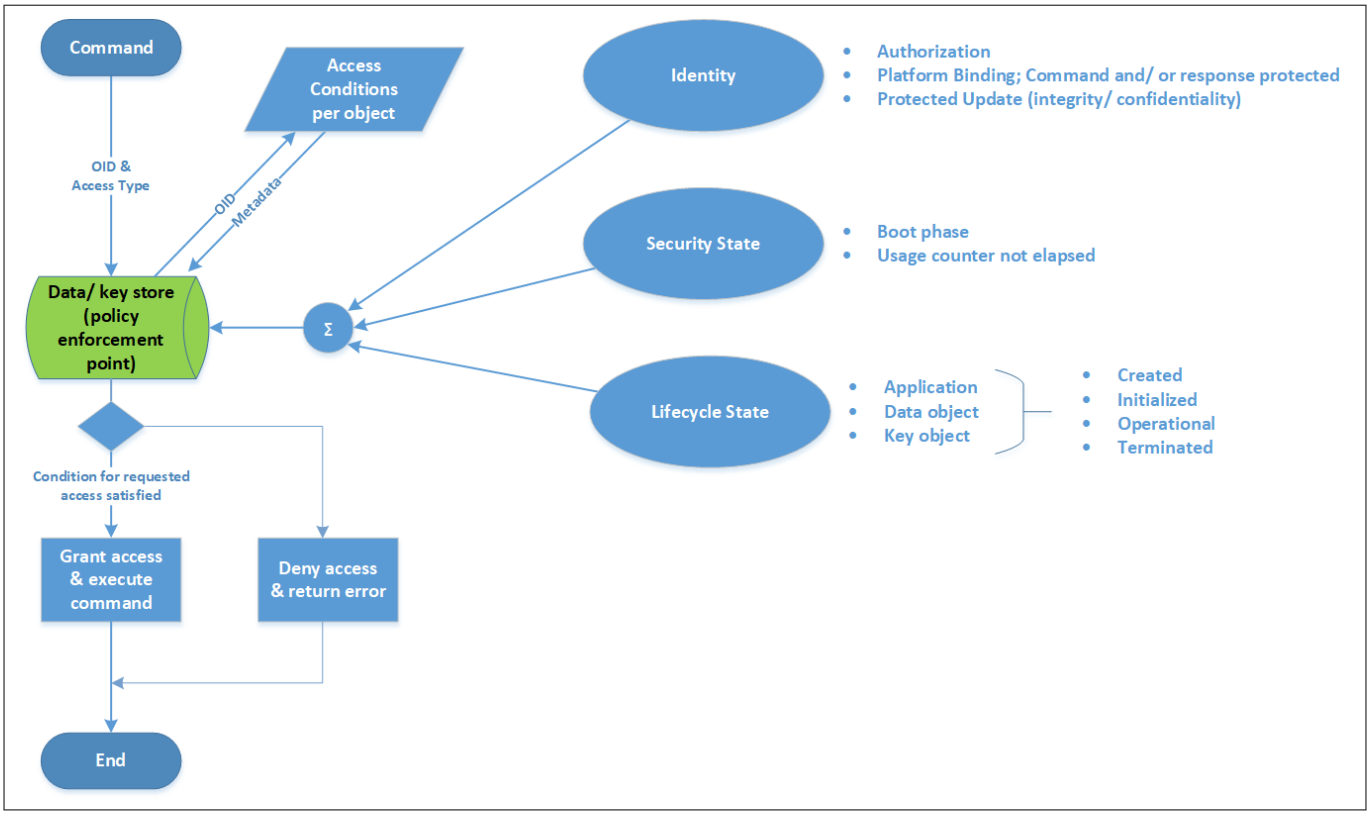

Figure 26 Security policy architecture

Note: The OPTIGA™ Trust M V1 does not support security policies like boot phase, authorization, and confidentiality (for protected update).

4.5.2 Policy attributes

The policy attributes are grouped in identity, security state, and life cycle state-based attributes:

- Identity (for example Identity of the host: Platform binding, namely OPTIGA™ shielded connection technology, is used for command/response)
- Security State (for example: Usage counter of a data or key object is not elapsed)
- Life cycle state (Lcs); for example 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). Figure 27 shows the flow, which leads to granting or denying the respective access.

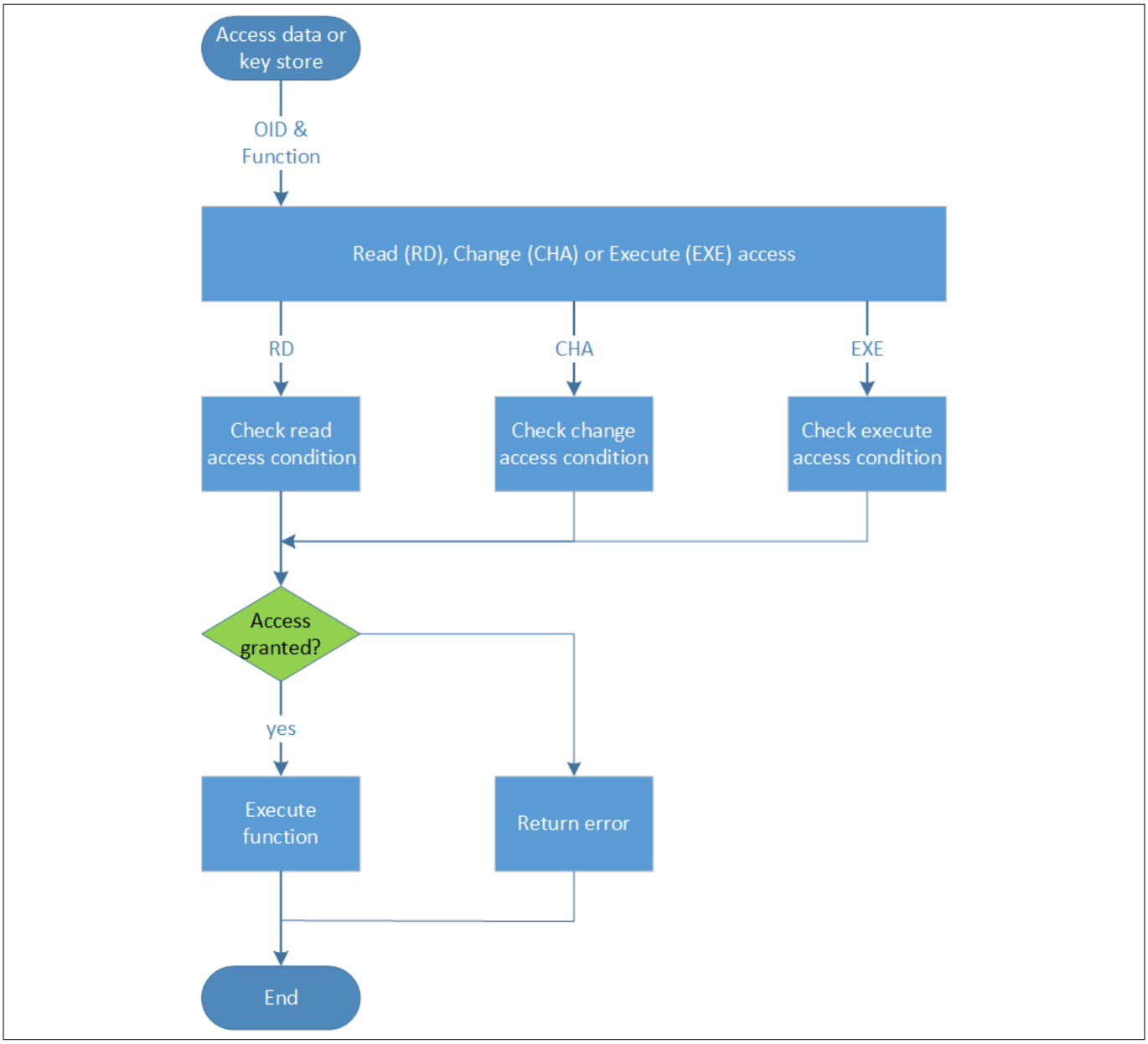

Figure 27 Policy enforcement flow

The access conditions and the policy attribute details are provided in Access conditions.

The Security guidance provides recommendations for security policy enforcement.

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 software and takes actions accordingly.

4.6.1 Security events

Table 65 defines permitted security events considered by the OPTIGA™ implementation.

Table 65 Security events

Name	Description
Decryption failure	The decryption failure event occurs in case decryption and/or integrity check of provided data lead to an integrity failure
Key derivation	The key derivation event occurs in case an operation, which executes a key derivation gets applied on a persistent data object which contains a pre-shared secret
Private key use	The private key use event occurs in case the internal services are going to use an OPTIGA™ hosted private key, except temporary keys from session context, are used for a key agreement like ECDH
Secret key use	The secret key use event occurs in case the internal services are going to use an OPTIGA™ hosted secret (symmetric) key (once per respective command), except temporary keys from session context are used
Suspect system behavior	The suspect system behavior event occurs in case the embedded software detects inconsistencies with the expected behavior of the system. Those inconsistencies might be redundant information, which does not fit to their counterpart

4.6.2 Security monitor policy

The security monitor policy provides all details of the policy selected for the OPTIGA™ project.

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 occurrences over time and in case those violate the permitted usage profile of the system it takes actions to throttle down the performance and so the possible frequency of attacks.

The permitted usage profile is defined as follows:

1. One protected operation (refer to Security events) events per t_{max} period
2. A suspect system behavior event is never permitted and will cause setting the SEC to its maximum
3. t_{max} is configurable, and the default value is 5 seconds (± 5%)

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

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

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

- After power-up or restart the SEC_{CREDIT} is cleared
- In case the t_{max} elapses without a security event and the SEC is > 0, the SEC gets decreased by one
- In case t_{max} elapses without a security event and the SEC is =0, the SEC_{CREDIT} gets increased by one to a maximum limit configured
- In case a security event occurs and the SEC_{CREDIT} is > 0, the SEC_{CREDIT} gets decreased by one
- In case the SEC_{CREDIT} is = 0 and a security event occurs, the SEC increased

4.6.3 Security monitor configurations

Note: The OPTIGA™ Trust M V1 does not support these security monitor configurations.

The possible security monitor configurations (available in security monitor configurations data object) are as follows:

- **t_{max}**
  The default value of t_{max} is 5 seconds. Due to use case demands, the t_{max} can be set to a different value.
  If t_{max} is set to 0, the security monitor gets disabled. In general, a higher t_{max} value lowers the possible frequency of attacks.
  The maximum value of t_{max} that will be applied internally is 5 seconds even if t_{max} is configured to a higher value than 5 seconds.
  In case, the current SEC is > 0 and t_{max} is configured to 0, the SEC gets set to 0.
- **Maximum SEC_{CREDIT} (SEC_{CREDIT_MAX})**
  The maximum SEC_{CREDIT} that can be achieved is configurable. The default value is 5.
  If this value is set to 0, the SEC_{CREDIT} will be set to 0 and will not be incremented.
  For example: If t_{max} = 4000 milliseconds, if there are 720 sign operations distributed across an hour (3600 seconds) (on average, for every 5 seconds, there is one sign operation), Then there is a possibility of SEC_{CREDIT} gets incremented about 180 times.
- **Delayed SEC decrement synchronization count**
  The SEC is as well maintained in RAM (SEC_{CURR}) in addition to the NVM (security event counter (SEC)) (SEC_{NVM}) and the synchronization (writing to SEC data object in NVM) of decrement events (example: t_{max} elapsed or decryption failure) can be delayed by configuring this value (> 1). If there are multiple security events with in t_{max} due to use case demand, the number of NVM write operations can be avoided by configuring this count appropriately.
  The default and minimum value is 1. In case, this value is configured to 0, minimum 1 will be applied.
  This option reduces SEC NVM write operations reasonably across the life time, if there are too frequent security events.
  For example: If this configuration is set to 4, and there are 4 security events (example: Sign operations) immediate after reset which led to SEC increment, then the total number of SEC NVM write operations are 5 instead of 8 (in case of default (= 1) configuration).

The offset details of above-specified configurations in the security monitor configurations data object are specified in Table 77.

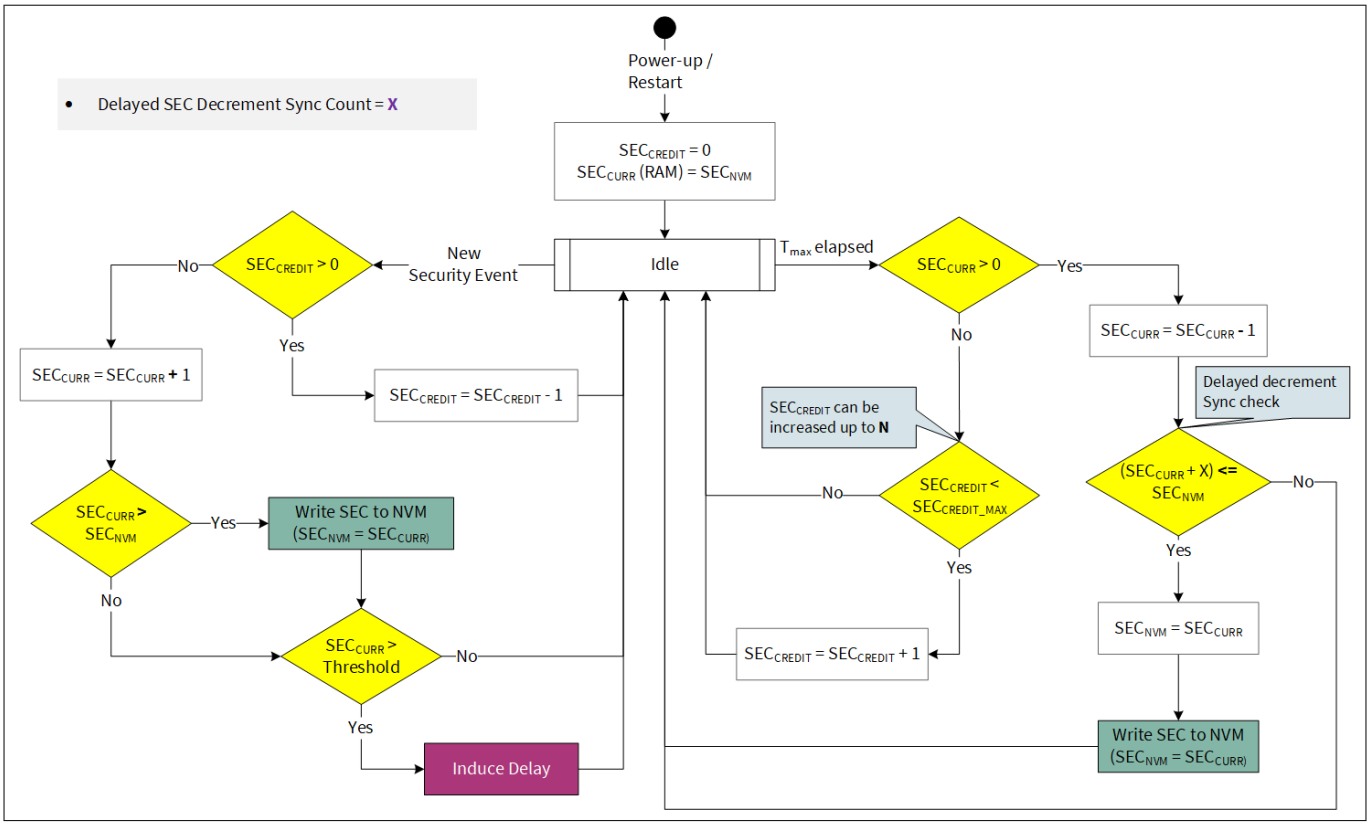

Figure 28 Security monitor flow diagram

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

The MSB (invalid flag) of the version tag and EXE access conditions are ignored for security monitor configurations data object while using internally.

