Integration manual - TO136 on standard MCU device

Release 6.3.10

March 6, 2023

Authored by *Trusted Objects*

Contents

1. Legal

1.1 Disclaimer

These softwares, source codes, header files, and documentations (hereinafter referred to as materials) are provided AS IS, without warranty of any kind, express or implied, including but not limited to the warranties of merchantability, fitness for a particular purpose and non-infringement.

In no event shall the authors or Copyright holders be liable for any claim, damages or other liability, whether in an action of contract, tort or otherwise, arising from, out of or in connection with the materials or the use or other dealings in the materials.

2. Introduction

The libTO is a library used as an abstraction layer between Secure Element and your software, in order to make its usage as simple as possible.

The libTO is to be integrated as part of your software. libTO provides to your application an interface to easily deal with Secure Element features. libTO helps developers to work with the Secure Element, as an abstraction layer between its API and I2C communications.

The library is designed to be able to run on MCUs, as on Linux embedded hardwares. Dynamic allocation is not used by the library.

You can find in this documentation details about the library, installation and settings instructions, information on I2C wrappers, and API references.

2.1 Overall architecture

Below is the detailed librarys architecture.

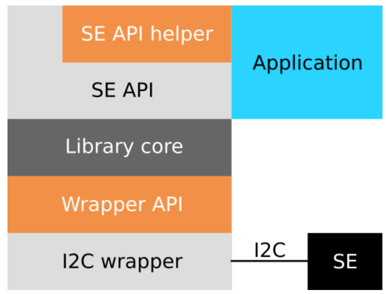


Fig. 1: Library architecture

Two developers APIs are available to use from your application: *Secure Element API* and *Helper API*.

These APIs are using library internal mechanisms to abstract the Secure Element communication protocol. However, this internal layer provides *Library core APIs*, which you may want to use for debugging or advanced uses.

The communication flow can (optionally) rely on a Secure Link protocol, which aims to encrypt and authenticate communication between Secure Element and MCU. If needed, request documentation about Secure Link to Trusted Objects.

Finally, everything relies on an *I2C wrapper*, which is hardware dependent, internally accessed through the *I2C wrapper API*.

2.2 Library files tree

The libTO library files tree structure is the following:

- **/include:** headers providing library APIs, see *Provided API*
- **/src:** library sources
- **/src/:** Secure Element bindings
- **/src/wrapper:** I2C wrappers, to abstract Secure Element I2C communications, a *.C* file is provided for every supported platform, and you are free to implement your own, see *I2C wrapper*
- **/examples:** some examples to use the library from your project

2.3 Limitations

Warning: Due to the underlying I2C bus, the library is **not** designed to be used simultaneously by different processes, so doing that may cause undefined behavior.

Your application must manage concurrent accesses to libTO functions calls, and cumulative APIs sequences (init, update, final) must not be interrupted by another call.

3. Library setup and configuration

3.1 Use the library in an MCU project

In order to work with this library in an MCU project, please follow the integration instructions below.

Note: The following prerequisites are expected in this article:

- a Secure Element soldered onto a development board and connected to the I2C bus
- the ability to build C code for the target hardware

Install library in your sources tree

The following directories are to be considered:

- **include**: header files, providing definitions and APIs
- **src**: implementation
- **wrapper**: I2C wrapper (platform dependent), to allow the library and the Secure Element communications, see *Use an existing I2C wrapper or develop your own one*

There are two ways to install the library in your sources tree:

- you can simply put the Secure Element library folder in your sources tree, it will be easier to update it on future deliveries, by replacing the folder
- or you can integrate different library parts among your project files, for example copy the *include* directory content with your own headers, or *src* directory contents with your HAL APIs sources

From the *src* folder, the following files are to be included into your build process:

- **src/*.c** library files
- **src/core.c**, the library core
- **src/api_*.c**, Secure Element API
- **src/helper_*.c**, library helpers API, based on Secure Element API
- **src/seclink_*.c**, Secure Link support API

Note: The Secure Link protocol you choose must be enabled in your delivered Secure Element chips (read Secure Link documentation for more details)

3.1.1 Configure your project

Your build process needs some configurations to be able to build and use the library.

3.1.1.1 Headers include paths

No matter the way you installed the library into your source tree, be sure its headers (the files provided in the *include* directory of the library) are accessible from one of your include path.

3.1.1.2 Preprocessor definitions

The **TO_LOG_LEVEL_MAX** preprocessor definition is available to set maximal logs level. Debug level prints out additional logs to help you debugging your application.

Read *Library configuration for an MCU project* for details on all available preprocessor definitions. This document also details endianness settings.

3.1.2 Use an existing I2C wrapper or develop your own one

The I2C wrapper is handling the Secure Element I2C inputs/outputs. It is an underlying stack of the Secure Element library used by every provided API. The wrapper is platform dependent, and you need to use an already existing implementation for your platform, or implement your own. More details from *I2C wrapper*.

3.1.2.1 Use an existing I2C wrapper

Available I2C wrappers are provided into the library *src/wrapper* directory. Just ensure to build the right one for your platform. If there is no wrapper for your platform, continue with *Implement your own I2C wrapper*.

3.1.2.2 Implement your own I2C wrapper

No wrapper is already available for your hardware, then you need to implement a wrapper for your specific platform, according to the provided I2C wrapper API. Please read *I2C wrapper implementation guidelines*.

3.2 Library configuration for an MCU project

The library allows various settings with different granularity in order to customize global settings and select features to enable. These settings may be important, especially to minimize library memory usage.

Note: It is assumed you have read the library setup guide, *Use the library in an MCU project*.

The settings below can be defined through preprocessor definitions from your build environment, or by editing the following files provided with library header files:

- *TO_cfg.h*: provides a way to configure libTO build
- *TODRV_HSE_cfg.h*: provides a way to configure the driver part of libTO
- *TOSE_helper_cfg.h*: provides a way to configure libTO helpers

3.2.1 User configuration file

It might be convenient to define your settings in your configuration file, outside the library tree, in order to isolate your configuration in a single file, and not having your configuration spread into various IDE menus.

For this, you can define the symbol *TO_USER_CONFIG*, in your IDE or on the command line.

When *TO_USER_CONFIG* is defined, the file *TO_user_config.h* will be included by the library.

3.2.2 Global settings

The following preprocessor definitions are available:

Table 1: Global MCU settings

Flag	Description
TO_LOG_LEVEL_MAX	Select maximal log level to compile (log level is also configureable at runtime with `TO_set_log_level()`): -1 (disabled), 0 (error), 1 (warning, default), 2 (info), 3 (debug)
TO_BIG_ENDIAN	Force big endian
TO_LITTLE_ENDIAN	Force little endian
HAVE_ENDIAN_H	Toolchain provides endian.h
HAVE_BYTESWAP_H	Toolchain provides byteswap.h
HAVE_NO_STDINT_H	Toolchain does not provide stdint.h
TO_USER_CONFIG	User provides file TO_user_config.h
TO_I2C_WRAPPER_CONFIG	Ability to configure I2C wrapper, see `TO_data_config()`
TODRV_HSE_LIB_INTERNAL_IO_BUFFER_SIZE	(expert) Customize internal I/O buffer size (maximum 640 bytes due to Secure Element limitations)
TO_CMD_MAX_PARAMS	(expert) Customize maximum number of parameters taken by commands, for internal library use
TO_TLS_SESSIONS_NB	TLS sessions number (default: 2)
TOSE_HELPER_TLS_IO_BUFFER_SIZE	(expert) Customize internal TLS I/O buffer size, must be at least as big as biggest handshake message (defragmented, with handshake header, without record header) except messages containing certificates
TOSE_HELPER_TLS_RX_BUFFER_SIZE	(expert) Customize internal TLS I/O buffer size reserved for reception (default value: half of TOSE_HELPER_TLS_IO_BUFFER_SIZE)
TOSE_HELPER_TLS_FLIGHT_BUFFER_SIZE	(expert) Customize internal TLS flight buffer size, must be at least as big as biggest client flight (defragmented, with handshake header, without record header, adding 4 bytes per handshake message). Unused without DTLS retransmission feature.
TOSE_HELPER_TLS_RECEIVE_TIMEOUT	(expert) Customize internal TLS receive timeout

For the enable/disable flags, just define to enable the expected setting.

3.2.2.1 Endianness

If your target system build environment provides *endian.h* header file (defining functions such as *be32toh()* or *htobe32()*), you can just define the *HAVE_ENDIAN_H* preprocessor macro to 1. If your target system build environment provides *byteswap.h* header file (defining functions such as *__bswap16()* or *__bswap32()*), you can just define the *HAVE_BYTESWAP_H* preprocessor macro to 1. Else, endianness settings may be computed by the library from preprocessor pre-defined macros if available.

If previous solutions are not available, endianness is going to be detected at run time, when `TOSE_init()` function is called by client application.

In all cases, if you know your target endianness, you can force it by defining *TO_BIG_ENDIAN* or *TO_LITTLE_ENDIAN* preprocessor macros to 1 according to your architecture characteristics.

3.2.2.2 Integers (stdint)

If your target system does not provide *stdint.h* header file, you must define *HAVE_NO_STDINT_H* preprocessor macro to 1. The library will declare its needed integer declarations from *TO_stdint.h*.

3.2.3 Features settings

It may be interesting to only enable features required in order to minimize library memory usage.

3.2.3.1 Macroscopic settings

These settings are used to enable or disable large sets of features (macroscopic settings). The following preprocessor definitions are available:

Table 2: Macroscopic settings

Flag	Description
TO_ENDIAN_RUNTIME_DETECT	Runtime endianness detection (default: disabled)
TO_DISABLE_LORA	LoRa APIs (default: enabled)
TO_DISABLE_LORA_OPTIMIZED	LoRa optimized API (default: enabled)
TO_DISABLE_TLS	TLS APIs (default: enabled)
TO_DISABLE_TLS_STACK	TLS stack (default: enabled)
TO_DISABLE_TLS_HELPER	TLS handshake helper (default: enabled)
TO_ENABLE_DTLS	DTLS APIs (default: disabled)
TO_DISABLE_DTLS_RETRANSMISSION	DTLS retransmission (default: enabled)
TO_DISABLE_TLS_OPTIMIZED	TLS optimized API (default: enabled)
TO_DISABLE_ECIES_HELPER	ECIES sequence helper (default: enabled)
TO_DISABLE_TO_INFO	Secure Element informations APIs (get_sn, get_pn,) (default: enabled)
TO_DISABLE_API_GET_RANDOM	Random number generator API (default: enabled)
TO_DISABLE_CERT_MGMT	Certificate management APIs (default: enabled)
TO_DISABLE_SIGNING	Signing and verification APIs (default: enabled)
TO_DISABLE_AES_ENCRYPT	AES encryption/decryption APIs (default: enabled)
TO_DISABLE_SEC_MSG	Secure messaging APIs (default: enabled)
TO_DISABLE_SEC_MSG_HELPER	Secure messaging helper (default: enabled)
TO_DISABLE_SHA256	SHA256 hash APIs (default: enabled)
TO_DISABLE_KEYS_MGMT	Keys management APIs (default: enabled)
TO_DISABLE_FINGERPRINT	Fingerprint APIs (default: enabled)
TO_DISABLE_HMAC	HMAC computation/verification APIs (default: enabled)
TO_DISABLE_CMAC	CMAC computation/verification APIs (default: enabled)
TO_DISABLE_NVM	NVM secure storage APIs (default: enabled)
TO_DISABLE_STATUS_PIO_CONFIG	Secure Element status PIO settings API

Some features are disabled by default and enabled if the relevant flag is defined, the other ones are enabled by default and disabled by defining a flag.

The value of these flags does not matter, only the definition is taken into account.

3.2.3.2 Microscopic settings

These settings are used to enable or disable features with a per-API granularity (microscopic settings).

Every API has its own disable flag to tell compiler to not build the related function.

Disable flags have the following form: *TO_DISABLE_API_<API_NAME>*. For example, *get_serial_number()* API can be disabled by defining the *TO_DISABLE_API_GET_SERIAL_NUMBER* flag.

Some APIs can be disabled by groups:

- ***_init/update/final()** form APIs, as *sha256_init()*, *sha256_update()* and *sha256_final()*, which can be disabled by group using **TO_DISABLE_API_<API_NAME>_INIT_UPDATE_FINAL** definition
- **LoRa** APIs
- **TLS** APIs
- **TLS Optimized** APIs

4. I2C wrapper

4.1 I2C wrapper

To be able to communicate with the Secure Element, libTO needs to rely on an I2C wrapper, the library layer responsible of I2C communications. On every library Secure Element API function call, the underlying I2C wrapper is used to write the command to the Secure Element, and read its response. I2C wrapper depends on target platform I2C hardware.

I2C wrappers are mainly available for MCUs, but it is possible to have PC targets implementation (as CP2112 for Linux and Windows).

4.1.1 Available wrappers

The available wrappers implementations are present into the library *src/wrapper* directory:

- **mbed_os.c**: Mbed generic wrapper
- **stm32_hal.c**: implementation of STM32 wrapper for I2C, using ST HAL
- **arduino.cpp**: implementation of I2C wrapper for Arduino

If the wrapper you need is not already available, you can implement your own for your platform by following *I2C wrapper implementation guidelines*.

4.2 I2C wrapper implementation guidelines

To implement an I2C wrapper according to your I2C hardware, please refer to *I2C wrapper API* and implement your own wrapper functions by following this API documentation.

Once your implementation is complete, you should be able to call *Secure Element API* functions to interact with the Secure Element.

4.2.1 Timeout

Defining timeouts may be important to avoid blocking your code in case of I2C bus communication error with the Secure Element.

So, in your wrapper implementation, it is recommended to define read/write timeouts. We suggest to define 5 seconds timeouts, knowing that this value will never be reached in normal use.

4.2.2 Library debug logs

You may want to enable libTO debug logs to help you implement your I2C wrapper. It prints out I2C read and written data on standard output, so you can refer to the Secure Element datasheet to compare the printed logs with what is expected according to the Secure Element protocol.

For an MCU project, **TO_LOG_LEVEL_MAX** preprocessor flag can be defined to **TO_LOG_LEVEL_DBG** to enable debug mode. If you are building the library with Autotools, use *./configure* with *log_level_max=3* option.

4.2.3 I2C wrapper integration

4.2.3.1 MCU project

For an MCU target, just build your new I2C wrapper C file with your project, to make the library able to rely on it for I2C communications with the Secure Element.

5. TO136 usage examples

5.1 Initialization

Here is how you initialize TO136:

```
static TOSE_ctx_t* se_ctx;

int user_init (void)
{
        int ret;

        // Retrieve driver instance
        se_ctx = TODRV_HSE_get_ctx();

        // Initialize
        ret = TOSE_init(se_ctx);

        if (ret != TO_OK)
        {
                // Handle the situation
                // ...
                return -1;
        }

        return 0;
}
```

Now, you can access TO136 functions.

5.1.1 API usage example

After handshake succeeded, use TOSE_helper_tls_send() and TO_helper_tls_receive() to send and receive data on the TLS link.

```
ret = TOSE_helper_tls_send(tls_ctx, (uint8_t *)rest_command, http_len);
if (ret != TO_OK)
{
        // Handle the situation
        // ...
        return -1;
}

ret = TOSE_helper_tls_receive(tls_ctx, (uint8_t *)response, buffer_len, response_len,
↪5000);
if (ret != TO_OK)
{
```

(continues on next page)

(continued from previous page)

```
        // Handle the situation
        // ...
        return -1;
}
```

5.2 Uninitialization

When you are done, uninitialize TO136

```
int user_terminate(void)
{
        int ret;

        ret = TOSE_fini(se_ctx);

        if (ret != TO_OK)
        {
                // Handle the situation
                // ...
                return -1;
        }

        return 0;
}
```

6. Provided API

6.1 Helper API

Helper APIs are high level APIs designed to make integration easier.

```
#include "TO_helper.h"
```

6.1.1 ECIES sequence

The following functions are an easy-to-use ECIES sequence abstraction. They are to be called successively to complete the sequence. ECIES is a cipher suite standardized by ISO 18033.

Steps:

- authenticate the Secure Element
- authenticate remote device against the Secure Element
- prepare secure messaging

The two first steps are for mutual authentication between remote device and the Secure Element, to prevent man-in-the-middle attacks when messaging.

To complete the ECIES sequence, execute the functions below, in order.

To understand what are short and standalone certificates, please see Datasheet - Certificates description.

6.1.1.1 Authenticate the Secure Element

TO_lib_ret_t **TOSE_helper_ecies_seq_auth_TO**(*TOSE_ctx_t* *ctx, uint8_t certificate_index, uint8_t challenge[*TO_CHALLENGE_SIZE*], uint8_t TO_certificate[sizeof(*TO_cert_short_t*)], uint8_t challenge_signature[*TO_SIGNATURE_SIZE*])

ECIES sequence (1st step): authenticate Secure Element.

This is the ECIES sequence first step, which aims to authenticate Secure Element. It provides a challenge to Secure Element, and get back its certificate and the challenge signed using the private key associated to the certificate.

Refer to Secure Element Datasheet Application Notes - Authenticate Secure Element (and also optimized scheme).

Before call you need to:

- randomly generate a challenge After call you need to:
- check return value (see below)
- verify Secure Element certificate signature using CA public key
- verify challenge signature using Secure Element certificate public key if previous steps are validated, continue with the next ECIES step: TOSE_helper_ecies_seq_auth_remote_1(TOSE_ctx_t *ctx,) to authenticate the remote device.

Parameters

- **ctx** – [**in**] Pointer to the SE context
- **certificate_index** – [**in**] Index of the Secure Element certificate to use
- **challenge** – [**in**] Challenge (randomly generated) to be provided to the Secure Element
- **TO_certificate** – [**out**] Short certificate returned by Secure Element
- **challenge_signature** – [**out**] Signature of the challenge by Secure Element

Returns

TO_OK if this step is passed successfully.

6.1.1.2 Authenticate remote

TO_lib_ret_t **TOSE_helper_ecies_seq_auth_remote_1**(*TOSE_ctx_t* *ctx, uint8_t ca_pubkey_index, uint8_t remote_certificate[sizeof(*TO_cert_standalone_t*)], uint8_t challenge[*TO_CHALLENGE_SIZE*])

ECIES sequence (2nd step): authenticate remote device against Secure Element (part 1)

This is the ECIES sequence second step, which aims to authenticate remote device (server or other connected object). This first part provides remote device certificate to Secure Element, and get back a random challenge which is going to be used later to authenticate remote device.

There is only one remote certificate at a time. If several shared keys are needed, we can overwrite remote certificate after shared keys computing.

Refer to Secure Element Datasheet Application Notes - Authenticate Remote Device.

Before call you need to:

- have completed previous ECIES sequence steps
- have the remote device certificate After call you need to:
- check return value (see below)
- sign the returned challenge using the remote device certificate private key if previous steps are validated, continue with TOSE_helper_ecies_seq_auth_remote_2(TOSE_ctx_t *ctx,) to finalize remote device authentication.

Parameters

- **ctx** – [**in**] Pointer to the SE context
- **ca_pubkey_index** – [**in**] Index of Certificate Authority public key
- **remote_certificate** – [**in**] Remote device standalone certificate
- **challenge** – [**out**] Challenge returned by Secure Element to authenticate remote device

Returns

TO_OK if this step is passed successfully, else:

- TORSP_BAD_SIGNATURE: the remote device certificate CA signature is invalid

TO_lib_ret_t **TOSE_helper_ecies_seq_auth_remote_2**(*TOSE_ctx_t* *ctx, uint8_t challenge_signature[*TO_SIGNATURE_SIZE*])

ECIES sequence (2nd step): authenticate remote device against Secure Element (part 2)

This is the ECIES sequence second step, which aims to authenticate remote device (server or other connected object). This second part provides challenge signed using remote device certificate private key.

Refer to Secure Element Datasheet Application Notes - Authenticate Remote Device.

Before call you need to:

- have completed previous ECIES sequence steps
- compute the challenge signature After call you need to:
- check return value (see below) if previous steps are validated, continue with TOSE_helper_ecies_seq_secure_messaging(TOSE_ctx_t *ctx,).

Parameters

- **ctx** – **[in]** Pointer to the SE context
- **challenge_signature** – **[in]** Challenge signed using remote device certificate private key

Returns

TO_OK if this step is passed successfully, else:

- TORSP_BAD_SIGNATURE: the challenge signature is invalid

6.1.1.3 ECIES secure messaging

TO_lib_ret_t **TOSE_helper_ecies_seq_secure_messaging**(*TOSE_ctx_t* *ctx, uint8_t remote_pubkey_index, uint8_t ecc_keypair_index, uint8_t remote_eph_pubkey[*TO_ECC_PUB_KEYSIZE*], uint8_t remote_eph_pubkey_signature[*TO_SIGNATURE_SIZE*], uint8_t TO_eph_pubkey[*TO_ECC_PUB_KEYSIZE*], uint8_t TO_eph_pubkey_signature[*TO_SIGNATURE_SIZE*])

ECIES sequence (3rd step): prepare secure data exchange.

This is the ECIES sequence third step, which aims to prepare secure messaging. Server and connected object will be able to securely exchange data. It provides remote device ephemeral public key signed using remote device certificate private key, and get back Secure Element ephemeral public key.

Secure Element public keys, AES keys, and HMAC keys have the same index to use them from Secure Element APIs.

Refer to Secure Element Datasheet Application Notes - Secure Messaging.

Before call you need to:

- have completed previous ECIES sequence steps
- generate ephemeral key pair
- sign the ephemeral public key using remote device certificate private key.

After call you need to:

- check return value (see below)
- check Secure Element ephemeral public key signature using Secure Element certificate public key
- compute shared secret using remote device and Secure Element ephemeral public keys
- derive shared secret with SHA256 to get AES and HMAC keys

If previous steps are validated, AES and HMAC keys can be used for secure messaging.

Parameters

- **ctx** – **[in]** Pointer to the SE context
- **remote_pubkey_index** – **[in]** Index where the public key will be stored
- **ecc_keypair_index** – **[in]** Index of the ECC key pair to renew
- **remote_eph_pubkey** – **[in]** Remote device ephemeral public key
- **remote_eph_pubkey_signature** – **[in]** Remote device ephemeral public key signature
- **TO_eph_pubkey** – **[out]** Returned Secure Element ephemeral public key
- **TO_eph_pubkey_signature** – **[out]** Secure Element ephemeral public key signature

Returns

TO_OK if this step is passed successfully, else:

- TORSP_BAD_SIGNATURE: the remote device public key signature is invalid

6.1.2 Secure messaging

TO_lib_ret_t **TOSE_helper_secure_payload**(*TOSE_ctx_t* *ctx, const uint8_t key_index, const *TO_enc_alg_t* enc_alg, const *TO_mac_alg_t* mac_alg, const uint8_t *data, const uint16_t data_len, uint8_t *payload, uint16_t *payload_len)

Transforms a message into a secured payload.

Input (`data`) and output (`payload`) buffers must not be exactly the same. If you want to use the same buffer, you need to shift data from input buffer by `TO_SEQUENCE_SIZE` + `TO_INITIALVECTOR_SIZE` bytes (and send the shifted pointer in `data`).

The MAC tag is calculated on clear data, including sequence counter. This function will add padding after MAC if clear data size is not aligned. Padding scheme is PKCS7 with extra padding length byte (TLS like).

Initial vector is generated by the Secure Element and not included in the data length

You can use following macros to extract parts of payload for advanced usage:

- *TOSE_HELPER_SECURE_PAYLOAD_GET_SEQUENCE()*

- *TOSE_HELPER_SECURE_PAYLOAD_GET_SEQUENCE_LEN()*
- *TOSE_HELPER_SECURE_PAYLOAD_GET_INITIAL_VECTOR()*
- *TOSE_HELPER_SECURE_PAYLOAD_GET_INITIAL_VECTOR_LEN()*
- *TOSE_HELPER_SECURE_PAYLOAD_GET_CRYPTOGRAM()*
- *TOSE_HELPER_SECURE_PAYLOAD_GET_CRYPTOGRAM_LEN()*

Payload length is given by *TO_PAYLOAD_SECURED_PAYLOAD_SIZE()*.

Parameters

- **`ctx`** – [**in**] Pointer to the SE context
- **`key_index`** – [**in**] Index of the keys to use for data encryption and MAC, starting from 0
- **`enc_alg`** – [**in**] Encryption algorithm to use
- **`mac_alg`** – [**in**] MAC algorithm to use
- **`data`** – [**in**] Message to be secured
- **`data_len`** – [**in**] Message length
- **`payload`** – [**out**] Payload
- **`payload_len`** – [**in**] Payload length

Returns

- TORSP_SUCCESS on success
- TORSP_ARG_OUT_OF_RANGE: invalid key index
- TORSP_NOT_AVAILABLE: algorithm not available
- TO_DEVICE_WRITE_ERROR: error writing data to Secure Element
- TO_DEVICE_READ_ERROR: error reading data from Secure Element
- TO_INVALID_RESPONSE_LENGTH: unexpected response length from device
- TO_MEMORY_ERROR: internal I/O buffer overflow
- TO_ERROR: generic error

TO_lib_ret_t **`TOSE_helper_unsecure_payload(`** *TOSE_ctx_t* *ctx, const uint8_t key_index, const *TO_enc_alg_t* enc_alg, const *TO_mac_alg_t* mac_alg, const uint8_t *payload, const uint16_t payload_len, uint8_t *data, uint16_t *data_len)

Get back a message from a secured payload.

Input (`payload`) and output (`data`) buffers can be the same buffer.

The MAC tag is verified on clear data, including sequence counter. This function will remove padding. Padding scheme is PKCS7 with extra padding length byte (TLS like).

You can use following macros to extract parts of payload for advanced usage:

- *TOSE_HELPER_SECURE_PAYLOAD_GET_SEQUENCE()*
- *TOSE_HELPER_SECURE_PAYLOAD_GET_SEQUENCE_LEN()*

- *TOSE_HELPER_SECURE_PAYLOAD_GET_INITIAL_VECTOR()*
- *TOSE_HELPER_SECURE_PAYLOAD_GET_INITIAL_VECTOR_LEN()*
- *TOSE_HELPER_SECURE_PAYLOAD_GET_CRYPTOGRAM()*
- *TOSE_HELPER_SECURE_PAYLOAD_GET_CRYPTOGRAM_LEN()*

Maximal data length is given by *TO_PAYLOAD_CLEAR_DATA_SIZE()*.

Parameters

- **`ctx`** – [**in**] Pointer to the SE context
- **`key_index`** – [**in**] Index of the keys to use for data encryption and MAC, starting from 0
- **`enc_alg`** – [**in**] Encryption algorithm to use
- **`mac_alg`** – [**in**] MAC algorithm to use
- **`payload`** – [**in**] Payload
- **`payload_len`** – [**in**] Payload length
- **`data`** – [**out**] Message unsecured
- **`data_len`** – [**out**] Message length

Returns

- TORSP_SUCCESS on success
- TORSP_ARG_OUT_OF_RANGE: invalid key index
- TORSP_NOT_AVAILABLE: algorithm not available
- TO_DEVICE_WRITE_ERROR: error writing data to Secure Element
- TO_DEVICE_READ_ERROR: error reading data from Secure Element
- TO_INVALID_RESPONSE_LENGTH: unexpected response length from device
- TO_MEMORY_ERROR: internal I/O buffer overflow
- TO_ERROR: generic error

`TOSE_HELPER_SECURE_PAYLOAD_GET_SEQUENCE`(payload) ((payload) + 0)

Get sequence pointer from payload pointer

`TOSE_HELPER_SECURE_PAYLOAD_GET_SEQUENCE_LEN()` (TO_SEQUENCE_SIZE)

Get sequence length

`TOSE_HELPER_SECURE_PAYLOAD_GET_INITIAL_VECTOR`(payload) (*TOSE_HELPER_SECURE_PAYLOAD_GET_SEQUENCE*(payload) + TO_SEQUENCE_SIZE)

Get initial vector pointer from payload pointer

`TOSE_HELPER_SECURE_PAYLOAD_GET_INITIAL_VECTOR_LEN`(enc_alg) *TO_PAYLOAD_IV_SIZE*(enc_alg)

Get initial_vector length

TOSE_HELPER_SECURE_PAYLOAD_GET_CRYPTOGRAM(enc_alg, payload) (*TOSE_HELPER_SECURE_PAYLOAD_GET_INITIAL_VECTOR*(payload) + *TOSE_HELPER_SECURE_PAYLOAD_GET_INITIAL_VECTOR_LEN*(enc_alg))

Get cryptogram pointer from payload pointer

TOSE_HELPER_SECURE_PAYLOAD_GET_CRYPTOGRAM_LEN(enc_alg, payload_len) ((payload_len) - (TO_SEQUENCE_SIZE) - *TOSE_HELPER_SECURE_PAYLOAD_GET_INITIAL_VECTOR_LEN*(enc_alg))

Get cryptogram length

6.1.3 Certificates

TO_lib_ret_t **TOSE_helper_verify_chain_certificate_and_store**(*TOSE_ctx_t* *ctx, const uint8_t ca_key_index, const uint8_t *chain_certificate, const uint16_t chain_certificate_length)

Handle certificate chain at once.

