



# **Integration manual - T0136 on standard MCU device**

**Release 6.3.10**

**March 1, 2023**

**Authored by *Trusted Objects***

# Contents

<b>1</b>	<b>Legal</b>	<b>1</b>
1.1	Disclaimer . . . . .	1
<b>2</b>	<b>Introduction</b>	<b>2</b>
2.1	Overall architecture . . . . .	3
2.2	Library files tree . . . . .	3
2.3	Limitations . . . . .	4
<b>3</b>	<b>Library setup and configuration</b>	<b>5</b>
3.1	Use the library in an MCU project . . . . .	5
3.1.1	Configure your project . . . . .	6
3.1.1.1	Headers include paths . . . . .	6
3.1.1.2	Preprocessor definitions . . . . .	6
3.1.2	Use an existing I2C wrapper or develop your own one . . . . .	6
3.1.2.1	Use an existing I2C wrapper . . . . .	6
3.1.2.2	Implement your own I2C wrapper . . . . .	6
3.2	Library configuration for an MCU project . . . . .	6
3.2.1	User configuration file . . . . .	7
3.2.2	Global settings . . . . .	7
3.2.2.1	Endianness . . . . .	8
3.2.2.2	Integers (stdint) . . . . .	9
3.2.3	Features settings . . . . .	9
3.2.3.1	Macroscopic settings . . . . .	9
3.2.3.2	Microscopic settings . . . . .	10
<b>4</b>	<b>I2C wrapper</b>	<b>11</b>
4.1	I2C wrapper . . . . .	11
4.1.1	Available wrappers . . . . .	11
4.2	I2C wrapper implementation guidelines . . . . .	11
4.2.1	Timeout . . . . .	11
4.2.2	Library debug logs . . . . .	12
4.2.3	I2C wrapper integration . . . . .	12
4.2.3.1	MCU project . . . . .	12
<b>5</b>	<b>TO136 usage examples</b>	<b>13</b>
5.1	Initialization . . . . .	13
5.1.1	API usage example . . . . .	13
5.2	Uninitialization . . . . .	14
<b>6</b>	<b>Provided API</b>	<b>15</b>
6.1	Helper API . . . . .	15
6.1.1	ECIES sequence . . . . .	15
6.1.1.1	Authenticate the Secure Element . . . . .	15
6.1.1.2	Authenticate remote . . . . .	16
6.1.1.3	ECIES secure messaging . . . . .	17
6.1.2	Secure messaging . . . . .	18
6.1.3	Certificates . . . . .	21
6.1.4	TLS handshake . . . . .	23
6.1.4.1	TLS callback functions to define . . . . .	23
6.1.4.2	Handshake API . . . . .	26

6.1.4.3	Messaging API	32
6.2	Secure Element API	33
6.2.1	I2C communication	33
6.2.1.1	I2C setup	33
6.2.1.2	Basic messaging	34
6.2.2	Secure Element functions	35
6.2.2.1	NVM	35
6.2.2.2	Initialization	37
6.2.2.3	System	37
6.2.2.4	Hashes	39
6.2.2.5	Authentication	41
6.2.2.6	Encryption	49
6.2.2.7	MAC	57
6.2.2.8	Secure messaging CAPI	62
6.2.2.8.1	Old API	66
6.2.2.9	TLS	70
6.3	I2C wrapper API	81
6.3.1	Types and definitions	82
6.3.2	I2C bus setup	82
6.3.3	Data transfers	83
6.3.4	Miscellaneous	84
6.4	Library core APIs	84
6.4.1	Data buffers	84
6.4.2	Command data preparation	85
6.4.3	Send command	86
6.4.4	Hooks	86
6.4.4.1	Hooks functions prototypes	86
6.4.4.2	Hooks setup functions	87
6.4.5	Logs	88
6.5	Types and definitions	88
6.5.1	Library error codes	89
6.5.2	Secure Element error codes	90
6.5.3	Combined error codes	92
6.5.4	Keys types	92
6.5.5	Certificates	93
6.5.6	Algorithms	96
6.5.7	Payloads	98
6.5.8	Constants	98
6.5.9	Generic context	108
6.5.10	Log levels	109
6.5.11	Miscellaneous constants	112
6.5.12	LoRaWAN constants	114
6.5.13	Certificates constants	115
6.5.14	TLS constants	116
6.5.15	I2C constants	123
6.5.16	Status PIO constants	124
6.5.17	Seclink constants	124
6.5.18	Admin constants	124
7	Miscellany guides	126
7.1	Production optimizations	126
7.1.1	Features	126