4.6.4 Security monitor characteristics

The security monitor characteristics provides the throttle-down characteristics for the protected operations implemented by the security monitor. The security monitor uses the SEC to count Security events to figure out not permitted usage profiles. Figure 30 shows 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. In other words, the delay starts as soon as SEC reaches the value of 128 and will be t_{max} in case the SEC reaches its maximum value of 255.

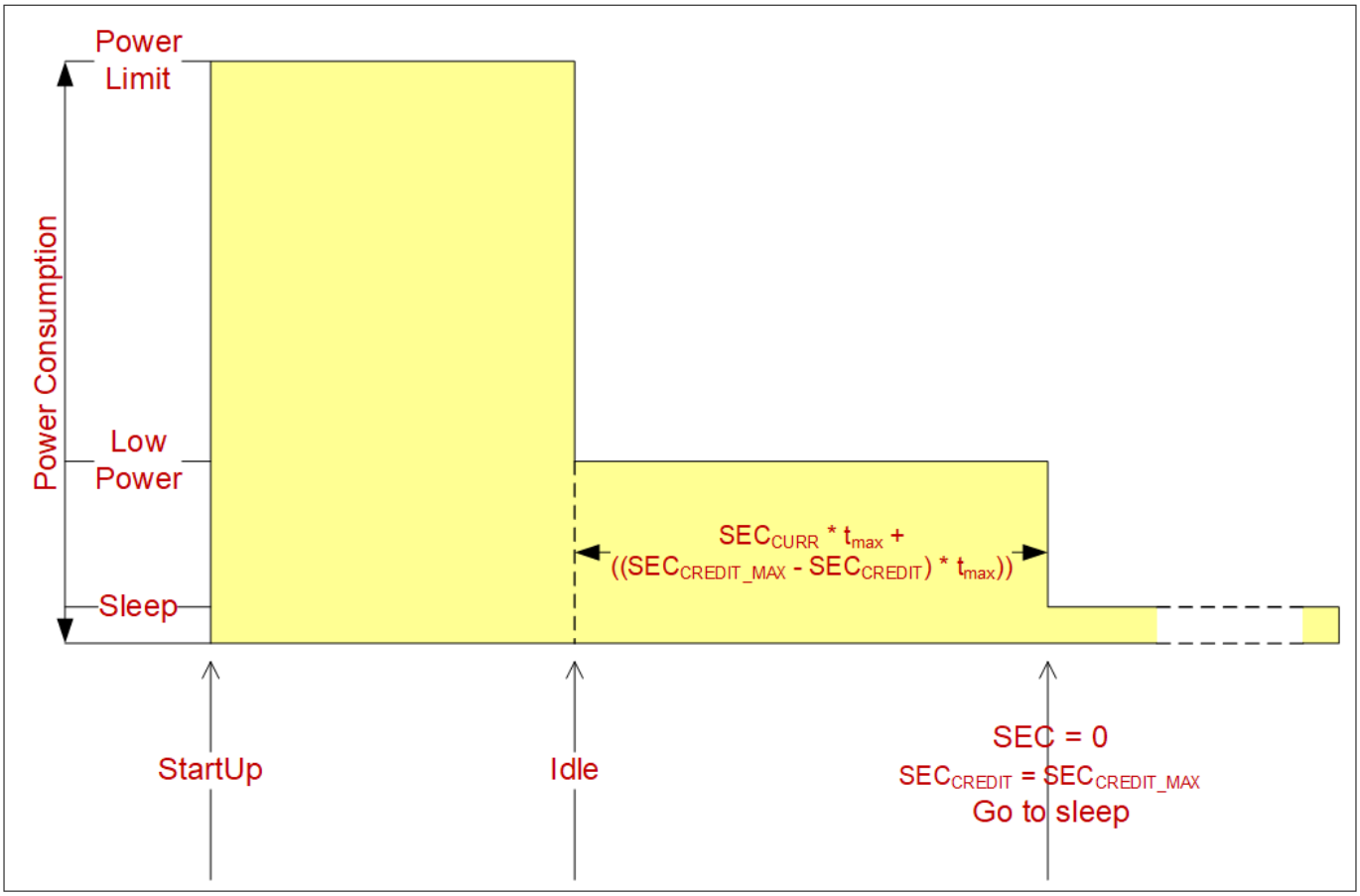

Figure 29 Power profile

Figure 29 shows the power profile of a regular start-up sequence, caused either by power-up, 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 73). As soon as the OPTIGA™ enters idle state (nothing to compute or communicate) the OPTIGA™ reduces its power consumption to the low power limit (refer to chapter “System Halt Power Consumption” in [11]). 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. 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 (refer to Figure 30) for functionalities that potentially triggering Security events. However, the method of switching VCC off and on is limited to 200000 times over a lifetime.

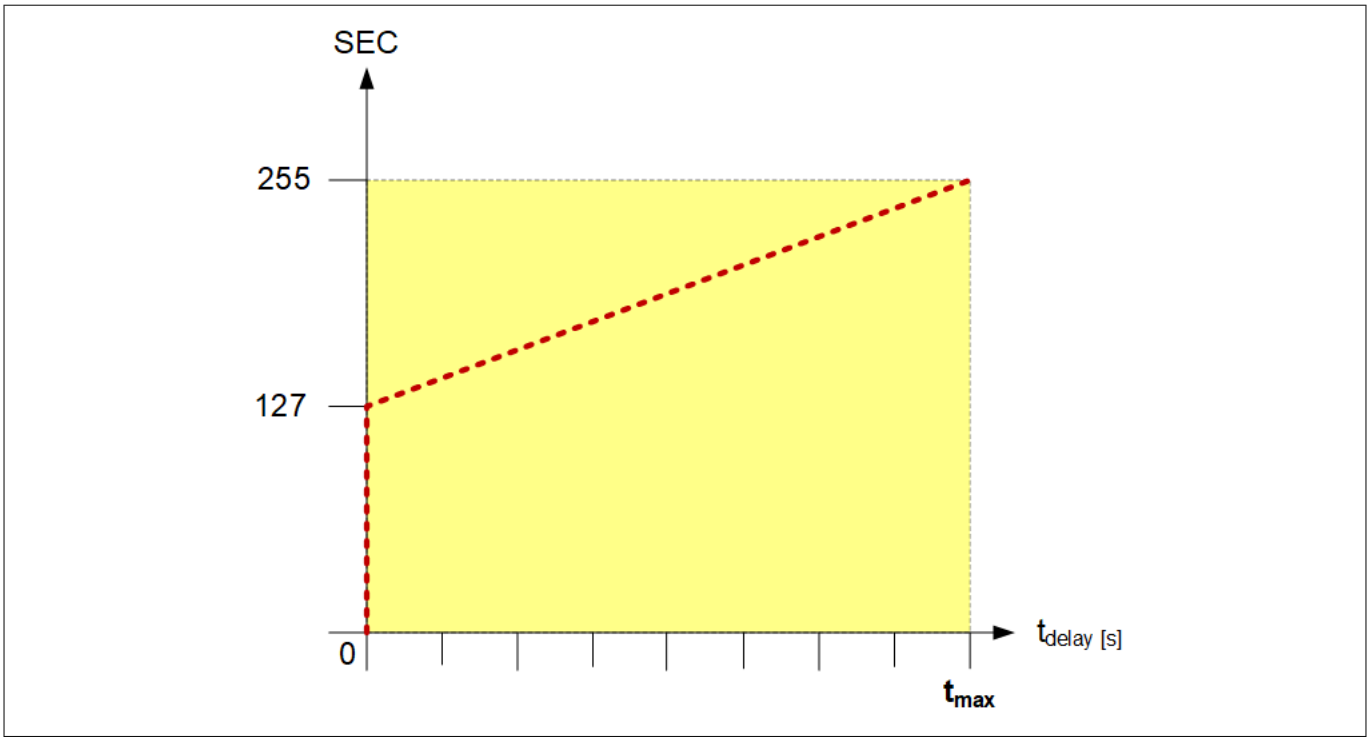

Figure 30 Throttling down profile

5 OPTIGA™ Trust M data structures

5.1 Overview data and key store

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

- Data-retention-after-testing defines how long NVM programmed during production (at wafer tester) is preserved, in case the NVM experiences no additional programming attempts, this time is the same as the device lifetime defined in [11]
- Data-retention-after-cycling defines how long NVM data, programmed throughout the lifetime, are preserved. The number of experienced programming cycles is important for how long the data are preserved. For maximum of 100 times, the same as data-retention-after-testing applies. After for example 20000 times the NVM data retention declines linearly down to 2 years and even more down to ½ year after about 40000 NVM programming cycles. ½ year data-retention-after-cycling is the worst case. In 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 20000 times, the cycles shall be homogeneously distributed across the lifetime
- Hardening is a process that is embedded in any erase cycle (each NVM programming cycle causes one or multiple erase cycles) and refreshes a randomly selected NVM page. After a certain number of erase cycles, which is dependent on the physical NVM size (for OPTIGA™ Trust M about 5000), all NVM is “refreshed”. Hardening is performed without erasing or physically moving the hardened data, that is hardening is neither susceptible to tearing nor does it count as a programming cycle
- Maximum endurance defines how often NVM data could be programmed until it wears out
- The 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 mentioning, 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
- Protected update of data object causes a minimum of 3 tearing-safe-cycles and it will go up to 5 depending on the size of a data object
- Protected update of the key object causes tearing-safe-cycles (for ECC - 4 cycles, for RSA - 6 cycles, for AES - 4 cycles)
- Update of a key object causes one to six tearing-safe-programming-cycles (average for ECC is one and for RSA is five)
- Usage (EXE or CHA or RD) of a data or key object which is linked to a usage counter causes one (additional) tearing-safe-programming-cycle
- The security event counter (SEC) gets increased and subsequently decreased (refer to Table 65) and causes two tearing-safe-programming-cycle
- One hibernate cycle causes five tearing-safe-programming-cycles

Figure 31 gives 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 met and for higher cycled data objects the homogeneous distributed of applied programming cycles across the lifetime are respected, the data integrity over a lifetime could be considered safe.