Certificates must be in X509 DER (binary) format. Certificates must be ordered as following:

- Final certificate
- Intermediate CA certificates (if any)
- Root CA certificate (optional as it must already be trusted by the Secure Element)

Each certificate must be signed by the next.

Parameters

- **ctx** – **[in]** Pointer to the SE context
- **ca_key_index** – **[in]** CA key index (use TO_CA_IDX_AUTO to enable Authority Key Identifier based CA detection)
- **chain_certificate** – **[in]** Certificate chain
- **chain_certificate_length** – **[in]** Certificate chain length

Returns

TO_OK if data has been sent successfully, else TO_ERROR

TO_lib_ret_t **TOSE_helper_verify_chain_ca_certificate_and_store**(*TOSE_ctx_t* *ctx, const uint8_t ca_key_index, const uint8_t subca_key_index, const uint8_t *chain_certificate, const uint16_t chain_certificate_length)

Handle CA certificate chain at once.

Certificates must be in X509 DER (binary) format. Certificates must be ordered as following:

- Intermediate CA certificates

- Root CA certificate (optional as it must already be trusted by the Secure Element)

Each certificate must be signed by the next.

Parameters

- **ctx** – **[in]** Pointer to the SE context
- **ca_key_index** – **[in]** CA key index (use TO_CA_IDX_AUTO to enable Authority Key Identifier based CA detection)
- **subca_key_index** – **[in]** subCA index to store subCA
- **chain_certificate** – **[in]** Certificate chain
- **chain_certificate_length** – **[in]** Certificate chain length

Returns

TO_OK if data has been sent successfully, else TO_ERROR

TO_lib_ret_t **TOSE_helper_set_certificate_x509**(*TOSE_ctx_t* *ctx, const uint8_t certificate_index, const uint8_t *certificate, const uint16_t size)

Set new certificate from previously generated CSR.

Set a x509 DER formated certificate according to the given index. The new certificate must be signed by a CA trusted by the Secure Element. Secure Element certificate size cannot exceed TO_CERT_X509_MAXSIZE.

Parameters

- **ctx** – **[in]** Pointer to the SE context
- **certificate_index** – **[in]** Requested certificate index
- **certificate** – **[in]** New certificate data (x509 DER formated)
- **size** – **[in]** New certificate size

Returns

TO_OK if certificate has been sent successfully, else TO_ERROR

TO_lib_ret_t **TOSE_helper_get_certificate_x509_and_sign**(*TOSE_ctx_t* *ctx, const uint8_t certificate_index, const uint8_t *challenge, const uint16_t challenge_length, uint8_t *certificate, uint16_t *size, uint8_t signature[*TO_SIGNATURE_SIZE*])

Returns one of the Secure Element x509 DER formated certificates, and optionnaly a challenge signed with the certificate private key.

Parameters

- **ctx** – **[inout]** SE context
- **certificate_index** – **[in]** Index of the certificate to return, starting from 0
- **challenge** – **[in]** Challenge to be signed, NULL if nothing to sign
- **challenge_length** – **[in]** Length of the challenge to be signed, 0 if nothing to sign
- **certificate** – **[out]** Returned certificate data, this buffer should be at least TO_CERT_X509_MAXSIZE

- **size** – [**inout**] input: the certificates buffer size, output: the certificates real size
- **signature** – [**out**] Returned signature, NULL if nothing to sign

Returns

- TORSP_SUCCESS on success
- TORSP_INVALID_LEN: wrong length
- TORSP_NOT_AVAILABLE: certificate Format not supported
- TORSP_ARG_OUT_OF_RANGE: invalid Certificate Number
- TO_DEVICE_WRITE_ERROR: error writing data to Secure Element
- TO_DEVICE_READ_ERROR: error reading data from Secure Element
- TO_ERROR: generic error

6.1.4 TLS handshake

TLS Helper is a set of functions making TLS handshake easy to integrate.

It only requires to provide 2 callbacks to physically send and receive data.

6.1.4.1 TLS callback functions to define

These callbacks need to be implemented and passed to TLS helper APIs to be able to send / receive data to / from the server.

typedef *TO_lib_ret_t* (***TOSE_helper_tls_send_func**)(void *priv_ctx, const uint8_t *data, const uint32_t len)

Handshake helper network send function.

This function is used by TOSE_helper_tls_handshake to send data on the network.

Param priv_ctx
[**in**] Opaque context given to TOSE_helper_tls_handshake

Param data
[**in**] Data to send

Param len
[**in**] Length of data

Return
TO_OK if data has been sent successfully, else TO_ERROR

typedef *TO_lib_ret_t* (***TOSE_helper_tls_receive_func**)(void *priv_ctx, uint8_t *data, const uint32_t len, uint32_t *read_len, int32_t timeout)

Handshake helper network receive function.

This function is used by TOSE_helper_tls_handshake to receive data from the network.

Note: TO_AGAIN may be returned for example when this callback is implemented with POSIX recv(), and recv() returns #EINTR

Param priv_ctx
[**in**] Opaque context given to TOSE_helper_tls_handshake

Param data
[**in**] Data output

Param len
[**in**] Length of data to read

Param read_len
[**out**] Length of data read

Param timeout
[**in**] Receive timeout in milliseconds (-1 for no timeout)

Retval TO_OK
if some data has been received successfully, read_len is updated and >0

Retval TO_TIMEOUT
timed out elapsed before any data was available

Retval TO_AGAIN
the function has been interrupted before receiving any data

Retval TO_ERROR
Other error

typedef *TO_ret_t* (***`TOSE_helper_tls_unsecure_record`**)(void *ctx, uint16_t header_length, uint8_t *in, uint16_t in_length, uint8_t **out, uint16_t *out_length)

callback to unsecure a received protected record (HANDSHAKE_ONLY_MODE)

Note: The in parameter isnt const because the callback can reuse it to unsecure in place provided it doesnt write above in_length. For example if it uses hardware decryption with constraints on memory regions used by the DMA.

Param ctx
[**inout**] cipher context

Param header_length
[**in**] length of the records header

Param in
[**in**] input buffer containing the entire protected record (e.g. with the header)

Param in_length
[**in**] length of the protected record in the input buffer

Param out
[**out**] buffer with the plain text content of the record (e.g. without the header)

Param out_length
[**out**] length of the plain text content

Retval TO_OK
if the record is authenticated and decrypted

Retval TO_ERROR
Otherwise

typedef *TO_ret_t* (***TOSE_helper_tls_secure_record**)(void *ctx, uint8_t *hdr, uint16_t hdr_length, const uint8_t *in, uint16_t in_length, uint8_t **out, uint16_t *out_length)

callback to secure a plain text record before sending (HANDSHAKE_ONLY_MODE)

> **Note:** input and output buffers provided by the caller may overlap with a gap of at least 1 AES block (*out + hdr_length + AES_BLOCK_LEN <= in).

Param ctx
[**inout**] cipher context

Param hdr
[**in**] plain text record header buffer

Param hdr_length
[**in**] plain text record header length

Param in
[**in**] input buffer with the plain text records content data (e.g. withot header)

Param in_length
[**in**] length of the plain text records content data

Param out
[**inout**] output buffer with the ciphered content of the protected record

Retval TO_OK
if the record cannot be encrypted

Retval TO_ERROR
Otherwise

typedef *TO_ret_t* (***TOSE_helper_tls_setup_cipher_ctx**)(void *ctx, uint16_t cipher_suite, uint8_t **key_block, uint8_t *key_block_length, uint16_t *cipher_overhead_length, *TOSE_helper_tls_unsecure_record* *unsecure_record, *TOSE_helper_tls_secure_record* *secure_record)

callback to setup the cipher context (HANDSHAKE_ONLY_MODE)

> **Note:** This callback is called during the handshake after the cipher suite is negotiated with the server and before extracting the derived key from the Secure Element.

Param ctx
[**inout**] cipher context

Param cipher_suite
[**in**] the negociated cipher_suite identifier (as specified in TLS RFCs)

Param key_block
[**out**] pointer on the key block where key derivation from master secret is stored

Param key_block_length
[**out**] length of the key block, depends upon the negociated cipher suite

Param cipher_overhead_length
[**inout**] the maximum difference of length between the plain text content and the ciphered text content. The caller provides its own value if possible, the callee can lower it to 0 if it provides its own buffer to store protected records.

Param unsecure_record
[**out**] callback used to authenticate and decrypt incoming records

Param secure_record
[**out**] callback used to encrypt data to the outcoming records

Retval TO_OK
if setup completed correctly

Retval TO_ERROR
Otherwise

typedef struct TOSE_helper_tls_ctx_s **`TOSE_helper_tls_ctx_t`**

Opaque TLS helper context

6.1.4.2 Handshake API

Calling one function will do all the steps of the TLS handshake.

void ***`default_cipher_ctx`**

default cipher context when NULL is passed to *TOSE_helper_tls_set_mode_handshake_only()*

TOSE_helper_tls_setup_cipher_ctx **`default_setup_cipher_ctx`**

default setup cipher context when NULL is passed to *TOSE_helper_tls_set_mode_handshake_only()*

TO_lib_ret_t **`TOSE_helper_tls_init_session`**(*TOSE_ctx_t* *ctx, *TOSE_helper_tls_ctx_t* **tls_ctx, const uint8_t session, void *priv_ctx, *TOSE_helper_tls_send_func* send_func, *TOSE_helper_tls_receive_func* receive_func)

Initialize TLS handshake.

This function initialize TLS handshake. It configures the Secure Element and initialize static envrionment.

Each initialized session must be cleaned with *TOSE_helper_tls_cleanup()*.

Parameters

- **`ctx`** – [**in**] Pointer to the SE context
- **`tls_ctx`** – [**in**] TLS context assigned

- **session** – [**in**] TLS session to use
- **priv_ctx** – [**in**] Opaque context to forward to given functions
- **send_func** – [**in**] Function to send on network
- **receive_func** – [**in**] Function to receive from network

Returns
TO_OK if initialization succeed, else TO_ERROR

TO_lib_ret_t **TOSE_helper_tls_close**(*TOSE_helper_tls_ctx_t* *tls_ctx)

Close TLS handshake.

This function closes TLS handshake by sending a close notify alert to the TLS server. Given context must not be used anymore. In TCP, the socket used by this session might not be usable anymore due to close notify alert.

Parameters

- **tls_ctx** – [**in**] TLS context

Returns
TO_OK if close succeed, else TO_ERROR

TO_lib_ret_t **TOSE_helper_tls_fini**(*TOSE_helper_tls_ctx_t* *tls_ctx)

Finalize TLS context.

It is needed to call this function if TCP socket closed for any reason.

Parameters

- **tls_ctx** – [**in**] TLS context

Returns
TO_OK if finalize succeed, else TO_ERROR

TO_lib_ret_t **TOSE_helper_tls_cleanup**(*TOSE_helper_tls_ctx_t* *tls_ctx)

Cleanup TLS handshake.

This function closes and finalizes TLS handshake and session using TOSE_helper_tls_close and TOSE_helper_tls_fini.

Parameters

- **tls_ctx** – [**in**] TLS context

Returns
TO_OK if cleanup succeed, else TO_ERROR

TO_lib_ret_t **TOSE_helper_tls_set_retransmission_timeout**(*TOSE_helper_tls_ctx_t* *tls_ctx, const uint32_t min_timeout, const uint32_t max_timeout)

Set DTLS retransmission timeout min/max values.

Parameters

- **tls_ctx** – [**in**] TLS context
- **min_timeout** – [**in**] Minimal (initial) retransmission timeout, in milliseconds
- **max_timeout** – [**in**] Maximal retransmission timeout, in milliseconds

Returns
TO_OK if cleanup succeed, else TO_ERROR

TO_lib_ret_t **`TOSE_helper_tls_set_retransmission_max`**(*TOSE_helper_tls_ctx_t* *tls_ctx, const uint32_t max_retransmissions)

Set DTLS retransmission max value.

Retransmission counter is reset in case of successful receive.

Parameters

- **`tls_ctx`** – [**in**] TLS context
- **`max_retransmissions`** – [**in**] Maximal retransmissions count

Returns
TO_OK if cleanup succeed, else TO_ERROR

TO_lib_ret_t **`TOSE_helper_tls_set_fragment_max_size`**(*TOSE_helper_tls_ctx_t* *tls_ctx, const uint16_t max_size)

Set DTLS fragment maximum size.

Parameters

- **`tls_ctx`** – [**in**] TLS context
- **`max_size`** – [**in**] Maximum fragment size in bytes (record & handshake headers excluded)

Returns
TO_OK if cleanup succeed, else TO_ERROR

TO_lib_ret_t **`TOSE_helper_tls_set_cipher_suites`**(*TOSE_helper_tls_ctx_t* *tls_ctx, const uint16_t *cipher_suites, const uint16_t cipher_suites_cnt)

Set cipher suites list.

cipher_suites values must be values defined in helper header (TO_TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA256, etc)

Parameters

- **`tls_ctx`** – [**in**] TLS context
- **`cipher_suites`** – [**in**] Array of cipher suites (array of 16-bits integer values. See TO_tls_cipher_suite_e.)
- **`cipher_suites_cnt`** – [**in**] Cipher suites count

Returns
TO_OK in case of success, else TO_ERROR

TO_lib_ret_t **`TOSE_helper_tls_set_config_mode`**(*TOSE_helper_tls_ctx_t* *tls_ctx, *TO_tls_mode_t* mode)

Set configuration mode of the TLS session.

Note: updating the mode is persistent across reboot.

Parameters

- **tls_ctx** – [**in**] TLS context
- **mode** – [**in**] configuration mode (see TO_tls_mode_e)

Returns
TO_OK in case of success, else TO_ERROR

TO_lib_ret_t **TOSE_helper_tls_set_config_certificate_slot**(*TOSE_helper_tls_ctx_t* *tls_ctx, uint8_t certificate_slot)

Configure client certificate slot of the TLS session.

Note: updating the certificate slot is persistent across reboot.

Parameters

- **tls_ctx** – [**in**] TLS context
- **mode** – [**in**] client certificate mode

Returns
TO_OK in case of success, else TO_ERROR

TO_lib_ret_t **TOSE_helper_tls_set_server_name**(*TOSE_helper_tls_ctx_t* *tls_ctx, const char *server_name)

Configure the servers domain name.

When the server name is configured, it is used during handshake within the SNI extension (section 3 - RFC 6066)

Note: server_name may be NULL or empty, in that case the TLS context is configured to not use the SNI extension.

Parameters

- **tls_ctx** – [**inout**] context of the TLS session
- **server_name** – [**in**] a string with the servers domain name

Return values

- **TO_OK** – the server name is configured inside the TLS context
- **TO_ERROR** – the server name configuration failed

TO_lib_ret_t **TOSE_helper_tls_set_mode_handshake_only**(*TOSE_helper_tls_ctx_t* *tls_ctx, void *cipher_ctx, *TOSE_helper_tls_setup_cipher_ctx* setup_cipher_ctx)

configure the TLS session in HANDSHAKE_ONLY_MODE

In this mode the encryption and decryption of TLS records is delegated to the upper layer. This layer shall provide a set of callbacks to be called by the libTO to transmit the key block and to secure/unsecure records.

Note: The callback `setup_cipher_ctx` can been NULL if the libTO has been built with a default callback enabled. In that case the parameter `cipher_ctx` is ignored.

Note: This function shall be called with a initialized tls_ctx, so after calling *TOSE_helper_tls_init_session()*, and it shall be called before starting a handshake, so before *TOSE_helper_tls_do_handshake()* The following sequence show the calls needed to use the mode Handshake Only with the default cipher in AES128-GCM:

```
// function returns are ignored for compactness but should be handled in
// production code.
#define CIPHER_SUITE_CNT 2
uint16_t cipher_suites[CIPHER_SUITE_CNT] =
            {TO_TLS_ECDHE_ECDSA_WITH_AES_128_GCM_SHA256,
            TO_TLS_PSK_WITH_AES_128_GCM_SHA256};
TOSE_helper_tls_ctx_t *tls_ctx;
TOSE_helper_tls_init_session(DEFAULT_CTX, &tls_ctx, session_slot,
            &your_recv_send_ctx, your_send_func, your_receive_func);
TOSE_helper_tls_set_cipher_suites(tls_ctx, // setting the cipher suite is optional
            cipher_suites,     // if the server is known to always
            CIPHER_SUITE_CNT); // choose an AES-GCM cipher suite
TOSE_helper_tls_set_mode_handshake_only(tls_ctx, NULL, NULL);
TOSE_helper_tls_do_handshake(tls_ctx);
```

Note: Once the handshake is completed. The Secure Element can be shutdown with *TOSE_fini()* as the encryption/decryption/authentication of payloads are done at the library layer.

Note: Setting the mode Handshake Only has for effect to change the persistent configuration of the Secure Element. In order to go back to the mode Full TLS, the session shall be re-configured using the following sequence:

```
// function returns are ignored for compactness but should be handled in
// production code.
TOSE_helper_tls_ctx_t *tls_ctx;
TOSE_helper_tls_init_session(DEFAULT_CTX, &tls_ctx, session_slot,
            &your_recv_send_ctx, your_send_func, your_receive_func);
TOSE_helper_tls_set_mode(tls_ctx, TO_TLS_MODE_TLS_1_2_FULL);
TOSE_helper_tls_do_handshake(tls_ctx);
```

Parameters

- **`tls_ctx`** – [**inout**] context of the TLS session
- **`cipher_ctx`** – [**in**] private cipher context given to the callbacks
- **`setup_cipher_ctx`** – [**in**] callback used to setup the cipher context, call during the TLS handshake after cipher suite have been negotiated.

Return values

- **TO_OK** – the TLS session switched to HANDSHAKE_ONLY_MODE
- **TO_ERROR** – the TLS session didnt switch to HANDSHAKE_ONLY_MODE

TO_lib_ret_t **TOSE_helper_tls_do_handshake_step**(*TOSE_helper_tls_ctx_t* *tls_ctx)

Do TLS handshake step.

This function does one step of a TLS handshake. It encapsulates Secure Element payloads from optimized API in a TLS record, and sends it on the network through given function. It decapsulates TLS records received from the network and sends it to the Secure Element.

Parameters

- **tls_ctx** – [**in**] TLS context

Returns

TO_AGAIN if intermediate step suceed, TO_OK if last step succeed, else TO_ERROR

TO_lib_ret_t **TOSE_helper_tls_do_handshake**(*TOSE_helper_tls_ctx_t* *tls_ctx)

Do TLS handshake.

This function does all the steps of a TLS handshake except initialization and cleanup. It encapsulates the Secure Element payloads from optimized API in a TLS record, and sends it on the network through given function. It decapsulates TLS records received from the network and sends it to the Secure Element. This function uses TOSE_helper_tls_handshake_init() and TOSE_helper_tls_handshake_step().

Parameters

- **tls_ctx** – [**in**] TLS context

Returns

TO_OK if data has been sent successfully, else TO_ERROR

TO_lib_ret_t **TOSE_helper_tls_get_certificate_slot**(*TOSE_helper_tls_ctx_t* *tls_ctx, uint8_t *slot)

Get certificate slot used during TLS handshake.

This function must be called after handshake.

Parameters

- **tls_ctx** – [**in**] TLS context
- **slot** – [**out**] Certificate slot

Returns

TO_OK if slot has been retrieved successfully, else TO_ERROR

6.1.4.3 Messaging API

Once handshake is done, these 2 functions will allow to send and receive with TLS encryption using just negociated session.

TO_lib_ret_t **TOSE_helper_tls_send**(*TOSE_helper_tls_ctx_t* *tls_ctx, const uint8_t *msg, const uint32_t msg_len)

Send TLS encrypted data.

This function uses TLS handshake keys to encrypt and send a message on the network through given function.

Parameters

- **tls_ctx** – [**in**] TLS context
- **msg** – [**in**] Message
- **msg_len** – [**in**] Message length

Returns

TO_OK if message has been sent successfully, else TO_ERROR

TO_lib_ret_t **TOSE_helper_tls_receive**(*TOSE_helper_tls_ctx_t* *tls_ctx, uint8_t *msg, uint32_t max_msg_len, uint32_t *msg_len, int32_t timeout)

Receive TLS encrypted data.

This function uses given function to receive a message from the network and decrypts it with TLS handshake keys.

Parameters

- **tls_ctx** – [**in**] TLS context
- **msg** – [**out**] Message output buffer
- **max_msg_len** – [**in**] Message output buffer length
- **msg_len** – [**out**] Receive message length
- **timeout** – [**in**] Receive timeout in milliseconds (-1 for no timeout)

Returns

TO_OK if message has been received successfully, TO_TIMEOUT if given timeout has been exceeded, else TO_ERROR

TO_lib_ret_t **TOSE_helper_tls_recv**(*TOSE_helper_tls_ctx_t* *tls_ctx, uint8_t *msg, uint32_t max_msg_len, uint32_t *msg_len, int32_t timeout_ms)

receive plain text application data

More precisely, receives at most a plain text record of type application data, less if the receiving buffer is too short or if a record has been partially received previously.

Note: the parameter timeout_ms is given to the receive_func() callback provided to *TOSE_helper_tls_init_session()*. To ensure to not block more than timeout_ms, the recv() callback is called just once, thus the TO_AGAIN retval if partial data has been received.

Parameters

- **tls_ctx** – [**inout**] the TLS context
- **msg** – [**out**] received data
- **max_msg_len** – [**in**] maximum length of data writable in msg
- **msg_len** – [**out**] number of bytes read
- **timeout_ms** – [**in**] the maximum time to wait data in milliseconds

Return values

- **TO_OK** – application data received with success, msg is updated and *msg_len is greater than 0
- **TO_AGAIN** – some data has been received but not enough to receive a complete record, or it was not application data (see note above)
- **TO_TIMEOUT** – timeout elapsed before any bytes were received
- **TO_ERROR** – data cannot be received, the connection shall be (re-)initialized

6.2 Secure Element API

This API is used to setup I2C communication and then send basic commands to Secure Element.

```
#include "TO.h"
```

6.2.1 I2C communication

The following functions are used to deal with Secure Element I2C communication, they rely on the underlying I2C wrapper (see *I2C wrapper*).

6.2.1.1 I2C setup

Functions to manage connection with Secure Element.

TO_ret_t **TODRV_HSE_trp_config**(unsigned char i2c_addr, unsigned char misc_settings)

Configure hardware Secure Element transport.

See TO_data_config() for more details.

Parameters

- **i2c_addr** – I2C address to use
- **misc_settings** – Misc. settings byte. It has the following bit form (from MSB to LSB): RES, RES, RES, RES, RES, RES, RES, last byte NACKed. The *last byte NACKed* bit must be set to 1 if remote device NACKs last written byte.

Returns

TO_OK if configuration was successful.

6.2.1.2 Basic messaging

Functions to read and write data to Secure Element. They should be used only for debug purposes, as every Secure Element API is supported by the library (see *Secure Element functions*.

Warning: I2C must be initialized, see `TOSE_init()`.

TO_ret_t **`TODRV_HSE_trp_write`**(const void *data, unsigned int length)

Write data to Secure Element.

This function uses the underlying TO_data_write() wrapper function. Refer to its documentation for more details.

Parameters

- **`data`** – Buffer containing data to send
- **`length`** – Amount of data to send in bytes

Returns

- TO_OK if data has been written sucessfully
- TO_DEVICE_WRITE_ERROR: error writing data to Secure Element
- TO_ERROR if an internal error has occured

TO_ret_t **`TODRV_HSE_trp_read`**(void *data, unsigned int length)

Read data from Secure Element.

This function uses the underlying TO_data_read() wrapper function. Refer to its documentation for more details.

Parameters

- **`data`** – Buffer to store received data
- **`length`** – Amount of data to read in bytes

Returns

- TO_OK if data has been read sucessfully
- TO_DEVICE_READ_ERROR: error reading data from Secure Element
- TO_ERROR if an internal error has occured

TO_ret_t **`TODRV_HSE_trp_last_command_duration`**(unsigned int *duration)

Last command duration from Secure Element.

This function uses the underlying TO_data_last_command_duration() wrapper function. Refer to its documentation for more details.

This function should only be called after a successful command or a successful TO_read() call. If it is called after a failed command or a failed TO_read(), or after a TO_write() call, the result is unspecified and may be irrelevant.

Parameters

- **`duration`** – Pointer to store last command duration in microseconds

Returns

- TO_OK if data has been read sucessfully
- TO_ERROR if an internal error has occured

TO_ret_t **`TODRV_HSE_trp_last_command_stack_usage`**(unsigned int *stack_depth)

Last command stack depths usage from Secure Element.

Parameters

- **`duration`** – Pointer to store last command stack depths usage in bytes

  –

Returns

- TO_OK if data has been read sucessfully
- TO_ERROR if an internal error has occured

6.2.2 Secure Element functions

Warning: To use every of these functions, I2C must be initialized, see *`TOSE_init()`*.

The following API is directly based on Secure Element API.

6.2.2.1 NVM

Functions to use Secure Element secure data storage.

This zone is reserved user non-volatile memory, to store any data. No control is performed by the secure element when manipulating this data through read and write commands, except that the data area read/written must be within the NVM area.

The Secure Element does an XOR of user key with an internal key, and uses the resulting key to encrypt/decrypt the user data, using AES128-CBC. The same user key must be given to write and to read back the data, or irrelevant data will be retrieved.

Warning:

NVM is a flash memory with a limited capacity in term of erase/write cycle per sector. Please refer to electrical characteristics for details. Please note that calling 2 writes to 2 different bytes of the same sector result in 2 erase/write cycles.

TO_ret_t **`TOSE_write_nvm`**(*TOSE_ctx_t* *ctx, const uint16_t offset, const void *data, unsigned int length, const uint8_t key[*TO_AES_KEYSIZE*])

Write data to Secure Element NVM reserved zone.

Parameters

- **`ctx`** – **[in]** Pointer to the SE context
- **`offset`** – **[in]** Offset in NVM reserved zone to write data
- **`data`** – **[in]** Buffer containing data to write

- **length** – [**in**] Amount of data to write in bytes (512 bytes max.)
- **key** – [**in**] Key used to write data

Returns

TO_OK if data has been written successfully

- TO_DEVICE_WRITE_ERROR: error writing data to Secure Element
- TO_DEVICE_READ_ERROR: error reading data from Secure Element
- TO_ERROR if an internal error has occurred

TO_ret_t **TOSE_read_nvm**(*TOSE_ctx_t* *ctx, const uint16_t offset, void *data, unsigned int length, const uint8_t key[*TO_AES_KEYSIZE*])

Read data from Secure Element NVM reserved zone.

Parameters

- **ctx** – [**in**] Pointer to the SE context
- **offset** – [**in**] Offset in NVM reserved zone to read data
- **data** – [**out**] Buffer to store data
- **length** – [**in**] Amount of data to read in bytes (512 bytes max.)
- **key** – [**in**] Key used to read data

Returns

TO_OK if data has been read successfully

- TO_DEVICE_WRITE_ERROR: error writing data to Secure Element
- TO_DEVICE_READ_ERROR: error reading data from Secure Element
- TO_ERROR if an internal error has occurred

TO_ret_t **TOSE_get_nvm_size**(*TOSE_ctx_t* *ctx, uint16_t *size)

Get NVM reserved zone available size.

Parameters

- **ctx** – [**in**] Pointer to the SE context
- **size** – [**in**] NVM size

Returns

TO_OK if size has been retrieved successfully

- TO_DEVICE_WRITE_ERROR: error writing data to Secure Element
- TO_DEVICE_READ_ERROR: error reading data from Secure Element
- TO_ERROR if an internal error has occurred

6.2.2.2 Initialization

The following functions are used to initialize the library with given driver configuration.

TOSE_ctx_t ***TODRV_HSE_get_ctx**(void)

Get HSE context.

Returns

HSE context pointer

TOSE_ctx_t ***TODRV_SSE_get_ctx**(void)

Get SSE context.

Returns

SSE context pointer

TO_ret_t **TOSE_init**(*TOSE_ctx_t* *ctx)

Initialize Secure Element.

Parameters

- **ctx** – [**in**] Pointer to the SE context

TO_ret_t **TOSE_fini**(*TOSE_ctx_t* *ctx)

Uninitialize Secure Element.

Parameters

- **ctx** – [**in**] Pointer to the SE context

6.2.2.3 System

Misc. system functions.

TO_ret_t **TOSE_get_serial_number**(*TOSE_ctx_t* *ctx, uint8_t serial_number[*TO_SN_SIZE*])

Returns the unique Secure Element serial number.

The Serial Number is encoded on 8 bytes :

- The first 3 bytes identify the application ID.
- The last 5 bytes are the chip ID. Each Secure Element has an unique serial number.

Parameters

- **ctx** – [**in**] Pointer to the SE context
- **serial_number** – [**out**] Secure Element serial number

TO_ret_t **TOSE_get_hardware_serial_number**(*TOSE_ctx_t* *ctx, uint8_t hardware_serial_number[*TO_HW_SN_SIZE*])

Returns the hardware serial number.

Parameters

- **ctx** – [**in**] Pointer to the SE context
- **hardware_serial_number** – [**out**] Hardware serial number

TO_ret_t **TOSE_get_product_number**(*TOSE_ctx_t* *ctx, uint8_t product_number[*TO_PN_SIZE*])

Returns the Secure Element product number.