7.1.2	Internal buffers	126
7.1.3	Logs	126
7.2	Migration	126
7.2.1	TO library migration guide from 5.7.x to 5.8.x	126
7.2.2	TO library migration guide from 5.6.x to 5.7.x	127
7.2.3	TO library migration guide from 5.5.x to 5.6.x	127
7.2.4	TO library migration guide from 5.4.x to 5.5.x	127
7.2.5	TO library migration guide from 5.3.x to 5.4.x	127
7.2.6	TO library migration guide from 5.2.x to 5.3.x	127
7.2.7	TO library migration guide from 5.1.x to 5.2.x	127
7.2.8	TO library migration guide from 5.0.x to 5.1.x	127
7.2.9	TO library migration guide from 4.x.x to 5.x.x	128
7.2.9.1	Internal architecture changes	128
7.2.9.2	Generic APIs renames	128
7.2.9.3	Hardware Secure Element renames	128
7.2.10	TO library migration guide from 4.18.x to 4.19.x	129
7.2.11	TO library migration guide from 4.17.x to 4.18.x	129
7.2.12	TO library migration guide from 4.16.x to 4.17.x	129
7.2.13	TO library migration guide from 4.15.x to 4.16.x	129
7.2.14	TO library migration guide from 4.14.x to 4.15.x	129
7.2.15	TO library migration guide from 4.13.x to 4.14.x	129
7.2.16	TO library migration guide from 4.12.x to 4.13.x	129
7.2.17	TO library migration guide from 4.11.x to 4.12.x	130
7.2.18	TO library migration guide from 4.10.x to 4.11.x	130
7.2.19	TO library migration guide from 4.9.x to 4.10.x	130
7.2.20	TO library migration guide from 4.8.x to 4.9.x	130
7.2.21	TO library migration guide from 4.6.x to 4.7.x	131
7.2.21.1	Preprocessor defines (MCU project)	131
7.2.22	TO library migration guide from 4.5.x to 4.6.x	131
7.2.22.1	TLS helper API	131
7.2.23	TO library migration guide from 4.4.x to 4.5.x	131
7.2.24	TO library migration guide from 4.3.x to 4.4.x	132
7.2.24.1	Preprocessor defines (MCU project)	132
7.2.25	TO library migration guide from 4.1.x to 4.2.x	132
7.2.25.1	Changed APIs	133
7.2.26	TO library migration guide from 4.0.x to 4.1.x	133
7.2.26.1	Renamed files	133
7.2.27	TO library migration guide from 3.x.x to 4.x.x	133
7.2.27.1	Renamed APIs	133
7.2.27.2	Preprocessor flags	133
7.2.27.3	Error codes	134
7.2.28	TO library migration guide from 2.x.x to 3.x.x	134
7.2.28.1	Headers	134
7.2.28.2	Defines	134
7.2.28.3	Autotools	134

# 1. Legal

Copyright (C) 2016-2023 Trusted Objects. All rights reserved.

## 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.

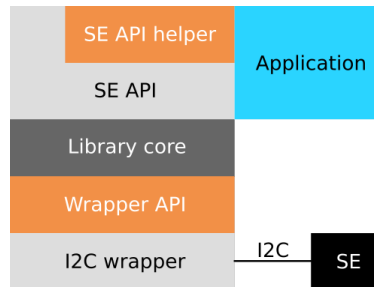


Fig. 1: Library architecture

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 <code>TO_set_log_level()</code> : -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 providestdint.h
TO_USER_CONFIG	User provides file TO_user_config.h
TO_I2C_WRAPPER_CONFIG	Ability to configure I2C wrapper, see <code>TO_data_config()</code>
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 formatted 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 formatted)
- **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 formatted 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 isn't const because the callback can reuse it to unsecure in place provided it doesn't 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. without 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 negotiated 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 negotiated 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 outgoing 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 environment.

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 be 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_rcv_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_rcv_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 succeed, **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 negotiated 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 successfully
- TO\_DEVICE\_WRITE\_ERROR: error writing data to Secure Element
- TO\_ERROR if an internal error has occurred

*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 successfully
- TO\_DEVICE\_READ\_ERROR: error reading data from Secure Element
- TO\_ERROR if an internal error has occurred

*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 successfully
- TO\_ERROR if an internal error has occurred

*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 successfully
- TO\_ERROR if an internal error has occurred

## 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 context
- **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 formatted 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 formatted certificate according to the given index. The new certificate must be signed by a CA trusted by the Secure Element.

#### Parameters

- **ctx** – [in] Pointer to the SE context
- **certificate\_index** – [in] Requested certificate index
- **certificate** – [in] New certificate data (x509 DER formatted)
- **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 formatted)
- **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 formatted 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 transferred 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

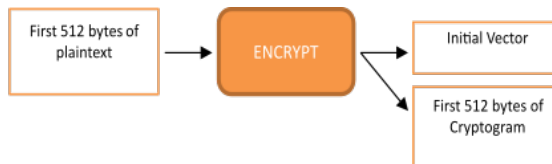
### 6.2.2.6 Encryption

Ciphred messaging functions.

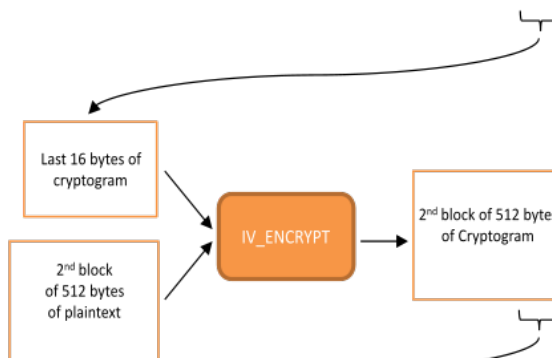
The best way to encrypt more than the maximum size of data supported by `TOSE_<xxx>_encrypt()` command is to manually chain encryption.