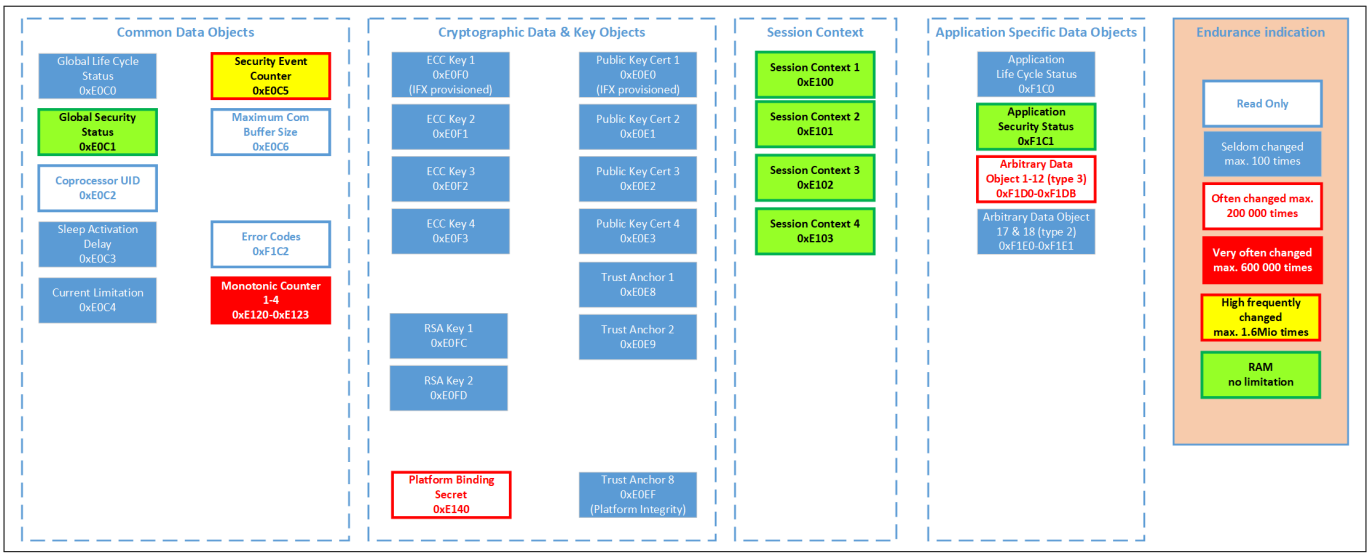

Figure 31 OPTIGA™ Trust M (V1): Overview data and key store

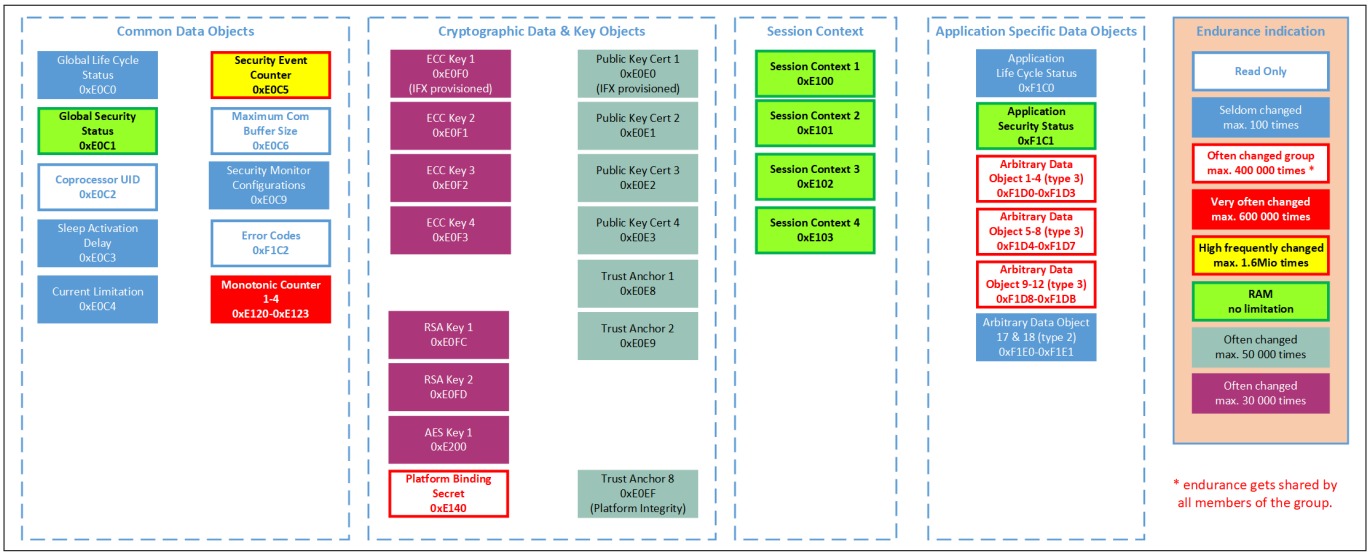

Figure 32 OPTIGA™ Trust M (V3): Overview data and key store

Note: In Figure 32, the endurance of arbitrary data objects (type 3) is shared between other data objects in the same group. In case of SetObjectProtected (for protected update), the data object memory gets updated 3 times for one complete update cycle. So, the endurance specified in Figure 32 is to be considered accordingly across the lifetime.

Examples:

- Each monotonic counter can be updated up to a maximum of 600000 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

At each level of the data structure, access conditions (ACs) 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 (example: GetDataObject)
CHA	Changing (writing or flushing) a data or key object by an external command (example: SetDataObject)
EXE	Utilize a data or key object implicitly by executing a command (example: CalcSign, GenKeyPair, etc..,)
MUPD	Updating metadata by an external command (example: SetObjectProtected) *Note: The OPTIGA™ Trust M V1 does not support MUPD access type.*

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 life Lifecycle status meets the condition given by X
LcsA(X)	The action is only possible in case the application-specific Lifecycle status meets the condition given by X
LcsO(X)	The action is only possible in case the data object-specific Lifecycle status meets the condition given by X
Conf(X)	The action is only possible in case the data involved (to be read/write) are confidentiality protected with key given by X
Int(X)	The action is only possible in case the data involved (to be read/write) are integrity protected with key given by X
Auto(X)	The action is only possible in case the authorization of the external world was successfully performed, providing the authorization value which matches the Table 67. Authorization reference is given by X *Note: Up to 4 independent Auto(X) states at a time are supported.*
SecStaG(X)	The action is only possible in case the global security status ANDed with X is equal to X
SecStaA(X)	The action is only possible in case the application-specific security status ANDed with X is equal to X

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

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

Table Access Condition Identifier and Operators defines the Access Condition Identifier and Operands to be used, to define ACs associated with data objects. Access Condition Identifier must be used with the commands trying to achieve associated ACs.

There are simple and complex Access Condition expressions defined (examples of 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)

$AT = sAC_1 \ldots \&\& sAC_n$ (n = 1…7)
...|| operators combine multiple ATs to a cAC
$cAC = AT_1 \ldots || AT_m$ (m = 1…3; $((n_1 + \ldots + n_m) * m) > 1$)

Note:
- *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 does not 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 with OPTIGA™ data and key objects are defined in Table 66.

Note: Note:The OPTIGA™ Trust M V1 does not support SecStaG(X), SecStaG(X), and Auto(X) identifiers.

Table 66 Access condition identifier and operators

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

(table continues...)

Table 66 (continued) Access condition identifier and operators

AC ID	Operator	Value	Description
Conf	-	0x20	3-byte; value, key reference (OID) (example: Conf first session key → 0x20, 0xE1, 0x40) • Read, Conf, binding secret (example: 0xD1, 0x03, 0x20, 0xE1, 0x40) In case of reading a data object (example: Using GetDataObject), the shielded connection must be established already using the specified binding secret (example: 0xE140) and the response is requested with protection (encrypted) • Change, Conf, binding secret (example: 0xD0, 0x03, 0x20, 0xE1, 0x40) In case of writing a data object (example: 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 (example: 0xD3, 0x03, 0x20, 0xE1, 0x40) In case of using a data object with an internal operation (example: Using DeriveKey from a pre-shared secret), the shielded connection must be established already using the specified binding secret (0xE140) and the command is sent protection (encrypted) • Change, Conf, protected update secret → (example: 0xD0, 0x03, 0x20, 0xF1, 0xD0) In case of writing a data object (using SetObjectProtected), the manifest must specify the same protected update secret (example: 0xF1, 0xD0) which is specified in the object metadata. This enforces to use of the defined protected update secret to decrypt the object data in fragments *Note: Conf (protected update secret) must be used in association (operator AND) with integrity (trust anchor), to enforce the right protected update secret to be used to decrypt the object data as part of SetObjectProtected. If Conf (protected update secret) is not specified in the metadata access conditions, SetObjectProtected uses the protected update secret specified in the manifest, to decrypt the object data as part of fragments. The usage of this identifier is to enforce the right secret used (integrity trust anchor, operator AND, confidentiality protected update secret OID). The protected update secret must not same as the target data object to be updated.*

(table continues...)

Table 66 (continued) Access condition identifier and operators

AC ID	Operator	Value	Description
Int	-	0x21	3-byte; value, key reference (example: Int first session key → 0x21, 0xF1, 0xF0) • Read, Int, binding secret (example: 0xD1, 0x03, 0x21, 0xE1, 0x40) In case of reading a data object (example: 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 (example: 0xD0, 0x03, 0x21, 0xE1, 0x40) In case of writing a data object (example: 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 (example: 0xD3, 0x03, 0x21, 0xE1, 0x40) In case of using a data object with an internal operation (example: Using DeriveKey from a pre-shared secret), the shielded connection must be established already using the specified pre-shared secret (0xE140) and the command is sent with protection (MAC) • Change, Int, trust anchor (example: 0xD0, 0x03, 0x21, 0xE0, 0xEF) In case of writing a data object (example: Using SetObjectProtected), the signature associated with the meta data in the manifest must be verified with the addressed trust anchor (example: 0xE0EF) in the access conditions. In case of SetObjectProtected command, the change access conditions of the target OID must have an integrity access condition identifier with the respective trust anchor
Auto	-	0x23	3-byte; value, reference (authorization reference OID) (example: Auto → 0x23, 0xF1, 0xD0)
Luc	-	0x40	3-byte; value, counter reference (example: Linked counter 1 → 0x40, 0xE1, 0x20) For example: The arbitrary data object holds a pre-shared secret and this secret is allowed to be used for key derivation (DeriveKey) operations for a limited number of times. To enable this, select a counter object (updated with the maximum allowed limit) and assign the counter data object in the EXE access condition of the arbitrary data object as shown below. (example: EXE, Luc, counter object → 0xD3, 0x03, 0x40, 0xE1, 0x20) The counter data objects get updated (counter value gets incremented by 1 up to the maximum limit) automatically when the DeriveKey command is performed.
LcsG	-	0x70	3-byte; value, qualifier, reference (example: LcsG < op → 0x70, 0xFC, 0x07)

(table continues...)

Table 66 (continued) Access condition identifier and operators

AC ID	Operator	Value	Description
SecStaA	-	0x90	2-byte; value (example: Enable access if boot phase flag in security status application is set → 0x90, 0x20) *Note: The SetDataObject with Param = erase&write clears all bits and with Param = write clears all corresponding bits not set to 1 in data to be written.*
LcsA	-	0xE0	3-byte; value, qualifier, reference (example: LcsA > in → 0xE0, 0xFB, 0x03)
LcsO	-	0xE1	3-byte; value, qualifier, reference (example: 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 67 lists the various types of data objects supported by OPTIGA™.

Table 67 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
UPCTR	0x01	The up-counter data type implements a counter with a current value which could be increased only and a threshold terminating the counter
TA	0x11	The trust anchor data type contains a single X.509 certificate which could be used in various commands requiring a root of trust
DEVCERT	0x12	The device identity data type contains a single X.509 certificate or a chain of certificates (TLS, USB Type C, etc..,) which was issued to vouch for the cryptographic identity of the end-device.
PRESSEC	0x21	The pre-shared secret contains a binary data string which makes up a pre-shared secret for various purposes (FW-decryption, etc..,)
PTFBIND	0x22	The platform binding contains a binary data string which makes up a pre-shared secret for platform binding (example: Used for OPTIGA™ shielded connection)
UPDATSEC	0x23	The protected update secret contains a binary data string which makes up a pre-shared secret for confidentiality-protected update of data or key objects. The maximum length is limited to 64 bytes, even if the hosting data object has a higher maximum length
AUTOREF	0x31	The authorization reference contains a binary data string which makes up a reference value for verifying an external entity (admin, user, etc..,) authorization

Note: The OPTIGA™ Trust M V1 does not support UPDATSEC and AUTOREF types.

5.3 Life cycle state

The device, 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 (example: 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 74.

5.4 Common and application-specific objects and ACs

Table 68 lists all common data structures defined for the OPTIGA™ with its TAGs and ACs.

Table 68 Common data objects with TAGs and ACs

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	0x20	NEV	ALW[93)]	ALW	Default LcsO = op
0xE0C2	Coprocessor UID OPTIGA™ Trust Family	-	NEV	NEV	ALW	Default LcsO = op
0xE0C3	Sleep mode activation delay (refer to Table 73)	0x14	NEV	ALW	ALW	Default LcsO = op
0xE0C4	Current limitation (refer to Table 73)	0x06	NEV	ALW	ALW	Default LcsO = op
0xE0C5	Security event counter (SEC) (refer to Table 73)	-	NEV	NEV	ALW	Default LcsO = op
0xE0C6	Maximum com buffer size (refer to Table 73)	0x0615	NEV	NEV	ALW	Default LcsO = op

(table continues...)

93 It is only possible to reset an achieved security status

Table 68 (continued) Common data objects with TAGs and ACs

Tag	Structure definition	Default Value	EXE	Change	Read	Note
0xE0C9	Security monitor configurations *Note: The OPTIGA™ Trust M V1 does not support the 0xE0C9 (Security Monitor Configurations) data object.*	50 00 05 01 00 00 00 00	NEV	LcsO < 7	ALW	Default LcsO = op; This data object holds the security monitor configurations (example: The maximum SEC_{CREDIT}, t_{max}, etc.,.). The Data Object Types are not allowed to be configured with this data object. For more details, refer to Security Monitor Configurations section.
0xE0E0	Public key certificate 1, issued by Infineon[94] (refer to Table 73)	-	ALW	NEV	ALW	Default LcsO = cr; default data type is "device identity"[95]
0xE0E1- 0xE0E3	Public key certificate 2-4 (refer to Table 73)	0x00	ALW	LcsO < op	ALW	Default LcsO = cr; default data type is "device identity[96]"
0xE0E8- 0xE0E9	Root CA public key certificate 1-2[97] (refer to Table 73)	0x00	ALW	LcsO < op	ALW	Default LcsO = cr; default data type is "trust anchor"
0xE0EF	Root CA public key certificate 8[98]. This trust anchor is assigned to platform integrity use cases (refer to Table 73).	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 linked characteristics the change (CHA) AC shall be never avoiding DoS attacks

(table continues...)

94 By default, Infineon issued device certificate is programmed. This slot can be reused to provision customer specific certificate

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

96 Due to its size the certificate is not written in an atomic way. In 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

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

98 Due to its size the public key or certificate is not written in an atomic way. In 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.

Table 68 (continued) Common data objects with TAGs and ACs

Tag	Structure definition	Default Value	EXE	Change	Read	Note
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 69 lists all common keys defined for the OPTIGA™ with its TAGs and ACs.

Table 69 Common key objects with TAGs and ACs

Tag	Structure definition	EXE	Change	Read	Note
0xE0F0	Device private ECC key 1; The GetDataObject and 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 and SetDataObject commands are not allowed for the data part of the key object even if the metadata states the access rights differently	ALW	LcsO < op	NEV	Default LcsO = cr
0xE0FC-0xE0FD	Device private RSA key 1-2; The GetDataObject and SetDataObject commands are not allowed for the data part of the key object even if the metadata states the access rights differently	ALW	LcsO < op	NEV	Default LcsO = cr

(table continues...)

Table 69 (continued) Common key objects with TAGs and ACs

Tag	Structure definition	EXE	Change	Read	Note
0xE100-0xE103	Session context 1-4 (OID to address one of the four-session contexts for example (D)TLS connection state). These OIDs are not applicable for the GetDataObject and SetDataObject commands. The session context holds either a private key or shared secret or is a target for toolbox commands like (GenKeyPair, CalcSSec, and DeriveKey)	ALW	NEV	NEV	Default LcsO = op
0xE200	Device symmetric key 1; The GetDataObject, and SetDataObject commands are not allowed for the data part of the key object even if the metadata state the access rights differently *Note: The OPTIGA™ Trust M V1 does not support the 0xE200 (device symmetric key object).*	ALW	NEV	NEV	Default LcsO = cr

Table 70 lists all data structures defined for the OPTIGA™ authentication application with its TAGs and ACs.