Product Number is a text string encoded on 12 bytes, e.g: TOSF-IS1-001

Parameters

- **ctx** – **[in]** Pointer to the SE context
- **product_number** – **[out]** Secure Element product number

TO_ret_t **TOSE_get_hardware_version**(*TOSE_ctx_t* *ctx, uint8_t hardware_version[*TO_HW_VERSION_SIZE*])

Returns the Secure Element hardware version.

Hardware version is encoded on 2 bytes. Available values are:

- 00 00: Software
- 00 01: SCO136i

Parameters

- **ctx** – **[in]** Pointer to the SE context
- **hardware_version** – **[out]** Secure Element hardware version

TO_ret_t **TOSE_get_software_version**(*TOSE_ctx_t* *ctx, uint8_t *major, uint8_t *minor, uint8_t *revision)

Returns the Secure Element software version.

Parameters

- **ctx** – **[in]** Pointer to the SE context
- **major** – **[out]** Major number. When this byte changes, API changes have occurred, incompatibility issues may be met, depending on your application.
- **minor** – **[out]** Minor number. This byte is incremented when changes happen without breaking the API.
- **revision** – **[out]** Revision number. This byte is incremented on each new build (when released).

TO_ret_t **TOSE_get_product_id**(*TOSE_ctx_t* *ctx, uint8_t product_id[*TO_PRODUCT_ID_SIZE*])

Returns the Secure Element product identifier.

The product identifier is a text string, encoded on maximum 15 ASCII bytes. It identifies the personalization profile.

Parameters

- **ctx** – **[in]** Pointer to the SE context
- **product_id** – **[out]** Secure Element product identifier

TO_ret_t **TOSE_get_random**(*TOSE_ctx_t* *ctx, const uint16_t random_length, uint8_t *random)

Returns a random number of the given length.

Request a random number to Secure Element random number generator.

Parameters

- **ctx** – [**in**] Pointer to the SE context
- **random_length** – [**in**] Requested random length
- **random** – [**out**] Returned random number

TO_ret_t **TOSE_access_dummy_data**(*TOSE_ctx_t* *ctx, const uint8_t write_data, uint32_t *dummy_data)

Returns the dummy data.

Parameters

- **ctx** – [**in**] Pointer to the SE context
- **write_data** – [**in**] Indicates whether we have to write (1) it or only read (0) it.
- **dummy_data** – [**out**] Pointer to the data to be read/written

6.2.2.4 Hashes

Hashing functions.

TO_ret_t **TOSE_sha256**(*TOSE_ctx_t* *ctx, const uint8_t *data, const uint16_t data_length, uint8_t sha256[*TO_SHA256_HASHSIZE*])

SHA256 computation.

Compute SHA256 hash on the given data.

Parameters

- **ctx** – [**in**] Pointer to the SE context
- **data** – [**in**] Data to compute SHA256 on
- **data_length** – [**in**] Data length, max. 512 bytes
- **sha256** – [**out**] returned computed SHA256

Returns

- TORSP_SUCCESS on success
- TORSP_INTERNAL_ERROR if a fault has been detected
- TORSP_INVALID_LEN if the data_length is too large (> 512)
- TO_DEVICE_WRITE_ERROR: error writing data to Secure Element
- TO_DEVICE_READ_ERROR: error reading data from Secure Element
- TO_INVALID_RESPONSE_LENGTH: unexpected response length from device
- TO_MEMORY_ERROR: internal I/O buffer overflow
- TO_ERROR: generic error

TO_ret_t **TOSE_sha256_init**(*TOSE_ctx_t* *ctx)

Compute SHA256 on more than 512 bytes of data.

This function must be followed by calls to *TOSE_sha256_update()* and *TOSE_sha256_final()*.

Parameters

- **ctx** – **[in]** Pointer to the SE context

Returns

- TORSP_SUCCESS on success
- TORSP_INTERNAL_ERROR if a fault has been detected
- TO_DEVICE_WRITE_ERROR: error writing data to Secure Element
- TO_DEVICE_READ_ERROR: error reading data from Secure Element
- TO_INVALID_RESPONSE_LENGTH: unexpected response length from device
- TO_MEMORY_ERROR: internal I/O buffer overflow
- TO_ERROR: generic error

TO_ret_t **TOSE_sha256_update**(*TOSE_ctx_t* *ctx, const uint8_t *data, const uint16_t length)

Update SHA256 computation with new data.

This function can be called several times to provide data to compute SHA256 on, and must be called after *TOSE_sha256_init()*.

This command is used to transmit data. It can be called several times, typically splitting the data into several blocks of 512 bytes.

Parameters

- **ctx** – **[in]** Pointer to the SE context
- **data** – **[in]** Data to compute SHA256 on
- **length** – **[in]** Data length, max. 512 bytes

Returns

- TORSP_SUCCESS on success
- TORSP_INTERNAL_ERROR if a fault has been detected
- TORSP_INVALID_LEN if the data_length is too large (> 512)
- TORSP_COND_OF_USE_NOT_SATISFIED if not called after *TOSE_sha256_init()* or *TOSE_sha256_update()*
- TO_DEVICE_WRITE_ERROR: error writing data to Secure Element
- TO_DEVICE_READ_ERROR: error reading data from Secure Element
- TO_INVALID_RESPONSE_LENGTH: unexpected response length from device
- TO_MEMORY_ERROR: internal I/O buffer overflow
- TO_ERROR: generic error

TO_ret_t **TOSE_sha256_final**(*TOSE_ctx_t* *ctx, uint8_t sha256[*TO_SHA256_HASHSIZE*])

Returns the SHA256 hash of the data previously given.

This function must be called after *TOSE_sha256_init()* and *TOSE_sha256_update()*.

This command finalizes the process and returns the SHA256 hash of the given data. This command handles the padding computation.

Parameters

- **ctx** – [**in**] Pointer to the SE context
- **sha256** – [**out**] returned computed SHA256

Returns

- TORSP_SUCCESS on success
- TORSP_INTERNAL_ERROR if a fault has been detected
- TORSP_COND_OF_USE_NOT_SATISFIED: if not called after *TOSE_sha256_update()*
- TO_DEVICE_WRITE_ERROR: error writing data to Secure Element
- TO_DEVICE_READ_ERROR: error reading data from Secure Element
- TO_INVALID_RESPONSE_LENGTH: unexpected response length from device
- TO_MEMORY_ERROR: internal I/O buffer overflow
- TO_ERROR: generic error

6.2.2.5 Authentication

Certificates management and signature functions.

Secure Element certificate index is starting from 0 (if the AVNET TO136 version supports several certificates).

For details on Secure Element certificate formats see Secure Element Datasheet.

TO_ret_t **TOSE_sign**(*TOSE_ctx_t* *ctx, const uint8_t key_index, const uint8_t *challenge, const uint16_t challenge_length, uint8_t signature[*TO_SIGNATURE_SIZE*])

Returns the Elliptic Curve Digital Signature of the given data.

Note that calling this function is equivalent to calling `TOSE_sha256()` followed by `TOSE_sign_hash()`.

Signature Size is twice the size of the ECC key in bytes. With a 256 bits key, signature is 64 bytes.

Parameters

- **ctx** – [**in**] Pointer to the SE contex
- **key_index** – [**in**] Key index to use for signature
- **challenge** – [**in**] Challenge to be signed
- **challenge_length** – [**in**] Challenge length (maximum 512)
- **signature** – [**out**] Returned challenge signature (64 bytes)

Returns

- TORSP_SUCCESS on success
- TORSP_ARG_OUT_OF_RANGE: invalid key index
- TO_DEVICE_WRITE_ERROR: error writing data to Secure Element
- TO_DEVICE_READ_ERROR: error reading data from Secure Element
- TO_INVALID_RESPONSE_LENGTH: unexpected response length from device

- TO_MEMORY_ERROR: internal I/O buffer overflow
- TO_ERROR: generic error

TO_ret_t **TOSE_verify**(*TOSE_ctx_t* *ctx, const uint8_t key_index, const uint8_t *data, const uint16_t data_length, const uint8_t signature[*TO_SIGNATURE_SIZE*])

Verifies the given Elliptic Curve Digital Signature of the given data.

The public key used for the signature verification must be previously provided using the TOSE_set_remote_public_key() call.

Parameters

- **ctx** – [**in**] Pointer to the SE context
- **key_index** – [**in**] Remote Public Key index to use for verification
- **data** – [**in**] Data to verify signature on
- **data_length** – [**in**] Data length (maximum 512)
- **signature** – [**in**] Expected data signature (64 bytes)

Returns

- TORSP_SUCCESS on success
- TORSP_ARG_OUT_OF_RANGE: invalid key index
- TORSP_BAD_SIGNATURE: invalid signature
- TO_DEVICE_WRITE_ERROR: error writing data to Secure Element
- TO_DEVICE_READ_ERROR: error reading data from Secure Element
- TO_INVALID_RESPONSE_LENGTH: unexpected response length from device
- TO_MEMORY_ERROR: internal I/O buffer overflow
- TO_ERROR: generic error

TO_ret_t **TOSE_sign_hash**(*TOSE_ctx_t* *ctx, const uint8_t key_index, const uint8_t hash[*TO_HASH_SIZE*], uint8_t signature[*TO_SIGNATURE_SIZE*])

Returns the Elliptic Curve Digital Signature of the given hash.

Signature Size is twice the size of the ECC key in bytes. With a 256 bits key, signature is 64 bytes.

Parameters

- **ctx** – [**in**] Pointer to the SE context
- **key_index** – [**in**] Key index to use for signature
- **hash** – [**in**] Hash to be signed
- **signature** – [**out**] Returned hash signature

Returns

- TORSP_SUCCESS on success
- TORSP_ARG_OUT_OF_RANGE: invalid key index
- TO_DEVICE_WRITE_ERROR: error writing data to Secure Element
- TO_DEVICE_READ_ERROR: error reading data from Secure Element

- TO_INVALID_RESPONSE_LENGTH: unexpected response length from device
- TO_MEMORY_ERROR: internal I/O buffer overflow
- TO_ERROR: generic error

TO_ret_t **`TOSE_verify_hash_signature`**(*TOSE_ctx_t* *ctx, const uint8_t key_index, const uint8_t hash[*TO_HASH_SIZE*], const uint8_t signature[*TO_SIGNATURE_SIZE*])

Verifies the given Elliptic Curve Digital Signature of the data that generates the given hash.

The public key used for the signature verification must be previously provided using the TOSE_set_remote_public_key() call.

Parameters

- **`ctx`** – **[in]** Pointer to the SE context
- **`key_index`** – **[in]** Remote Public Key index to use for verification
- **`hash`** – **[in]** Hash to verify signature on (32 bytes)
- **`signature`** – **[in]** Expected hash signature (64 bytes)

Returns

- TORSP_SUCCESS on success
- TORSP_ARG_OUT_OF_RANGE: invalid key index
- TORSP_BAD_SIGNATURE: invalid signature
- TO_DEVICE_WRITE_ERROR: error writing data to Secure Element
- TO_DEVICE_READ_ERROR: error reading data from Secure Element
- TO_INVALID_RESPONSE_LENGTH: unexpected response length from device
- TO_MEMORY_ERROR: internal I/O buffer overflow
- TO_ERROR: generic error

TO_ret_t **`TOSE_get_certificate_subject_cn`**(*TOSE_ctx_t* *ctx, const uint8_t certificate_index, char subject_cn[*TO_CERT_SUBJECT_CN_MAXSIZE* + 1])

Returns subject common name of one of the Secure Element certificates.

Request a certificate subject common name to Secure Element according to the given index.

Parameters

- **`ctx`** – **[in]** Pointer to the SE context
- **`certificate_index`** – **[in]** Requested certificate index
- **`subject_cn`** – **[out]** Returned certificate subject common name null terminated string

Returns

- TORSP_SUCCESS on success
- TORSP_NOT_AVAILABLE: certificate Format not supported
- TORSP_ARG_OUT_OF_RANGE: invalid Certificate Number

- TO_DEVICE_WRITE_ERROR: error writing data to Secure Element
- TO_DEVICE_READ_ERROR: error reading data from Secure Element
- TO_INVALID_RESPONSE_LENGTH: unexpected response length from device
- TO_MEMORY_ERROR: internal I/O buffer overflow
- TO_ERROR: generic error

TO_ret_t **`TOSE_set_certificate_signing_request_dn`**(*TOSE_ctx_t* *ctx, const uint8_t certificate_index, const uint8_t csr_dn[*TO_CERT_DN_MAXSIZE*], const uint16_t csr_dn_len)

Set CSR distinguished name.

Set certificate distinguished name which will be used in next CSR.

openssl can be used to generate a fake CSR and extract the Distinguished Name sequence in DER format, like:

- openssl ecparam -out acme.key -name prime256v1 -genkey
- openssl req -new -key acme.key -out acme.csr -subj /CN=*.ACME.com/O=ACME/OU=Security Services
- openssl asn1parse -in acme.csr Note the number of the first SEQUENCE with depth=2; in example above, this is item number 9
- openssl asn1parse -in acme.csr -strparse 9 -out extract_acme_DN.der and the file extract_acme_DN.der contains the Distinguished Name in DER format, that can be used as parameter to *TOSE_set_certificate_signing_request_dn()* Double-check that Distinguished Name size (check *extract_acme_DN.der* file size on the disk) does not exceed TO_CERT_DN_MAXSIZE; else this will be rejected by libTO.

Parameters

- **`ctx`** – **[in]** Pointer to the SE context
- **`certificate_index`** – **[in]** Certificate index
- **`csr_dn`** – **[in]** CSR distinguished name (without main sequence tag & length)
- **`csr_dn_len`** – **[in]** CSR distinguished name length

Returns

- TORSP_SUCCESS on success
- TORSP_ARG_OUT_OF_RANGE: invalid Certificate Number
- TORSP_INVALID_LEN: invalid Distinguished Name length (> TO_CERT_DN_MAXSIZE)
- TO_DEVICE_WRITE_ERROR: error writing data to Secure Element
- TO_DEVICE_READ_ERROR: error reading data from Secure Element
- TO_MEMORY_ERROR: internal I/O buffer overflow
- TO_ERROR: generic error

TO_ret_t **TOSE_get_certificate_signing_request**(*TOSE_ctx_t* *ctx, const uint8_t certificate_index, uint8_t *csr, uint16_t *size)

Get new certificate signing request.

Request a x509 DER formated certificate signing request according to the given index. CSR distinguished name can be set with *TOSE_set_certificate_signing_request_dn()*, otherwise existing certificate DN will be used (if any).

Parameters

- **ctx** – [**in**] Pointer to the SE context
- **certificate_index** – [**in**] Certificate index to renew
- **csr** – [**out**] Returned CSR data
- **size** – [**out**] Returned CSR real size

Returns

- TORSP_SUCCESS on success
- TORSP_ARG_OUT_OF_RANGE: invalid Certificate Number
- TO_DEVICE_WRITE_ERROR: error writing data to Secure Element
- TO_DEVICE_READ_ERROR: error reading data from Secure Element
- TO_INVALID_RESPONSE_LENGTH: unexpected response length from device
- TO_MEMORY_ERROR: internal I/O buffer overflow
- TO_ERROR: generic error

TO_ret_t **TOSE_set_certificate_x509**(*TOSE_ctx_t* *ctx, const uint8_t certificate_index, const uint8_t *certificate, const uint16_t size)

Set new certificate from previously generated CSR.

Set a x509 DER formated certificate according to the given index. The new certificate must be signed by a CA trusted by the Secure Element.

Parameters

- **certificate_index** – [**in**] Requested certificate index
- **ctx** – [**in**] Pointer to the SE context
- **certificate** – [**in**] New certificate data (x509 DER formated)
- **size** – [**in**] New certificate size

Returns

- TORSP_SUCCESS on success
- TORSP_NOT_AVAILABLE: certificate Format not supported
- TORSP_ARG_OUT_OF_RANGE: invalid Certificate Number
- TO_DEVICE_WRITE_ERROR: error writing data to Secure Element
- TO_DEVICE_READ_ERROR: error reading data from Secure Element
- TO_INVALID_RESPONSE_LENGTH: unexpected response length from device

- TO_MEMORY_ERROR: internal I/O buffer overflow
- TO_ERROR: generic error

TO_ret_t **`TOSE_set_certificate_x509_init(`***TOSE_ctx_t* *ctx, const uint8_t certificate_index)

Initialize to set new certificate from previously generated CSR.

See TOSE_set_certificate_x509

Parameters

- **`ctx`** – **[in]** Pointer to the SE context
- **`certificate_index`** – **[in]** Requested certificate index

Returns

- TORSP_SUCCESS on success
- TORSP_NOT_AVAILABLE: certificate Format not supported
- TORSP_ARG_OUT_OF_RANGE: invalid Certificate Number
- TO_DEVICE_WRITE_ERROR: error writing data to Secure Element
- TO_DEVICE_READ_ERROR: error reading data from Secure Element
- TO_INVALID_RESPONSE_LENGTH: unexpected response length from device
- TO_MEMORY_ERROR: internal I/O buffer overflow
- TO_ERROR: generic error

TO_ret_t **`TOSE_set_certificate_x509_update(`***TOSE_ctx_t* *ctx, const uint8_t *certificate, const uint16_t size)

Update to set new certificate from previously generated CSR.

See TOSE_set_certificate_x509

Parameters

- **`ctx`** – **[in]** Pointer to the SE context
- **`certificate`** – **[in]** New certificate partial data (from x509 DER formated)
- **`size`** – **[in]** New certificate partial data size

Returns

- TORSP_SUCCESS on success
- TORSP_NOT_AVAILABLE: certificate Format not supported
- TORSP_ARG_OUT_OF_RANGE: invalid Certificate Number
- TO_DEVICE_WRITE_ERROR: error writing data to Secure Element
- TO_DEVICE_READ_ERROR: error reading data from Secure Element
- TO_INVALID_RESPONSE_LENGTH: unexpected response length from device
- TO_MEMORY_ERROR: internal I/O buffer overflow
- TO_ERROR: generic error

TO_ret_t **TOSE_set_certificate_x509_final**(*TOSE_ctx_t* *ctx)

Finalize to set new certificate from previously generated CSR.

See TOSE_set_certificate_x509

Parameters

- **ctx** – **[in]** Pointer to the SE context

Returns

- TORSP_SUCCESS on success
- TORSP_NOT_AVAILABLE: certificate Format not supported
- TORSP_ARG_OUT_OF_RANGE: invalid Certificate Number
- TO_DEVICE_WRITE_ERROR: error writing data to Secure Element
- TO_DEVICE_READ_ERROR: error reading data from Secure Element
- TO_INVALID_RESPONSE_LENGTH: unexpected response length from device
- TO_MEMORY_ERROR: internal I/O buffer overflow
- TO_ERROR: generic error

TO_ret_t **TOSE_get_certificate**(*TOSE_ctx_t* *ctx, const uint8_t certificate_index, const *TO_certificate_format_t* certificate_format, uint8_t *certificate)

Returns one of the Secure Element certificates.

Request a certificate to Secure Element according to the given index and format.

Parameters

- **ctx** – **[in]** Pointer to the SE context
- **certificate_index** – **[in]** Requested certificate index
- **certificate** – **[out]** Pointer to a buffer, which will receive the Certificate under the desired format.
- **size** – **[out]** Size of the transfered data, is returned in output.

Returns

- TORSP_SUCCESS on success
- TORSP_NOT_AVAILABLE: certificate Format not supported
- TORSP_ARG_OUT_OF_RANGE: invalid Certificate Number
- TO_DEVICE_WRITE_ERROR: error writing data to Secure Element
- TO_DEVICE_READ_ERROR: error reading data from Secure Element
- TO_INVALID_RESPONSE_LENGTH: unexpected response length from device
- TO_MEMORY_ERROR: internal I/O buffer overflow
- TO_ERROR: generic error

TO_ret_t **`TOSE_get_certificate_x509`**(*TOSE_ctx_t* *ctx, const uint8_t certificate_index, uint8_t *certificate, uint16_t *size)

Returns one of the certificates, x509 DER formatted.

Request a x509 DER formated certificate according to the given index.

Parameters

- **`ctx`** – [**in**] Pointer to the SE context
- **`certificate_index`** – [**in**] Requested certificate index
- **`certificate`** – [**out**] Returned certificate data
- **`size`** – [**out**] Size of the transfered certificate, is returned in output.

Returns

- TORSP_SUCCESS on success
- TORSP_ARG_OUT_OF_RANGE: invalid Certificate Number
- TO_DEVICE_WRITE_ERROR: error writing data to Secure Element
- TO_DEVICE_READ_ERROR: error reading data from Secure Element
- TO_INVALID_RESPONSE_LENGTH: unexpected response length from device
- TO_MEMORY_ERROR: internal I/O buffer overflow
- TO_ERROR: generic error

TO_ret_t **`TOSE_verify_ca_certificate_and_store`**(*TOSE_ctx_t* *ctx, const uint8_t ca_key_index, const uint8_t subca_key_index, const uint8_t *certificate, const uint16_t certificate_len)

Requests to verify signature of the given subCA certificate; if verification succeeds, this certificate is stored into Secure Element CA slot.

Note: the only supported certificate format for this command is DER X509.

Parameters

- **`ctx`** – [**in**] Pointer to the SE context
- **`ca_key_index`** – [**in**] index of the CA slot used to verify subCA
- **`subca_key_index`** – [**in**] subCA index to store certificate
- **`certificate`** – [**in**] Certificate to be verified and stored
- **`certificate_len`** – [**in**] Certificate length

Returns

- TORSP_SUCCESS on success
- TORSP_ARG_OUT_OF_RANGE: invalid CA Key index
- TO_DEVICE_WRITE_ERROR: error writing data to Secure Element
- TO_DEVICE_READ_ERROR: error reading data from Secure Element
- TO_INVALID_RESPONSE_LENGTH: unexpected response length from device
- TO_MEMORY_ERROR: internal I/O buffer overflow

- TO_ERROR: generic error

TO_ret_t **TOSE_get_challenge_and_store**(*TOSE_ctx_t* *ctx, uint8_t challenge[*TO_CHALLENGE_SIZE*])

Returns a challenge (random number of fixed length) and store it into Secure Element memory.

This command must be called before TOSE_verify_challenge_signature().

Parameters

- **ctx** – [**in**] Pointer to the SE context
- **challenge** – [**out**] Returned challenge

Returns

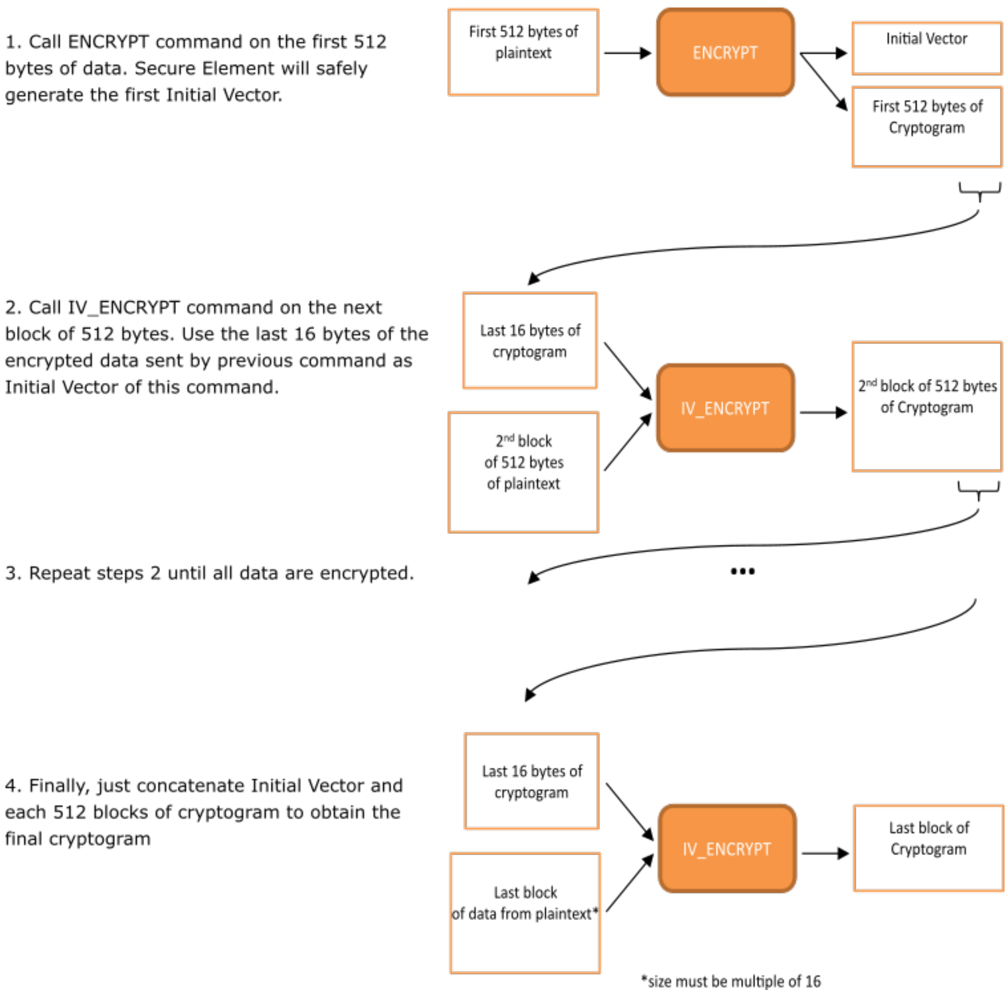
- TORSP_SUCCESS on success
- TO_DEVICE_WRITE_ERROR: error writing data to Secure Element
- TO_DEVICE_READ_ERROR: error reading data from Secure Element
- TO_INVALID_RESPONSE_LENGTH: unexpected response length from device
- TO_MEMORY_ERROR: internal I/O buffer overflow
- TO_ERROR: generic error

6.2.2.6 Encryption

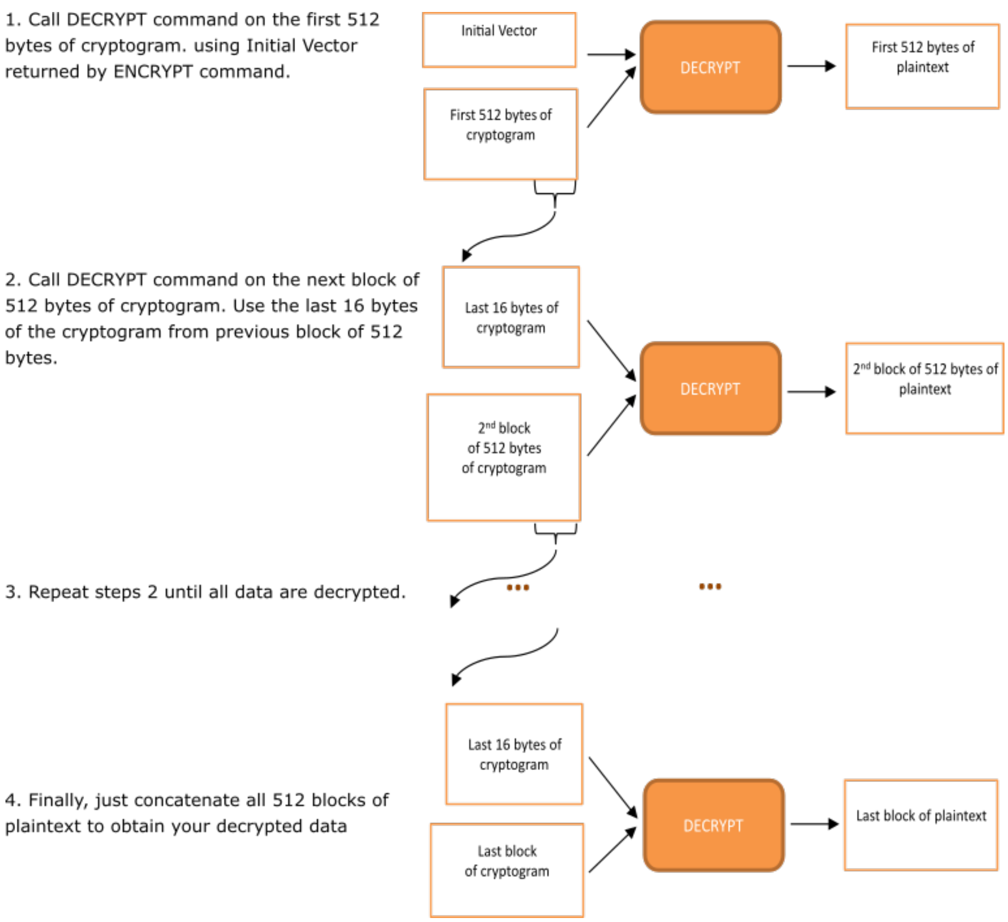
Ciphered messaging functions.

The best way to encrypt more than the maximum size of data supported by TOSE_<xxx>_encrypt() command is to manually chain encryption.