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

1. Call ENCRYPT command on the first 512 bytes of data. Secure Element will safely generate the first Initial Vector.



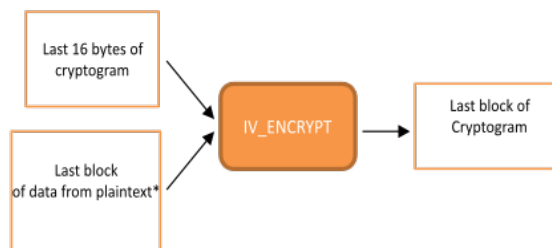
2. Call IV\_ENCRYPT command on the next block of 512 bytes. Use the last 16 bytes of the encrypted data sent by previous command as Initial Vector of this command.



3. Repeat steps 2 until all data are encrypted.

...

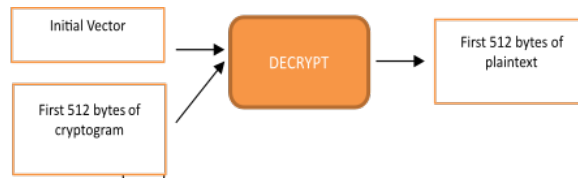
4. Finally, just concatenate Initial Vector and each 512 blocks of cryptogram to obtain the final cryptogram



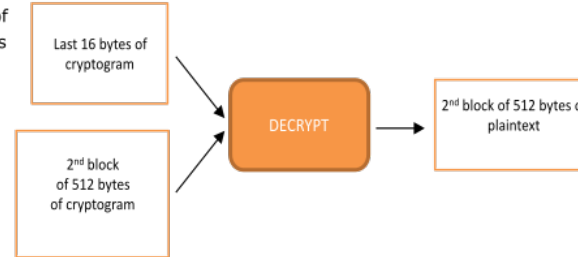
\*size must be multiple of 16

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

1. Call DECRYPT command on the first 512 bytes of cryptogram, using Initial Vector returned by ENCRYPT command.



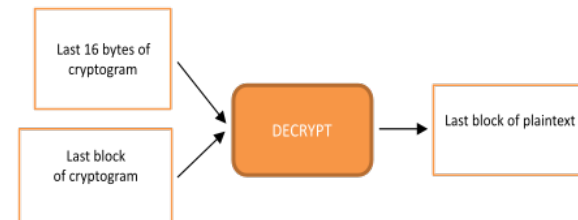
2. Call DECRYPT command on the next block of 512 bytes of cryptogram. Use the last 16 bytes of the cryptogram from previous block of 512 bytes.



3. Repeat steps 2 until all data are decrypted.

...

4. Finally, just concatenate all 512 blocks of plaintext to obtain your decrypted data



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)

Finalizes 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_CMAL_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_CMACE_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-negotiated, 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

- **ctx** – [in] Pointer to the SE context
- **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] ServerHello payload
- **server\_hello\_len** – [in] ServerHello 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**(), **update**() 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 don't 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 occurred. 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 successfully `TO_DEVICE_READ_ERROR`: error reading data from Secure Element `TO_ERROR` if an internal error has occurred

*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 occurred. 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 successfully TO\_DEVICE\_WRITE\_ERROR: error writing data to Secure Element TO\_ERROR if an internal error has occurred

### 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 occurred

## 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 occurred 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 detailed 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 readiness. 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 readiness 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 **T0136RSP\_COM\_ERROR**

Deprecated, use **TORSP\_COM\_ERROR** instead

enumerator **TORSP\_NEED\_AUTHENTICATION**

An authentication process has to be conducted 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 conducted 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 **T0\_ret\_t**

### 6.5.4 Keys types

*group* **key\_types**

#### Typedefs

typedef enum *TO\_key\_type\_e* **T0\_key\_type\_t**

Secure Element key types

#### Enums

enum **T0\_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 **T0\_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 **T0\_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 T0_log_level_e __attribute__((packed)) T0_log_level_t
```

Different log levels that are available to the application.

```
void print_log_function(const T0_log_level_t level, const char *log)
```

Default print log function, potentially 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 T0_log(T0_log_ctx_t *log_ctx, const T0_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 T0_set_log_level(T0_log_ctx_t *log_ctx, const T0_log_level_t level, T0_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. T0\_log)

```
T0_log_ctx_t *T0_log_get_ctx(void)
```

Get the LOG context.

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

### Returns

T0\_log\_ctx\_t\*