Table 70 Authentication application-specific data objects with TAGs and ACs

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	0x20	NEV	ALW[99)]	ALW	Default LcsO = op
0xF1C2	Error codes (refer to Table 39)	-	NEV	NEV[100)]	ALW	Default LcsO = op
0xF1D0-0xF1DB	Table 79 type 3	0x00	App-specific	App-specific	App-specific	Default LcsO = cr

(table continues...)

99 It is only possible to reset an achieved security status
100 cleared on read

Table 70 (continued) Authentication application-specific data objects with TAGs and ACs

Tag	Structure definition	Default value	EXE	Change	Read	Note
0xF1E0-0xF1E1	Table 79 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 (example: LCS, maximum length, etc..,) and access conditions (read, change, etc..,) to those data objects. The access conditions expressed in Table 66 describes under which condition metadata itself can 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 as NEV
- In case the LcsO is absent, the access conditions of the regarded data object is considered as operational (op) and could not be changed
- In case the used size (Tag 0xC5) is absent, the used size is same as maximum size
- The changed metadata gets effective as soon as the change gets consolidated at the data object

Table 71 lists all common data structures defined for the OPTIGA™ with its TAGs and ACs.

Table 71 Metadata associated with data and key objects

Tag	Structure definition	EXE	Change	Read	Note
0x20	Metadata constructed TLV-Object	NA	NA	ALW	-
0xC0	Life cycle state of the data/key object (LcsO)	NA	ALW	ALW	Refer to Table 74
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.	NA	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 whether the data object is valid (0) or invalid (1).
0xC4	Maximum size of the data object	NA	NEV	ALW	-
0xC5	Used size of the data object	NA	Auto	ALW	The used length gets updated automatically in case of change (CHA) access. In case it is not already present, it gets created

(table continues...)

Table 71 (continued) Metadata associated with data and key objects

Tag	Structure definition	EXE	Change	Read	Note
0xD0	Change access condition descriptor	NA	LcsO < op	ALW	-
0xD1	Read access condition descriptor	NA	LcsO < op	ALW	-
0xD3	Execute access condition descriptor	NA	LcsO < op	ALW	-
0xD8	Metadata update descriptor *Note: The OPTIGA™ Trust M V1 does not support this tag.*	NA	LcsO < op	ALW	This tag defines the condition under which the metadata update is permitted. (example: Metadata update descriptor, Int, trust anchor → 0xD8, 0x03, 0x21, 0xE0, 0xEF) In case of updating object metadata (example: Using SetObjectProtected), the signature associated with the metadata update must be verified with the addressed trust anchor (example: 0xE0EF) and the associated (factory) reset type (0xF0) gets applied before the metadata are newly set. In case no (factory) reset type is given by the current metadata, the update must fail.
0xE0	Algorithm associated with key container	NA	Auto[101)]	ALW	Refer to Table 57
0xE1	Key usage associated with key container	NA	LcsO < op	ALW	Refer to Table 58
0xE8	Data object type	NA	LcsO < op	ALW	Refer to Table 67 In case this tag is not present, the data object is represented by an unrestricted array of bytes (described by tags 0xC4 and 0xC5).
0xF0	(factory) reset type *Note: The OPTIGA™ Trust M V1 does not support this tag.*	NA	LcsO < op	ALW	It defines what happens with the object data in case of updating the metadata, refer to Table 72 for the values. Will be applied as part of SetObjectProtected implementation

Table 72 lists the metadata update identifier supported by the OPTIGA™. The identifier can be combined by ORing them (example: 0x11 = > LcsO = cr and flush object with zero). However, the bits in the upper nibble are not allowed to be combined. In case of protected metadata update the new LcsO gets provided, this overrules the setting given by Table 72.

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

Table 72 Metadata update identifier

Name	Description
0x0X	Setting the LcsO of either a key or data object. The definition of X is given by Life Cycle Status (lower nibble)
0x10	• Flushing of either a key or data object with zero and set the used length of data objects, if present, to 0x0000 • If none of the flushing options is set in metadata, then the SetObjectProtected manifest setting (if present) gets used • In case of a key object the algorithm associated gets cleared and sets again with a successful generation or writing (protected update) a new key
0x20	• Overwrite either a key or data object with random values and set the used length of data objects, if present, to 0x0000 • If none of the flushing options is set in metadata, then the SetObjectProtected manifest setting (if present) gets used • In case of a key object the algorithm associated gets cleared and sets again with a successful generation or writing (protected update) a new key

Note: The maximum size of metadata is 44 bytes.

Examples of commonly used access conditions:

- Arbitrary data record at 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 at shipping to customer

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

Note: There is no ordering rule for metadata tags

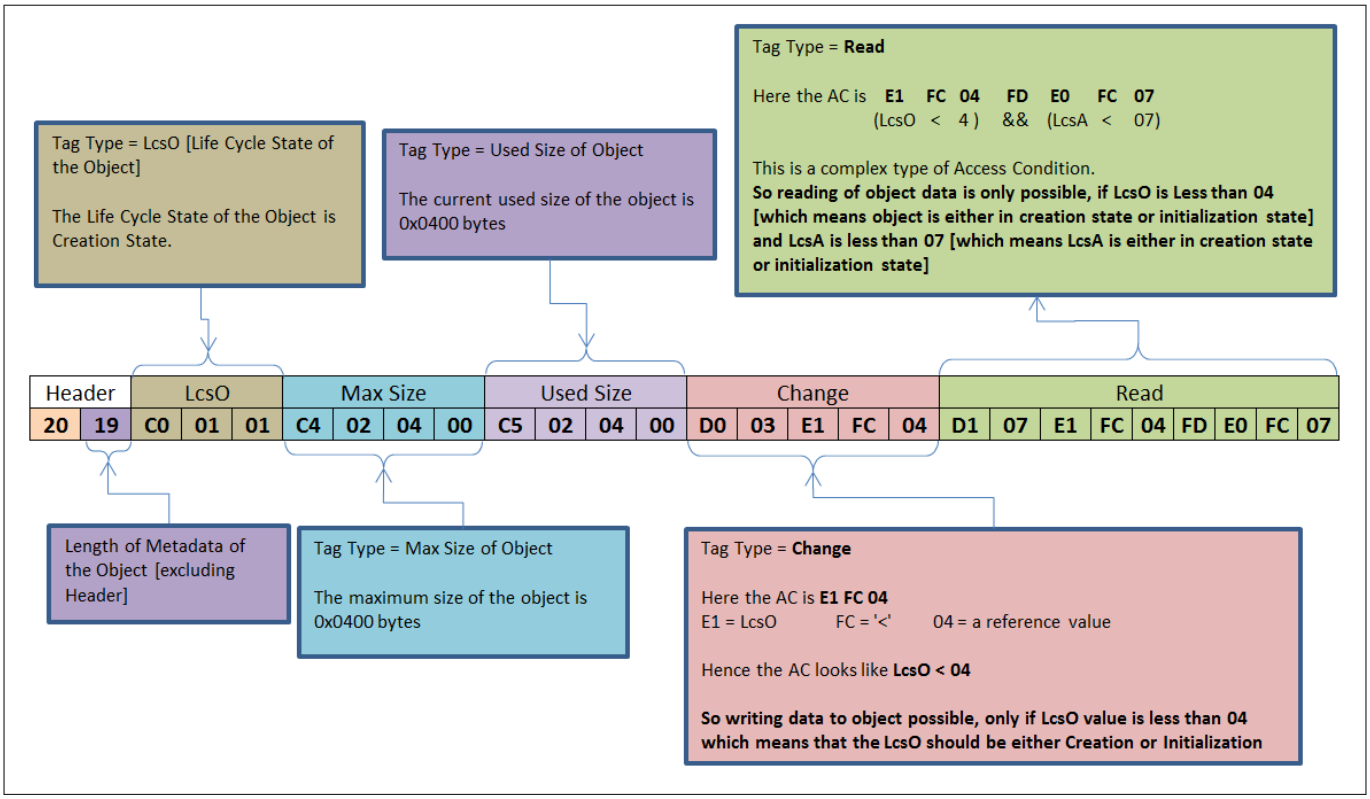

Figure 33 Metadata sample

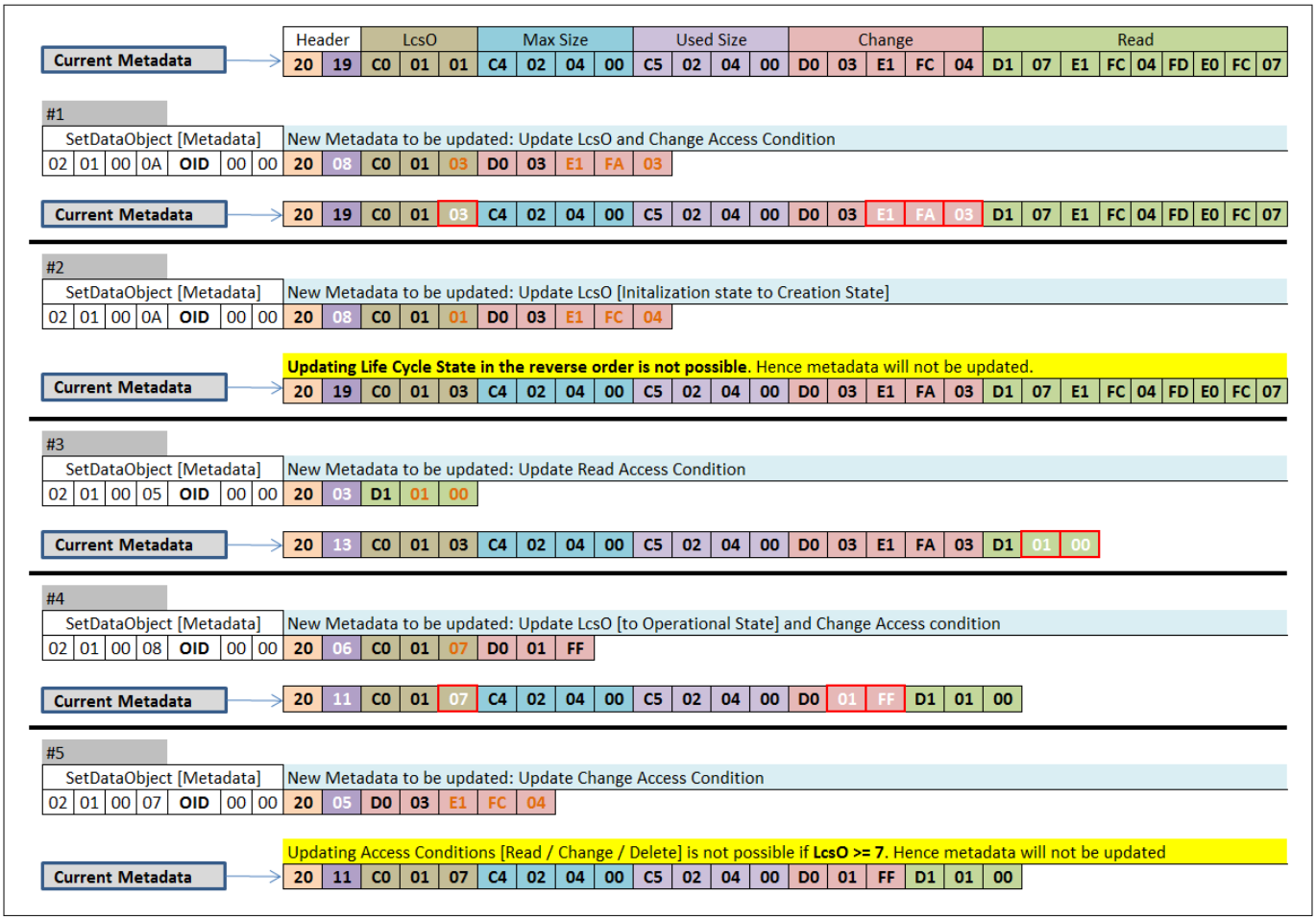

Figure 34 SetDataObject (metadata) examples

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

5.6 Common data structures

Table 73 lists all common data structures defined for the OPTIGA™.

Table 73 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 74)	Binary number 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 (example: Initialization (in) => operational (op) state, but not vice versa). The application-specific states, if used at all, are managed by the particular application

(table continues...)

Table 73 (continued) Common data structures

Data element	Coding	Length (bytes)	Description
Security state (refer to Table 75)	Binary number 8	1	The device and each application may have a security status associated. The device security status is further referenced to “global security status” and the application specific status by “application-specific security status”
Last error code (refer to Table 39)	Binary number 8	1	The last error code stores the most recent error code generated since the data object was last cleared. The availability of the last error code is indicated by the (GENERAL) ERROR (refer to Table 38), 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 (refer to Table 36). The possible error codes are listed in Table 39. If multiple commands happen to produce subsequent errors then only the highest-numbered error code is stored
Sleep mode activation delay in ms	Binary number 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 ms
Current limitation in mA	Binary number 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 it will be restored upon power up or reset. Its default content is 6 mA *Note: 15 mA will cause best-case performance. 9 mA will cause roughly 60% of the best case performance. Even the maximum communication speed might be degraded by current limitation (how the maximum possible communication speed gets indicated to the I2C master, refer to [2]).*
Buffer size in number of bytes	Binary number 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 Security monitor

(table continues...)

Table 73 (continued) Common data structures

Data element	Coding	Length (bytes)	Description
Public key certificate	X.509	1728 (maximum)	The public key certificate data object holds one or multiple of X.509 certificate (refer to [5]). The certificate was issued by IFX or a customer. An external entity (host, server) utilizes it as a device identity to authenticate the OPTIGA™ within the regarded PKI domain (IFX or customer). Tags to differentiate the public key certificate chain format: • Single X.509 certificate: Certificate DER coded The first byte of the DER encoded certificate is 0x30 and is used as Tag to differentiate from other public key certificate formats defined below • TLS identity: Tag = 0xC0 Length = Value length (2 bytes) Value = TLS certificate chain[102)] • USB Type-C Identity: Tag = 0xC2 Length = Value length (2 bytes) Value = USB Type-C certificate chain (refer to section 3.2 in [18])
Root CA public key certificate aka "trust anchor"	X.509 (maybe self-signed certificate)	1200 (maximum)	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 a "trust anchor" to authenticate external entities (example: Verify server signature)
Platform binding secret	Binary number 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	Binary number 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

102 Format of a "certificate structure message" used in TLS handshake

Table 74 lists all coding of the life cycle status defined for the OPTIGA™.