The following steps allows to encrypt any size of data using manual blocks chaining with the maximum security level:



The following steps allows to decrypt any size of data using manual blocks chaining:



Here is the API list:

TO_ret_t **TOSE_aes128cbc_encrypt**(*TOSE_ctx_t* *ctx, const uint8_t key_index, const uint8_t *data, const uint16_t data_length, uint8_t initial_vector[*TO_INITIALVECTOR_SIZE*], uint8_t *cryptogram)

Encrypts data using AES128 algorithm in CBC mode of operation.

As padding is not handled by the Secure Element, you must ensure that data length is a multiple of 16 and is not greater than maximum length value (512 bytes). Initial vector is generated by the Secure Element.

Parameters

- `ctx` – **[in]** Pointer to the SE context
- `key_index` – **[in]** Index of the key to use for data encryption, starting from 0
- `data` – **[in]** Data to encrypt
- `data_length` – **[in]** Length of the data to encrypt
- `initial_vector` – **[out]** Initial vector
- `cryptogram` – **[out]** Cryptogram, sent back by the Secure Element

Returns

- TORSP_SUCCESS on success
- TORSP_ARG_OUT_OF_RANGE: invalid key index
- TORSP_INVALID_LEN: Wrong length
- TORSP_ARG_OUT_OF_RANGE: invalid key index
- TO_DEVICE_WRITE_ERROR: error writing data to Secure Element
- TO_DEVICE_READ_ERROR: error reading data from Secure Element
- TO_ERROR: generic error

TO_ret_t **`TOSE_aes128cbc_iv_encrypt`**(*TOSE_ctx_t* *ctx, const uint8_t key_index, const uint8_t initial_vector[*TO_INITIALVECTOR_SIZE*], const uint8_t *data, const uint16_t data_length, uint8_t *cryptogram)

Similar to encrypt() except that Initial Vector is given by user.

It can be used to encrypt more than data size limit (512 bytes) by manually chaining blocs of 512 bytes (see Secure Element Datasheet - Encrypt or

decrypt more than 512 bytes chapter for more details).

> **Warning:** Using iv_encrypt() with a predictable Initial Vector can have security impact. Please let Secure Element generate Initial Vector by using encrypt() command when possible.

Parameters

- **`ctx`** – [**in**] Pointer to the SE context
- **`key_index`** – [**in**] Index of the key to use for data encryption, starting from 0
- **`initial_vector`** – [**in**] Random data (16 bytes)
- **`data`** – [**in**] Data to encrypt
- **`data_length`** – [**in**]
- **`cryptogram`** – [**out**] Returned encrypted data

Returns

- TORSP_SUCCESS on success
- TORSP_ARG_OUT_OF_RANGE: invalid key index
- TO_DEVICE_WRITE_ERROR: error writing data to Secure Element
- TO_DEVICE_READ_ERROR: error reading data from Secure Element
- TO_INVALID_RESPONSE_LENGTH: unexpected response length from device
- TO_MEMORY_ERROR: internal I/O buffer overflow
- TO_ERROR: generic error

TO_ret_t **`TOSE_aes128cbc_decrypt`**(*TOSE_ctx_t* *ctx, const uint8_t key_index, const uint8_t initial_vector[*TO_INITIALVECTOR_SIZE*], const uint8_t *cryptogram, const uint16_t cryptogram_length, uint8_t *data)

Decrypts data using AES128 algorithm in CBC mode of operation.

Requires the initial vector provided by the encryption function.

Padding is not handled by Secure Element firmware. It gives the possibility to avoid the case of a full padding block sometimes required by padding functions.

Parameters

- **`ctx`** – **[in]** Pointer to the SE context
- **`key_index`** – **[in]** Index of the key to use for data decryption, starting from 0
- **`initial_vector`** – **[in]** Initial vector
- **`cryptogram`** – **[in]** Data to decrypt
- **`cryptogram_length`** – **[in]** Cryptogram length, less or equal to 512 bytes
- **`data`** – **[out]** returned decrypted data

Returns

- TORSP_SUCCESS on success
- TORSP_ARG_OUT_OF_RANGE: invalid key index
- TO_DEVICE_WRITE_ERROR: error writing data to Secure Element
- TO_DEVICE_READ_ERROR: error reading data from Secure Element
- TO_INVALID_RESPONSE_LENGTH: unexpected response length from device
- TO_MEMORY_ERROR: internal I/O buffer overflow
- TO_ERROR: generic error

TO_ret_t **`TOSE_aes128gcm_encrypt`**(*TOSE_ctx_t* *ctx, const uint8_t key_index, const uint8_t *data, const uint16_t data_length, const uint8_t *aad, const uint16_t aad_length, uint8_t initial_vector[*TO_AESGCM_INITIALVECTOR_SIZE*], uint8_t *cryptogram, uint8_t tag[*TO_AESGCM_TAG_SIZE*])

Encrypts data using AES128 algorithm in GCM mode of operation.

Additional authentication data length and data length can not exceed driver IO buffer size (if applicable). Initial vector is generated by the Secure Element.

Parameters

- **`ctx`** – **[in]** Pointer to the SE context
- **`key_index`** – **[in]** Index of the key to use for data encryption, starting from 0
- **`data`** – **[in]** Data to encrypt
- **`data_length`** – **[in]** Length of the data to encrypt
- **`aad`** – **[in]** Additional authentication data
- **`aad_length`** – **[in]** Length of the additional authentication data
- **`initial_vector`** – **[out]** Initial vector
- **`cryptogram`** – **[out]** Cryptogram

- **tag** – [**out**] Authentication tag

Returns

- TORSP_SUCCESS on success
- TORSP_ARG_OUT_OF_RANGE: invalid key index
- TORSP_INVALID_LEN: Wrong length
- TORSP_ARG_OUT_OF_RANGE: invalid key index
- TO_DEVICE_WRITE_ERROR: error writing data to Secure Element
- TO_DEVICE_READ_ERROR: error reading data from Secure Element
- TO_ERROR: generic error

TO_ret_t **TOSE_aes128gcm_decrypt**(*TOSE_ctx_t* *ctx, const uint8_t key_index, const uint8_t initial_vector[*TO_AESGCM_INITIALVECTOR_SIZE*], const uint8_t *aad, const uint16_t aad_length, const uint8_t *cryptogram, const uint16_t cryptogram_length, const uint8_t tag[*TO_AESGCM_TAG_SIZE*], uint8_t *data)

Decrypts data using AES128 algorithm in GCM mode of operation.

Requires the initial vector provided by the encryption function. Additional authentication data length and cryptogram length can not exceed driver IO buffer size (if applicable).

Parameters

- **ctx** – [**in**] Pointer to the SE context
- **key_index** – [**in**] Index of the key to use for data decryption, starting from 0
- **initial_vector** – [**in**] Initial vector
- **aad** – [**in**] Additional authentication data
- **aad_length** – [**in**] Length of the additional authentication data
- **cryptogram** – [**in**] Data to decrypt
- **cryptogram_length** – [**in**] Cryptogram length, less or equal to 512 bytes
- **tag** – [**in**] Authentication tag
- **data** – [**out**] returned decrypted data

Returns

- TORSP_SUCCESS on success
- TORSP_ARG_OUT_OF_RANGE: invalid key index
- TO_DEVICE_WRITE_ERROR: error writing data to Secure Element
- TO_DEVICE_READ_ERROR: error reading data from Secure Element
- TO_INVALID_RESPONSE_LENGTH: unexpected response length from device
- TO_MEMORY_ERROR: internal I/O buffer overflow
- TO_ERROR: generic error

TO_ret_t **TOSE_aes128ccm_encrypt**(*TOSE_ctx_t* *ctx, const uint8_t key_index, const uint8_t *data, const uint16_t data_length, const uint8_t *aad, const uint16_t aad_length, uint8_t nonce[*TO_AESCCM_NONCE_SIZE*], uint8_t *cryptogram, uint8_t tag[*TO_AESCCM_TAG_SIZE*])

Encrypts data using AES128 algorithm in CCM mode of operation.

Additional authentication data length and data length can not exceed driver IO buffer size (if applicable). Nonce is generated by the Secure Element.

Parameters

- **ctx** – **[in]** Pointer to the SE context
- **key_index** – **[in]** Index of the key to use for data encryption, starting from 0
- **data** – **[in]** Data to encrypt
- **data_length** – **[in]** Length of the data to encrypt
- **aad** – **[in]** Additional authentication data
- **aad_length** – **[in]** Length of the additional authentication data
- **nonce** – **[in]** Nonce
- **cryptogram** – **[out]** Cryptogram
- **tag** – **[out]** Authentication tag

Returns

- TORSP_SUCCESS on success
- TORSP_ARG_OUT_OF_RANGE: invalid key index
- TORSP_INVALID_LEN: Wrong length
- TORSP_ARG_OUT_OF_RANGE: invalid key index
- TO_DEVICE_WRITE_ERROR: error writing data to Secure Element
- TO_DEVICE_READ_ERROR: error reading data from Secure Element
- TO_ERROR: generic error

TO_ret_t **TOSE_aes128ccm_decrypt**(*TOSE_ctx_t* *ctx, const uint8_t key_index, const uint8_t nonce[*TO_AESCCM_NONCE_SIZE*], const uint8_t *aad, const uint16_t aad_length, const uint8_t *cryptogram, const uint16_t cryptogram_length, const uint8_t tag[*TO_AESCCM_TAG_SIZE*], uint8_t *data)

Decrypts data using AES128 algorithm in CCM mode of operation.

Requires the nonce provided by the encryption function. Additional authentication data length and cryptogram length can not exceed driver IO buffer size (if applicable).

Parameters

- **ctx** – **[in]** Pointer to the SE context
- **key_index** – **[in]** Index of the key to use for data decryption, starting from 0
- **nonce** – **[in]** Nonce
- **aad** – **[in]** Additional authentication data

- **aad_length** – **[in]** Length of the additional authentication data
- **cryptogram** – **[in]** Data to decrypt
- **cryptogram_length** – **[in]** Cryptogram length, less or equal to 512 bytes
- **tag** – **[in]** Authentication tag
- **data** – **[out]** returned decrypted data

Returns

- TORSP_SUCCESS on success
- TORSP_ARG_OUT_OF_RANGE: invalid key index
- TO_DEVICE_WRITE_ERROR: error writing data to Secure Element
- TO_DEVICE_READ_ERROR: error reading data from Secure Element
- TO_INVALID_RESPONSE_LENGTH: unexpected response length from device
- TO_MEMORY_ERROR: internal I/O buffer overflow
- TO_ERROR: generic error

TO_ret_t **TOSE_aes128ecb_encrypt**(*TOSE_ctx_t* *ctx, const uint8_t key_index, const uint8_t *data, const uint16_t data_length, uint8_t *cryptogram)

Encrypts data using AES128 algorithm in ECB mode of operation.

As padding is not handled by the Secure Element, you must ensure that data length is a multiple of 16 and is not greater than maximum length value (512 bytes).

Parameters

- **ctx** – **[in]** Pointer to the SE context
- **key_index** – **[in]** Index of the key to use for data encryption, starting from 0
- **data** – **[in]** Data to encrypt
- **data_length** – **[in]** Length of the data to encrypt
- **cryptogram** – **[out]** Cryptogram

Returns

- TORSP_SUCCESS on success
- TORSP_ARG_OUT_OF_RANGE: invalid key index
- TORSP_INVALID_LEN: Wrong length
- TORSP_ARG_OUT_OF_RANGE: invalid key index
- TO_DEVICE_WRITE_ERROR: error writing data to Secure Element
- TO_DEVICE_READ_ERROR: error reading data from Secure Element
- TO_ERROR: generic error

TO_ret_t **TOSE_aes128ecb_decrypt**(*TOSE_ctx_t* *ctx, const uint8_t key_index, const uint8_t *cryptogram, const uint16_t cryptogram_length, uint8_t *data)

Decrypts data using AES128 algorithm in ECB mode of operation.

Padding is not handled by Secure Element firmware. It gives the possibility to avoid the case of a full padding block sometime required by padding functions.

Parameters

- **ctx** – [**in**] Pointer to the SE context
- **key_index** – [**in**] Index of the key to use for data decryption, starting from 0
- **cryptogram** – [**in**] Data to decrypt
- **cryptogram_length** – [**in**] Cryptogram length, less or equal to 512 bytes
- **data** – [**out**] returned decrypted data

Returns

- TORSP_SUCCESS on success
- TORSP_ARG_OUT_OF_RANGE: invalid key index
- TO_DEVICE_WRITE_ERROR: error writing data to Secure Element
- TO_DEVICE_READ_ERROR: error reading data from Secure Element
- TO_INVALID_RESPONSE_LENGTH: unexpected response length from device
- TO_MEMORY_ERROR: internal I/O buffer overflow
- TO_ERROR: generic error

6.2.2.7 MAC

Message Authentication Code functions (HMAC and CMAC).

TO_ret_t **TOSE_compute_hmac**(*TOSE_ctx_t* *ctx, const uint8_t key_index, const uint8_t *data, const uint16_t data_length, uint8_t hmac_data[*TO_HMAC_SIZE*])

Computes a 256-bit HMAC tag based on SHA256 hash function.

If you need to compute HMAC on more than 512 bytes, please use the sequence *TOSE_compute_hmac_init()*, *TOSE_compute_hmac_update()*, , *TOSE_compute_hmac_final()*.

Parameters

- **ctx** – [**in**] Pointer to the SE context
- **key_index** – [**in**] Index of the key to use for HMAC calculation, starting from 0
- **data** – [**in**] Data to compute HMAC on
- **data_length** – [**in**]
- **hmac_data** – [**out**] Computed HMAC

Returns

- TORSP_SUCCESS on success
- TORSP_ARG_OUT_OF_RANGE: invalid key index
- TO_DEVICE_WRITE_ERROR: error writing data to Secure Element

- TO_DEVICE_READ_ERROR: error reading data from Secure Element
- TO_INVALID_RESPONSE_LENGTH: unexpected response length from device
- TO_MEMORY_ERROR: internal I/O buffer overflow
- TO_ERROR: generic error

TO_ret_t **`TOSE_compute_hmac_init`**(*TOSE_ctx_t* *ctx, uint8_t key_index)

Compute HMAC on more than 512 bytes of data.

This is the first command of the sequence *TOSE_compute_hmac_init()*, *TOSE_compute_hmac_update()*, , *TOSE_compute_hmac_final()*. It is used to Secure Element send Key_index.

Parameters

- **`ctx`** – **[in]** Pointer to the SE context
- **`key_index`** – **[in]** Index of the key to use for HMAC calculation, starting from 0

Returns

- TORSP_SUCCESS on success
- TORSP_ARG_OUT_OF_RANGE: invalid key index
- TO_DEVICE_WRITE_ERROR: error writing data to Secure Element
- TO_DEVICE_READ_ERROR: error reading data from Secure Element
- TO_INVALID_RESPONSE_LENGTH: unexpected response length from device
- TO_MEMORY_ERROR: internal I/O buffer overflow
- TO_ERROR: generic error

TO_ret_t **`TOSE_compute_hmac_update`**(*TOSE_ctx_t* *ctx, const uint8_t *data, uint16_t length)

Used to send data to compute HMAC on.

This command can be called several times, new data are added to the data previously sent.

Parameters

- **`ctx`** – **[in]** Pointer to the SE context
- **`data`** – **[in]** Data to compute HMAC on
- **`length`** – **[in]** Data length

Returns

- TORSP_SUCCESS on success
- TORSP_COND_OF_USE_NOT_SATISFIED: need to call *TOSE_compute_hmac_init()* first
- TO_DEVICE_WRITE_ERROR: error writing data to Secure Element
- TO_DEVICE_READ_ERROR: error reading data from Secure Element
- TO_INVALID_RESPONSE_LENGTH: unexpected response length from device
- TO_MEMORY_ERROR: internal I/O buffer overflow
- TO_ERROR: generic error

TO_ret_t **TOSE_compute_hmac_final**(*TOSE_ctx_t* *ctx, uint8_t hmac[*TO_HMAC_SIZE*])

Returns computed HMAC.

This is the last command of the sequence *TOSE_compute_hmac_init()*, *TOSE_compute_hmac_update()*, , *TOSE_compute_hmac_final()*.

Parameters

- **ctx** – [**in**] Pointer to the SE context
- **hmac** – [**out**] Returned computed HMAC

Returns

- TORSP_SUCCESS on success
- TORSP_COND_OF_USE_NOT_SATISFIED: need to call *TOSE_compute_hmac_init()* and *TOSE_compute_hmac_update()* first
- TO_DEVICE_WRITE_ERROR: error writing data to Secure Element
- TO_DEVICE_READ_ERROR: error reading data from Secure Element
- TO_INVALID_RESPONSE_LENGTH: unexpected response length from device
- TO_MEMORY_ERROR: internal I/O buffer overflow
- TO_ERROR: generic error

TO_ret_t **TOSE_verify_hmac**(*TOSE_ctx_t* *ctx, const uint8_t key_index, const uint8_t *data, const uint16_t data_length, const uint8_t hmac_data[*TO_HMAC_SIZE*])

Verifies if the HMAC tag is correct for the given data.

If you need to verify HMAC of more than 512 bytes, please use the combination of *TOSE_verify_hmac_init()*, *TOSE_verify_hmac_update()*, , *TOSE_verify_hmac_final()*

Parameters

- **ctx** – [**in**] Pointer to the SE context
- **key_index** – [**in**] Index of the key to use for HMAC calculation, starting from 0
- **data** – [**in**] Data to verify HMAC on
- **data_length** – [**in**]
- **hmac_data** – [**in**] expected HMAC value

Returns

- TORSP_SUCCESS on success
- TORSP_BAD_SIGNATURE: verification failed
- TORSP_ARG_OUT_OF_RANGE: invalid key index
- TO_DEVICE_WRITE_ERROR: error writing data to Secure Element
- TO_DEVICE_READ_ERROR: error reading data from Secure Element
- TO_INVALID_RESPONSE_LENGTH: unexpected response length from device
- TO_MEMORY_ERROR: internal I/O buffer overflow
- TO_ERROR: generic error

TO_ret_t **TOSE_verify_hmac_init**(*TOSE_ctx_t* *ctx, uint8_t key_index)

Verify HMAC on more than 512 bytes of data.

When you need to verify HMAC of more than 512 bytes you need to call this function first with the key index - as sent to verify_hmac(). Data will be sent with verify_hmac_update() and HMAC will be sent with verify_hmac_final().

Parameters

- **ctx** – [**in**] Pointer to the SE context
- **key_index** – [**in**] Index of the key to use for HMAC calculation, starting from 0

Returns

- TORSP_SUCCESS on success
- TORSP_ARG_OUT_OF_RANGE: invalid key index
- TO_DEVICE_WRITE_ERROR: error writing data to Secure Element
- TO_DEVICE_READ_ERROR: error reading data from Secure Element
- TO_INVALID_RESPONSE_LENGTH: unexpected response length from device
- TO_MEMORY_ERROR: internal I/O buffer overflow
- TO_ERROR: generic error

TO_ret_t **TOSE_verify_hmac_update**(*TOSE_ctx_t* *ctx, const uint8_t *data, uint16_t length)

Used to send data to verify HMAC on.

After calling *TOSE_verify_hmac_init()* to provide key index, you can call *TOSE_verify_hmac_update()* to send the data to verify HMAC on. This command can be called several times, and new data are added to the previous one for HMAC verification. Last command to use is *TOSE_verify_hmac_final()*.

Parameters

- **ctx** – [**in**] Pointer to the SE context
- **data** – [**in**] Data to verify HMAC on
- **length** – [**in**] Data length

Returns

- TORSP_SUCCESS on success
- TORSP_COND_OF_USE_NOT_SATISFIED: need to call VERIFY_HMAC_INIT first
- TO_DEVICE_WRITE_ERROR: error writing data to Secure Element
- TO_DEVICE_READ_ERROR: error reading data from Secure Element
- TO_INVALID_RESPONSE_LENGTH: unexpected response length from device
- TO_MEMORY_ERROR: internal I/O buffer overflow
- TO_ERROR: generic error

TO_ret_t **TOSE_verify_hmac_final**(*TOSE_ctx_t* *ctx, const uint8_t hmac[*TO_HMAC_SIZE*])

This command is used to send HMAC to verify.

Data was previously sent by the sequence *TOSE_verify_hmac_init()*, *TOSE_verify_hmac_update()*, , *TOSE_verify_hmac_final()*. This command succeeds if the HMAC is correct for the given data.

Parameters

- **ctx** – **[in]** Pointer to the SE context
- **hmac** – **[in]** HMAC to verify

Returns

- TORSP_SUCCESS on success
- TORSP_BAD_SIGNATURE: verification failed
- TORSP_COND_OF_USE_NOT_SATISFIED: *TOSE_verify_hmac_init()* or *TOSE_verify_hmac_update()* were not called before this command
- TO_DEVICE_WRITE_ERROR: error writing data to Secure Element
- TO_DEVICE_READ_ERROR: error reading data from Secure Element
- TO_INVALID_RESPONSE_LENGTH: unexpected response length from device
- TO_MEMORY_ERROR: internal I/O buffer overflow
- TO_ERROR: generic error

TO_ret_t **TOSE_compute_cmac**(*TOSE_ctx_t* *ctx, const uint8_t key_index, const uint8_t *data, const uint16_t data_length, uint8_t cmac_data[*TO_CMAC_SIZE*])

Compute CMAC.

Compute a 128-bit CMAC tag based on AES128 algorithm.

Parameters

- **ctx** – **[in]** Pointer to the SE context
- **key_index** – **[in]** Index of the key to use for CMAC calculation, starting from 0
- **data** – **[in]** Data to compute CMAC on
- **data_length** – **[in]** Length of the data which signature has to be verified
- **cmac_data** – **[out]** Returned computed CMAC

Returns

- TORSP_SUCCESS on success
- TORSP_ARG_OUT_OF_RANGE: invalid key index
- TO_DEVICE_WRITE_ERROR: error writing data to Secure Element
- TO_DEVICE_READ_ERROR: error reading data from Secure Element
- TO_INVALID_RESPONSE_LENGTH: unexpected response length from device
- TO_MEMORY_ERROR: internal I/O buffer overflow
- TO_ERROR: generic error

TO_ret_t **TOSE_verify_cmac**(*TOSE_ctx_t* *ctx, const uint8_t key_index, const uint8_t *data, const uint16_t data_length, const uint8_t cmac_data[*TO_CMAC_SIZE*])

Verify the CMAC signature of a given block of data.

Verify if the CMAC tag is correct for the given data.

Parameters

- **ctx** – [**in**] Pointer to the SE context
- **key_index** – [**in**] Index of the key to use to compute the CMAC tag, starting from 0
- **data** – [**in**] Data to verify CMAC on
- **data_length** – [**in**] Length of the data which signature has to be verified
- **cmac_data** – [**in**] expected CMAC

Returns

- TORSP_SUCCESS on success
- TORSP_BAD_SIGNATURE: verification failed
- TORSP_ARG_OUT_OF_RANGE: invalid key index
- TO_DEVICE_WRITE_ERROR: error writing data to Secure Element
- TO_DEVICE_READ_ERROR: error reading data from Secure Element
- TO_INVALID_RESPONSE_LENGTH: unexpected response length from device
- TO_MEMORY_ERROR: internal I/O buffer overflow
- TO_ERROR: generic error

6.2.2.8 Secure messaging CAPI

Secure messaging functions.

TO_ret_t **TOSE_secure_payload_init**(*TOSE_ctx_t* *ctx, const uint8_t key_index, const *TO_enc_alg_t* enc_alg, const *TO_mac_alg_t* mac_alg, const uint16_t data_len, uint8_t sequence[*TO_SEQUENCE_SIZE*], uint8_t *iv, uint16_t *iv_len)

Initializes transform of a message into a secured payload.

Initial vector is generated by the Secure Element and not included in the data length

Initial vector length is also given by *TO_PAYLOAD_IV_SIZE()*.

Parameters

- **ctx** – [**in**] Pointer to the SE context
- **key_index** – [**in**] Index of the keys to use for data encryption and MAC, starting from 0
- **enc_alg** – [**in**] Encryption algorithm to use
- **mac_alg** – [**in**] MAC algorithm to use
- **data_len** – [**in**] Full data length

- **sequence** – [**in**] Sequence counter to avoid replay attacks
- **iv** – [**out**] Initial vector
- **iv_len** – [**out**] Initial vector length

Returns

- TORSP_SUCCESS on success
- TORSP_ARG_OUT_OF_RANGE: invalid key index
- TORSP_NOT_AVAILABLE: algorithm not available
- TO_DEVICE_WRITE_ERROR: error writing data to Secure Element
- TO_DEVICE_READ_ERROR: error reading data from Secure Element
- TO_INVALID_RESPONSE_LENGTH: unexpected response length from device
- TO_MEMORY_ERROR: internal I/O buffer overflow
- TO_ERROR: generic error

TO_ret_t **TOSE_secure_payload_update**(*TOSE_ctx_t* *ctx, const uint8_t *data, const uint16_t data_len, uint8_t *cryptogram)

Updates transform of a message into a secured payload.

Parameters

- **ctx** – [**in**] Pointer to the SE context
- **data** – [**in**] Message part to be secured
- **data_len** – [**in**] Message part length (must be multiple of 16)
- **cryptogram** – [**out**] Message cryptogram (same length)

Returns

- TORSP_SUCCESS on success
- TORSP_ARG_OUT_OF_RANGE: invalid key index
- TORSP_NOT_AVAILABLE: algorithm not available
- TO_DEVICE_WRITE_ERROR: error writing data to Secure Element
- TO_DEVICE_READ_ERROR: error reading data from Secure Element
- TO_INVALID_RESPONSE_LENGTH: unexpected response length from device
- TO_MEMORY_ERROR: internal I/O buffer overflow
- TO_ERROR: generic error

TO_ret_t **TOSE_secure_payload_final**(*TOSE_ctx_t* *ctx, const uint8_t *data, const uint16_t data_len, uint8_t *cryptogram, uint16_t *cryptogram_len)

Finaliazes transform of a message into a secured payload.

The MAC tag is calculated on clear data, including sequence counter. This function will add padding after MAC if clear data size is not aligned. Padding scheme is PKCS7 with extra padding length byte (TLS like).

Parameters

- **ctx** – [**in**] Pointer to the SE context
- **data** – [**in**] Final message part to be secured
- **data_len** – [**in**] Final message part length (0 <= data_len < 16)
- **cryptogram** – [**out**] Final message cryptogram (containing MAC and padding)
- **cryptogram_len** – [**out**] Final message cryptogram length

Returns

- TORSP_SUCCESS on success
- TORSP_ARG_OUT_OF_RANGE: invalid key index
- TORSP_NOT_AVAILABLE: algorithm not available
- TO_DEVICE_WRITE_ERROR: error writing data to Secure Element
- TO_DEVICE_READ_ERROR: error reading data from Secure Element
- TO_INVALID_RESPONSE_LENGTH: unexpected response length from device
- TO_MEMORY_ERROR: internal I/O buffer overflow
- TO_ERROR: generic error

TO_ret_t **TOSE_unsecure_payload_init_cbc**(*TOSE_ctx_t* *ctx, const uint8_t key_index, const *TO_enc_alg_t* enc_alg, const *TO_mac_alg_t* mac_alg, const uint16_t cryptogram_len, const uint8_t sequence[*TO_SEQUENCE_SIZE*], const uint8_t initial_vector[*TO_INITIALVECTOR_SIZE*], const uint8_t last_block_iv[*TO_INITIALVECTOR_SIZE*], const uint8_t last_block[*TO_AES_BLOCK_SIZE*])

Initializes to get back a message from a secured payload (CBC).

Parameters

- **ctx** – [**in**] Pointer to the SE context
- **key_index** – [**in**] Index of the keys to use for data encryption and MAC, starting from 0
- **enc_alg** – [**in**] Encryption algorithm to use
- **mac_alg** – [**in**] MAC algorithm to use
- **cryptogram_len** – [**in**] Cryptogram length
- **sequence** – [**in**] Sequence counter to avoid replay attacks
- **initial_vector** – [**in**] Block of 16 random bytes generated by the Secure Element and required to decrypt the data
- **last_block_iv** – [**in**] Last AES block initial vector (penultimate block)
- **last_block** – [**in**] Last AES block

Returns

- TORSP_SUCCESS on success
- TORSP_ARG_OUT_OF_RANGE: invalid key index

- TORSP_NOT_AVAILABLE: algorithm not available
- TO_DEVICE_WRITE_ERROR: error writing data to Secure Element
- TO_DEVICE_READ_ERROR: error reading data from Secure Element
- TO_INVALID_RESPONSE_LENGTH: unexpected response length from device
- TO_MEMORY_ERROR: internal I/O buffer overflow
- TO_ERROR: generic error

TO_ret_t **`TOSE_unsecure_payload_init_aead`**(*TOSE_ctx_t* *ctx, const uint8_t key_index, const *TO_enc_alg_t* enc_alg, const *TO_mac_alg_t* mac_alg, const uint16_t cryptogram_len, const uint8_t sequence[*TO_SEQUENCE_SIZE*])

Initializes to get back a message from a secured payload (AEAD).

Do not use this function directly, use *TOSE_helper_unsecure_payload()* instead.