```
struct T0_log_ctx_s
```

*#include <T0\_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 theoretical 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* T0\_tls\_mode\_t

Different modes available for TO-Protect TLS.

typedef enum *TO\_tls\_config\_id\_e* T0\_tls\_config\_id\_t

typedef enum *TO\_tls\_record\_type\_e* T0\_tls\_record\_type\_t

typedef enum *TO\_tls\_cipher\_suite\_e* T0\_tls\_cipher\_suite\_t

typedef enum *TO\_tls\_cipher\_suite\_type\_e* T0\_tls\_cipher\_suite\_type\_t

typedef enum *TO\_tls\_encryption\_type\_e* T0\_tls\_encryption\_type\_t

typedef enum *TO\_tls\_handshake\_type\_e* T0\_tls\_handshake\_type\_t

typedef enum *TO\_tls\_state\_e* T0\_tls\_state\_t

typedef enum *TO\_tls\_extensions\_e* T0\_tls\_extension\_t

## Enums

enum T0\_tls\_mode\_e

Different modes available for TO-Protect TLS.

*Values:*

enumerator T0\_TLS\_MODE\_UNKNOWN

Unknown mode (uninitialized)

enumerator T0\_TLS\_MODE\_HANDSHAKE\_ONLY

Handshake Only mode

enumerator T0\_TLS\_MODE\_TLS\_1\_2

TLS 1.2 only

enumerator T0\_TLS\_MODE\_TLS\_1\_2\_HANDSHAKE\_ONLY

TLS 1.2, in Handshake only

enumerator T0\_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\_CRYPTOGAM\_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 successful `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 don't need TLS or LoRa in your project, you now have to explicitly disable these features.

DTLS remains disabled and has to be explicitly 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
<code>TLS_IO_BUFFER_SIZE</code>	<code>TO_TLS_IO_BUFFER_SIZE</code>
<code>TLS_FLIGHT_BUFFER_SIZE</code>	<code>TO_TLS_FLIGHT_BUFFER_SIZE</code>

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

default\_cipher\_ctx (*C var*), 26  
default\_setup\_cipher\_ctx (*C var*), 26

## P

print\_log\_function (*C function*), 111

## T

TO\_ADMIN\_CHALLENGE\_SIZE (*C macro*), 125  
TO\_ADMIN\_CRYPTOGAM\_SIZE (*C macro*), 125  
TO\_ADMIN\_DATAIDX\_SIZE (*C macro*), 125  
TO\_ADMIN\_DIVERS\_DATA\_SIZE (*C macro*), 125  
TO\_ADMIN\_MAC\_SIZE (*C macro*), 125  
TO\_ADMIN\_OPTIONS\_SIZE (*C macro*), 125  
TO\_ADMIN\_PROTO\_INFO\_SIZE (*C macro*), 125  
TO\_AES\_BLOCK\_SIZE (*C macro*), 112  
TO\_AES\_KEYSIZE (*C macro*), 112  
TO\_AESC\_CM\_8\_TAG\_SIZE (*C macro*), 112  
TO\_AESC\_CM\_AAD\_LEN\_SIZE (*C macro*), 112  
TO\_AESC\_CM\_NONCE\_SIZE (*C macro*), 112  
TO\_AESC\_CM\_TAG\_SIZE (*C macro*), 112  
TO\_AESGCM\_AAD\_LEN\_SIZE (*C macro*), 112  
TO\_AESGCM\_INITIALVECTOR\_SIZE (*C macro*), 112  
TO\_AESGCM\_TAG\_SIZE (*C macro*), 112  
TO\_ARC4\_INITIALVECTOR\_SIZE (*C macro*), 124  
TO\_ARC4\_KEY\_SIZE (*C macro*), 124  
TO\_CA\_IDX\_AUTO (*C macro*), 94  
TO\_CA\_PUBKEY\_CAID\_SIZE (*C macro*), 116  
TO\_CA\_PUBKEY\_SIZE (*C macro*), 116  
TO\_cert\_CA\_capabilities\_e (*C enum*), 95  
TO\_cert\_CA\_capabilities\_e.TO\_CERT\_CA\_CAP\_ADMIN (*C enumerator*), 95  
TO\_cert\_CA\_capabilities\_e.TO\_CERT\_CA\_CAP\_EMPTY (*C enumerator*), 95  
TO\_cert\_CA\_capabilities\_e.TO\_CERT\_CA\_CAP\_UPDATE\_CA (*C enumerator*), 95  
TO\_cert\_CA\_capabilities\_t (*C type*), 94  
TO\_CERT\_DATE\_SIZE (*C macro*), 115  
TO\_CERT\_DN\_MAXSIZE (*C macro*), 116  
TO\_CERT\_GENERALIZED\_TIME\_SIZE (*C macro*), 115  