Table 74 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)[103]

Table 75 shows the security status defined for the device either global or application-specific. The default is set after reset for the global and after OpenApplication for the application-specific security status[104][105].

Table 75 Security status

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

Table 76 shows UID definition for the OPTIGA™.

Table 76 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

Table 77 shows the possible configurations with respect to security monitor.

103 This state is not applicable for the LcsA
104 Bit = 0 status not satisfied
105 Bit = 1 status satisfied

Table 77 Security monitor configurations

Offset	Name	Data type	Notes
0	t_{max} configuration	uint8_t	= Required t_{max} in ms/100 For example: If the required t_{max} is 4 seconds (4000 milliseconds), this will be configured as 40 Default value = 50
1	RFU[1]	uint8_t	Default is 0x00
2	SEC_{CREDIT} maximum value (SEC_{CREDIT_MAX})	uint8_t	The maximum value of SEC_{CREDIT} that can be achieved Default is 5
3	Delayed SEC decrement sync count	uint8_t	The SEC is as well maintained in RAM in addition to the NVM security event counter (SEC) and the synchronization (writing to a data object in NVM) of decrement event (example: T_{max} elapsed) can be delayed by configuring this value greater than 1 Default is 1
4	RFU[4]	uint8_t	Default is 00 00 00 00.

5.7 Application-specific data structures

Table 78 shows a unique identifier for the OPTIGA™ application which gets used with the OpenApplication command.

Table 78 Data structure unique application identifier

Data element	Value	Coding	Length (in bytes)
RID	0xD276000004[106)]	Hex number 5	5
PIX	'GenAuthAppl' 0x47656E417574684170706C	ASCII 11	11

Table 79 lists the supported types of arbitrary data objects.

Table 79 Data structure arbitrary data object

Data element	Value	Coding	Length (in bytes)
Arbitrary data object type 2	0x00 - 0xFF	Application-specific	1500
Arbitrary data object type 3	0x00 - 0xFF	Application-specific	140

106 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 35 GetDataObject [read data] example

Note: The values specified in Figure 35 are in hex format.

- SetDataObject

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

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

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

Figure 36 GetDataObject [read data] example

Note: The values specified in are in hex format.

6.2 Data encoding format examples

This section provides the examples of the encoding format of asymmetric key pairs and signature used across the Enabler APIs.

6.2.1 ECC private key

The examples for the format of ECC private key exported as part of generate key pair are given below.

- Example for ECC NIST P-256 private key [values are in hex]

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

- Example for ECC NIST P-384 private key [values are in hex]

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

6.2.2 ECC public key

The examples for the format of ECC public key (referred by generate key pair, verify signature, and ECDH operations) are given below.

- Example for ECC NIST P-256 public key [values are in hex]

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

- Example for ECC NIST P-384 public key [values are in hex]

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

6.2.3 ECDSA signature

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

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

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

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

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

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

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

Note:

- *The size of R and S components is based on the key size selected (example: 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*

6.2.4 RSA private key

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

For RSA 1024, the length of the private exponent is 128 bytes.

For RSA 2048, the length of the private exponent is 256 bytes.

- Example for RSA 1024 exponential - private key (private exponent) [values are in hex]

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

- Example for RSA 2048 exponential - private key (private exponent) [values are in hex]

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

6.2.5 RSA public key

The examples for the format of RSA public key (referred by generate key pair, verify signature, and asymmetric encryption operations) are given below.

- Example for RSA 1024 exponential public key (modulus and public exponent) [values are in hex]

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

Note:
- *The size of modulus component is based on the key size selected (example: In case of RSA 1024, size is 128 bytes and for RSA 2048, size is 256 bytes)*
- *If the first byte of modulus is greater than 0x7F (negative integer), then 0x00 gets prepended while coding as it is in integer format*

6.2.6 RSA signature

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

There is no additional encoding (format) considered for the RSA signature. The length of the signature is based on the key length selected.

For RSA 1024, the length of the signature is 128 bytes.

For RSA 2048, the length of the signature is 256 bytes.

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

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

6.3 Limitations

6.3.1 Memory constraints

- The maximum command InData length is 1553. Any attempt to exceed these limit will lead to Invalid length field error
- The 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

The followings are the basic parameter checks performed while parsing the certificate:

- X.509 DER coding and length checks
- Must be v3 only
- Serial number
  - The length must be 1 to 20 byes
- Public key
  - Only uncompressed format is supported
  - ECC curves supported
    - ECC NIST P 256/384/521
    - ECC Brainpool P256r1/P384r1/P512r1

    Note: The OPTIGA™ Trust M V1 does not support ECC Brainpool and ECC NIST P 521

  - RSA 1024/2048 exponential
  - The public key must be aligned to the key length. In case the public key size is less than the respective key size, then the respective component must be prepended with zeroes before generating the certificate
- The signature algorithm identifier must not contain the parameters (not even as NULL) in case of ECDSA. (section 2.2.3 in [RFC3279])`AlgorithmIdentifier ::= SEQUENCE {`
  `algorithm OBJECT IDENTIFIER,`
  `parameters ANY DEFINED BY algorithm OPTIONAL }`
- Issuer DN must not be empty

- Basic constraints
  - CA-Flag to indicate CA certificate (TRUE/asserted for CA)
  - Path length constraint
    - Exists only for CA certificate (if CA flag is TRUE)
- The signature algorithm identifier in tbscertificate and signature tags must be the same

[5] specifies the number of fields (specified as MUST/SHOULD etc..,). All of these will not be validated for the 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 on how to use and configure the customer solution in a reasonable manner.

6.5.1 Use case: Mutual authentication -toolbox-

Server trust anchor which is stored in a data object can be modified by an attacker. By doing that, the attacker can mimic the legitimate server and misuse the services provided or consumed by the nodes.

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

6.5.2 Use case: Host firmware update -toolbox-

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

Firmware update shared secret, which is stored in one of the data objects could be modified and read out by an attacker. By reading the global shared secret, the attacker can create a faulty image containing malicious code which could be multicast installed to all nodes of the same type. By writing the global shared secret, the attacker can create a faulty image containing malicious code which could be installed on the modified node.

- After setting the shared secret, the respective data object access conditions RD and CHA shall be configured with never allowed (NEV) and the life cycle state of the data object (LcsO) shall be set to operational (op) to prevent changes to the access conditions

Platform integrity trust anchor which is stored in a data object can be modified by an attacker. By doing that the attacker can create new metadata which will be accepted by the modified node. This might be used to undermine the rollback prevention of the solution and could lead to installing known security flaws.

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

Chapter 6 Security considerations in [19] shall be reviewed and taken as a guideline with regards to:

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

6.5.3 Key usage associated to toolbox functionality

Key usage which is stored in key object metadata can be modified by an attacker. By 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 restricted usage (only with the respective host)
- After setting the key usage, the life cycle state of the key object (LcsO) shall be set to operational (op) to prevent changes to the key usage

6.5.4 Key pair generation associated to toolbox functionality

The generated key pair and the associated public key certificate are stored in key object and public key certificate data object. The attacker attempts to re-generate the key pair. By doing that the attacker is dropping the identity which was associated to that key pair and could be considered as a DoS attack.

Note: A similar result could be achieved in case only the certificate data object gets corrupted.

- After installing the identity, the respective key object and public key certificate access conditions CHA shall be configured with never allowed (NEV) or allow just protected update with integrity and confidentiality protection and the life cycle state of the key and data object (LcsO) shall be set to operational (op) to prevent changes to the access conditions

6.5.5 Static key generation associated to toolbox functionality

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

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

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

- A shared secret (can be a pre-shared secret or authorization reference which gets fed in a key derivation or MAC generation and/or verification (example: HMAC with SHA256) operations, either from the session context or from a data object shall be at least of a size of 16 bytes
- Restrict the maximum usage of the secret using the monotonic counters to 2048 times is recommended
- EXE access condition tied with platform binding shared secret (CONF E140), enforces the shielded connection for the usage of pre-shared secret in OPTIGA™ during the respective critical operations (example: Key derivation or MAC generation) which does not allow the usage with unknown hosts
- If the shared secret gets updated on the field either by protected update (integrity and confidentiality) or shielded connection, the regarded usage counter must be updated accordingly to allow the usage of the shared secret further for the required number of times

6.5.7 Auto states

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

6.5.8 Shielded connection

- The security level of the shielded connection is as high as a 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 run-time using the shielded connection is recommended as shown in Figure 14
  - Update the platform binding secret and followed by re-establishing the shielded connection, can also be used by the host to validate (to ensure freshness in shielded connection session) when required. But this procedure would lead to NVM write operations (example: Writing to platform binding secret data object, security events due to the usage of shared secrets in performing the shielded connection). So the endurance of the data object and overall endurance as specified in overview data and key store must be considered accordingly
  - Enable the monotonic counter and reduce the maximum number of usages to XYZ using the monotonic counter and update the secret on the 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 (example: RSA keys, AES keys, and shared secrets) if assets are not intended to use extremely
- It is recommended to use the shielded connection for EncryptAsym (which uses a session context) command-based operations, which enforces the usage of session context
- The host can validate OPTIGA™ using the sequence specified in Host authenticates OPTIGA™ to ensure freshness

6.5.9 Algorithm usage

- The recommendation is to use RSA 2048 against RSA 1024

6.6 Shielded connection V1 guidance

The OPTIGA™ shielded connection enables protected (integrity and confidentiality) communication between the OPTIGA™ and a corresponding host platform as depicted in the figure below.

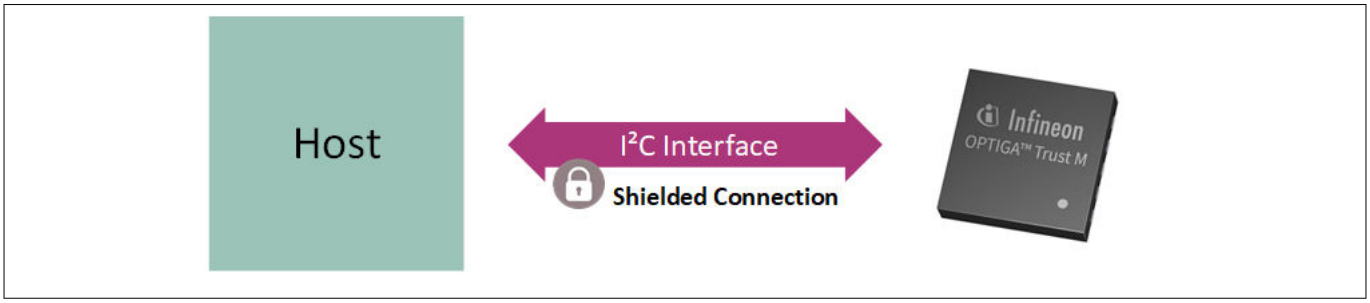

Figure 37 Overview OPTIGA™ shielded connection

This section provides information regarding the setup 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. [2] explains internal details of establishing the shielded connection (example: Negotiation and crypto algorithms used for the protection in the section presentation layer).

Note: The shielded connection does not provide an option to add random challenge/nonce from the host to ensure the freshness of the established shielded connection. This can be achieved with a few more additional steps as described in host authenticates OPTIGA™.

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 the first boot/initialization sequence. Figure 12 shows 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 the 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: That is command and response, protection for deriving the decryption keys in firmware update use case using the pre-shared secret from OPTIGA™:

- Invoke OPTIGA_CRYPT_SET_COMMS_PROTECTION_LEVEL (me_crypt, OPTIGA_COMMS_FULL_PROTECTION)
- Invoke optiga_crypt_tls_prf_sha256 (me_crypt, shard_secret_oid, etc..,)

In case of re-establishing the shielded connection periodically, the protection_level | OPTIGA_COMMS_RE_ESTABLISH can be set to the instance using the 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, etc..,)

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; that is OPTIGA_COMMS_DEFAULT_PROTECTION_LEVEL once the operation is invoked.

Figure 14 shows a process of updating the platform binding secret periodically using the shielded connection at runtime.

6.6.3 Host authenticates OPTIGA™

After the shielded connection is established, if the host intends to validate/authenticate OPTIGA™ (random challenge driven by the host) or to ensure the freshness in the established shielded connection, one of the below-specified mechanisms can be performed by local_host_application. This must be performed once (at least) after the shielded connection is established and/or whenever required.

6.6.3.1 Write and read nonce to/from a data object

The local_host_application writes generated nonce to OPTIGA™ and reads the same data object to authenticate OPTIGA™ and to ensure the freshness (on the host side) in shielded connection.

Note: The write operation shown below leads to writing to NVM at OPTIGA™. So the endurance of the data object and overall endurance as specified in Overview data and key store must be considered accordingly.

Pre-condition:

- The OPTIGA™ application is already launched and the shielded connection between the host and OPTIGA™ is established
- The optiga_util APIs shown in Figure 38 must be invoked with command and response protection using the shielded connection

Post-condition:

- The local_host_application considers OPTIGA™ as an authentic member of the target platform

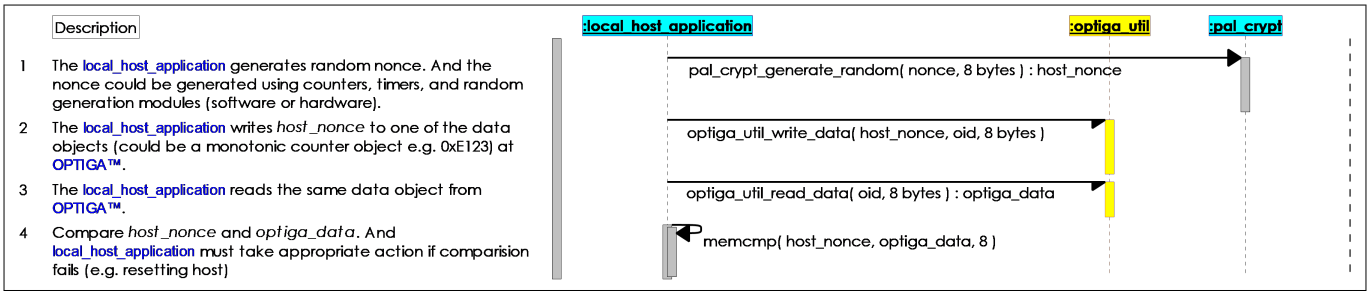

Figure 38 Write and read nonce to/from a data object

6.6.3.2 Derive keys using nonce during run time

The local_host_application establishes a session with intermediate secrets at OPTIGA™ using respective operations. And host further uses this intermediate secret in session to ensure the freshness (on the host side) in the shielded connection.