Parameters

- **`ctx`** – [**in**] Pointer to the SE context
- **`key_index`** – [**in**] Index of the keys to use for data encryption and MAC, starting from 0
- **`enc_alg`** – [**in**] Encryption algorithm to use
- **`mac_alg`** – [**in**] MAC algorithm to use
- **`cryptogram_len`** – [**in**] Cryptogram length
- **`sequence`** – [**in**] Sequence counter to avoid replay attacks

Returns

- TORSP_SUCCESS on success
- TORSP_ARG_OUT_OF_RANGE: invalid key index
- TORSP_NOT_AVAILABLE: algorithm not available
- TO_DEVICE_WRITE_ERROR: error writing data to Secure Element
- TO_DEVICE_READ_ERROR: error reading data from Secure Element
- TO_INVALID_RESPONSE_LENGTH: unexpected response length from device
- TO_MEMORY_ERROR: internal I/O buffer overflow
- TO_ERROR: generic error

TO_ret_t **`TOSE_unsecure_payload_update`**(*TOSE_ctx_t* *ctx, const uint8_t *cryptogram, const uint16_t cryptogram_len, uint8_t *data, uint16_t *data_len)

Updates to get back a message from a secured payload.

Do not use this function directly, use *TOSE_helper_unsecure_payload()* instead.

Parameters

- **`ctx`** – [**in**] Pointer to the SE context
- **`cryptogram`** – [**in**] Message cryptogram

- **cryptogram_len** – [**in**] Message cryptogram length
- **data** – [**out**] Message unsecured
- **data_len** – [**out**] Message length

Returns

- TORSP_SUCCESS on success
- TORSP_ARG_OUT_OF_RANGE: invalid key index
- TORSP_NOT_AVAILABLE: algorithm not available
- TO_DEVICE_WRITE_ERROR: error writing data to Secure Element
- TO_DEVICE_READ_ERROR: error reading data from Secure Element
- TO_INVALID_RESPONSE_LENGTH: unexpected response length from device
- TO_MEMORY_ERROR: internal I/O buffer overflow
- TO_ERROR: generic error

TO_ret_t **TOSE_unsecure_payload_final**(*TOSE_ctx_t* *ctx)

Finalizes to get back a message from a secured payload.

Do not use this function directly, use *TOSE_helper_unsecure_payload()* instead.

Parameters

- **ctx** – [**in**] Pointer to the SE context

Returns

- TORSP_SUCCESS on success
- TORSP_ARG_OUT_OF_RANGE: invalid key index
- TORSP_NOT_AVAILABLE: algorithm not available
- TO_DEVICE_WRITE_ERROR: error writing data to Secure Element
- TO_DEVICE_READ_ERROR: error reading data from Secure Element
- TO_INVALID_RESPONSE_LENGTH: unexpected response length from device
- TO_MEMORY_ERROR: internal I/O buffer overflow
- TO_ERROR: generic error

6.2.2.8.1 Old API

Deprecated since version 4.10.0: Use new **`Secure messaging`__**.

```
TO_ret_t TOSE_aes128cbc_hmac_secure_message (TOSE_ctx_t *ctx,
const uint8_t aes_key_index, const uint8_t hmac_key_index, const uint8_t *data,
const uint16_t data_length, uint8_t initial_vector[TO_INITIALVECTOR_SIZE],
uint8_t *cryptogram, uint8_t hmac[TO_HMAC_SIZE]) TO_DEPRECATED
```

Transforms a message into a secured message (AES128-CBC cryptogram and HMAC tag).

It is equivalent to call *TOSE_aes128cbc_encrypt()* command, then *TOSE_compute_hmac()* on the result. The HMAC tag is calculated on encrypted data. Typical use is to have the same value to both AES and HMAC Key indexes. If remote public key is known and trusted by the Secure Element, the Secure Elements public key could be added to the result of this command and could be used on to have one way only communication network (from Secure Element to remote only).

Note: As padding is not handled by the the Secure Element, you must ensure that data length is a multiple of 16 and is not greater than maximum length value (512 bytes). Initial vector is generated by the Secure Element and not included in the data length

This API is deprecated, use TOSE_secure_payload() instead.

Parameters

- **`ctx`** – [**in**] Pointer to the SE context
- **`aes_key_index`** – [**in**] Index of the key to use for data encryption, starting from 0
- **`hmac_key_index`** – [**in**] Index of the key to use for HMAC, starting from 0
- **`data`** – [**in**] Message to be secured
- **`data_length`** – [**in**]
- **`initial_vector`** – [**out**] Block of 16 random bytes generated by the Secure Element and required to decrypt the data
- **`cryptogram`** – [**out**] Message cryptogram (same size as data)
- **`hmac`** – [**out**] Message HMAC

Returns

- TORSP_SUCCESS on success
- TORSP_ARG_OUT_OF_RANGE: invalid key index
- TO_DEVICE_WRITE_ERROR: error writing data to Secure Element
- TO_DEVICE_READ_ERROR: error reading data from Secure Element
- TO_INVALID_RESPONSE_LENGTH: unexpected response length from device
- TO_MEMORY_ERROR: internal I/O buffer overflow
- TO_ERROR: generic error

```
TO_ret_t TOSE_aes128cbc_hmac_unsecure_message (TOSE_ctx_t *ctx,
const uint8_t aes_key_index, const uint8_t hmac_key_index,
const uint8_t initial_vector[TO_INITIALVECTOR_SIZE], const uint8_t *cryptogram,
const uint16_t cryptogram_length, const uint8_t hmac[TO_HMAC_SIZE],
uint8_t *data) TO_DEPRECATED
```

Get back a message from a secured message (AES128-CBC cryptogram and HMAC tag).

Data are decrypted only if the HMAC tag is valid.

This API is deprecated, use TOSE_unsecure_payload() instead.

Parameters

- **`ctx`** – [**in**] Pointer to the SE context
- **`aes_key_index`** – [**in**] Index of the key to use for data decryption, starting from 0

- **hmac_key_index** – [**in**] Index of the key to use for HMAC verification, starting from 0
- **initial_vector** – [**in**] Initial vector for decryption
- **cryptogram** – [**in**] Message cryptogram
- **cryptogram_length** – [**in**]
- **hmac** – [**in**] Expected HMAC
- **data** – [**out**] Decrypted data

Returns

- TORSP_SUCCESS on success
- TORSP_ARG_OUT_OF_RANGE: invalid key index
- TO_DEVICE_WRITE_ERROR: error writing data to Secure Element
- TO_DEVICE_READ_ERROR: error reading data from Secure Element
- TO_INVALID_RESPONSE_LENGTH: unexpected response length from device
- TO_MEMORY_ERROR: internal I/O buffer overflow
- TO_ERROR: generic error

TO_ret_t TOSE_aes128cbc_cmac_secure_message (TOSE_ctx_t *ctx, const uint8_t aes_key_index, const uint8_t cmac_key_index, const uint8_t *data, const uint16_t data_length, uint8_t initial_vector[TO_INITIALVECTOR_SIZE], uint8_t *cryptogram, uint8_t cmac[TO_CMAC_SIZE]) TO_DEPRECATED

Transforms a message into a secured message (AES128-CBC cryptogram and CMAC tag).

It is equivalent to call *TOSE_aes128cbc_encrypt()* command, then *TOSE_compute_cmac()* on the result. The CMAC tag is calculated on encrypted data. Typical use is to have the same value to both AES and CMAC Key indexes. If remote public key is known and trusted by the Secure Element, the Secure Elements public key could be added to the result of this command and could be used on to have one way only communication network (from Secure Element to remote only).

Note: As padding is not handled by the Secure Element, you must ensure that data length is a multiple of 16 and is not greater than maximum length value (512 bytes). Initial vector is generated by the Secure Element and not included in the data length

This API is deprecated, use TOSE_secure_payload() instead.

Parameters

- **ctx** – [**in**] Pointer to the SE context
- **aes_key_index** – [**in**] Index of the key to use for data encryption, starting from 0
- **cmac_key_index** – [**in**] Index of the key to use for CMAC, starting from 0
- **data** – [**in**] Message to be secured
- **data_length** – [**in**]
- **initial_vector** – [**out**] Block of 16 random bytes generated by the Secure Element and required to decrypt the data
- **cryptogram** – [**out**] Message cryptogram (same size as data)

- **cmac** – [**out**] Message CMAC

Returns

- TORSP_SUCCESS on success
- TORSP_ARG_OUT_OF_RANGE: invalid key index
- TO_DEVICE_WRITE_ERROR: error writing data to Secure Element
- TO_DEVICE_READ_ERROR: error reading data from Secure Element
- TO_INVALID_RESPONSE_LENGTH: unexpected response length from device
- TO_MEMORY_ERROR: internal I/O buffer overflow
- TO_ERROR: generic error

TO_ret_t TOSE_aes128cbc_cmac_unsecure_message (TOSE_ctx_t *ctx, const uint8_t aes_key_index, const uint8_t cmac_key_index, const uint8_t initial_vector[TO_INITIALVECTOR_SIZE], const uint8_t *cryptogram, const uint16_t cryptogram_length, const uint8_t cmac[TO_CMAC_SIZE], uint8_t *data) TO_DEPRECATED

Get back a message from a secured message (AES128-CBC cryptogram and CMAC tag).

Data are decrypted only if the CMAC tag is valid.

This API is deprecated, use TOSE_unsecure_payload() instead.

Parameters

- **ctx** – [**in**] Pointer to the SE context
- **aes_key_index** – [**in**] Index of the key to use for data decryption, starting from 0
- **cmac_key_index** – [**in**] Index of the key to use for CMAC verification, starting from 0
- **initial_vector** – [**in**] Initial vector for decryption
- **cryptogram** – [**in**] Message cryptogram
- **cryptogram_length** – [**in**]
- **cmac** – [**in**] Expected CMAC
- **data** – [**out**] Decrypted data

Returns

- TORSP_SUCCESS on success
- TORSP_ARG_OUT_OF_RANGE: invalid key index
- TO_DEVICE_WRITE_ERROR: error writing data to Secure Element
- TO_DEVICE_READ_ERROR: error reading data from Secure Element
- TO_INVALID_RESPONSE_LENGTH: unexpected response length from device
- TO_MEMORY_ERROR: internal I/O buffer overflow
- TO_ERROR: generic error

6.2.2.9 TLS

TO_ret_t **TOSE_tls_reset**(*TOSE_ctx_t* *ctx)

Resets the current TLS/DTLS session.

Note:

After resetting the session, a full handshake will have to be re-negociated, as the session keys and master secrets are reset for this session. It does not have any influence on the other sessions that may be opened.

It can be used also to fix a malfunctioning TLS slot.

Parameters

- **ctx** – **[in]** Pointer to the SE context

TO_ret_t TOSE_tls_set_mode (TOSE_ctx_t *ctx, const TO_tls_mode_t mode) TO_DEPRECATED

Selects between TLS and DTLS mode and resets the session for the current selected slot.

Deprecated:

Parameters

- **ctx** – **[in]** Pointer to the SE context
- **mode** – **[in]** TLS mode. Currently only *TO_TLS_MODE_TLS_1_2* and *TO_TLS_MODE_DTLS_1_2* are supported.

TO_ret_t **TOSE_tls_set_config**(*TOSE_ctx_t* *ctx, const *TO_tls_config_id_t* config_id, const uint8_t *config, const uint16_t config_len)

Set TLS config (either mode or cipher suite selection).

Permits to switch to TLS or DTLS, to select a cipher suite for the handshake and resets the current session (if the configuration has changed).

Parameters

- **ctx** – **[in]** Pointer to the SE context
- **config_id** – **[in]** TLS configuration ID (either *TO_TLS_CONFIG_ID_MODE* or *TO_TLS_CONFIG_ID_CIPHER_SUITES*)
- **config** – **[in]** Pointer to the desired new TLS configuration
- **config_len** – **[in]** TLS configuration length (1 for the mode, 2 for the cipher suite)

TO_ret_t **TOSE_tls_set_session**(*TOSE_ctx_t* *ctx, const uint8_t session)

Selects the current TLS session slot to be used.

Note: There are several session slots available which can be connected to different servers. Depending on your application you may have to switch between those session slots.

Parameters

- **ctx** – [**in**] Pointer to the SE context
- **session** – [**in**] TLS session ID

TO_ret_t **TOSE_tls_set_cid_ext_id**(*TOSE_ctx_t* *ctx, const *TO_tls_extension_t* cid_ext_id)

Set sets the type of the extension ID corresponding to the connection ID.

Currently, the ID corresponding to the connection ID is still part of a draft standard (dec. 2021). Until the moment the RFC standard is published, this entry-point is used to provide this information.

Parameters

- **cid_ext_id** – [**in**] Connection ID extension ID

TO_ret_t **TOSE_tls_get_client_hello**(*TOSE_ctx_t* *ctx, const uint8_t timestamp[*TO_TIMESTAMP_SIZE*], uint8_t *client_hello, uint16_t *client_hello_len)

Generates the TLS Client_Hello (client) message.

When a client first connects to a server, it is required to send the ClientHello as its first message. The client can also send a ClientHello in response to a HelloRequest or on its own initiative in order to renegotiate the security parameters in an existing connection.

Parameters

- **ctx** – [**in**] Pointer to the SE context
- **timestamp** – [**in**] Timestamp (seconds since epoch)
- **client_hello** – [**out**] Pointer to a buffer receiving the ClientHello payload (up to 79 bytes in TLS, 120 bytes in DTLS)
- **client_hello_len** – [**out**] Pointer to receive the ClientHello payload length

TO_ret_t **TOSE_tls_get_client_hello_ext**(*TOSE_ctx_t* *ctx, const uint8_t timestamp[*TO_TIMESTAMP_SIZE*], const uint8_t *ext_data, uint16_t ext_length, uint8_t *client_hello, uint16_t *client_hello_len)

Get TLS ClientHello with extension.

Return the TLS handshake payload of the standard TLS ClientHello message. This payload must be encapsulated in a TLS record. The length of the response can be different depending on the use case.

Parameters

- **ctx** – [**inout**] SE context
- **timestamp** – [**in**] Timestamp (seconds since epoch)
- **ext_data** – [**in**] extension data
- **ext_length** – [**in**] extension length
- **client_hello** – [**out**] ClientHello payload
- **client_hello_len** – [**out**] ClientHello payload length

Return values

- `TORSP_SUCCESS` – on success
- `TO_DEVICE_WRITE_ERROR` – error writing data to Secure Element
- `TO_DEVICE_READ_ERROR` – error reading data from Secure Element
- `TO_INVALID_RESPONSE_LENGTH` – unexpected response length from device
- `TO_MEMORY_ERROR` – internal I/O buffer overflow
- `TO_ERROR` – generic error

TO_ret_t **TOSE_tls_get_client_hello_init**(*TOSE_ctx_t* *ctx, const uint8_t timestamp[*TO_TIMESTAMP_SIZE*], const uint8_t *ext_data, uint16_t ext_length, uint16_t *client_hello_len, uint8_t *final_flag)

Get TLS ClientHello - CAPI version - Init.

Initialize retrieval of the TLS handshake payload of the standard TLS ClientHello message. This payload must be encapsulated in a TLS record.

Parameters

- `ctx` – [**inout**] SE context
- `timestamp` – [**in**] Timestamp (seconds since epoch)
- `ext_data` – [**in**] extension data
- `ext_length` – [**in**] extension length
- `client_hello_len` – [**out**] ClientHello payload length
- `final_flag` – [**out**] signal the final chunk of ClientHello to be received with *TOSE_tls_get_client_hello_final()*

Return values

- `TORSP_SUCCESS` – on success
- `TO_DEVICE_WRITE_ERROR` – error writing data to Secure Element
- `TO_DEVICE_READ_ERROR` – error reading data from Secure Element
- `TO_INVALID_RESPONSE_LENGTH` – unexpected response length from device
- `TO_MEMORY_ERROR` – internal I/O buffer overflow
- `TO_ERROR` – generic error

TO_ret_t **TOSE_tls_get_client_hello_update**(*TOSE_ctx_t* *ctx, uint8_t *data, uint16_t *part_len, uint8_t *final_flag)

Get TLS ClientHello - CAPI version - Update.

Return a part of the TLS handshake payload of the standard TLS ClientHello message. This payload must be encapsulated in a TLS record.

Parameters

- `ctx` – [**inout**] SE context
- `data` – [**out**] ClientHello payload part
- `part_len` – [**out**] ClientHello payload part length

- **final_flag** – [**out**] signal the final chunk of ClientHello to be received with *TOSE_tls_get_client_hello_final()*

Return values

- **TORSP_SUCCESS** – on success
- **TO_DEVICE_WRITE_ERROR** – error writing data to Secure Element
- **TO_DEVICE_READ_ERROR** – error reading data from Secure Element
- **TO_INVALID_RESPONSE_LENGTH** – unexpected response length from device
- **TO_MEMORY_ERROR** – internal I/O buffer overflow
- **TO_ERROR** – generic error

TO_ret_t **TOSE_tls_get_client_hello_final**(*TOSE_ctx_t* *ctx, uint8_t *data)

Get TLS ClientHello - CAPI version - Final.

Return the last part of the TLS handshake payload of the standard TLS ClientHello message. This payload must be encapsulated in a TLS record.

Parameters

- **ctx** – [**inout**] SE context
- **data** – [**out**] last ClientHello payload part

Return values

- **TORSP_SUCCESS** – on success
- **TO_DEVICE_WRITE_ERROR** – error writing data to Secure Element
- **TO_DEVICE_READ_ERROR** – error reading data from Secure Element
- **TO_INVALID_RESPONSE_LENGTH** – unexpected response length from device
- **TO_MEMORY_ERROR** – internal I/O buffer overflow
- **TO_ERROR** – generic error

TO_ret_t **TOSE_tls_handle_hello_verify_request**(*TOSE_ctx_t* *ctx, const uint8_t *hello_verify_request, const uint32_t hello_verify_request_len)

Handles the DTLS HelloVerifyRequest (server) message.

When the client sends its ClientHello message to the server, the server MAY respond with a HelloVerifyRequest message. This message contains a stateless cookie.

Note: This message processing is only needed in the case of DTLS

Parameters

- **ctx** – [**in**] Pointer to the SE context
- **hello_verify_request** – [**in**] HelloVerifyRequest message
- **hello_verify_request_len** – [**in**] HelloVerifyRequest message length

TO_ret_t **TOSE_tls_handle_server_hello**(*TOSE_ctx_t* *ctx, const uint8_t *server_hello, const uint32_t server_hello_len)

Handles the ServerHello (server) message.

The server will send this message in response to a ClientHello message when it was able to find an acceptable set of algorithms. If it cannot find such a match, it will respond with a handshake failure alert.

Parameters

- **ctx** – [**in**] Pointer to the SE context
- **server_hello** – [**in**] ServertHello payload
- **server_hello_len** – [**in**] ServertHello payload length

TO_ret_t **TOSE_tls_handle_server_hello_init**(*TOSE_ctx_t* *ctx, const uint32_t server_hello_len)

Handle TLS ServerHello - CAPI version - Init.

Initialize handling of the TLS handshake payload of the standard TLS ServerHello message received during TLS handshake.

Parameters

- **ctx** – [**inout**] SE context
- **server_hello_len** – [**in**] ServerHello payload length

Return values

- **TORSP_SUCCESS** – on success
- **TO_DEVICE_WRITE_ERROR** – error writing data to Secure Element
- **TO_DEVICE_READ_ERROR** – error reading data from Secure Element
- **TORSP_ARG_OUT_OF_RANGE** – bad content
- **TO_MEMORY_ERROR** – internal I/O buffer overflow
- **TO_ERROR** – generic error

TO_ret_t **TOSE_tls_handle_server_hello_update**(*TOSE_ctx_t* *ctx, const uint8_t *data, const uint32_t part_len)

Handle TLS ServerHello - CAPI version - Update.

Handle a part of the TLS handshake payload of the standard TLS ServerHello message received during TLS handshake.

Parameters

- **ctx** – [**inout**] SE context
- **data** – [**in**] part of ServerHello payload
- **part_len** – [**in**] part length

Return values

- **TORSP_SUCCESS** – on success
- **TO_DEVICE_WRITE_ERROR** – error writing data to Secure Element
- **TO_DEVICE_READ_ERROR** – error reading data from Secure Element

- **TORSP_ARG_OUT_OF_RANGE** – bad content
- **TO_MEMORY_ERROR** – internal I/O buffer overflow
- **TO_ERROR** – generic error

TO_ret_t **TOSE_tls_handle_server_hello_final**(*TOSE_ctx_t* *ctx, const uint8_t *data, const uint32_t last_len)

Handle TLS ServerHello - CAPI version - Final.

Handle the last part of the TLS handshake payload of the standard TLS ServerHello message received during TLS handshake.

Parameters

- **ctx** – [**inout**] SE context
- **data** – [**in**] last part of ServerHello payload
- **last_len** – [**in**] last part len

Return values

- **TORSP_SUCCESS** – on success
- **TO_DEVICE_WRITE_ERROR** – error writing data to Secure Element
- **TO_DEVICE_READ_ERROR** – error reading data from Secure Element
- **TORSP_ARG_OUT_OF_RANGE** – bad content
- **TO_MEMORY_ERROR** – internal I/O buffer overflow
- **TO_ERROR** – generic error

TO_ret_t **TOSE_tls_handle_server_certificate**(*TOSE_ctx_t* *ctx, const uint8_t *server_certificate, const uint32_t server_certificate_len)

Handles the TLS Certificate (server) message.

The server MUST send a Certificate message whenever the agreed- upon key exchange method uses certificates for authentication (this includes all key exchange methods defined in this document except DH_anon). This message will always immediately follow the ServerHello message.

Parameters

- **ctx** – [**in**] Pointer to the SE context
- **server_certificate** – [**in**] Certificate payload
- **server_certificate_len** – [**in**] Certificate payload length

TO_ret_t **TOSE_tls_handle_server_certificate_init**(*TOSE_ctx_t* *ctx, const uint8_t *server_certificate_init, const uint32_t server_certificate_init_len)

Handles the TLS Server Certificate header (server)

Handle TLS Server Certificate header from TLS handshake payload of the standard TLS ServerCertificate message. The goal of TOSE_tls_handle_server_certificate_ init(), udate() and final(), is to validate a certificate chain, and to store the public key of the first certificate. You must decapsulate it from TLS record prior to use this command.

Parameters

- **ctx** – [**in**] Pointer to the SE context
- **server_certificate_init** – [**in**] Certificate payload header (handshake header
  – certificates list length)
- **server_certificate_init_len** – [**in**] Certificate payload header length

TO_ret_t **TOSE_tls_handle_server_certificate_update**(*TOSE_ctx_t* *ctx, const uint8_t *server_certificate_update, const uint32_t server_certificate_update_len)

Handles the TLS Server Certificate partial payload (server)

Handle TLS Server Certificate partial payload from TLS handshake payload of the standard TLS ServerCertificate message, and if possible, verify the signature and memories the key of the current certificate of the certificates chain. You must decapsulate it from TLS record prior to use this command. This command can be called several times. *TOSE_tls_handle_server_certificate_init()* must be called prior to this call.

Parameters

- **ctx** – [**in**] Pointer to the SE context
- **server_certificate_update** – [**in**] Certificate partial payload
- **server_certificate_update_len** – [**in**] Certificate partial payload length

TO_ret_t **TOSE_tls_handle_server_certificate_final**(*TOSE_ctx_t* *ctx)

Finishes the TLS Server Certificate handling (server)

Parameters

- **ctx** – [**in**] Pointer to the SE context * Finish Server Certificate TLS handshake payload handling by verifying signature of last certificate and store the public key of the first certificate of the chain. You must decapsulate it from TLS record prior to use this command. Functions *TOSE_tls_handle_server_certificate_init()*, and *TOSE_tls_handle_server_certificate_update()* must be called prior to this call.

TO_ret_t **TOSE_tls_handle_server_key_exchange**(*TOSE_ctx_t* *ctx, const uint8_t *server_key_exchange, const uint32_t server_key_exchange_len)

Handle the TLS ServerKeyExchange (server) message.

Handle TLS handshake payload of the standard TLS ServerKeyExchange message.

Parameters

- **ctx** – [**in**] Pointer to the SE context
- **server_key_exchange** – [**in**] ServerKeyExchange payload
- **server_key_exchange_len** – [**in**] ServerKeyExchange payload length

TO_ret_t **TOSE_tls_handle_server_key_exchange_init**(*TOSE_ctx_t* *ctx, const uint8_t *server_key_exchange_init, const uint32_t server_key_exchange_init_len)

Handles the TLS Server ServerKeyExchange (server) header.

Parameters

- **ctx** – [**in**] Pointer to the SE context

- **server_key_exchange_init** – **[in]** ServerKeyExchange payload header (handshake header

  – key_exchanges list length)
- **server_key_exchange_init_len** – **[in]** ServerKeyExchange payload header length

TO_ret_t **TOSE_tls_handle_server_key_exchange_update**(*TOSE_ctx_t* *ctx, const uint8_t *server_key_exchange_update, const uint32_t server_key_exchange_update_len)

Handles the TLS Server ServerKeyExchange partial payload (server)

Parameters

- **ctx** – **[in]** Pointer to the SE context
- **server_key_exchange_update** – **[in]** ServerKeyExchange partial payload
- **server_key_exchange_update_len** – **[in]** ServerKeyExchange partial payload length

TO_ret_t **TOSE_tls_handle_server_key_exchange_final**(*TOSE_ctx_t* *ctx)

Finishes TLS Server ServerKeyExchange handling (server)

Parameters

- **ctx** – **[in]** Pointer to the SE context

TO_ret_t **TOSE_tls_handle_certificate_request**(*TOSE_ctx_t* *ctx, const uint8_t *certificate_request, const uint32_t certificate_request_len)

Handles the TLS CertificateRequest (server) message.

The server MUST send a Certificate message whenever the agreed- upon key exchange method uses certificates for authentication (this includes all key exchange methods defined in this document except DH_anon). This message will always immediately follow the ServerHello message.

Parameters

- **ctx** – **[in]** Pointer to the SE context
- **certificate_request** – **[in]** CertificateRequest payload
- **certificate_request_len** – **[in]** CertificateRequest payload length

TO_ret_t **TOSE_tls_handle_server_hello_done**(*TOSE_ctx_t* *ctx, const uint8_t *server_hello_done, const uint32_t server_hello_done_len)

Handles the DTLS ServerHelloDone (server) message.

Parameters

- **ctx** – **[in]** Pointer to the SE context
- **server_hello_done** – **[in]** ServerHelloDone payload
- **server_hello_done_len** – **[in]** ServerHelloDone payload length

TO_ret_t **TOSE_tls_get_certificate**(*TOSE_ctx_t* *ctx, uint8_t *certificate, uint16_t *certificate_len)

Generates the TLS Certificate (client) message.

This is the first message the client can send after receiving a ServerHelloDone message. This message is only sent if the server requests a certificate. If no suitable certificate is available, the client MUST send a certificate message containing no certificates.

Parameters

- **ctx** – **[in]** Pointer to the SE context
- **certificate** – **[out]** Certificate payload
- **certificate_len** – **[out]** Certificate payload length

TO_ret_t **TOSE_tls_get_certificate_init**(*TOSE_ctx_t* *ctx, uint8_t *certificate, uint16_t *certificate_len)

Get the TLS Certificate initialization (client)

This function is used with *TOSE_tls_get_certificate_update()* and *TOSE_tls_get_certificate_final()* to get TLS Certificate of more than 512 bytes without limitation. This first command initiates the process.

Parameters

- **ctx** – **[in]** Pointer to the SE context
- **certificate** – **[out]** Certificate payload
- **certificate_len** – **[out]** Certificate payload length

TO_ret_t **TOSE_tls_get_certificate_update**(*TOSE_ctx_t* *ctx, uint8_t *certificate, uint16_t *certificate_len)

Gets the TLS Certificate update (client)

This command can be called several times. Function *TOSE_tls_get_certificate_init()* must be called prior to this command.

Parameters

- **ctx** – **[in]** Pointer to the SE context
- **certificate** – **[out]** Certificate payload
- **certificate_len** – **[out]** Certificate payload length

TO_ret_t **TOSE_tls_get_certificate_final**(*TOSE_ctx_t* *ctx)

Gets the TLS Certificate finalize (client)

Parameters

- **ctx** – **[in]** Pointer to the SE context

TO_ret_t **TOSE_tls_get_client_key_exchange**(*TOSE_ctx_t* *ctx, uint8_t *client_key_exchange, uint16_t *client_key_exchange_len)

Gets the TLS ClientKeyExchange (client) message.

Get TLS handshake payload of the standard TLS message ClientKeyExchange, containing internal Secure Elements ephemeral public key if using ECDHE cipher suite.

Parameters

- **ctx** – [**in**] Pointer to the SE context
- **client_key_exchange** – [**out**] ClientKeyExchange payload
- **client_key_exchange_len** – [**out**] ClientKeyExchange payload length

TO_ret_t **TOSE_tls_get_certificate_verify**(*TOSE_ctx_t* *ctx, uint8_t *certificate_verify, uint16_t *certificate_verify_len)

Generates the TLS Certificate_Verify (client) message.