TO\_CERT\_PRIVKEY\_SIZE (*C macro*), 115  
TO\_cert\_short\_s (*C struct*), 95  
TO\_cert\_short\_s.ca\_id (*C var*), 96  
TO\_cert\_short\_s.public\_key (*C var*), 96  
TO\_cert\_short\_s.serial\_number (*C var*), 96  
TO\_cert\_short\_s.signature (*C var*), 96  
TO\_cert\_short\_t (*C type*), 94  
TO\_cert\_short\_v2\_s (*C struct*), 96  
TO\_cert\_short\_v2\_s.ca\_id (*C var*), 96  
TO\_cert\_short\_v2\_s.date (*C var*), 96  
TO\_cert\_short\_v2\_s.public\_key (*C var*), 96  
TO\_cert\_short\_v2\_s.serial\_number (*C var*), 96  
TO\_cert\_short\_v2\_s.signature (*C var*), 96  
TO\_cert\_short\_v2\_t (*C type*), 94  
TO\_cert\_standalone\_s (*C struct*), 95  
TO\_cert\_standalone\_s.ca\_id (*C var*), 95  
TO\_cert\_standalone\_s.public\_key (*C var*), 95  
TO\_cert\_standalone\_s.serial\_number (*C var*), 95  
TO\_cert\_standalone\_s.signature (*C var*), 95  
TO\_cert\_standalone\_t (*C type*), 94  
TO\_CERT\_SUBJECT\_CN\_MAXSIZE (*C macro*), 116  
TO\_CERT\_SUBJECT\_CN\_PREFIX\_MAXSIZE (*C macro*), 116  
TO\_CERT\_SUBJECT\_PREFIX\_SIZE (*C macro*), 115  
TO\_certificate\_format\_e (*C enum*), 94  
TO\_certificate\_format\_e.TO\_CERTIFICATE\_NONE (*C enumerator*), 95  
TO\_certificate\_format\_e.TO\_CERTIFICATE\_SHORT (*C enumerator*), 94  
TO\_certificate\_format\_e.TO\_CERTIFICATE\_SHORT\_V2 (*C enumerator*), 95  
TO\_certificate\_format\_e.TO\_CERTIFICATE\_STANDALONE (*C enumerator*), 94  
TO\_certificate\_format\_e.TO\_CERTIFICATE\_UNKNOWN (*C enumerator*), 95  
TO\_certificate\_format\_e.TO\_CERTIFICATE\_X509 (*C enumerator*), 94  
TO\_certificate\_format\_t (*C type*), 94  
TO\_CERTIFICATE\_SIZE (*C macro*), 115  
TO\_CHALLENGE\_SIZE (*C macro*), 113  
TO\_CMACE\_KEYSIZE (*C macro*), 113  
TO\_CMACE\_MIN\_SIZE (*C macro*), 113  
TO\_CMACE\_SIZE (*C macro*), 113  
TO\_CONFIG\_NACK\_LAST\_BYTE (*C macro*), 82  
TO\_CRC\_SIZE (*C macro*), 113  
TO\_data\_config (*C function*), 83  
TO\_data\_fini (*C function*), 82  
TO\_data\_init (*C function*), 82  
TO\_data\_last\_command\_duration (*C function*), 84  
TO\_data\_read (*C function*), 83  
TO\_data\_write (*C function*), 83  
TO\_DTLS\_CONNECTION\_ID\_MAXSIZE (*C macro*), 117  
TO\_DTLS\_HANDSHAKE\_HEADER\_MAXSIZE (*C macro*), 117  
TO\_DTLS\_HANDSHAKE\_HEADER\_SIZE (*C macro*), 117  
TO\_DTLS\_HEADER\_MAXSIZE (*C macro*), 117  
TO\_DTLS\_HEADER\_SIZE (*C macro*), 117  
TO\_DTLS\_MAJOR (*C macro*), 117  
TO\_DTLS\_MINOR (*C macro*), 117  
TO\_ECC\_PRIV\_KEYSIZE (*C macro*), 115  
TO\_ECC\_PUB\_KEYSIZE (*C macro*), 115

TO\_enc\_alg\_e (*C enum*), 96  
 TO\_enc\_alg\_e.TO\_ENC\_ALG\_AES128CBC (*C enumerator*), 97  
 TO\_enc\_alg\_e.TO\_ENC\_ALG\_AES128CCM (*C enumerator*), 97  
 TO\_enc\_alg\_e.TO\_ENC\_ALG\_AES128GCM (*C enumerator*), 97  
 TO\_enc\_alg\_e.TO\_ENC\_ALG\_ARC4 (*C enumerator*), 97  
 TO\_enc\_alg\_e.TO\_ENC\_ALG\_MAX (*C enumerator*), 97  
 TO\_enc\_alg\_e.TO\_ENC\_ALG\_UNDEFINED (*C enumerator*), 96  
 TO\_enc\_alg\_t (*C type*), 97  
 TO\_FORMAT\_SIZE (*C macro*), 112  
 TO\_HASH\_SIZE (*C macro*), 113  
 TO\_HMAC\_KEYSIZE (*C macro*), 112  
 TO\_HMAC\_MINSIZE (*C macro*), 113  
 TO\_HMAC\_SIZE (*C macro*), 113  
 TO\_HW\_SN\_SIZE (*C macro*), 113  
 TO\_HW\_VERSION\_SIZE (*C macro*), 113  