Note: In this way, there is no additional NVM writes during the respective operations.

Pre-condition:

- The OPTIGA™ application is already launched and the shielded connection between the host and OPTIGA™ is established
- The optiga_crypt APIs shown in Figure 39 must be invoked with command and response protection using the shielded connection

Post-condition:

- The local_host_application considers OPTIGA™ as an authentic member of the target platform

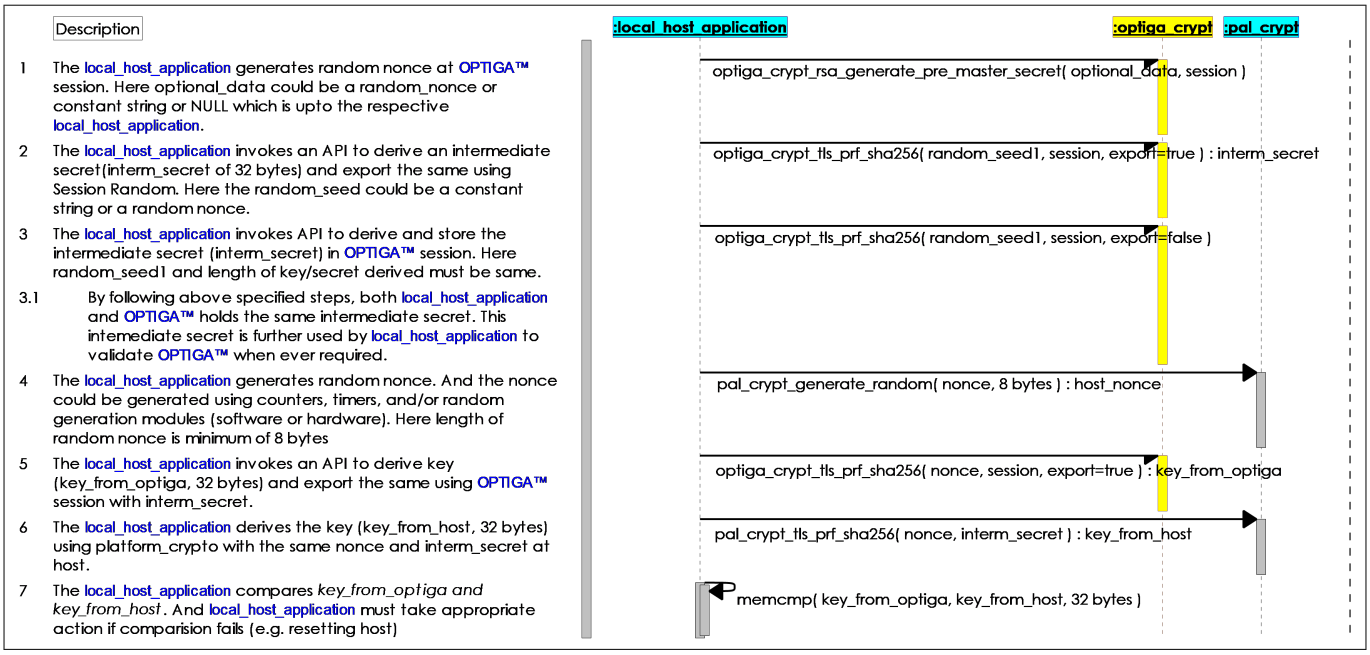

Figure 39 Derive keys using nonce during run time

6.6.3.3 Derive keys using nonce and a static (additional) pre-shared secret

The local_host_application uses an additional pre-shared secret (stored at both sides in NVM) which is already exchanged by local_host_application and OPTIGA™. The local_host_application generates a secret nonce and followed by key derivation using respective operations and uses the same to authenticate OPTIGA™ and to ensure the freshness (on the host side) in the shielded connection.

Note: The key derivation using the static shared secret (from a data object) leads to a security event at OPTIGA™, which leads to NVM write operations (if SEC_{CREDIT} = 0) at OPTIGA™. Hence the endurance of the SEC data object and overall endurance as specified in Overview data and key store must be considered accordingly.

Pre-condition:

- The local_host_application and OPTIGA™ hold an additional static pre-shared secret and the access conditions (e.g. read disabled) of a pre-shared secret at OPTIGA™ are set accordingly
- The OPTIGA™ application is already launched and the shielded connection between the host and OPTIGA™ is established
- The optiga_crypt APIs shown in the below diagram must be invoked with command and response protection using the shielded connection

Post-condition:

- The local_host_application considers OPTIGA™ as an authentic member of the target platform

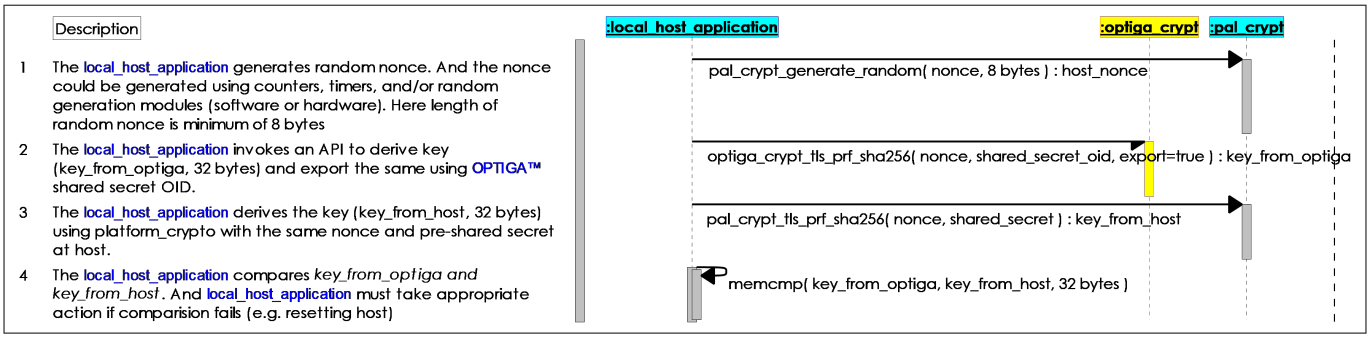

Figure 40 Derive keys using nonce and a static (additional) pre-shared secret

6.7 Protected update

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

Figure 41 shows the high-level structure of the update data set for data or key objects. It consists of a manifest and connected binary data.

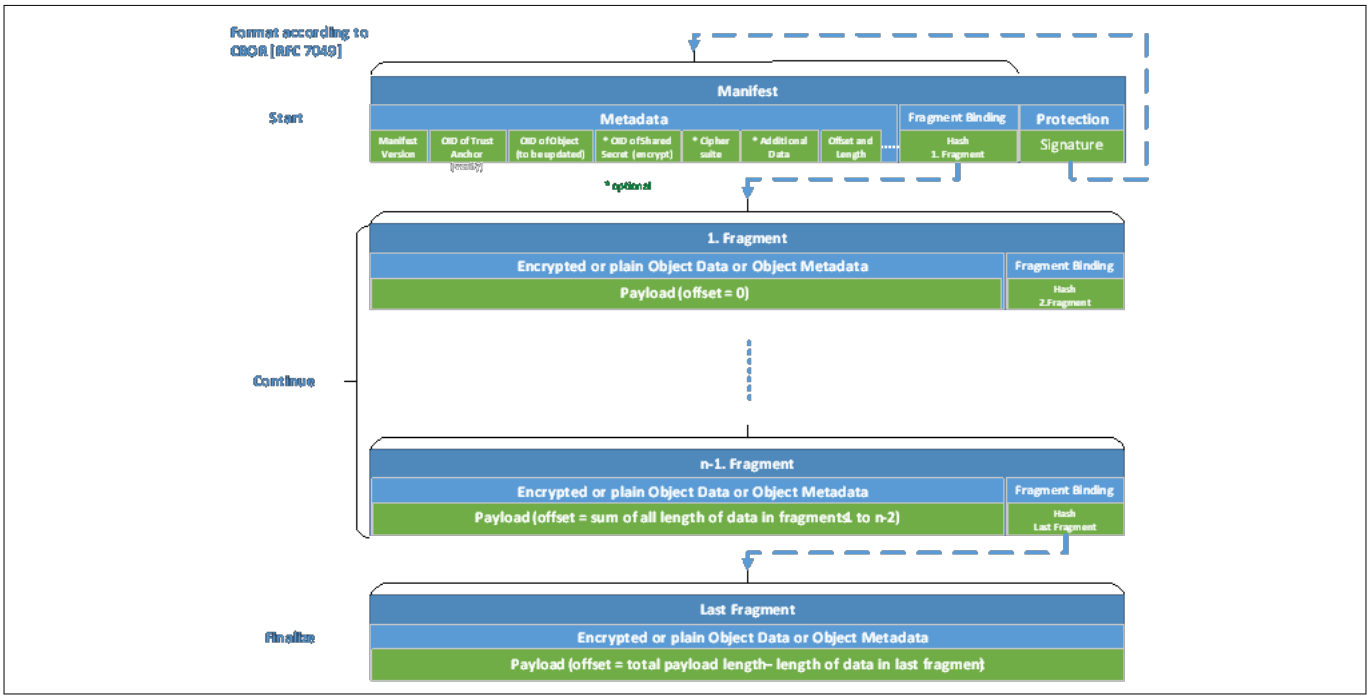

Figure 41 Protected update high-level structure

The coding of the structure is according to [13]. The [14] 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 in 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 provides 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 specifies all cryptographic algorithms (signature, encryption, hash, key derivation, etc..,) used during executing the update
- In case of encrypted object data, OID of the shared secret to be used for the derivation of the decryption key
- In case of encrypted object data, additional data used in the key derivation
- Offset within the target object and length of the object data

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

The confidentiality protection is based on a shared secret installed in the OPTIGA™, and additional data is used to derive the object data decryption key. The session key gets applied as soon as the integrity of the current block of the object data is successfully verified.

Note: The OPTIGA™ Trust M V1 does not support confidentiality protection and update of keys and metadata.

6.7.1 Payload confidentiality

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

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

Key derivation

Figure 42 shows the derivation of the encryption key.

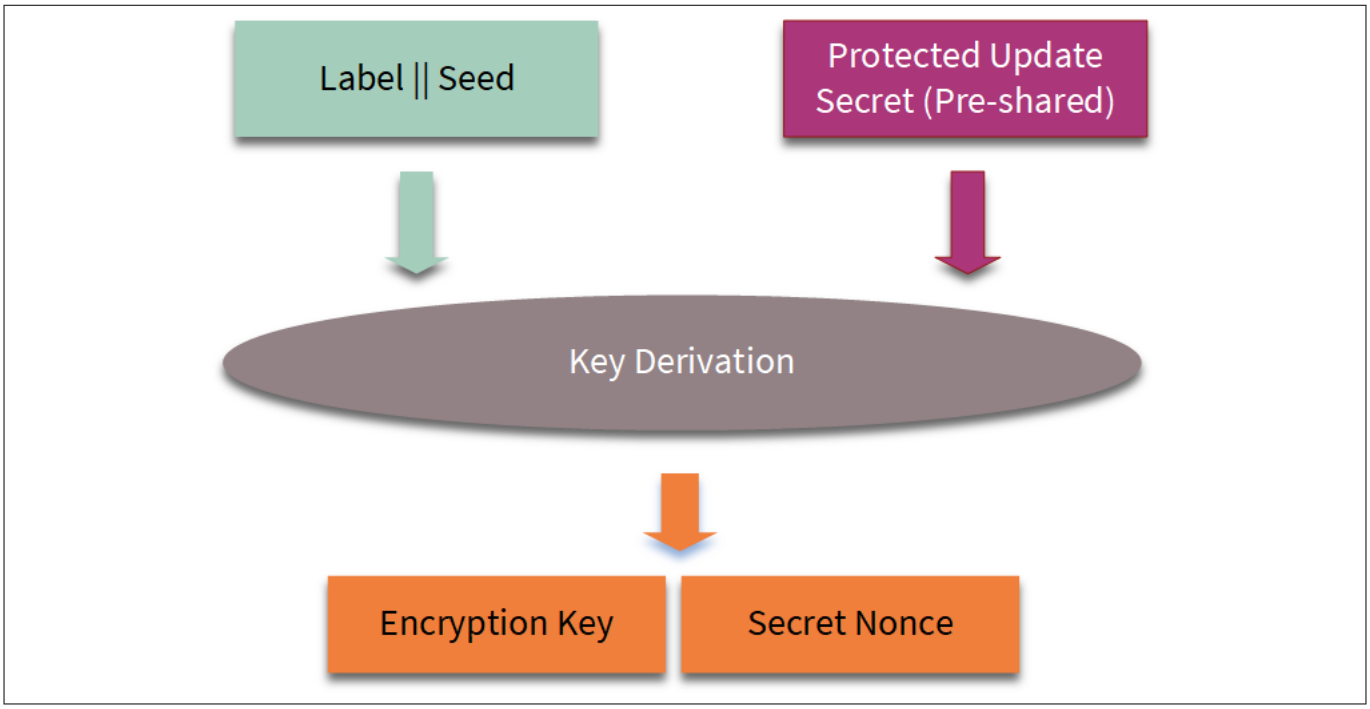

Figure 42 Protected update - derivation for payload encryption key

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

Encryption and decryption

Figure 43 shows the components (input and output) involved as part of the encryption.