This message is used to provide explicit verification of a client certificate. This message is only sent following a client certificate that has signing capability (i.e., all certificates except those containing fixed Diffie-Hellman parameters). When sent, it MUST immediately follow the client key exchange message.

Parameters

- **ctx** – [**in**] Pointer to the SE context
- **certificate_verify** – [**out**] CertificateVerify payload
- **certificate_verify_len** – [**out**] CertificateVerify payload length

TO_ret_t **TOSE_tls_get_change_cipher_spec**(*TOSE_ctx_t* *ctx, uint8_t *change_cipher_spec, uint16_t *change_cipher_spec_len)

Generates the TLS Change_Cipher_Spec (client) message.

The ChangeCipherSpec message is sent by both the client and the server to notify the receiving party that subsequent records will be protected under the newly negotiated CipherSpec and keys. This message is technically not part of the handshake.

Parameters

- **ctx** – [**in**] Pointer to the SE context
- **change_cipher_spec** – [**out**] ChangeCipherSpec payload
- **change_cipher_spec_len** – [**out**] ChangeCipherSpec payload length

TO_ret_t **TOSE_tls_get_finished**(*TOSE_ctx_t* *ctx, uint8_t *finished, uint16_t *finished_len)

Generates the TLS Finished (client) message.

The Finished message is the first one protected with the just negotiated algorithms, keys, and secrets. Recipients of Finished messages MUST verify that the contents are correct. Once a side has sent its Finished message and received and validated the Finished message from its peer, it may begin to send and receive application data over the connection.

Parameters

- **ctx** – [**in**] Pointer to the SE context
- **finished** – [**out**] Finish payload
- **finished_len** – [**out**] Finish payload length

TO_ret_t **TOSE_tls_handle_change_cipher_spec**(*TOSE_ctx_t* *ctx, const uint8_t *change_cipher_spec, const uint32_t change_cipher_spec_len)

Handles the TLS ChangeCipherSpec (server) message.

The change cipher spec protocol exists to signal transitions in ciphering strategies. The protocol consists of a single message, which is encrypted and compressed under the current (not the pending) connection state.

Parameters

- **ctx** – **[in]** Pointer to the SE context
- **change_cipher_spec** – **[in]** ChangeCipherSpec payload
- **change_cipher_spec_len** – **[in]** ChangeCipherSpec payload length

TO_ret_t **TOSE_tls_handle_finished**(*TOSE_ctx_t* *ctx, const uint8_t *finished, const uint32_t finished_len)

Handles the TLS Finished (server) message.

The Finished message is the first one protected with the just negotiated algorithms, keys, and secrets. Recipients of Finished messages MUST verify that the contents are correct. Once a side has sent its Finished message and received and validated the Finished message from its peer, it may begin to send and receive application data over the connection.

Parameters

- **ctx** – **[in]** Pointer to the SE context
- **finished** – **[in]** Finished payload
- **finished_len** – **[in]** Finish payload length

TO_ret_t **TOSE_tls_get_certificate_slot**(*TOSE_ctx_t* *ctx, uint8_t *slot)

Generates the TLS certificate slot used during handshake (client) message.

Parameters

- **ctx** – **[in]** Pointer to the SE context
- **slot** – **[out]** Certificate slot

Post

Handshake must have been proceeded before calling this function.

TO_ret_t **TOSE_tls_secure_payload**(*TOSE_ctx_t* *ctx, const uint8_t *header, const uint16_t header_len, const uint8_t *data, const uint16_t data_len, uint8_t *payload, uint16_t *payload_len)

Secures a (client) message with TLS.

Parameters

- **ctx** – **[in]** Pointer to the SE context
- **header** – **[in]** TLS header
- **header_len** – **[in]** TLS header length
- **data** – **[in]** TLS data
- **data_len** – **[in]** TLS data length
- **payload** – **[out]** Secured message (without header)

- **`payload_len`** – **[out]** Secured message (without header) length

Post

Handshake must have been proceeded before calling this function.

TO_ret_t **`TOSE_tls_unsecure_payload`**(*TOSE_ctx_t* *ctx, const uint8_t *header, const uint16_t header_len, const uint8_t *payload, const uint16_t payload_len, uint8_t *data, uint16_t *data_len)

Unsecure message with TLS.

Decrypt data received from server through TLS. Take a TLS record as input with encrypted content and return a TLS record with clear content.

Parameters

- **`ctx`** – **[in]** Pointer to the SE context
- **`header`** – **[in]** TLS header
- **`header_len`** – **[in]** TLS header length
- **`payload`** – **[in]** Secured message (without header)
- **`payload_len`** – **[in]** Secured message (without header) length
- **`data`** – **[out]** TLS data
- **`data_len`** – **[out]** TLS data length

Post

Handshake must have been proceeded before calling this function.

TO_ret_t **`TOSE_tls_handle_mediator_certificate`**(*TOSE_ctx_t* *ctx, const uint8_t *mediator_certificate, const uint32_t mediator_certificate_len)

Handles the TLS proprietary MediatorCertificate (server) message.

This is a TO-specific message, used to handle the mediator certificate. This message is not part of any standard (TLS or DTLS).

Parameters

- **`ctx`** – **[in]** Pointer to the SE context
- **`mediator_certificate`** – **[in]** MediatorCertificate payload
- **`mediator_certificate_len`** – **[in]** MediatorCertificate payload length

6.3 I2C wrapper API

Warning: These APIs are **not** to be called externally, only the library should rely on them.

This API is implemented by every libTO I2C wrapper. The following functions have to be implemented in order to develop a new wrapper for a new I2C master device.

```
#include "TODRV_HSE_i2c_wrapper.h"
```

6.3.1 Types and definitions

The following structure type is used to configure I2C wrapper:

struct **TO_i2c_config_s**

I2C wrapper configuration.

To be used through TO_data_config().

Public Members

unsigned char **i2c_addr**

Device I2C address on 7 bits (MSB=0)

unsigned char **misc_settings**

Misc. device I2C settings bitfield: | RES | RES | RES | RES | RES | RES | RES | last byte NACKed |

typedef struct *TO_i2c_config_s* **TO_i2c_config_t**

misc. settings bitfield definitions:

TO_CONFIG_NACK_LAST_BYTE 0x01

TO_i2c_config_s misc. setting: last byte is NACKed by remote device

6.3.2 I2C bus setup

TO_lib_ret_t **TO_data_init**(void)

Initialize Secure Element communication bus session.

Initializes I2C bus for Secure Element communications. If required, this is the recommended place to handle SecureElement power-on.

Returns

TO_OK if initialization was successful, else TO_ERROR

TO_lib_ret_t **TO_data_fini**(void)

Finish Secure Element communication bus session.

Reset (stop) I2C bus used for Secure Element communications. If required, this is the recommended place to handle SecureElement power-off.

Returns

TO_OK if reset was successful, else TO_ERROR

If you power-up TO136 only when you use it, please respect TO136 boot time after power-up before sending commands. You should also respect sufficient time on power-down, to guarantee that a consecutive power-up is not going to make power glitch.

TO_lib_ret_t **`TO_data_config`**(const *TO_i2c_config_t* *config)

I2C configuration (optional function)

Take given I2C configuration and apply it on the I2C wrapper. If the function returns successfully, it means the configuration has been applied and taken into account. The wrapper must NOT assume this function will be called, and must run correctly even if this function is never used.

This function is optional, and even if enabled by TODRV_HSE_I2C_WRAPPER_CONFIG it can still return TO_OK without doing anything. It is left to the wrapper developer discretion. This function is not called internally by TO library.

See *TO_i2c_config_s*.

Parameters

- **`config`** – I2C configuration to use

Returns

TO_OK if configuration has been applied, else TO_ERROR

This function uses the following structure to receive settings:

Note: TO_data_config() API is not mandatory, if you dont need it do not define TO_I2C_WRAPPER_CONFIG in your project preprocessor flags.

6.3.3 Data transfers

TO_lib_ret_t **`TO_data_read`**(void *data, unsigned int length)

Read data from Secure Element on I2C bus.

Reads specified amount of data from the Secure Element on I2C bus. This function returns when data has been read and is available in the data buffer, or if an error occured. The condition start has to be sent only one time to read the full Secure Element response, the reading can not be divided.

Parameters

- **`data`** – Buffer to store received data
- **`length`** – Amount of data to read in bytes

Returns

TO_OK if data has been read sucessfully TO_DEVICE_READ_ERROR: error reading data from Secure Element TO_ERROR if an internal error has occured

TO_lib_ret_t **`TO_data_write`**(const void *data, unsigned int length)

Write data to Secure Element on I2C bus.

Writes specified amount of data to the Secure Element on I2C bus. This function returns when all data in the buffer has been written, or if an error occured. The condition start has to be sent only one time to write the full Secure Element command, the writing can not be divided.

Parameters

- **data** – Buffer containing data to send
- **length** – Amount of data to send in bytes

Returns
TO_OK if data has been written sucessfully TO_DEVICE_WRITE_ERROR: error writing data to Secure Element TO_ERROR if an internal error has occured

6.3.4 Miscellaneous

TO_lib_ret_t **TO_data_last_command_duration**(unsigned int *duration)

Get last command duration (from I2C send to I2C receive)

Measure the delay of the last executed command with MCU point of view. This function is optional, if implemented you have to define TODRV_HSE_I2C_WRAPPER_LAST_COMMAND_DURATION in your project in order to use it through TO_last_command_duration() API.

This function should only be called after a successful TO_read() call. If it is called after a failed TO_read(), or after a TO_write() call, the result is unspecified and may be irrelevant.

Parameters

- **duration** – Pointer to store last command duration in microseconds

Returns
TO_OK if last command duration is available TO_ERROR if an internal error has occured

6.4 Library core APIs

These APIs are available if it is needed to add some custom tuning on the library behavior. For example, the *Secure Element functions* can be completely rewritten using the following APIs, if the way some of them are implemented does not fit your needs.

```
#include "TODRV_HSE_cmd.h"
```

6.4.1 Data buffers

The following buffers are accessible.

unsigned char ***TODRV_HSE_command_data**

Helper to access internal I/O buffer command data section, only valid before TO_send_command() call (even if an error occured while sending command).

unsigned char ***TODRV_HSE_response_data**

Helper to access internal I/O buffer response data section, only valid after TO_send_command() call.

6.4.2 Command data preparation

The following functions are used to prepare data before sending command to the Secure Element.

TO_lib_ret_t **TODRV_HSE_prepare_command_data**(uint16_t offset, const unsigned char *data, uint16_t len)

Prepare command data.

Insert data into the internal I/O buffer at the specified offset.

Warning: do not free data pointer parameter or overwrite data before having called *TODRV_HSE_send_command()*, or before aborted command with *TODRV_HSE_reset_command_data()*.

Parameters

- **offset** – Buffer offset where to insert data
- **data** – Data to be copied into the buffer
- **len** – Data length

Returns

TO_OK on success TO_MEMORY_ERROR: data overflows internal I/O buffer, in this case internal command data buffers are invalidated (as if *TODRV_HSE_reset_command_data()* has been called).

TO_lib_ret_t **TODRV_HSE_prepare_command_data_byte**(uint16_t offset, const char byte)

Prepare command data byte.

Insert data byte into the internal I/O buffer at the specified offset.

Parameters

- **offset** – Buffer offset where to insert data
- **byte** – Data byte to be copied into the buffer

Returns

TO_OK on success TO_MEMORY_ERROR: data byte overflows internal I/O buffer, in this case internal command data buffers are invalidated (as if *TODRV_HSE_reset_command_data()* has been called).

TO_lib_ret_t **TODRV_HSE_set_command_data**(uint16_t offset, const char byte, uint16_t len)

Set data range.

Set internal I/O buffer range bytes to a defined value.

Parameters

- **offset** – Buffer offset where to begin range
- **byte** – Value to be set for each byte in the range
- **len** – Range length

Returns

TO_OK on success TO_MEMORY_ERROR: range overflows internal I/O buffer, in this case internal command data buffers are invalidated (as if *TODRV_HSE_reset_command_data()* has been called).

And to reset command context:

void **TODRV_HSE_reset_command_data**(void)

Reset command data.

This function resets command data. It MUST be called if command data has been prepared without subsequent call to *TODRV_HSE_send_command()* (if command has been aborted for example).

6.4.3 Send command

The following function is used to send a command to the Secure Element, after *Command data preparation*.

TO_lib_ret_t **TODRV_HSE_send_command**(const uint16_t cmd, uint16_t cmd_data_len, uint16_t *resp_data_len, *TO_se_ret_t* *resp_status)

Send command to the Secure Element device.

Send a command to the Secure Element device and get response data. Internal command data buffers must be considered as invalidated after calling this function.

Parameters

- **cmd** – Command code (see TODRV_HSE_CMD_* definitions)
- **cmd_data_len** – Command data len (got from internal I/O buffer)
- **resp_data_len** – Response data len (expected)
- **resp_status** – Status of the command

Returns

TO_OK on success TO_MEMORY_ERROR: data overflows internal I/O buffer TO_DEVICE_WRITE_ERROR: unable to send command TO_DEVICE_READ_ERROR: unable to read response data TO_INVALID_RESPONSE_LENGTH: expected response length differs from headers

6.4.4 Hooks

The following hooks can be set to automatically call client application functions when reaching particular steps in the library internal flow. This mechanism allows client application to run custom code interlaced with libTO code.

6.4.4.1 Hooks functions prototypes

Below are detailled functions hooks prototypes, to be implemented by client application if required. Implemented hook functions have to be setup using *Hooks setup functions*.

typedef void (***TODRV_HSE_pre_command_hook**)(uint16_t cmd, uint16_t cmd_data_len)

Hook function prototype to be called by *TODRV_HSE_send_command()* just before sending a command to the Secure Element.

Once return, the command response is read from Secure Element.

Warning: do NOT call any libTO function from this kind of hook.

Param cmd
Command code, see *Hardware Secure Element command codes*

Param cmd_data_len
Command data length

typedef void (*`TODRV_HSE_post_write_hook`)(uint16_t cmd, uint16_t cmd_data_len)

Hook function prototype to be called by *TODRV_HSE_send_command()* just after writing command to the Secure Element, and before reading its response.

This hook can be used by client application for power optimization, for example making the system sleep for a while or until Secure Element status GPIO signals response readyness. For this second use case, it is recommended to arm GPIO wakeup interrupt by setting a hook with *TODRV_HSE_pre_command_hook()*, to be sure to do not miss the response readyness GPIO toggle.

Once return, the command response is read from Secure Element.

Warning: do NOT call any libTO function from this kind of hook.

Param cmd
Command code, see *Hardware Secure Element command codes*

Param cmd_data_len
Command data length

typedef void (*`TODRV_HSE_post_command_hook`)(uint16_t cmd, uint16_t cmd_data_len, uint16_t cmd_rsp_len, *TO_se_ret_t* cmd_status)

Hook function prototype to be called by *TODRV_HSE_send_command()* just after reading command response from the Secure Element.

Warning: do NOT call any libTO function from this kind of hook.

Param cmd
Command code, see *Hardware Secure Element command codes*

Param cmd_data_len
Command data length

Param cmd_rsp_len
Command response length

Param cmd_status
Command status

6.4.4.2 Hooks setup functions

For each hook type, a function has to be called to setup it and to allow libTO to call it.

void `TODRV_HSE_set_lib_hook_pre_command`(*TODRV_HSE_pre_command_hook* hook)

Set a pre command hook (see TODRV_HSE_pre_command_hook).

Parameters

- `hook` – Pre command hook function to set (NULL to disable).

void **TODRV_HSE_set_lib_hook_post_write**(*TODRV_HSE_post_write_hook* hook)

Set a post write hook (see TODRV_HSE_post_write_hook).

Parameters

- **hook** – Post write hook function to set (NULL to disable).

void **TODRV_HSE_set_lib_hook_post_command**(*TODRV_HSE_post_command_hook* hook)

Set a post cmd hook (see TODRV_HSE_post_command_hook).

Parameters

- **hook** – Post cmd hook function to set (NULL to disable).

void **TODRV_HSE_cmd_name_from_number**(int number, char *name)

Note: Hooks are set permanently until reboot, or until you set a NULL hook function pointer using the hook setup function.

6.4.5 Logs

The following function is used to set library log level.

void **TO_set_log_level**(*TO_log_ctx_t* *log_ctx, const TO_log_level_t level, TO_log_func_t *log_function)

Sets the Log function and log level.

This function permits to change the log level and the log function.

Parameters

- **log_ctx** – Current log context
- **level** – Desired log level
- **log_function** – Log function (eg. TO_log)

6.5 Types and definitions

LibTO types and definitions.

```
#include "TO_defs.h"
```

6.5.1 Library error codes

group **lib_codes**

Error codes

Typedefs

typedef enum *TO_lib_ret_e* **TO_lib_ret_t**

Enums

enum **TO_lib_ret_e**

Values:

enumerator **TO_OK**

enumerator **TO_MEMORY_ERROR**

enumerator **TO_DEVICE_WRITE_ERROR**

enumerator **TO_DEVICE_READ_ERROR**

enumerator **TO_INVALID_CA_ID**

enumerator **TO_INVALID_CERTIFICATE_FORMAT**

enumerator **TO_INVALID_CERTIFICATE_NUMBER**

enumerator **TO_INVALID_RESPONSE_LENGTH**

enumerator **TO_SECLINK_ERROR**

enumerator **TO_TIMEOUT**

enumerator **TO_AGAIN**

enumerator **TO_INVALID_PARAM**

enumerator **TO_NOT_IMPLEMENTED**

enumerator **TO_ERROR**

Note: Less significant byte is left empty because it is reserved for Secure Element error codes, then it is possible to return Secure Element and library error codes in one single variable. See *Secure Element error codes*.

6.5.2 Secure Element error codes

group **se_codes**

Typedefs

typedef enum *TO_se_ret_e* **TO_se_ret_t**

Secure Element response codes.

These return codes are common to all TO Secure elements, including the TO-136 and TO-Protect. Therefore, some of these return values may have a different meaning depending on the SE you are using, and the context you are receiving it. Refer yourself to the called function to have a more precise information.

Enums

enum **TO_se_ret_e**

Secure Element response codes.

These return codes are common to all TO Secure elements, including the TO-136 and TO-Protect. Therefore, some of these return values may have a different meaning depending on the SE you are using, and the context you are receiving it. Refer yourself to the called function to have a more precise information.

Values:

enumerator **TORSP_UNKNOWN_CMD**

Indicates that the SE does not know how to handle this command

enumerator **TORSP_BAD_SIGNATURE**

The digital signature is wrong

enumerator **TORSP_INVALID_LEN**

The provided length is wrong

enumerator **TORSP_NOT_AVAILABLE**

The requested data cannot be retrieved

enumerator **TORSP_INVALID_PADDING**

The expected padding is not respected

enumerator **TORSP_COM_ERROR**

A communication error has occurred

enumerator **TO136RSP_COM_ERROR**

Deprecated, use TORSP_COM_ERROR instead

enumerator **TORSP_NEED_AUTHENTICATION**

An authentication process has to be conduced to pursue

enumerator **TORSP_COND_OF_USE_NOT_SATISFIED**

This command cannot be used in this context

enumerator **TORSP_ARG_OUT_OF_RANGE**

An argument is not in the expected range

enumerator **TORSP_SUCCESS**

The Commands execution has been conduced correctly

enumerator **TORSP_NO_VALID_SECURE_STORAGE_FOUND**

No valid secure storage found

enumerator **TORSP_INCONSISTENT_SECURE_STORAGES**

Inconsistent secure storages

enumerator **TORSP_WRONG_SECURE_STORAGE_VERSION**

Secure Storages version is wrong

enumerator **TORSP_WRONG_INSTALLATION**

Wrong code integrity

enumerator **TORSP_SECLINK_RENEW_KEY**

The SecLink key has to be renewed

enumerator **TORSP_INTERNAL_ERROR**

An internal error has occurred. It may be the proof that something unexpected has happened (for instance, a fault has been detected).

6.5.3 Combined error codes

group **error_codes**

Defines

TO_LIB_ERRCODE(errcode) ((errcode) & 0xFF00)
Mask error code to extract library error

TO_SE_ERRCODE(errcode) ((errcode) & 0x00FF)
Mask error code to extract SE error

Typedefs

typedef uint16_t **TO_ret_t**

6.5.4 Keys types

group **key_types**

Typedefs

typedef enum *TO_key_type_e* **TO_key_type_t**
Secure Element key types

Enums

enum **TO_key_type_e**
Secure Element key types
Values:

enumerator **KTYPE_CERT_KPUB**

enumerator **KTYPE_CERT_KPRIV**

enumerator **KTYPE_CA_KPUB**

enumerator **KTYPE_REMOTE_KPUB**

enumerator **KTYPE_ECIES_KPUB**

enumerator **KTYPE_ECIES_KPRIV**

enumerator **KTYPE_ECIES_KAES**

enumerator **KTYPE_ECIES_KMAC**

enumerator **KTYPE_LORA_KAPP**

enumerator **KTYPE_LORA_KNET**

enumerator **KTYPE_LORA_KSAPP**

enumerator **KTYPE_LORA_KSNET**

6.5.5 Certificates

group **certificates**

Defines

TOCERTF_STANDALONE ((unsigned char)0x00)

TOCERTF_SHORT ((unsigned char)0x01)

TOCERTF_X509 ((unsigned char)0x02)

TOCERTF_SHORT_V2 ((unsigned char)0x03)

TOCERTF_NONE ((unsigned char)0xFF)

TOCERTF_UNKNOWN ((unsigned char)0xFE)

TOCERTF_VALIDITY_DATE_SIZE 7UL

TOCERTF_SUBJECT_NAME_SIZE 15UL

TO_IDX_AUTO 0xFF

Index to enable automatic certificate detection

TO_CA_IDX_AUTO *TO_IDX_AUTO*

Deprecated:

Retro-compatibility

Typedefs

typedef enum *TO_certificate_format_e* **TO_certificate_format_t**

Certificates formats

- TO_CERTIFICATE_X509 is used for Secure Element and remote certificate verification
- TO_CERTIFICATE_STANDALONE is only used for remote certificate verification
- TO_CERTIFICATE_SHORT is only used for Secure Element certificates

typedef struct *TO_cert_standalone_s* **TO_cert_standalone_t**

typedef struct *TO_cert_short_s* **TO_cert_short_t**

typedef struct *TO_cert_short_v2_s* **TO_cert_short_v2_t**

typedef enum *TO_cert_CA_capabilities_e* **TO_cert_CA_capabilities_t**

Enums

enum **TO_certificate_format_e**

Certificates formats

- TO_CERTIFICATE_X509 is used for Secure Element and remote certificate verification
- TO_CERTIFICATE_STANDALONE is only used for remote certificate verification
- TO_CERTIFICATE_SHORT is only used for Secure Element certificates

Values:

enumerator **TO_CERTIFICATE_STANDALONE**

enumerator **TO_CERTIFICATE_SHORT**

enumerator **TO_CERTIFICATE_X509**

enumerator **TO_CERTIFICATE_SHORT_V2**

enumerator **TO_CERTIFICATE_NONE**

enumerator **TO_CERTIFICATE_UNKNOWN**

enum **TO_cert_CA_capabilities_e**

Values:

enumerator **TO_CERT_CA_CAP_EMPTY**

No capability

enumerator **TO_CERT_CA_CAP_ADMIN**

Admin capability

enumerator **TO_CERT_CA_CAP_UPDATE_CA**

CA update capability

struct **TO_cert_standalone_s**

#include <TO_defs.h> Standalone certificate structure

Public Members

uint8_t **ca_id**[*TO_SN_CA_ID_SIZE*]

Certificate Authority ID

uint8_t **serial_number**[*TO_SN_NB_SIZE*]

SE serial number

uint8_t **public_key**[*TO_ECC_PUB_KEYSIZE*]

Public key

uint8_t **signature**[*TO_SIGNATURE_SIZE*]

Certificate signature

struct **TO_cert_short_s**

#include <TO_defs.h> Short certificate structure

Public Members

uint8_t **ca_id**[*TO_SN_CA_ID_SIZE*]
Certificate Authority ID

uint8_t **serial_number**[*TO_SN_NB_SIZE*]
SE serial number

uint8_t **public_key**[*TO_ECC_PUB_KEYSIZE*]
Public key

uint8_t **signature**[*TO_SIGNATURE_SIZE*]
Certificate signature

struct **TO_cert_short_v2_s**
#include <TO_defs.h> Short v2 certificate structure

Public Members

uint8_t **ca_id**[*TO_SN_CA_ID_SIZE*]
Certificate Authority ID

uint8_t **serial_number**[*TO_SN_NB_SIZE*]
SE serial number

uint8_t **date**[*TOCERTF_VALIDITY_DATE_SIZE*]
Validity date (not to be used after xxx) (Zulu date (UTC))

uint8_t **public_key**[*TO_ECC_PUB_KEYSIZE*]
Public key

uint8_t **signature**[*TO_SIGNATURE_SIZE*]
Certificate signature

6.5.6 Algorithms

enum **TO_enc_alg_e**
Encryption algorithms

Values:

enumerator **TO_ENC_ALG_UNDEFINED**

Undefined

enumerator **TO_ENC_ALG_AES128CBC**

AES128 CBC

enumerator **TO_ENC_ALG_AES128GCM**

AES128 GCM

enumerator **TO_ENC_ALG_ARC4**

ARC4

enumerator **TO_ENC_ALG_AES128CCM**

AES128 CCM

enumerator **TO_ENC_ALG_MAX**

enum **TO_mac_alg_e**

MAC algorithms

Values:

enumerator **TO_MAC_ALG_UNDEFINED**

Deprecated:

, use *TO_MAC_ALG_NONE* instead

enumerator **TO_MAC_ALG_NONE**

no MAC for AEAD cipher suites

enumerator **TO_MAC_ALG_HMAC**

HMAC

enumerator **TO_MAC_ALG_CMAC**

CMAC

enumerator **TO_MAC_ALG_MAX**

typedef enum *TO_enc_alg_e* **TO_enc_alg_t**

Encryption algorithms

typedef enum *TO_mac_alg_e* **TO_mac_alg_t**

MAC algorithms

6.5.7 Payloads

TO_PAYLOAD_MAC_SIZE(enc_alg, mac_alg) ((enc_alg) == *TO_ENC_ALG_AES128CCM* ? TO_AESCCM_TAG_SIZE : ((enc_alg) == *TO_ENC_ALG_AES128GCM* ? TO_AESGCM_TAG_SIZE : ((enc_alg) == *TO_ENC_ALG_AES128CBC* ? ((mac_alg) == *TO_MAC_ALG_HMAC* ? TO_HMAC_SIZE : ((mac_alg) == *TO_MAC_ALG_CMAC* ? TO_CMAC_SIZE : 0)) : 0)))

Payload MAC size

TO_PAYLOAD_PADDING_SIZE(enc_alg, data_len) ((enc_alg) == *TO_ENC_ALG_AES128CBC* ? (((TO_AES_BLOCK_SIZE - (((data_len) + 1) % TO_AES_BLOCK_SIZE)) % TO_AES_BLOCK_SIZE) + 1) : 0)

Payload padding size

TO_PAYLOAD_IV_SIZE(enc_alg) ((enc_alg) == *TO_ENC_ALG_AES128CBC* ? TO_INITIALVECTOR_SIZE : 0)

Payload initial vector size

TO_PAYLOAD_SECURED_PAYLOAD_SIZE(enc_alg, mac_alg, data_len) (TO_SEQUENCE_SIZE + *TO_PAYLOAD_IV_SIZE*(enc_alg) + data_len + *TO_PAYLOAD_MAC_SIZE*(enc_alg, mac_alg) + *TO_PAYLOAD_PADDING_SIZE*(enc_alg, data_len))

Secured payload size from clear data size

TO_PAYLOAD_CLEAR_DATA_SIZE(enc_alg, mac_alg, payload_len) ((payload_len) - TO_SEQUENCE_SIZE - *TO_PAYLOAD_IV_SIZE*(enc_alg) - *TO_PAYLOAD_MAC_SIZE*(enc_alg, mac_alg))