 TO\_i2c\_config\_s (*C struct*), 82  
 TO\_i2c\_config\_s.i2c\_addr (*C var*), 82  
 TO\_i2c\_config\_s.misc\_settings (*C var*), 82  
 TO\_i2c\_config\_t (*C type*), 82  
 TO\_I2C\_ERROR\_MSTIMEOUT (*C macro*), 124  
 TO\_I2C\_MSTIMEOUT (*C macro*), 123  
 TO\_I2C\_RECV\_MSTIMEOUT (*C macro*), 123  
 TO\_I2C\_RESPONSE\_MSTIMEOUT (*C macro*), 123  
 TO\_I2C\_SEND\_MSTIMEOUT (*C macro*), 123  
 TO\_I2CADDR\_SIZE (*C macro*), 123  
 TO\_IDX\_AUTO (*C macro*), 93  
 TO\_INDEX\_SIZE (*C macro*), 112  
 TO\_INITIALVECTOR\_SIZE (*C macro*), 112  
 TO\_KEY\_FINGERPRINT\_SIZE (*C macro*), 113  
 TO\_KEY\_IDENTIFIER\_SHORT\_SIZE (*C macro*), 116  
 TO\_KEY\_IDENTIFIER\_SIZE (*C macro*), 116  
 TO\_key\_type\_e (*C enum*), 92  
 TO\_key\_type\_e.KTYPE\_CA\_KPUB (*C enumerator*), 92  
 TO\_key\_type\_e.KTYPE\_CERT\_KPRIV (*C enumerator*), 92  
 TO\_key\_type\_e.KTYPE\_CERT\_KPUB (*C enumerator*), 92  
 TO\_key\_type\_e.KTYPE\_ECIES\_KAES (*C enumerator*), 93  
 TO\_key\_type\_e.KTYPE\_ECIES\_KMAC (*C enumerator*), 93  
 TO\_key\_type\_e.KTYPE\_ECIES\_KPRIV (*C enumerator*), 93  
 TO\_key\_type\_e.KTYPE\_ECIES\_KPUB (*C enumerator*), 92  
 TO\_key\_type\_e.KTYPE\_LORA\_KAPP (*C enumerator*), 93  
 TO\_key\_type\_e.KTYPE\_LORA\_KNET (*C enumerator*), 93  
 TO\_key\_type\_e.KTYPE\_LORA\_KSAPP (*C enumerator*), 93  
 TO\_key\_type\_e.KTYPE\_LORA\_KSNET (*C enumerator*), 93  
 TO\_key\_type\_e.KTYPE\_REMOTE\_KPUB (*C enumerator*), 92  
 TO\_key\_type\_t (*C type*), 92  
 TO\_KEYTYPE\_SIZE (*C macro*), 116  
 TO\_LIB\_ERRCODE (*C macro*), 92  
 TO\_lib\_ret\_e (*C enum*), 89  
 TO\_lib\_ret\_e.TO\_AGAIN (*C enumerator*), 89  
 TO\_lib\_ret\_e.TO\_DEVICE\_READ\_ERROR (*C enumerator*), 89  
 TO\_lib\_ret\_e.TO\_DEVICE\_WRITE\_ERROR (*C enumerator*), 89  
 TO\_lib\_ret\_e.TO\_ERROR (*C enumerator*), 89  
 TO\_lib\_ret\_e.TO\_INVALID\_CA\_ID (*C enumerator*), 89  
 TO\_lib\_ret\_e.TO\_INVALID\_CERTIFICATE\_FORMAT (*C enumerator*), 89  
 TO\_lib\_ret\_e.TO\_INVALID\_CERTIFICATE\_NUMBER (*C enumerator*), 89  
 TO\_lib\_ret\_e.TO\_INVALID\_PARAM (*C enumerator*), 89  
 TO\_lib\_ret\_e.TO\_INVALID\_RESPONSE\_LENGTH (*C enumerator*), 89  
 TO\_lib\_ret\_e.TO\_MEMORY\_ERROR (*C enumerator*), 89  
 TO\_lib\_ret\_e.TO\_NOT\_IMPLEMENTED (*C enumerator*), 89  
 TO\_lib\_ret\_e.TO\_OK (*C enumerator*), 89  
 TO\_lib\_ret\_e.TO\_SECLINK\_ERROR (*C enumerator*), 89  
 TO\_lib\_ret\_e.TO\_TIMEOUT (*C enumerator*), 89  
 TO\_lib\_ret\_t (*C type*), 89  
 TO\_log (*C function*), 111  
 TO\_LOG\_BUFFER (*C macro*), 110  
 TO\_log\_ctx\_s (*C struct*), 111  
 TO\_log\_ctx\_s.log\_function (*C var*), 112  
 TO\_log\_ctx\_s.log\_level (*C var*), 112  
 TO\_log\_ctx\_t (*C type*), 110  
 TO\_log\_get\_ctx (*C function*), 111  
 TO\_LOG\_HEX\_DISP (*C macro*), 110  
 TO\_LOG\_LEVEL\_DBG (*C macro*), 110  
 TO\_log\_level\_e (*C enum*), 110  
 TO\_LOG\_LEVEL\_ERR (*C macro*), 109  
 TO\_LOG\_LEVEL\_INF (*C macro*), 110  
 TO\_LOG\_LEVEL\_MASK (*C macro*), 110

TO\_LOG\_LEVEL\_NONE (*C macro*), 109  
 TO\_LOG\_LEVEL\_WRN (*C macro*), 109  
 TO\_LOG\_STRING (*C macro*), 110  
 TO\_LOG\_TRACE (*C macro*), 110  
 TO\_LORA\_APEUI\_SIZE (*C macro*), 114  
 TO\_LORA\_APPKEY\_SIZE (*C macro*), 114  