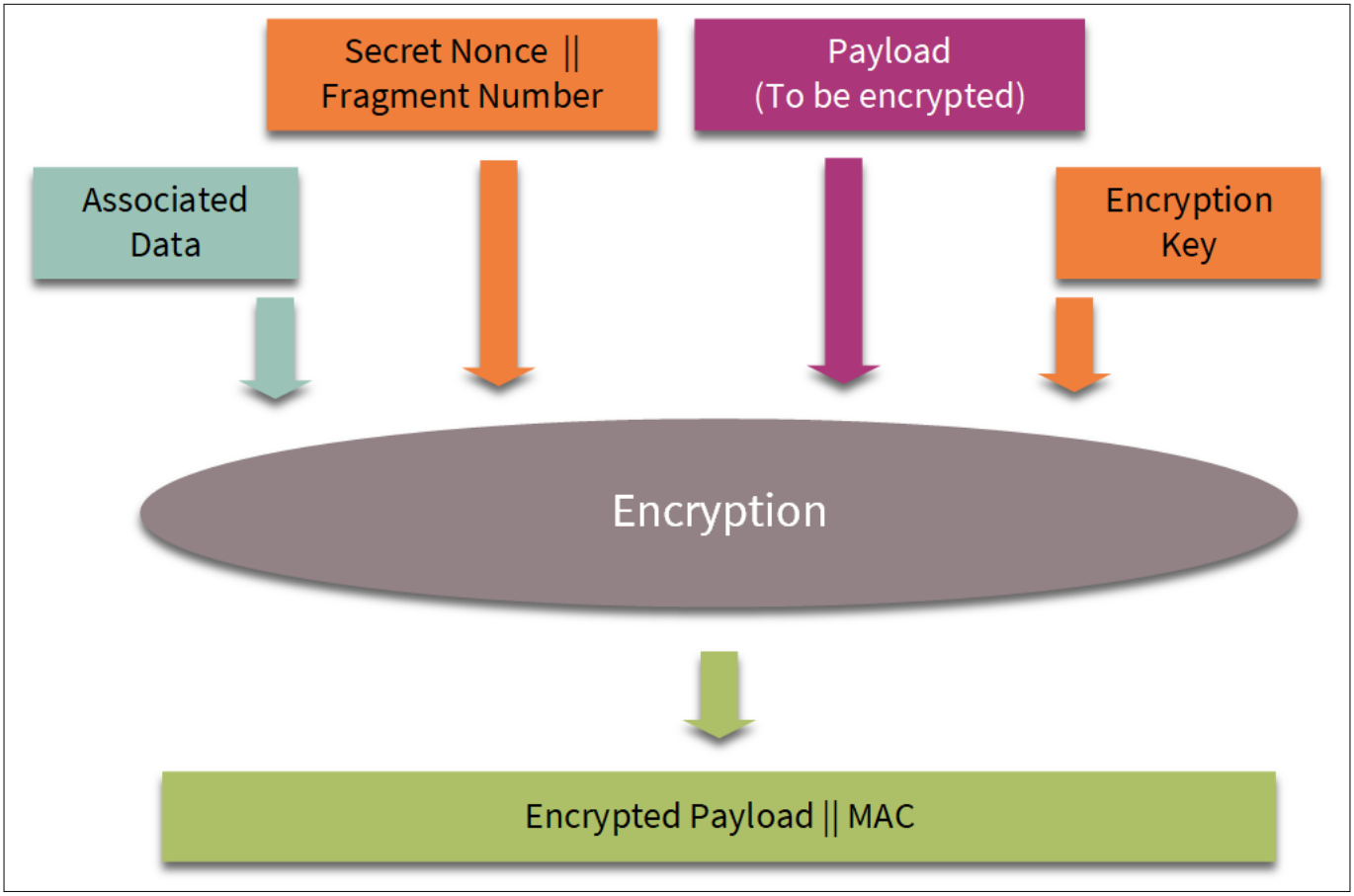

Figure 43 Protected update - payload encryption

Figure 44 shows the components (input and output) involved as part of the decryption.

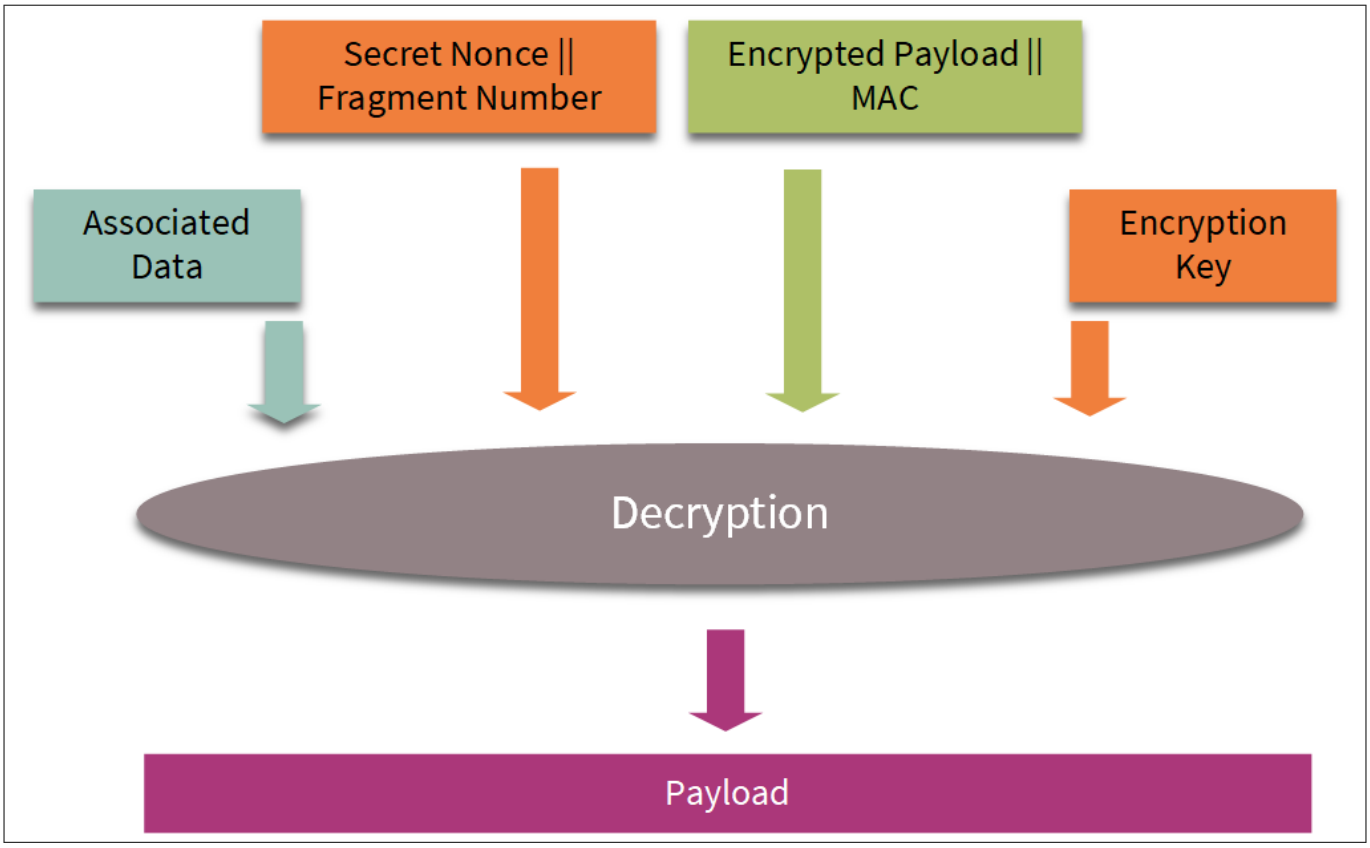

Figure 44 Protected update - payload decryption

- The fragment number is represented in two bytes in octet string format (example: 0x02 is represented as “0x00 0x02”). This is used as part of the nonce which gets concatenated with the secret nonce derived using a key derivation algorithm
  - AES-CCM-16-64-128 from [20], the nonce length to be used is 13 bytes. So, we derive 11 bytes of secret nonce and concatenate with 2 bytes of fragment number
- The size of the MAC is based on the encryption algorithm
- The format of association data as shown below:
  - Associated data = (payload version [2] ||
  - Offset of payload in fragment [3] ||
  - Total payload length [3])

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

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

6.7.2 Format of keys in payload

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

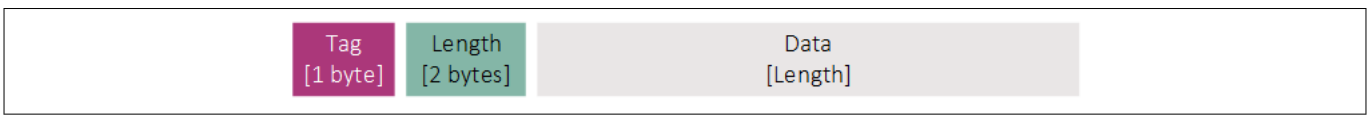

Figure 45 Protected update - encoding of keys in payload

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

6.7.2.1 ECC

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

  The value for the private key tag is 0x01

  The value for the public key tag is 0x02

  For example: The ECC NIST P 256 keys are as given below [values shown below are in hex format]

  - `<01> < 00 20> <29 2E FD 39 …. 4C 74 BC C8>`
  - `<02> < 00 40> <91 8A 12 34 …. 8A 98 1A 52 || 1A 1B 1C 1D …. 2A 2B 2C 2D>`

  For example: The ECC NIST P 256 keys are as given below with padding if the respective component size is less than the key size [values shown below are in hex format]

  - `<01> < 00 20> <00 00 FD 39 …. 4C 74 BC C8>`
  - `<02> < 00 40> <00 8A 12 34 …. 8A 98 1A 52 || 1A 1B 1C 1D …. 2A 2B 2C 2D>`

6.7.2.2 RSA

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

  The tag value for the private exponent is 0x01

  The tag value for the modulus is 0x02

  The tag value for the public exponent is 0x03

  For example: RSA 1024 keys are as given below [values shown below are in hex format]

  - Private exponent: `<01> <00 80> <data = 1A 2F 3D 37 …. 5A 4B E5 F0>`
  - Modulus: `<02> <00 80> <data = 29 2E FD 39 …. 4C 74 BC C8>`
  - Public exponent: `<03> <00 04> <data = 00 10 00 01 >`

6.7.2.3 AES

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

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

  The tag value for the symmetric key is 0x01

  For example: The AES 128 key is as specified below [values shown below are in hex format]

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

6.7.3 Metadata update

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

  The following metadata tags must not be part of the payload:

  - Version [0xC1]
  - Maximum size of the data object [0xC4]
  - Used size of the data object [0xC5]
  - Algorithm associated with key a container [0xE0]
- The format of metadata in the payload is as same as in the GetDataObject command

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

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

6.7.4 CDDL tool

The details of CDDL tool are provided as part of [14] Appendix.

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

6.8 Authorization reference

This section describes how to setup and use authorization reference in the target device application.

Authorization reference is a pre-shared secret that is used to authorize an external entity. Access to a data object can be restricted using this.

Setup:

Authorization reference secret is setup in a data object (e.g. 0xF1D0) by the user/customer during production. The data object type in the metadata must be set to AUTOREF.

Note: The OPTIGA™ Trust M V1 does not support AUTOREF types.

The Use case: Confidentiality protected update of key or a data object depicts a process of updating the keys and data objects (for example, authorization reference).

Refer to Shared secret for key derivation or MAC generation associated to toolbox and protected update functionalities for security guidance on usage and configuration of authorization reference secret.

To restrict access to an object using authorization reference, the respective access condition in the metadata must be specified as AUTO (authorization reference OID).

Usage:

The Use case: Verify authorization (HMAC with SHA2) depicts the process of authorization using authorization reference secret.

Authorization can be achieved by using the specified authorization reference secret to perform HMAC verification in the OPTIGA™ on the provided HMAC (by the Host). OPTIGA™ keeps the achieved AUTO state in RAM after successfully verifying the provided HMAC.

A random value must be requested from OPTIGA™ for HMAC generation on the Host. The requested random value must be present in the input data for HMAC generation, and the secret used must match the authorization reference secret on OPTIGA™.

Example:

Figure 46 and Figure 47 depict how to enable read access to a data object (0xF1D1) by using authorization (authorization reference in data object 0xF1D0) and then reading the data once the access condition is met.

Figure 46 Protecting a data or key object using authorization reference

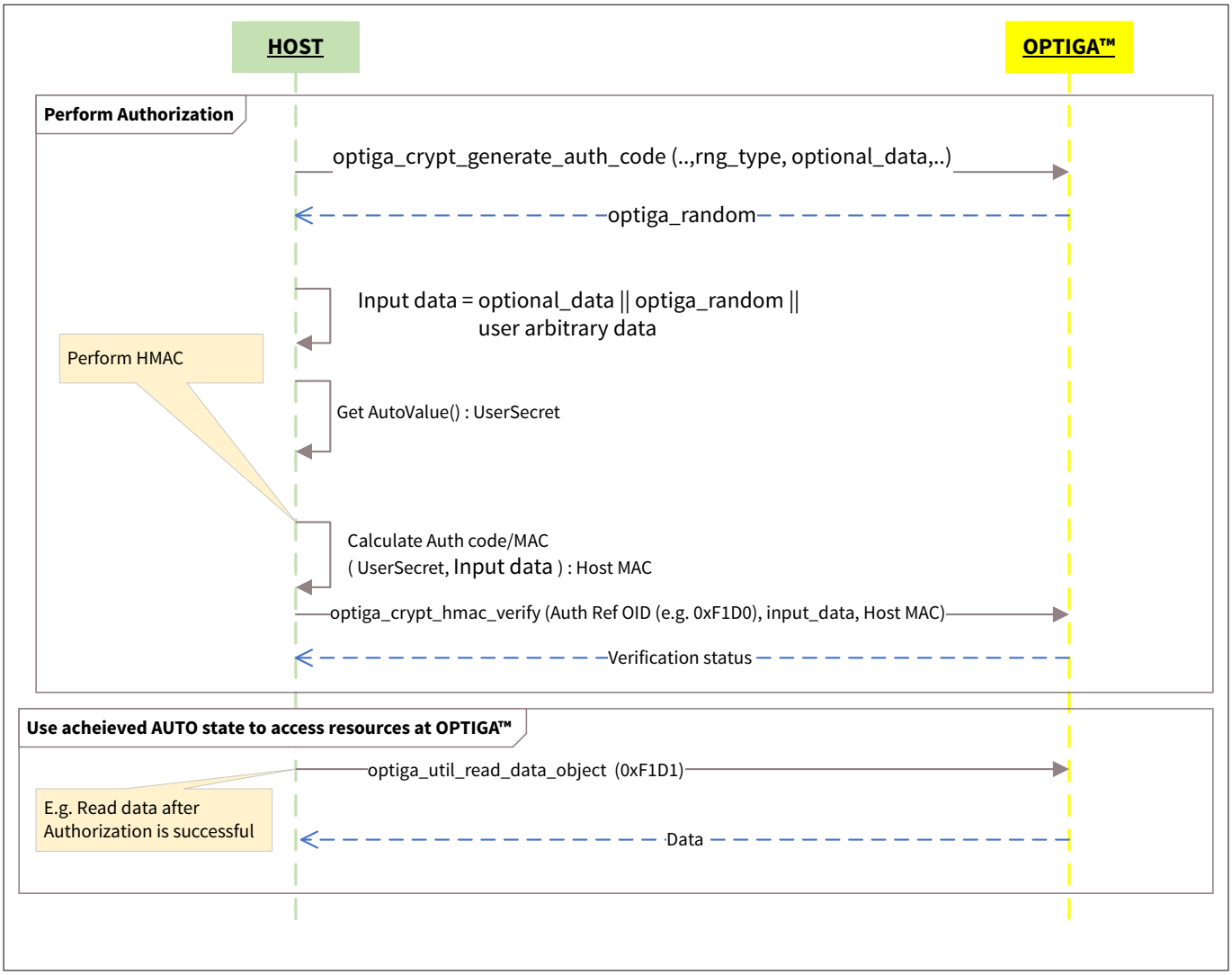

Figure 47 Reading the data after authorization reference

6.9 Terminology

This section 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 terminology to capture project-specific terms, and to ensure that all kind of specifications make correct and consistent use of those terms.