Clear data max size from secured payload size

6.5.8 Constants

group **command_codes**

Hardware Secure Element command codes

Defines

TODRV_HSE_CMD_INIT TO_SE_CMD_INIT

TODRV_HSE_CMD_GET_SN TO_SE_CMD_GET_SN

TODRV_HSE_CMD_GET_HW_SN TO_SE_CMD_GET_HW_SN

TODRV_HSE_CMD_RES TO_SE_CMD_RES

TODRV_HSE_CMD_GET_PN TO_SE_CMD_GET_PN

TODRV_HSE_CMD_GET_HW_VERSION TO_SE_CMD_GET_HW_VERSION

TODRV_HSE_CMD_GET_SW_VERSION TO_SE_CMD_GET_SW_VERSION

TODRV_HSE_CMD_GET_PRODUCT_ID TO_SE_CMD_GET_PRODUCT_ID

TODRV_HSE_CMD_ACCESS_DUMMY_DATA TO_SE_CMD_ACCESS_DUMMY_DATA

TODRV_HSE_CMD_GET_RANDOM TO_SE_CMD_GET_RANDOM

TODRV_HSE_CMD_ECHO TO_SE_CMD_ECHO

TODRV_HSE_CMD_SLEEP TO_SE_CMD_SLEEP

TODRV_HSE_CMD_READ_NVM TO_SE_CMD_READ_NVM

TODRV_HSE_CMD_WRITE_NVM TO_SE_CMD_WRITE_NVM

TODRV_HSE_CMD_GET_NVM_SIZE TO_SE_CMD_GET_NVM_SIZE

TODRV_HSE_CMD_SET_STATUS_PIO_CONFIG TO_SE_CMD_SET_STATUS_PIO_CONFIG

TODRV_HSE_CMD_GET_STATUS_PIO_CONFIG TO_SE_CMD_GET_STATUS_PIO_CONFIG

TODRV_HSE_CMD_SET_CERTIFICATE_SIGNING_REQUEST_DN
TO_SE_CMD_SET_CERTIFICATE_SIGNING_REQUEST_DN

TODRV_HSE_CMD_GET_CERTIFICATE_SIGNING_REQUEST
TO_SE_CMD_GET_CERTIFICATE_SIGNING_REQUEST

TODRV_HSE_CMD_GET_CERTIFICATE_SUBJECT_CN
TO_SE_CMD_GET_CERTIFICATE_SUBJECT_CN

TODRV_HSE_CMD_GET_CERTIFICATE TO_SE_CMD_GET_CERTIFICATE

TODRV_HSE_CMD_SET_CERTIFICATE TO_SE_CMD_SET_CERTIFICATE

TODRV_HSE_CMD_SET_CERTIFICATE_INIT TO_SE_CMD_SET_CERTIFICATE_INIT

TODRV_HSE_CMD_SET_CERTIFICATE_UPDATE TO_SE_CMD_SET_CERTIFICATE_UPDATE

TODRV_HSE_CMD_SET_CERTIFICATE_FINAL TO_SE_CMD_SET_CERTIFICATE_FINAL

TODRV_HSE_CMD_GET_CERTIFICATE_INIT TO_SE_CMD_GET_CERTIFICATE_INIT

TODRV_HSE_CMD_GET_CERTIFICATE_UPDATE TO_SE_CMD_GET_CERTIFICATE_UPDATE

TODRV_HSE_CMD_GET_CERTIFICATE_FINAL TO_SE_CMD_GET_CERTIFICATE_FINAL

TODRV_HSE_CMD_SIGN TO_SE_CMD_SIGN

TODRV_HSE_CMD_VERIFY TO_SE_CMD_VERIFY

TODRV_HSE_CMD_SIGN_HASH TO_SE_CMD_SIGN_HASH

TODRV_HSE_CMD_VERIFY_HASH_SIGNATURE TO_SE_CMD_VERIFY_HASH_SIGNATURE

TODRV_HSE_CMD_GET_CERTIFICATE_AND_SIGN TO_SE_CMD_GET_CERTIFICATE_AND_SIGN

TODRV_HSE_CMD_GET_CERTIFICATE_X509 TO_SE_CMD_GET_CERTIFICATE_X509

TODRV_HSE_CMD_SET_CERTIFICATE_X509 TO_SE_CMD_SET_CERTIFICATE_X509

TODRV_HSE_CMD_VERIFY_CERTIFICATE_AND_STORE TO_SE_CMD_VERIFY_CERTIFICATE_AND_STORE

TODRV_HSE_CMD_VERIFY_CA_CERTIFICATE_AND_STORE TO_SE_CMD_VERIFY_CA_CERTIFICATE_AND_STORE

TODRV_HSE_CMD_GET_CHALLENGE_AND_STORE TO_SE_CMD_GET_CHALLENGE_AND_STORE

TODRV_HSE_CMD_VERIFY_CHALLENGE_SIGNATURE TO_SE_CMD_VERIFY_CHALLENGE_SIGNATURE

TODRV_HSE_CMD_VERIFY_CHAIN_CERTIFICATE_AND_STORE_INIT TO_SE_CMD_VERIFY_CHAIN_CERTIFICATE_AND_STORE_INIT

TODRV_HSE_CMD_VERIFY_CHAIN_CERTIFICATE_AND_STORE_UPDATE TO_SE_CMD_VERIFY_CHAIN_CERTIFICATE_AND_STORE_UPDATE

TODRV_HSE_CMD_VERIFY_CHAIN_CERTIFICATE_AND_STORE_FINAL TO_SE_CMD_VERIFY_CHAIN_CERTIFICATE_AND_STORE_FINAL

TODRV_HSE_CMD_VERIFY_CHAIN_CA_CERTIFICATE_AND_STORE_INIT
TO_SE_CMD_VERIFY_CHAIN_CA_CERTIFICATE_AND_STORE_INIT

TODRV_HSE_CMD_VERIFY_CHAIN_CA_CERTIFICATE_AND_STORE_UPDATE
TO_SE_CMD_VERIFY_CHAIN_CA_CERTIFICATE_AND_STORE_UPDATE

TODRV_HSE_CMD_VERIFY_CHAIN_CA_CERTIFICATE_AND_STORE_FINAL
TO_SE_CMD_VERIFY_CHAIN_CA_CERTIFICATE_AND_STORE_FINAL

TODRV_HSE_CMD_COMPUTE_HMAC TO_SE_CMD_COMPUTE_HMAC

TODRV_HSE_CMD_COMPUTE_HMAC_INIT TO_SE_CMD_COMPUTE_HMAC_INIT

TODRV_HSE_CMD_COMPUTE_HMAC_UPDATE TO_SE_CMD_COMPUTE_HMAC_UPDATE

TODRV_HSE_CMD_COMPUTE_HMAC_FINAL TO_SE_CMD_COMPUTE_HMAC_FINAL

TODRV_HSE_CMD_VERIFY_HMAC TO_SE_CMD_VERIFY_HMAC

TODRV_HSE_CMD_VERIFY_HMAC_INIT TO_SE_CMD_VERIFY_HMAC_INIT

TODRV_HSE_CMD_VERIFY_HMAC_UPDATE TO_SE_CMD_VERIFY_HMAC_UPDATE

TODRV_HSE_CMD_VERIFY_HMAC_FINAL TO_SE_CMD_VERIFY_HMAC_FINAL

TODRV_HSE_CMD_AES128CBC_ENCRYPT TO_SE_CMD_AES128CBC_ENCRYPT

TODRV_HSE_CMD_AES128CBC_DECRYPT TO_SE_CMD_AES128CBC_DECRYPT

TODRV_HSE_CMD_AES128CBC_IV_ENCRYPT TO_SE_CMD_AES128CBC_IV_ENCRYPT

TODRV_HSE_CMD_AES128GCM_ENCRYPT TO_SE_CMD_AES128GCM_ENCRYPT

TODRV_HSE_CMD_AES128GCM_DECRYPT TO_SE_CMD_AES128GCM_DECRYPT

TODRV_HSE_CMD_AES128CCM_ENCRYPT TO_SE_CMD_AES128CCM_ENCRYPT

TODRV_HSE_CMD_AES128CCM_DECRYPT TO_SE_CMD_AES128CCM_DECRYPT

TODRV_HSE_CMD_AES128ECB_ENCRYPT TO_SE_CMD_AES128ECB_ENCRYPT

TODRV_HSE_CMD_AES128ECB_DECRYPT TO_SE_CMD_AES128ECB_DECRYPT

TODRV_HSE_CMD_COMPUTE_CMAC TO_SE_CMD_COMPUTE_CMAC

TODRV_HSE_CMD_VERIFY_CMAC TO_SE_CMD_VERIFY_CMAC

TODRV_HSE_CMD_SHA256 TO_SE_CMD_SHA256

TODRV_HSE_CMD_SHA256_INIT TO_SE_CMD_SHA256_INIT

TODRV_HSE_CMD_SHA256_UPDATE TO_SE_CMD_SHA256_UPDATE

TODRV_HSE_CMD_SHA256_FINAL TO_SE_CMD_SHA256_FINAL

TODRV_HSE_CMD_AES128CBC_HMAC_SECURE_MESSAGE
TO_SE_CMD_AES128CBC_HMAC_SECURE_MESSAGE

TODRV_HSE_CMD_AES128CBC_HMAC_UNSECURE_MESSAGE
TO_SE_CMD_AES128CBC_HMAC_UNSECURE_MESSAGE

TODRV_HSE_CMD_AES128CBC_CMAC_SECURE_MESSAGE
TO_SE_CMD_AES128CBC_CMAC_SECURE_MESSAGE

TODRV_HSE_CMD_AES128CBC_CMAC_UNSECURE_MESSAGE
TO_SE_CMD_AES128CBC_CMAC_UNSECURE_MESSAGE

TODRV_HSE_CMD_SET_REMOTE_PUBLIC_KEY TO_SE_CMD_SET_REMOTE_PUBLIC_KEY

TODRV_HSE_CMD_RENEW_ECC_KEYS TO_SE_CMD_RENEW_ECC_KEYS

TODRV_HSE_CMD_GET_PUBLIC_KEY TO_SE_CMD_GET_PUBLIC_KEY

TODRV_HSE_CMD_GET_UNSIGNED_PUBLIC_KEY TO_SE_CMD_GET_UNSIGNED_PUBLIC_KEY

TODRV_HSE_CMD_RENEW_SHARED_KEYS TO_SE_CMD_RENEW_SHARED_KEYS

TODRV_HSE_CMD_GET_KEY_FINGERPRINT TO_SE_CMD_GET_KEY_FINGERPRINT

TODRV_HSE_CMD_TLS_GET_RANDOM_AND_STORE
TO_SE_CMD_TLS_GET_RANDOM_AND_STORE

TODRV_HSE_CMD_TLS_RENEW_KEYS TO_SE_CMD_TLS_RENEW_KEYS

TODRV_HSE_CMD_TLS_GET_MASTER_SECRET TO_CMD_ID_GET_TLS_MASTER_SECRET

TODRV_HSE_CMD_TLS_GET_MASTER_SECRET_DERIVED_KEYS
TO_SE_CMD_TLS_GET_MASTER_SECRET_DERIVED_KEYS

TODRV_HSE_CMD_TLS_SET_SERVER_RANDOM TO_SE_CMD_TLS_SET_SERVER_RANDOM

TODRV_HSE_CMD_TLS_SET_SERVER_EPUBLIC_KEY
TO_SE_CMD_TLS_SET_SERVER_EPUBLIC_KEY

TODRV_HSE_CMD_TLS_RENEW_KEYS_ECDHE TO_SE_CMD_TLS_RENEW_KEYS_ECDHE

TODRV_HSE_CMD_TLS_CALCULATE_FINISHED TO_SE_CMD_TLS_CALCULATE_FINISHED

TODRV_HSE_CMD_TLS_RESET TO_SE_CMD_TLS_RESET

TODRV_HSE_CMD_TLS_SET_MODE TO_SE_CMD_TLS_SET_MODE

TODRV_HSE_CMD_TLS_SET_CONFIG TO_SE_CMD_TLS_SET_CONFIG

TODRV_HSE_CMD_TLS_SET_SESSION TO_SE_CMD_TLS_SET_SESSION

TODRV_HSE_CMD_TLS_SET_CONNECTION_ID_EXT_ID
TO_SE_CMD_TLS_SET_CONNECTION_ID_EXT_ID

TODRV_HSE_CMD_TLS_GET_CLIENT_HELLO TO_SE_CMD_TLS_GET_CLIENT_HELLO

TODRV_HSE_CMD_TLS_GET_CLIENT_HELLO_INIT
TO_SE_CMD_TLS_GET_CLIENT_HELLO_INIT

TODRV_HSE_CMD_TLS_GET_CLIENT_HELLO_UPDATE
TO_SE_CMD_TLS_GET_CLIENT_HELLO_UPDATE

TODRV_HSE_CMD_TLS_GET_CLIENT_HELLO_FINAL
TO_SE_CMD_TLS_GET_CLIENT_HELLO_FINAL

TODRV_HSE_CMD_TLS_HANDLE_HELLO_VERIFY_REQUEST
TO_SE_CMD_TLS_HANDLE_HELLO_VERIFY_REQUEST

TODRV_HSE_CMD_TLS_HANDLE_SERVER_HELLO TO_SE_CMD_TLS_HANDLE_SERVER_HELLO

TODRV_HSE_CMD_TLS_HANDLE_SERVER_HELLO_INIT
TO_SE_CMD_TLS_HANDLE_SERVER_HELLO_INIT

TODRV_HSE_CMD_TLS_HANDLE_SERVER_HELLO_UPDATE
TO_SE_CMD_TLS_HANDLE_SERVER_HELLO_UPDATE

TODRV_HSE_CMD_TLS_HANDLE_SERVER_HELLO_FINAL
TO_SE_CMD_TLS_HANDLE_SERVER_HELLO_FINAL

TODRV_HSE_CMD_TLS_HANDLE_SERVER_CERTIFICATE
TO_SE_CMD_TLS_HANDLE_SERVER_CERTIFICATE

TODRV_HSE_CMD_TLS_HANDLE_SERVER_CERTIFICATE_INIT
TO_SE_CMD_TLS_HANDLE_SERVER_CERTIFICATE_INIT

TODRV_HSE_CMD_TLS_HANDLE_SERVER_CERTIFICATE_UPDATE
TO_SE_CMD_TLS_HANDLE_SERVER_CERTIFICATE_UPDATE

TODRV_HSE_CMD_TLS_HANDLE_SERVER_CERTIFICATE_FINAL
TO_SE_CMD_TLS_HANDLE_SERVER_CERTIFICATE_FINAL

TODRV_HSE_CMD_TLS_HANDLE_SERVER_KEY_EXCHANGE
TO_SE_CMD_TLS_HANDLE_SERVER_KEY_EXCHANGE

TODRV_HSE_CMD_TLS_HANDLE_SERVER_KEY_EXCHANGE_INIT
TO_SE_CMD_TLS_HANDLE_SERVER_KEY_EXCHANGE_INIT

TODRV_HSE_CMD_TLS_HANDLE_SERVER_KEY_EXCHANGE_UPDATE
TO_SE_CMD_TLS_HANDLE_SERVER_KEY_EXCHANGE_UPDATE

TODRV_HSE_CMD_TLS_HANDLE_SERVER_KEY_EXCHANGE_FINAL
TO_SE_CMD_TLS_HANDLE_SERVER_KEY_EXCHANGE_FINAL

TODRV_HSE_CMD_TLS_HANDLE_CERTIFICATE_REQUEST
TO_SE_CMD_TLS_HANDLE_CERTIFICATE_REQUEST

TODRV_HSE_CMD_TLS_HANDLE_SERVER_HELLO_DONE
TO_SE_CMD_TLS_HANDLE_SERVER_HELLO_DONE

TODRV_HSE_CMD_TLS_HANDLE_MEDIATOR_CERTIFICATE
TO_SE_CMD_TLS_HANDLE_MEDIATOR_CERTIFICATE

TODRV_HSE_CMD_TLS_GET_CERTIFICATE TO_SE_CMD_TLS_GET_CERTIFICATE

TODRV_HSE_CMD_TLS_GET_CERTIFICATE_INIT TO_SE_CMD_TLS_GET_CERTIFICATE_INIT

TODRV_HSE_CMD_TLS_GET_CERTIFICATE_UPDATE
TO_SE_CMD_TLS_GET_CERTIFICATE_UPDATE

TODRV_HSE_CMD_TLS_GET_CERTIFICATE_FINAL
TO_SE_CMD_TLS_GET_CERTIFICATE_FINAL

TODRV_HSE_CMD_TLS_GET_CLIENT_KEY_EXCHANGE
TO_SE_CMD_TLS_GET_CLIENT_KEY_EXCHANGE

TODRV_HSE_CMD_TLS_GET_CERTIFICATE_VERIFY
TO_SE_CMD_TLS_GET_CERTIFICATE_VERIFY

TODRV_HSE_CMD_TLS_GET_CHANGE_CIPHER_SPEC
TO_SE_CMD_TLS_GET_CHANGE_CIPHER_SPEC

TODRV_HSE_CMD_TLS_GET_FINISHED TO_SE_CMD_TLS_GET_FINISHED

TODRV_HSE_CMD_TLS_HANDLE_CHANGE_CIPHER_SPEC
TO_SE_CMD_TLS_HANDLE_CHANGE_CIPHER_SPEC

TODRV_HSE_CMD_TLS_HANDLE_FINISHED TO_SE_CMD_TLS_HANDLE_FINISHED

TODRV_HSE_CMD_TLS_GET_CERTIFICATE_SLOT TO_SE_CMD_TLS_GET_CERTIFICATE_SLOT

TODRV_HSE_CMD_TLS_SECURE_MESSAGE TO_SE_CMD_TLS_SECURE_MESSAGE

TODRV_HSE_CMD_TLS_SECURE_MESSAGE_INIT TO_SE_CMD_TLS_SECURE_MESSAGE_INIT

TODRV_HSE_CMD_TLS_SECURE_MESSAGE_UPDATE
TO_SE_CMD_TLS_SECURE_MESSAGE_UPDATE

TODRV_HSE_CMD_TLS_SECURE_MESSAGE_FINAL TO_SE_CMD_TLS_SECURE_MESSAGE_FINAL

TODRV_HSE_CMD_TLS_UNSECURE_MESSAGE TO_SE_CMD_TLS_UNSECURE_MESSAGE

TODRV_HSE_CMD_TLS_UNSECURE_MESSAGE_INIT
TO_SE_CMD_TLS_UNSECURE_MESSAGE_INIT

TODRV_HSE_CMD_TLS_UNSECURE_MESSAGE_UPDATE
TO_SE_CMD_TLS_UNSECURE_MESSAGE_UPDATE

TODRV_HSE_CMD_TLS_UNSECURE_MESSAGE_FINAL
TO_SE_CMD_TLS_UNSECURE_MESSAGE_FINAL

TODRV_HSE_CMD_SECURE_MESSAGE TO_SE_CMD_SECURE_MESSAGE

TODRV_HSE_CMD_SECURE_MESSAGE_INIT TO_SE_CMD_SECURE_MESSAGE_INIT

TODRV_HSE_CMD_SECURE_MESSAGE_UPDATE TO_SE_CMD_SECURE_MESSAGE_UPDATE

TODRV_HSE_CMD_SECURE_MESSAGE_FINAL TO_SE_CMD_SECURE_MESSAGE_FINAL

TODRV_HSE_CMD_UNSECURE_MESSAGE TO_SE_CMD_UNSECURE_MESSAGE

TODRV_HSE_CMD_UNSECURE_MESSAGE_INIT TO_SE_CMD_UNSECURE_MESSAGE_INIT

TODRV_HSE_CMD_UNSECURE_MESSAGE_UPDATE TO_SE_CMD_UNSECURE_MESSAGE_UPDATE

TODRV_HSE_CMD_UNSECURE_MESSAGE_FINAL TO_SE_CMD_UNSECURE_MESSAGE_FINAL

TODRV_HSE_CMD_LORA_GET_APPEUI TO_SE_CMD_LORA_GET_APPEUI

TODRV_HSE_CMD_LORA_GET_DEVEUI TO_SE_CMD_LORA_GET_DEVEUI

TODRV_HSE_CMD_LORA_COMPUTE_MIC TO_SE_CMD_LORA_COMPUTE_MIC

TODRV_HSE_CMD_LORA_ENCRYPT_PAYLOAD TO_SE_CMD_LORA_ENCRYPT_PAYLOAD

TODRV_HSE_CMD_LORA_DECRYPT_JOIN TO_SE_CMD_LORA_DECRYPT_JOIN

TODRV_HSE_CMD_LORA_COMPUTE_SHARED_KEYS
TO_SE_CMD_LORA_COMPUTE_SHARED_KEYS

TODRV_HSE_CMD_LORA_GET_DEVADDR TO_SE_CMD_LORA_GET_DEVADDR

TODRV_HSE_CMD_LORA_GET_JOIN_REQUEST TO_SE_CMD_LORA_GET_JOIN_REQUEST

TODRV_HSE_CMD_LORA_HANDLE_JOIN_ACCEPT TO_SE_CMD_LORA_HANDLE_JOIN_ACCEPT

TODRV_HSE_CMD_LORA_SECURE_PHYPAYLOAD TO_SE_CMD_LORA_SECURE_PHYPAYLOAD

TODRV_HSE_CMD_LORA_UNSECURE_PHYPAYLOAD
TO_SE_CMD_LORA_UNSECURE_PHYPAYLOAD

TODRV_HSE_CMD_SET_PRE_PERSONALIZATION_DATA
TO_SE_CMD_SET_PRE_PERSONALIZATION_DATA

TODRV_HSE_CMD_SET_NEXT_STATE TO_SE_CMD_SET_NEXT_STATE

TODRV_HSE_CMD_GET_STATE TO_SE_CMD_GET_STATE

TODRV_HSE_CMD_ADMIN_SET_SLOT TO_SE_CMD_ADMIN_SET_SLOT

TODRV_HSE_CMD_INIT_ADMIN_SESSION TO_SE_CMD_INIT_ADMIN_SESSION

TODRV_HSE_CMD_AUTH_ADMIN_SESSION TO_SE_CMD_AUTH_ADMIN_SESSION

TODRV_HSE_CMD_FINI_ADMIN_SESSION TO_SE_CMD_FINI_ADMIN_SESSION

TODRV_HSE_CMD_ADMIN_COMMAND TO_SE_CMD_ADMIN_COMMAND

TODRV_HSE_CMD_ADMIN_COMMAND_WITH_RESPONSE TO_SE_CMD_ADMIN_COMMAND_WITH_RESPONSE

TODRV_HSE_CMD_LOCK TO_SE_CMD_LOCK

TODRV_HSE_CMD_UNLOCK TO_SE_CMD_UNLOCK

TODRV_HSE_CMD_SET_AES_KEY TO_SE_CMD_SET_AES_KEY

TODRV_HSE_CMD_SET_HMAC_KEY TO_SE_CMD_SET_HMAC_KEY

TODRV_HSE_CMD_SET_CMAC_KEY TO_SE_CMD_SET_CMAC_KEY

TODRV_HSE_CMD_SECLINK_ARC4 TO_SE_CMD_SECLINK_ARC4

TODRV_HSE_CMD_SECLINK_ARC4_GET_IV TO_SE_CMD_SECLINK_ARC4_GET_IV

TODRV_HSE_CMD_SECLINK_ARC4_GET_NEW_KEY TO_SE_CMD_SECLINK_ARC4_GET_NEW_KEY

TODRV_HSE_CMD_SECLINK_AESHMAC TO_SE_CMD_SECLINK_AESHMAC

TODRV_HSE_CMD_SECLINK_AESHMAC_GET_IV TO_SE_CMD_SECLINK_AESHMAC_GET_IV

TODRV_HSE_CMD_SECLINK_AESHMAC_GET_NEW_KEYS TO_SE_CMD_SECLINK_AESHMAC_GET_NEW_KEYS

TODRV_HSE_CMD_LOADER_BCAST_GET_INFO TO_SE_CMD_LOADER_BCAST_GET_INFO

TODRV_HSE_CMD_LOADER_BCAST_RESTORE TO_SE_CMD_LOADER_BCAST_RESTORE

TODRV_HSE_CMD_LOADER_BCAST_INITIALIZE_UPGRADE TO_SE_CMD_LOADER_BCAST_INITIALIZE_UPGRADE

TODRV_HSE_CMD_LOADER_BCAST_WRITE_DATA TO_SE_CMD_LOADER_BCAST_WRITE_DATA

TODRV_HSE_CMD_LOADER_BCAST_COMMIT_RELEASE TO_SE_CMD_LOADER_BCAST_COMMIT_RELEASE

TODRV_HSE_CMD_DATA_MIGRATION TO_SE_CMD_DATA_MIGRATION

TODRV_HSE_CMD_SET_MEASURE_BOOT TO_SE_CMD_SET_MEASURE_BOOT

TODRV_HSE_CMD_VALIDATE_NEW_FW_HASH TO_SE_CMD_VALIDATE_NEW_FW_HASH

TODRV_HSE_CMD_COMMIT_NEW_FW_HASH TO_SE_CMD_COMMIT_NEW_FW_HASH

TODRV_HSE_CMD_STORE_NEW_TRUSTED_FW_HASH TO_SE_CMD_STORE_NEW_TRUSTED_FW_HASH

TODRV_HSE_CMD_GET_BOOT_MEASUREMENT TO_SE_CMD_GET_BOOT_MEASUREMENT

TODRV_HSE_CMD_GET_SE_MEASUREMENT TO_SE_CMD_GET_SE_MEASUREMENT

TODRV_HSE_CMD_INVALIDATE_NEW_HASH TO_SE_CMD_INVALIDATE_NEW_HASH

6.5.9 Generic context

group `tose_defs`

Typedefs

typedef struct *TOSE_drv_ctx_s* **TOSE_drv_ctx_t**

Context structure for Secure Elements (Hardware / Software / other)

typedef struct *TOSE_ctx_s* **TOSE_ctx_t**

Context structure for Secure Elements (Hardware / Software / other) struct **TOSE_drv_ctx_s**

#include <TO_defs.h> Context structure for Secure Elements (Hardware / Software / other)

Public Members

const struct TODRV_api_s ***api**

Driver API

uint32_t **func_offset**

Offset to add to driver API functions

TO_log_ctx_t ***log_ctx**

Running-platform specific log function

void ***priv_ctx**

Driver private context

struct **TOSE_ctx_s**

#include <TO_defs.h> Context structure for Secure Elements (Hardware / Software / other)

Public Members

TOSE_drv_ctx_t ***drv**

Driver context

uint8_t **initialized**

Context initialization state

6.5.10 Log levels

group **log_level**

Defines

TO_LOG_LEVEL_NONE -1

Log levels

TO_LOG_LEVEL_ERR 0

TO_LOG_LEVEL_WRN 1

TO_LOG_LEVEL_INF 2

TO_LOG_LEVEL_DBG 3

TO_LOG_LEVEL_MASK 0x0f

TO_LOG_STRING 0x00

TO_LOG_BUFFER 0x10

TO_LOG_HEX_DISP 0x20

TO_LOG_TRACE 0x30

Typedefs

typedef struct *TO_log_ctx_s* **TO_log_ctx_t**

The Log context, propagated to all layers.

void() TO_log_func_t (TO_log_ctx_t *log_ctx, const TO_log_level_t level, void *ptr,...)

Pointer to a log function.

This function will be responsible for displaying/processing logs in a way suitable for your application.

Enums

enum **TO_log_level_e**

Different log levels that are available to the application.

Values:

Functions

`enum TO_log_level_e __attribute__ ((packed)) TO_log_level_t`

Different log levels that are available to the application.

void **`print_log_function`**(const TO_log_level_t level, const char *log)

Default print log function, potentialy to be customized per-target.

Depending on your target and the way to send string messages out, you may have to rewrite it. In this case, just declare a function having the same names/parameters, printing-out the messages.

Parameters

- **`level`** – Importance level of the message
- **`log`** – String to be displayed

void **`TO_log`**(*TO_log_ctx_t* *log_ctx, const TO_log_level_t level, void *ptr, ...)

Default LOG display function.

Parameters

- **`log_ctx`** – The LOG context
- **`level`** – The desired log display level
- **`ptr`** – Pointer to the string (mandatory parameter)

void **`TO_set_log_level`**(*TO_log_ctx_t* *log_ctx, const TO_log_level_t level, TO_log_func_t *log_function)

Sets the Log function and log level.

This function permits to change the log level and the log function.

Parameters

- **`log_ctx`** – Current log context
- **`level`** – Desired log level
- **`log_function`** – Log function (eg. TO_log)

TO_log_ctx_t ***`TO_log_get_ctx`**(void)

Get the LOG context.

This function is weak, and can be replaced by your own implementation.

Returns

TO_log_ctx_t*

struct **`TO_log_ctx_s`**

#include <TO_log.h> The Log context, propagated to all layers.