 TO\_LORA\_APPNONCE\_SIZE (*C macro*), 114  
 TO\_LORA\_APPSKEY\_SIZE (*C macro*), 114  
 TO\_LORA\_CFLIST\_SIZE (*C macro*), 115  
 TO\_LORA\_DEVADDR\_SIZE (*C macro*), 114  
 TO\_LORA\_DEVEUI\_SIZE (*C macro*), 114  
 TO\_LORA\_DEVNONCE\_SIZE (*C macro*), 114  
 TO\_LORA\_DLSETTINGS\_SIZE (*C macro*), 114  
 TO\_LORA\_FCNT\_SIZE (*C macro*), 114  
 TO\_LORA\_FCTRL\_SIZE (*C macro*), 114  
 TO\_LORA\_JOINACCEPT\_CLEAR\_MAXSIZE (*C macro*), 115  
 TO\_LORA\_JOINACCEPT\_MAXSIZE (*C macro*), 115  
 TO\_LORA\_JOINREQUEST\_SIZE (*C macro*), 115  
 TO\_LORA\_MHDR\_SIZE (*C macro*), 114  
 TO\_LORA\_MIC\_SIZE (*C macro*), 114  
 TO\_LORA\_NETID\_SIZE (*C macro*), 114  
 TO\_LORA\_NWKSKEY\_SIZE (*C macro*), 114  
 TO\_LORA\_PHYPAYLOAD\_MIN\_SIZE (*C macro*), 114  
 TO\_LORA\_RXDELAY\_SIZE (*C macro*), 115  
 TO\_mac\_alg\_e (*C enum*), 97  
 TO\_mac\_alg\_e.TO\_MAC\_ALG\_CMIC (*C enumerator*), 97  
 TO\_mac\_alg\_e.TO\_MAC\_ALG\_HMAC (*C enumerator*), 97  
 TO\_mac\_alg\_e.TO\_MAC\_ALG\_MAX (*C enumerator*), 97  
 TO\_mac\_alg\_e.TO\_MAC\_ALG\_NONE (*C enumerator*), 97  
 TO\_mac\_alg\_e.TO\_MAC\_ALG\_UNDEFINED (*C enumerator*), 97  
 TO\_mac\_alg\_t (*C type*), 97  
 TO\_PAYLOAD\_CLEAR\_DATA\_SIZE (*C macro*), 98  
 TO\_PAYLOAD\_IV\_SIZE (*C macro*), 98  
 TO\_PAYLOAD\_MAC\_SIZE (*C macro*), 98  
 TO\_PAYLOAD\_PADDING\_SIZE (*C macro*), 98  
 TO\_PAYLOAD\_SECURED\_PAYLOAD\_SIZE (*C macro*), 98  
 TO\_PN\_SIZE (*C macro*), 113  
 TO\_PRODUCT\_ID\_SIZE (*C macro*), 113  
 TO\_REMOTE\_CAID\_SIZE (*C macro*), 116  
 TO\_REMOTE\_CERTIFICATE\_SIZE (*C macro*), 116  
 TO\_ret\_t (*C type*), 92  
 TO\_SE\_ERRCODE (*C macro*), 92  
 TO\_se\_ret\_e (*C enum*), 90  
 TO\_se\_ret\_e.TO136RSP\_COM\_ERROR (*C enumerator*), 91  
 TO\_se\_ret\_e.TORSP\_ARG\_OUT\_OF\_RANGE (*C enumerator*), 91  
 TO\_se\_ret\_e.TORSP\_BAD\_SIGNATURE (*C enumerator*), 90  
 TO\_se\_ret\_e.TORSP\_COM\_ERROR (*C enumerator*), 91  
 TO\_se\_ret\_e.TORSP\_COND\_OF\_USE\_NOT\_SATISFIED (*C enumerator*), 91  
 TO\_se\_ret\_e.TORSP\_INCONSISTENT\_SECURE\_STORAGES (*C enumerator*), 91  
 TO\_se\_ret\_e.TORSP\_INTERNAL\_ERROR (*C enumerator*), 91  
 TO\_se\_ret\_e.TORSP\_INVALID\_LEN (*C enumerator*), 90  
 TO\_se\_ret\_e.TORSP\_INVALID\_PADDING (*C enumerator*), 91  
 TO\_se\_ret\_e.TORSP\_NEED\_AUTHENTICATION (*C enumerator*), 91  
 TO\_se\_ret\_e.TORSP\_NO\_VALID\_SECURE\_STORAGE\_FOUND (*C enumerator*), 91  
 TO\_se\_ret\_e.TORSP\_NOT\_AVAILABLE (*C enumerator*), 90  
 TO\_se\_ret\_e.TORSP\_SECLINK\_RENEW\_KEY (*C enumerator*), 91  
 TO\_se\_ret\_e.TORSP\_SUCCESS (*C enumerator*), 91  
 TO\_se\_ret\_e.TORSP\_UNKNOWN\_CMD (*C enumerator*), 90  
 TO\_se\_ret\_e.TORSP\_WRONG\_INSTALLATION (*C enumerator*), 91  
 TO\_se\_ret\_e.TORSP\_WRONG\_SECURE\_STORAGE\_VERSION (*C enumerator*), 91  
 TO\_se\_ret\_t (*C type*), 90  
 TO\_SEED\_SIZE (*C macro*), 114  
 TO\_SEQUENCE\_SIZE (*C macro*), 113  
 TO\_set\_log\_level (*C function*), 88, 111  
 TO\_SHA256\_HASHSIZE (*C macro*), 113  
 TO\_SHORTV2\_CERT\_SIZE (*C macro*), 115  
 TO\_SIGNATURE\_SIZE (*C macro*), 115  
 TO\_SN\_CA\_ID\_SIZE (*C macro*), 113  
 TO\_SN\_NB\_SIZE (*C macro*), 113  
 TO\_SN\_SIZE (*C macro*), 113  
 TO\_STATUS\_PIO\_ENABLE (*C macro*), 124  