Table 80 Terms of OPTIGA™ vocabulary

Term	Description	Abbreviation
Class	is a fundamental concept of object-oriented programming. It contains 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	

(table continues...)

Table 80 (continued) Terms of OPTIGA™ vocabulary

Term	Description	Abbreviation
Datagram transport layer security	Datagram transport layer security (DTLS) protocol provides communications privacy for datagram protocols. The protocol allows client/server applications to communicate in a way that is designed to prevent eavesdropping, tampering, or message forgery. The DTLS protocol is based on transport layer security (TLS) protocol and provides equivalent security guarantees [21]	DTLS
Deadline	In real-time concepts, a Deadline is the time after action initiation by which the action must be completed	
Denial of service	In computing, a denial-of-service attack (DoS attack) is a cyber-attack in which the perpetrator seeks to make a machine or network resource unavailable to its intended users by temporarily or indefinitely disrupting the service of a host or network	DoS
Designed for re-use	Designed for re-use is a 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, etc..,) 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	The 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 (example: 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 memory	Random access memory is a form of computer data storage. A random access memory device allows data items to be read and written in roughly the same amount of time regardless of the order in which data items are accessed	RAM

(table continues...)

Table 80 (continued) Terms of OPTIGA™ vocabulary

Term	Description	Abbreviation
Secure microcontroller	Secure microcontroller is a Microcontroller particularly designed for embedded security applications and is hardened against a huge variety of attacks that threaten the contained assets	SecMC
System	A system is a set of interacting or interdependent components forming an integrated whole. Every system is circumscribed by its spatial and temporal boundaries, surrounded and influenced by its environment, described by its structure and purpose, and expressed in its functioning	
Transport layer security	Transport layer security (TLS) protocol provides communications privacy for IP based (example: TCP/IP) protocols. The protocol allows client/ server applications to communicate in a way that is designed to prevent eavesdropping, tampering, or message forgery	TLS
Trust anchor	A trust anchor represents an authoritative entity via a public key and associated data. The public key is used to verify digital signatures, and the associated data is used to constrain the types of information for which the trust anchor is authoritative	
Trust anchor store	A trust anchor store is a set of one or more trust anchors stored in a device. A device may have more than one trust anchor in the store, each of which may be used by one or more applications	

References

[1] NXP: *UM10204 I2C-bus specification and user manual*; http://www.nxp.com/documents/user_manual/UM10204.pdf, http://www.nxp.com/documents/user_manual/UM10204.pdf

[2] [IFX_I2C] Infineon Technologies AG: *IFX I2C Protocol Specification*

[3] [RFC8017] PKCS #1: *RSA Cryptography Specifications Version 2.2*; https://tools.ietf.org/html/rfc8017

[4] [SP 800-38A]: *Recommendation for Block cipher modes of operation*; https://nvlpubs.nist.gov/nistpubs/Legacy/SP/nistspecialpublication800-38a.pdf

[5] [RFC5280]: *Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile*; https://tools.ietf.org/html/rfc5280

[6] [RFC5869]: *HMAC-based Extract-and-Expand Key Derivation Function (HKDF)*; https://tools.ietf.org/html/rfc5869

[7] [RFC2104]: *HMAC: Keyed-Hashing for Message Authentication*; https://tools.ietf.org/pdf/rfc2104.pdf

[8] [ISO 9797-1] ISO/IEC 9797-1:2011: *Security techniques - Message Authentication Codes (MACs) - Part 1: Mechanisms using a block cipher (Second edition)*; 2011-03; https://www.iso.org/standard/50375.html

[9] [SP 800-38B]: *Recommendation for Block Cipher Modes of Operation: The CMAC Mode for Authentication*; https://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.800-38b.pdf

[10] [RFC5246]: *The Transport Layer Security (TLS) Protocol, Version 1.2, August 2008*; https://tools.ietf.org/html/rfc5246

[11] [Data Sheet M] Infineon Technologies AG: *OPTIGA™ Trust M, Datasheet*

[12] [I2C] NXP: *UM10204 I2C - bus specification and user manual*; http://www.nxp.com/documents/user_manual/UM10204.pdf, http://www.nxp.com/documents/user_manual/UM10204.pdf

[13] [CBOR]: *Concise Binary Object Representation(CBOR)*; https://tools.ietf.org/html/rfc7049

[14] [CDDL]: *Concise data definition language (CDDL): a notational convention to express CBOR and JSON data structures [Draft version]*; https://tools.ietf.org/html/draft-ietf-cbor-cddl-05

[15] [AIS-31]: *A proposal for: Functionality classes for random number generators*; https://www.bsi.bund.de/SharedDocs/Downloads/DE/BSI/Zertifizierung/Interpretationen/AIS_31_Functionality_classes_for_random_number_generators_e.pdf?__blob=publicationFile

[16] [SP 800-90A]: *Recommendation for Random Number Generation Using Deterministic Random Bit Generators (SP 800-90A Rev1)*; http://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.800-90Ar1.pdf

[17] [SP 800-56A]: *Recommendation for Pair-Wise Key Establishment Schemes Using Discrete Logarithm Cryptography*; https://csrc.nist.gov/publications/detail/sp/800-56a/rev-3/final

[18] [USB Auth]: *Universal Serial Bus Type-C Authentication Specification*; http://www.usb.org/developers/docs

[19] [RFC4108]: *Using Cryptographic Message Syntax (CMS) to Protect Firmware Packages*; https://www.rfc-editor.org/rfc/rfc4108.html#page-51

[20] [COSE]: *CBOR Object Signing and Encryption*; https://tools.ietf.org/html/rfc8152

[21] [RFC6347]: *Datagram Transport Layer Security Version 1.2*; https://tools.ietf.org/html/rfc6347

[22] [COSE RSA]: *Using RSA Algorithms with CBOR Object Signing and Encryption messages*; https://tools.ietf.org/html/rfc8230

[23] [DAVE™] Infineon Technologies AG: *DAVE™ and complementary tools supporting the entire development process from evaluation-to-production (E2P)*; https://infineoncommunity.com/dave-download_ID645

[24] [FIPS PUB 140-2]: *FIPS 140-2, Security Requirements for Cryptographic Modules (May 25, 2001; Change Notice 2, 12/3/2002)*; http://csrc.nist.gov/publications/fips/fips140-2/fips1402.pdf

[25] [FIPS PUB 186-3]: *Updated Digital Signature Standard Approved as Federal Information Processing Standard (FIPS)186-3*; https://www.nist.gov/publications/updated-digital-signature-standard-approved-federal-information-processing-standard

[26] [IANA]: *Transport Layer Security (TLS) Parameters*; http://www.iana.org/assignments/tls-parameters/tls-parameters.xhtml

[27] [RFC2631]: *Diffie-Hellman Key Agreement Method*; https://tools.ietf.org/pdf/rfc2631.pdf

[28] [RFC2986]: *Certificate Request Syntax Specification Version 1.7*; https://tools.ietf.org/html/rfc2986

[29] [RFC5116]: *An Interface and Algorithms for Authenticated Encryption*; https://tools.ietf.org/html/rfc5116

[30] [RFC6655]: *AES-CCM Cipher Suites for Transport Layer Security (TLS)*; https://tools.ietf.org/html/rfc6655

References

[31] [RFC7251]: *AES-CCM Elliptic Curve Cryptography (ECC) Cipher Suites for TLS*; https://tools.ietf.org/html/rfc7251

[32] [RFC7301]: *Transport Layer Security (TLS) - Application-Layer Protocol Negotiation Extension*; https://tools.ietf.org/html/rfc7301

[33] [RFC7925]: *Transport Layer Security (TLS)/ Datagram Transport Layer Security (DTLS) Profiles for the Internet of Things*; https://tools.ietf.org/html/rfc7925

[34] [SP 800-38C]: *Recommendation for Block Cipher Modes of Operation: the CCM Mode for Authentication and Confidentiality*; http://csrc.nist.gov/publications/nistpubs/800-38C/SP800-38C.pdf

[35] [SUIT_DRAFTv2]: *A CBOR-based Manifest Serialization Format [Draft version]*; https://tools.ietf.org/html/draft-moran-suit-manifest-02

[36] [SysML]: *Object Management Group: "OMG Systems Modeling Language (OMG SysML™) - Version 1.2", June 2010, formal/2010-06-01*; http://www.omg.org/spec/SysML/1.2/PDF/

[37] [UML]: *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*; http://www.omg.org/spec/UML/2.4.1

[38] Infineon Technologies AG: *OPTIGA™ Trust M, Configuration Guide (Revision 1.2);* 2022-11-09

[39] Infineon Technologies AG: *OPTIGA™ Trust M Cloud ID, User Guide (Revision 1.2);* 2022-11-09

Glossary

AC
access condition (AC)

AES
Advanced Encryption Standard (AES)
The standard for the encryption of electronic data established by the U.S. National Institute of Standards and Technology (NIST) in 2001. The algorithm described by AES is a symmetric-key algorithm (i.e. the same key is used for both encryption and decryption).

APDU
application protocol data unit (APDU)
The communication unit between a smart card reader and a smart card.

API
application programming interface (API)

ASCII
American Standard Code for Information Interchange (ASCII)
A worldwide code for computer representation of character sets. The basic set has 128 characters (7-bit code).

AUT
authentication key (AUT)

BCD
binary coded decimal (BCD)

BDD
block definition diagram (BDD)

CA
certificate authority (CA)

CBC
cipher block chaining (CBC)

CBOR
concise binary object representation (CBOR)

CDDL
concise data definition language (CDDL)

CERT
certificate (CERT)

CMAC
cipher-based MAC (CMAC)

CRL
certificate revocation list (CRL)

DDK
device driver kit (DDK)

DER
distinguished encoding rules(DER)

DO
data object (DO)

DoS
denial of service (DoS)

DRNG
deterministic random number generator (DRNG)

DTLS
datagram transport layer security (DTLS)

EAL
evaluation assurance level (EAL)

ECB
electronic code book (ECB)

ECC
elliptic curve cryptography (ECC)

ECDSA
elliptic curve digital signature algorithm (ECDSA)

ENC
encryption (ENC)

ESW
embedded software (ESW)
A general term referring to software components running on a secure microcontroller. These can be, but are not limited to, application and operating system software.

FIPS
Federal Information Processing Standards (FIPS)
Publicly announced standards developed by the United States federal government for use in computer systems by non-military government agencies and government contractors.

GPIO
general purpose input/output (GPIO)

HMAC
hash-based message authentication code (HMAC)

I2C
inter-integrated circuit (I2C)

IFX

Infineon Technologies AG (IFX)
The stock market acronym for Infineon Technologies AG shares. It is sometimes used in diagrams or tables where the long term hinders readability.

IoT

Internet of Things (IoT)

LCS

life cycle state (LCS)

MAC

message authentication code (MAC)
Used to prove message integrity.

MCU

Microcontroller Unit (MCU)

NIST

National Institute of Standards and Technology (NIST)

NVM

non-volatile memory (NVM)

NW

network (NW)

OID

object identifier (OID)

OID

object identifier (OID)

PAL

platform abstraction layer (PAL)

PKCS

public key cryptography standards (PKCS)

PKI

public key infrastructure (PKI)
A set of roles, policies, hardware, software, and procedures needed to create, manage, distribute, use, store, and revoke digital certificates and manage public-key encryption.

PP

protection profile (PP)

PrK

private key (PrK)
Equivalent to SCD.

PuK
public key (PuK)
Equivalent to SVD.

RAM
random access memory (RAM)

RSA
Rivest Shamir Adleman (RSA)
An asymmetric cryptographic algorithm in which the encryption key is public and differs from the decryption key, which is kept secret (private).

SecMC
secure microcontroller (SecMC)

SEC
security event counter (SEC)

SHA
Secure Hash Algorithm (SHA)

SW
software (SW)

TBD
to be defined (TBD)

TBS
to be specified (TBS)

TLS
transport layer security (TLS)

TRNG
true random number generator (TRNG)

UID
unique identifier (UID)

USB
universal serial bus (USB)

Revision history

Revision	Date	Description
3.50	2022-11-09	• Layout change
3.40	2022-10-20	• Editorial changes • Migrated to latest template
3.30	2022-10-11	• Updated data and key store endurance related details • Editorial updates • Migrated to latest template
3.15	2020-09-22	• Editorial updates
3.10	2020-09-18	• Updated guidance with respect to shielded connection (Section 6.5.8 and Section 6.6) • Editorial updates
3.00	2020-06-30	• Release version

Trademarks

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

Edition 2022-11-09

Published by

Infineon Technologies AG

81726 Munich, Germany

Do you have a question about any aspect of this document?

Email:
CSSCustomerService@infineon.com

Document reference
IFX-idt1662982472652

Important notice

The information contained in this application note is given as a hint for the implementation of the product only and shall in no event be regarded as a description or warranty of a certain functionality, condition or quality of the product. Before implementation of the product, the recipient of this application note must verify any function and other technical information given herein in the real application. 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) with respect to any and all information given in this application note.

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.

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.