Public Members

TO_log_func_t ***log_function**

Pointer to a log function

TO_log_level_t **log_level**

Dynamic level management

6.5.11 Miscellaneous constants

group **constants**

Misc constants

Defines

TO_INDEX_SIZE 1UL

TO_FORMAT_SIZE 1UL

TO_AES_BLOCK_SIZE 16UL

TO_INITIALVECTOR_SIZE TO_AES_BLOCK_SIZE

TO_AES_KEYSIZE 16UL

TO_AESGCM_INITIALVECTOR_SIZE 12UL

TO_AESGCM_TAG_SIZE 16UL

TO_AESGCM_AAD_LEN_SIZE 2UL

TO_AESCCM_NONCE_SIZE 13UL

TO_AESCCM_TAG_SIZE 16UL

TO_AESCCM_8_TAG_SIZE 8UL

TO_AESCCM_AAD_LEN_SIZE 2UL

TO_HMAC_KEYSIZE 16UL

TO_HMAC_SIZE TO_SHA256_HASHSIZE

TO_HMAC_MINSIZE 10UL

TO_CMAC_KEYSIZE 16UL

TO_CMAC_SIZE TO_AES_BLOCK_SIZE

TO_CMAC_MIN_SIZE 4UL

TO_SEQUENCE_SIZE 4UL

TO_SHA256_HASHSIZE 32UL

TO_HASH_SIZE TO_SHA256_HASHSIZE

TO_CHALLENGE_SIZE 32UL

TO_KEY_FINGERPRINT_SIZE 3UL

TO_TIMESTAMP_SIZE 4UL

TO_CRC_SIZE 2UL

TO_SN_SIZE (TO_SN_CA_ID_SIZE+TO_SN_NB_SIZE)

TO_HW_SN_SIZE 23UL

TO_SN_CA_ID_SIZE 3UL

TO_SN_NB_SIZE 5UL

TO_PN_SIZE 12UL

TO_HW_VERSION_SIZE 2UL

TO_SW_VERSION_SIZE 3UL

TO_PRODUCT_ID_SIZE 15UL

TO_SEED_SIZE 32UL

6.5.12 LoRaWAN constants

group **lora_constants**

LoRa constants

Defines

TO_LORA_PHYPAYLOAD_MINSIZE 10UL

TO_LORA_MHDR_SIZE 1UL

TO_LORA_APPEUI_SIZE 8UL

TO_LORA_DEVEUI_SIZE 8UL

TO_LORA_DEVADDR_SIZE 4UL

TO_LORA_DEVNONCE_SIZE 2UL

TO_LORA_APPNONCE_SIZE 3UL

TO_LORA_NETID_SIZE 3UL

TO_LORA_MIC_SIZE 4UL

TO_LORA_FCTRL_SIZE 1UL

TO_LORA_FCNT_SIZE 4UL

TO_LORA_APPKEY_SIZE 16UL

TO_LORA_APPSKEY_SIZE 16UL

TO_LORA_NWKSKEY_SIZE 16UL

TO_LORA_DLSETTINGS_SIZE 1UL

TO_LORA_RXDELAY_SIZE 1UL

TO_LORA_CFLIST_SIZE 16UL

TO_LORA_JOINREQUEST_SIZE (TO_LORA_MHDR_SIZE + TO_LORA_APPEUI_SIZE + TO_LORA_DEVEUI_SIZE + TO_LORA_DEVNONCE_SIZE + TO_LORA_MIC_SIZE)

TO_LORA_JOINACCEPT_CLEAR_MAXSIZE (TO_LORA_MHDR_SIZE + TO_LORA_APPNONCE_SIZE + TO_LORA_NETID_SIZE + TO_LORA_DEVADDR_SIZE + TO_LORA_DLSETTINGS_SIZE + TO_LORA_RXDELAY_SIZE + TO_LORA_CFLIST_SIZE)

TO_LORA_JOINACCEPT_MAXSIZE (TO_LORA_JOINACCEPT_CLEAR_MAXSIZE + TO_LORA_MIC_SIZE)

6.5.13 Certificates constants

group **cert_constants**

Certificate constants

Defines

TO_CERTIFICATE_SIZE (TO_SN_SIZE+TO_ECC_PUB_KEYSIZE+TO_SIGNATURE_SIZE)

TO_CERT_PRIVKEY_SIZE 32UL

TO_ECC_PRIV_KEYSIZE TO_CERT_PRIVKEY_SIZE

TO_ECC_PUB_KEYSIZE (2*TO_ECC_PRIV_KEYSIZE)

TO_SIGNATURE_SIZE TO_ECC_PUB_KEYSIZE

TO_CERT_GENERALIZED_TIME_SIZE 15UL /* YYYYMMDDHHMMSSZ */

TO_CERT_DATE_SIZE ((TO_CERT_GENERALIZED_TIME_SIZE - 1) / 2)

TO_CERT_SUBJECT_PREFIX_SIZE 15UL

TO_SHORTV2_CERT_SIZE (TO_CERTIFICATE_SIZE + TO_CERT_DATE_SIZE)

TO_REMOTE_CERTIFICATE_SIZE (TO_SN_SIZE+TO_ECC_PUB_KEYSIZE)

TO_REMOTE_CAID_SIZE TO_SN_CA_ID_SIZE

TO_CERT_SUBJECT_CN_MAXSIZE 64UL

TO_CERT_SUBJECT_CN_PREFIX_MAXSIZE (TO_CERT_SUBJECT_CN_MAXSIZE - TO_SN_SIZE * 2)

TO_CERT_DN_MAXSIZE 127UL

TO_KEYTYPE_SIZE TO_SN_CA_ID_SIZE

TO_CA_PUBKEY_SIZE TO_ECC_PUB_KEYSIZE

TO_CA_PUBKEY_CAID_SIZE TO_SN_CA_ID_SIZE

TO_KEY_IDENTIFIER_SIZE 20UL

TO_KEY_IDENTIFIER_SHORT_SIZE 8UL

6.5.14 TLS constants

group **tls_constants**

TLS constants

Defines

TO_TLS_RECORD_CIPHER_OVERHEAD_MAX (352UL)

maximum data overhead between a protected record and its plain text version

The theorical maximum data overhead between a protected record and its plain text version is 1024 but in practice it should not exceed AES256 with SHA512 in CBC mode with max padding: (32 (IV) + 64 (hmac) + 256 (max padding)) => 352 This value is used when the real overhead is not known, to determine if a plain text record to send need to be fragmented before encryption

TO_TLS_MAJOR 3

TO_TLS_MINOR 3

TO_TLS_RECORD_MAX_SIZE (1UL « 14)

TO_TLS_RANDOM_SIZE (TO_TIMESTAMP_SIZE + 28UL)

TO_TLS_MASTER_SECRET_SIZE 48UL

TO_TLS_SERVER_PARAMS_SIZE 69UL

TO_TLS_HMAC_KEYSIZE 32UL

TO_TLS_FINISHED_SIZE 12UL

TO_TLS_CHANGE_CIPHER_SPEC_SIZE 1UL

TO_TLS_HEADER_SIZE 5UL

TO_TLS_HANDSHAKE_HEADER_SIZE 4UL

TO_TLS_AEAD_IMPLICIT_NONCE_SIZE 4UL

TO_TLS_AEAD_EXPLICIT_NONCE_SIZE 8UL

TO_TLS_SNI_LENGTH_MAX 253U

TO_DTLS_MAJOR 254

TO_DTLS_MINOR 253

TO_DTLS_CONNECTION_ID_MAXSIZE 8UL

TO_DTLS_HEADER_SIZE 13UL

TO_DTLS_HEADER_MAXSIZE (TO_DTLS_HEADER_SIZE + TO_DTLS_CONNECTION_ID_MAXSIZE)

TO_DTLS_HANDSHAKE_HEADER_SIZE 12UL

TO_DTLS_HANDSHAKE_HEADER_MAXSIZE (TO_DTLS_HANDSHAKE_HEADER_SIZE + TO_DTLS_CONNECTION_ID_MAXSIZE)

Typedefs

typedef enum *TO_tls_mode_e* **TO_tls_mode_t**
Different modes available for TO-Protect TLS.

typedef enum *TO_tls_config_id_e* **TO_tls_config_id_t**

typedef enum *TO_tls_record_type_e* **TO_tls_record_type_t**

typedef enum *TO_tls_cipher_suite_e* **TO_tls_cipher_suite_t**

typedef enum *TO_tls_cipher_suite_type_e* **TO_tls_cipher_suite_type_t**

typedef enum *TO_tls_encryption_type_e* **TO_tls_encryption_type_t**

typedef enum *TO_tls_handshake_type_e* **TO_tls_handshake_type_t**

typedef enum *TO_tls_state_e* **TO_tls_state_t**

typedef enum *TO_tls_extensions_e* **TO_tls_extension_t**

Enums

enum **TO_tls_mode_e**
Different modes available for TO-Protect TLS.
Values:

enumerator **TO_TLS_MODE_UNKNOWN**
Unknown mode (uninitialized)

enumerator **TO_TLS_MODE_HANDSHAKE_ONLY**
Handshake Only mode

enumerator **TO_TLS_MODE_TLS_1_2**
TLS 1.2 only

enumerator **TO_TLS_MODE_TLS_1_2_HANDSHAKE_ONLY**
TLS 1.2, in Handshake only

enumerator **TO_TLS_MODE_DTLS_1_2**
DTLS 1.2 only

enumerator **TO_TLS_MODE_DTLS_1_2_HANDSHAKE_ONLY**

DTLS 1.2, in Handshake only

enum **TO_tls_config_id_e**

Values:

enumerator **TO_TLS_CONFIG_ID_UNKNOWN**

enumerator **TO_TLS_CONFIG_ID_MODE**

Configure mode on 1 byte. See TO_tls_mode_e.

enumerator **TO_TLS_CONFIG_ID_CIPHER_SUITES**

Configure cipher suites list (each cipher suite on 2 bytes, big-endian)

enumerator **TO_TLS_CONFIG_ID_CERTIFICATE_SLOT**

Configure certificate slot

enumerator **TO_TLS_CONFIG_ID_MAX**

enumerator **TO_TLS_CONFIG_ID_LAST**

enum **TO_tls_record_type_e**

Values:

enumerator **TO_TLS_RECORD_TYPE_CHANGE_CIPHER_SPEC**

enumerator **TO_TLS_RECORD_TYPE_ALERT**

enumerator **TO_TLS_RECORD_TYPE_HANDSHAKE**

enumerator **TO_TLS_RECORD_TYPE_APPLICATION_DATA**

enumerator **TO_TLS_RECORD_TYPE_TLS_12_CID**

enum **TO_tls_cipher_suite_e**

Values:

enumerator **TO_TLS_PSK_WITH_AES_128_CBC_SHA256**

enumerator **TO_TLS_PSK_WITH_AES_128_CCM** enumerator **TO_TLS_PSK_WITH_AES_128_CCM_8**

enumerator **TO_TLS_PSK_WITH_AES_128_GCM_SHA256**

enumerator **TO_TLS_ECDHE_ECDSA_WITH_AES_128_CBC_SHA256**

enumerator **TO_TLS_ECDHE_ECDSA_WITH_AES_128_CCM**

enumerator **TO_TLS_ECDHE_ECDSA_WITH_AES_128_CCM_8**

enumerator **TO_TLS_ECDHE_ECDSA_WITH_AES_128_GCM_SHA256**

enumerator **TO_TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA256**

enumerator **TO_TLS_ECDHE_RSA_WITH_AES_128_GCM_SHA256**

enum **TO_tls_cipher_suite_type_e**

Values:

enumerator **TO_TLS_CIPHER_SUITE_ECDHE**

enumerator **TO_TLS_CIPHER_SUITE_PSK**

enum **TO_tls_encryption_type_e**

Values:

enumerator **TO_TLS_ENCRYPTION_AES_CBC**

enumerator **TO_TLS_ENCRYPTION_AES_CCM**

enumerator **TO_TLS_ENCRYPTION_AES_CCM_8**

enumerator **TO_TLS_ENCRYPTION_AES_GCM**

enum **TO_tls_handshake_type_e**

Values:

enumerator **TO_TLS_HANDSHAKE_TYPE_CLIENT_HELLO**

enumerator **TO_TLS_HANDSHAKE_TYPE_SERVER_HELLO** enumerator **TO_TLS_HANDSHAKE_TYPE_HELLO_VERIFY_REQUEST**

enumerator **TO_TLS_HANDSHAKE_TYPE_CERTIFICATE**

enumerator **TO_TLS_HANDSHAKE_TYPE_SERVER_KEY_EXCHANGE**

enumerator **TO_TLS_HANDSHAKE_TYPE_CERTIFICATE_REQUEST**

enumerator **TO_TLS_HANDSHAKE_TYPE_SERVER_HELLO_DONE**

enumerator **TO_TLS_HANDSHAKE_TYPE_CERTIFICATE_VERIFY**

enumerator **TO_TLS_HANDSHAKE_TYPE_CLIENT_KEY_EXCHANGE**

enumerator **TO_TLS_HANDSHAKE_TYPE_FINISHED**

enumerator **TO_TLS_HANDSHAKE_TYPE_MEDIATOR_CERTIFICATE**

enum **TO_tls_state_e**

Values:

enumerator **TO_TLS_STATE_HANDSHAKE_START**

enumerator **TO_TLS_STATE_FLIGHT_1**

enumerator **TO_TLS_STATE_CLIENT_HELLO**

enumerator **TO_TLS_STATE_FLIGHT_1_INIT**

enumerator **TO_TLS_STATE_FLIGHT_2**

enumerator **TO_TLS_STATE_SERVER_HELLO_VERIFY_REQUEST**

enumerator **TO_TLS_STATE_FLIGHT_2_INIT**

enumerator **TO_TLS_STATE_FLIGHT_3**

enumerator **TO_TLS_STATE_CLIENT_HELLO_WITH_COOKIE**

enumerator **TO_TLS_STATE_FLIGHT_3_INIT** enumerator **TO_TLS_STATE_FLIGHT_4**

enumerator **TO_TLS_STATE_SERVER_HELLO**

enumerator **TO_TLS_STATE_FLIGHT_4_INIT**

enumerator **TO_TLS_STATE_SERVER_CERTIFICATE**

enumerator **TO_TLS_STATE_SERVER_KEY_EXCHANGE**

enumerator **TO_TLS_STATE_SERVER_CERTIFICATE_REQUEST**

enumerator **TO_TLS_STATE_SERVER_HELLO_DONE**

enumerator **TO_TLS_STATE_MEDIATOR_CERTIFICATE**

enumerator **TO_TLS_STATE_FLIGHT_5**

enumerator **TO_TLS_STATE_CLIENT_CERTIFICATE**

enumerator **TO_TLS_STATE_FLIGHT_5_INIT**

enumerator **TO_TLS_STATE_CLIENT_KEY_EXCHANGE**

enumerator **TO_TLS_STATE_FLIGHT_5_INIT_NO_CLIENT_AUTH**

enumerator **TO_TLS_STATE_CLIENT_CERTIFICATE_VERIFY**

enumerator **TO_TLS_STATE_CLIENT_CHANGE_CIPHER_SPEC**

enumerator **TO_TLS_STATE_CLIENT_FINISHED**

enumerator **TO_TLS_STATE_FLIGHT_6**

enumerator **TO_TLS_STATE_SERVER_CHANGE_CIPHER_SPEC**

enumerator **TO_TLS_STATE_FLIGHT_6_INIT**

enumerator **TO_TLS_STATE_SERVER_FINISHED** enumerator **TO_TLS_STATE_HANDSHAKE_DONE**

enumerator **TO_TLS_STATE_HANDSHAKE_FAILED**

enumerator **TO_TLS_STATE_FATAL_RECEIVED**

enumerator **TO_TLS_STATE_CLOSE_RECEIVED**

enum **TO_tls_extensions_e**

Values:

enumerator **TO_TLS_EXTENSION_SERVER_NAME**

enumerator **TO_TLS_EXTENSION_SIG_ALG**

enumerator **TO_TLS_EXTENSION_ECC**

enumerator **TO_TLS_EXTENSION_ECC_POINT_FORMAT**

enumerator **TO_TLS_EXTENSION_TRUNCATED_HMAC**

enumerator **TO_TLS_EXTENSION_CONNECTION_ID**

enumerator **TO_TLS_EXTENSION_DUMMY_MAX**

6.5.15 I2C constants

group **i2c_constants**

I2C constants

Defines

TO_I2CADDR_SIZE 1UL

TO_I2C_SEND_MSTIMEOUT TO_I2C_MSTIMEOUT

TO_I2C_RECV_MSTIMEOUT TO_I2C_MSTIMEOUT

TO_I2C_MSTIMEOUT 5000UL

`TO_I2C_RESPONSE_MSTIMEOUT` 10000UL

`TO_I2C_ERROR_MSTIMEOUT` 10000UL

6.5.16 Status PIO constants

group **`status_pio_constants`**

Status PIO constants

Defines

`TO_STATUS_PIO_ENABLE` 0x80

`TO_STATUS_PIO_READY_LEVEL_MASK` 0x01

`TO_STATUS_PIO_HIGH_OPENDRAIN_MASK` 0x02

`TO_STATUS_PIO_IDLE_HZ_MASK` 0x04

6.5.17 Seclink constants

group **`seclink_constants`**

Seclink constants

Defines

`TO_ARC4_KEY_SIZE` 16UL

`TO_ARC4_INITIALVECTOR_SIZE` 16UL

6.5.18 Admin constants

group **`admin_constants`**

Admin constants

Defines

TO_ADMIN_DIVERS_DATA_SIZE TO_SN_SIZE

TO_ADMIN_PROTO_INFO_SIZE 4UL

TO_ADMIN_OPTIONS_SIZE 2UL

TO_ADMIN_CHALLENGE_SIZE 8UL

TO_ADMIN_CRYPTOGRAM_SIZE 8UL

TO_ADMIN_MAC_SIZE 8UL

TO_ADMIN_DATAIDX_SIZE 4

7. Miscellany guides

7.1 Production optimizations

7.1.1 Features

In production environment, all unneeded features must be disabled using corresponding configuration flags.

7.1.2 Internal buffers

Internal buffers sizes must be reduced to fit the biggest used size in order to optimize library footprint.

For example, in a use-case using only AES on cryptograms of maximum 128 bytes, IO buffer can be reduced to 5 (header) + 16 (iv) + 128 (data) bytes.

7.1.3 Logs

Logs must be disabled as it can slow down performances, and increase the footprint. It can be done by setting *TO_LOG_LEVEL_MAX* at disabled state.

Note: Changing log level at runtime with `TO_set_log_level()` will fix performance issue but will not reduce the footprint.

See *Library configuration for an MCU project* to properly configure libTO.

7.2 Migration

7.2.1 TO library migration guide from 5.7.x to 5.8.x

The following changes are to be taken into account to update from 5.7.x to 5.8.x.

TLS helper `TOSE_helper_tls_cleanup()` function is splitted in 2 new functions:

- `TOSE_helper_tls_close()`
- `TOSE_helper_tls_fini()`

Note: `TOSE_helper_tls_cleanup()` can still be used.

7.2.2 TO library migration guide from 5.6.x to 5.7.x

The following changes are to be taken into account to update from 5.6.x to 5.7.x.

CSR function *TOSE_set_certificate_x509()* is now also available with helper *TOSE_helper_set_certificate_x509()* when it needs to be used with a small I2C buffer.

7.2.3 TO library migration guide from 5.5.x to 5.6.x

The following changes are to be taken into account to update from 5.5.x to 5.6.x.

New function *TOSE_get_product_id()* to get product ID.

7.2.4 TO library migration guide from 5.4.x to 5.5.x

The following changes are to be taken into account to update from 5.4.x to 5.5.x.

A user configuration file can now be included by libTO using TO_CONFIG_FILE define.

7.2.5 TO library migration guide from 5.3.x to 5.4.x

The following changes are to be taken into account to update from 5.3.x to 5.4.x.

API `TOSE_helper_tls_init()` is now deprecated, as it uses a free TLS session independently of session characteristics. As TLS sessions can now have different storage characteristics (NVM/RAM/hybrid), it is mandatory to explicitly select which session to use. The new API *TOSE_helper_tls_init_session()* must now be used to initialize TLS sessions.

7.2.6 TO library migration guide from 5.2.x to 5.3.x

The following changes are to be taken into account to update from 5.2.x to 5.3.x.

7.2.7 TO library migration guide from 5.1.x to 5.2.x

The following changes are to be taken into account to update from 5.1.x to 5.2.x.

7.2.8 TO library migration guide from 5.0.x to 5.1.x

The following changes are to be taken into account to update from 5.0.x to 5.1.x.

7.2.9 TO library migration guide from 4.x.x to 5.x.x

The following changes are to be taken into account to update from 4.x.x to 5.x.x.

Old APIs are now deprecated, but still defined and mapped on new APIs by including TO.h.

7.2.9.1 Internal architecture changes

The source tree structure has changed:

- Core API based on Trusted Objects Secure Element has been moved from *src/to/* to *src*
- Wrappers directory has been moved from *src/to/wrapper/* to *src/wrapper/*.

7.2.9.2 Generic APIs renames

Generic **TO_()** functions have been renamed to **TOSE_()**.

New **TOSE_()** APIs now take a context parameter, given by driver function *TODRV_HSE_get_ctx()*.

7.2.9.3 Hardware Secure Element renames

Hardware Secure Element specific functions have need renamed to **TODRV_HSE_**:

- **TO_config()** renamed to *TODRV_HSE_trp_config()*
- **TO_last_command_duration()** renamed to *TODRV_HSE_trp_last_command_duration()*
- **TO_read()** renamed to *TODRV_HSE_trp_read()*
- **TO_write()** renamed to *TODRV_HSE_trp_write()*
- **TO_seclink_()** renamed to `TODRV_HSE_seclink_...()`
- **TO_prepare_command_data()** renamed to *TODRV_HSE_prepare_command_data()*
- **TO_prepare_command_data_byte()** renamed to *TODRV_HSE_prepare_command_data_byte()*
- **TO_set_command_data()** renamed to *TODRV_HSE_set_command_data()*
- **TO_send_command()** renamed to *TODRV_HSE_send_command()*
- **TO_set_lib_hook_pre_command()** renamed to *TODRV_HSE_set_lib_hook_pre_command()*
- **TO_set_lib_hook_post_write()** renamed to *TODRV_HSE_set_lib_hook_post_write()*
- **TO_set_lib_hook_post_command()** renamed to *TODRV_HSE_set_lib_hook_post_command()*

Hardware Secure Element specific files have need renamed to **TODRV_HSE_**:

- **TO_i2c_wrapper.h** renamed to **TODRV_HSE_i2c_wrapper.h**
- **TO_hooks.h** renamed to **TODRV_HSE_hooks.h**

7.2.10 TO library migration guide from 4.18.x to 4.19.x

The following changes are to be taken into account to update from 4.18.x to 4.19.x.

7.2.11 TO library migration guide from 4.17.x to 4.18.x

The following changes are to be taken into account to update from 4.17.x to 4.18.x.

7.2.12 TO library migration guide from 4.16.x to 4.17.x

The following changes are to be taken into account to update from 4.16.x to 4.17.x.

TLS standard API function *TO_renew_tls_keys* has been removed, as non-ephemeral TLS cipher suites are now considered weak.

TLS standard API is now enabled by default with autotools.

7.2.13 TO library migration guide from 4.15.x to 4.16.x

The following changes are to be taken into account to update from 4.15.x to 4.16.x.

Following TLS helper functions (dedicated to DTLS) are now compiled when *TO_ENABLE_DTLS* is set:

- `TO_helper_tls_set_retransmission_timeout()`
- `TO_helper_tls_set_fragment_max_size()`

7.2.14 TO library migration guide from 4.14.x to 4.15.x

The following changes are to be taken into account to update from 4.14.x to 4.15.x.

7.2.15 TO library migration guide from 4.13.x to 4.14.x

The following changes are to be taken into account to update from 4.13.x to 4.14.x.

7.2.16 TO library migration guide from 4.12.x to 4.13.x

The following changes are to be taken into account to update from 4.12.x to 4.13.x.

7.2.17 TO library migration guide from 4.11.x to 4.12.x

The following changes are to be taken into account to update from 4.11.x to 4.12.x.

Helpers now returns errors codes with least significant byte corresponding to command error if it is the source of the error.

Errors originally tested with:

```
if (ret == TO_ERROR) {
```

Must now be tested using *TO_LIB_ERRCODE()*:

```
if (TO_LIB_ERRCODE(ret) == TO_ERROR) {
```

Secure Element error codes can be extracted using *TO_SE_ERRCODE()*:

```
se_ret = TO_SE_ERRCODE(ret);
```

7.2.18 TO library migration guide from 4.10.x to 4.11.x

The following changes are to be taken into account to update from 4.10.x to 4.11.x.

7.2.19 TO library migration guide from 4.9.x to 4.10.x

The following changes are to be taken into account to update from 4.9.x to 4.10.x.

The *old secure messaging API* and old secure messaging helper API are now deprecated.

Secure messaging APIs changed to new payload based API and *new payload based helper API*.

7.2.20 TO library migration guide from 4.8.x to 4.9.x

The following changes are to be taken into account to update from 4.8.x to 4.9.x.

New secure messaging APIs are available, with following features:

- Cumulative APIs (any data length supported)
- Replay attacks protection

Use *helpers function* for simple usage.

Retrocompatibility alias of functions *TO_secure_message* and *TO_unsecure_message* has been removed as these functions are part of new implementation (`TO_secure_message()` and `TO_unsecure_message()`).

7.2.21 TO library migration guide from 4.6.x to 4.7.x

The following changes are to be taken into account to update from 4.6.x to 4.7.x.

Internal architecture changed:

- Core API based on Trusted Objects Secure Element has been moved to *src/to/*.
- Wrappers directory has been moved to *src/to/wrapper/*.

TLS helper API changed to *new context based API* supporting multiple sessions. Old API is now deprecated.

New API `TO_helper_tls_handshake_cleanup()` has been added to old deprecated API. This function needs to be called to each successfull `TO_helper_tls_handshake_init()`.

7.2.21.1 Preprocessor defines (MCU project)

Define *TO_DEBUG* is no more used.

Logs are now enabled by default at warning level. Maximum level is controllable by setting define *TO_LOG_LEVEL_MAX*. All logs with higher log level will not be compiled (no possibility to enable it at runtime).

See *Library configuration for an MCU project*. to properly configure libTO.

7.2.22 TO library migration guide from 4.5.x to 4.6.x

The following changes are to be taken into account to update from 4.5.x to 4.6.x.

7.2.22.1 TLS helper API

size_t and *ssize_t* have been replaced respectively by *uint32_t* and *int32_t*.

Impacted functions are :

- `TO_helper_tls_send_message()`
- `TO_helper_tls_handshake_send_func`
- `TO_helper_tls_receive_message()`
- `TO_helper_tls_handshake_receive_func`

7.2.23 TO library migration guide from 4.4.x to 4.5.x

The following changes are to be taken into account to update from 4.4.x to 4.5.x.

7.2.24 TO library migration guide from 4.3.x to 4.4.x

The following changes are to be taken into account to update from 4.3.x to 4.4.x.

TLS and LoRa features have been enabled by default. Then, if you dont need TLS or LoRa in your project, you now have to explicitely disable these features.

DTLS remains disabled and has to be explicitely enabled if needed.

7.2.24.1 Preprocessor defines (MCU project)

The following definitions are useless because this is now the default setting:

- TO_ENABLE_LORA
- TO_ENABLE_LORA_OPTIMIZED
- TO_ENABLE_TLS
- TO_ENABLE_TLS_OPTIMIZED
- TO_ENABLE_TLS_HELPER

If not required, to disable these features for your project consider using the following definitions:

- TO_DISABLE_LORA
- TO_DISABLE_LORA_OPTIMIZED
- TO_DISABLE_TLS
- TO_DISABLE_TLS_OPTIMIZED
- TO_DISABLE_TLS_HELPER

The following defines have been renamed:

Old name	New name
TLS_IO_BUFFER_SIZE	TO_TLS_IO_BUFFER_SIZE
TLS_FLIGHT_BUFFER_SIZE	TO_TLS_FLIGHT_BUFFER_SIZE

See *Library configuration for an MCU project*. to properly configure libTO.

7.2.25 TO library migration guide from 4.1.x to 4.2.x

The following changes are to be taken into account to update from 4.1.x to 4.2.x.

7.2.25.1 Changed APIs

The API `TO_tls_get_certificate()` has changed, with a new length output parameter.

7.2.26 TO library migration guide from 4.0.x to 4.1.x

The following changes are to be taken into account to update from 4.0.x to 4.1.x.

7.2.26.1 Renamed files

The library core files, *src/main.c* and *src/main.h*, has been renamed *src/core.c* and *src/core.h*.

7.2.27 TO library migration guide from 3.x.x to 4.x.x

The following changes are to be taken into account to update from 3.x.x to 4.x.x.

7.2.27.1 Renamed APIs

The following header files have been renamed:

- **include/to136.h** to **include/TO.h**
- **include/to136_defs.h** to **TO_defs.h**
- **include/to136_helper.h** to **include/TO_helper.h**
- **include/to136_i2c_wrapper.h** to **include/TO_i2c_wrapper.h**

TO136_() functions have been renamed to **TO_()**.

TO136_ definitions have been renamed to **TO_**.

to136_ structures, types and enums have been renamed to **TO_**.

7.2.27.2 Preprocessor flags

ENABLE_/DISABLE_ flags have been renamed to **TO_ENABLE_/TO_DISABLE_**.

*TO_USE_** flags have been renamed to **TO_ENABLE_**.

Removed **USE_ECIES___SIGNATURE** flags.

7.2.27.3 Error codes

TO_OK (previously TO136_OK) have its value changed from **1** to **0x0000**. This change was motivated to always keep LSB free to code Secure Element error codes.

7.2.28 TO library migration guide from 2.x.x to 3.x.x

Please follow these quick steps to update TO library from 2.x.x to 3.x.x.

7.2.28.1 Headers

Include TO.h instead of TO_cli.h.

7.2.28.2 Defines

TO_I2C_WRAPPER_CONFIG replaces TO_CLI_I2C_WRAPPER_CONFIG. TO_LIB_INTERNAL_IO_BUFFER_SIZE replaces TO_CLI_INTERNAL_IO_BUFFER_SIZE.

7.2.28.3 Autotools

For Unix platforms, pkg-config file TO.pc replaces TO_client.pc.

Index

D

P

T