 TO\_STATUS\_PIO\_HIGH\_OPENDRAIN\_MASK (*C macro*), 124  
 TO\_STATUS\_PIO\_IDLE\_HZ\_MASK (*C macro*), 124  
 TO\_STATUS\_PIO\_READY\_LEVEL\_MASK (*C macro*), 124  
 TO\_SW\_VERSION\_SIZE (*C macro*), 113  
 TO\_TIMESTAMP\_SIZE (*C macro*), 113  
 TO\_TLS\_AEAD\_EXPLICIT\_NONCE\_SIZE (*C macro*), 117  
 TO\_TLS\_AEAD\_IMPLICIT\_NONCE\_SIZE (*C macro*), 117  
 TO\_TLS\_CHANGE\_CIPHER\_SPEC\_SIZE (*C macro*), 117  
 TO\_tls\_cipher\_suite\_e (*C enum*), 119  
 TO\_tls\_cipher\_suite\_e.TO\_TLS\_ECDHE\_ECDSA\_WITH\_AES\_128\_CBC\_

**TRUSTED OBJECTS** This document and the information it contains is the property of Trusted Objects. It can't be used, reproduced or transmitted to a third party without prior written approval. **138 / 145**



TO\_tls\_mode\_e.TO\_TLS\_MODE\_HANDSHAKE\_ONLY (C enumerator), 118

TO\_tls\_mode\_e.TO\_TLS\_MODE\_TLS\_1\_2 (C enumerator), 118

TO\_tls\_mode\_e.TO\_TLS\_MODE\_TLS\_1\_2\_HANDSHAKE\_ONLY (C enumerator), 118

TO\_tls\_mode\_e.TO\_TLS\_MODE\_UNKNOWN (C enumerator), 118

TO\_tls\_mode\_t (C type), 118

TO\_TLS\_RANDOM\_SIZE (C macro), 117

TO\_TLS\_RECORD\_CIPHER\_OVERHEAD\_MAX (C macro), 116

TO\_TLS\_RECORD\_MAX\_SIZE (C macro), 117

TO\_tls\_record\_type\_e (C enum), 119

TO\_tls\_record\_type\_e.TO\_TLS\_RECORD\_TYPE\_ALERT (C enumerator), 119

TO\_tls\_record\_type\_e.TO\_TLS\_RECORD\_TYPE\_APPLICATION\_DATA (C enumerator), 119

TO\_tls\_record\_type\_e.TO\_TLS\_RECORD\_TYPE\_CHANGE\_CIPHER\_SPEC (C enumerator), 119

TO\_tls\_record\_type\_e.TO\_TLS\_RECORD\_TYPE\_HANDSHAKE (C enumerator), 119

TO\_tls\_record\_type\_e.TO\_TLS\_RECORD\_TYPE\_TLS\_1\_2\_CID (C enumerator), 119

TO\_tls\_record\_type\_t (C type), 118

TO\_TLS\_SERVER\_PARAMS\_SIZE (C macro), 117

TO\_TLS\_SNI\_LENGTH\_MAX (C macro), 117

TO\_tls\_state\_e (C enum), 121

TO\_tls\_state\_e.TO\_TLS\_STATE\_CLIENT\_CERTIFICATE (C enumerator), 122

TO\_tls\_state\_e.TO\_TLS\_STATE\_CLIENT\_CERTIFICATE\_VERIFY (C enumerator), 122

TO\_tls\_state\_e.TO\_TLS\_STATE\_CLIENT\_CHANGE\_CIPHER\_SPEC (C enumerator), 122

TO\_tls\_state\_e.TO\_TLS\_STATE\_CLIENT\_FINISHED (C enumerator), 122

TO\_tls\_state\_e.TO\_TLS\_STATE\_CLIENT\_HELLO (C enumerator), 121

TO\_tls\_state\_e.TO\_TLS\_STATE\_CLIENT\_HELLO\_WITH\_COOKIE (C enumerator), 121

TO\_tls\_state\_e.TO\_TLS\_STATE\_CLIENT\_KEY\_EXCHANGE (C enumerator), 122

TO\_tls\_state\_e.TO\_TLS\_STATE\_CLOSE\_RECEIVED (C enumerator), 123

TO\_tls\_state\_e.TO\_TLS\_STATE\_FATAL\_RECEIVED (C enumerator), 123

TO\_tls\_state\_e.TO\_TLS\_STATE\_FLIGHT\_1 (C enumerator), 121

TO\_tls\_state\_e.TO\_TLS\_STATE\_FLIGHT\_1\_INIT (C enumerator), 121

TO\_tls\_state\_e.TO\_TLS\_STATE\_FLIGHT\_2 (C enumerator), 121

TO\_tls\_state\_e.TO\_TLS\_STATE\_FLIGHT\_2\_INIT (C enumerator), 121

TO\_tls\_state\_e.TO\_TLS\_STATE\_FLIGHT\_3 (C enumerator), 121

TO\_tls\_state\_e.TO\_TLS\_STATE\_FLIGHT\_3\_INIT (C enumerator), 121

TO\_tls\_state\_e.TO\_TLS\_STATE\_FLIGHT\_4 (C enumerator), 121

TO\_tls\_state\_e.TO\_TLS\_STATE\_FLIGHT\_4\_INIT (C enumerator), 122

TO\_tls\_state\_e.TO\_TLS\_STATE\_FLIGHT\_5 (C enumerator), 122

TO\_tls\_state\_e.TO\_TLS\_STATE\_FLIGHT\_5\_INIT (C enumerator), 122

TO\_tls\_state\_e.TO\_TLS\_STATE\_FLIGHT\_5\_INIT\_NO\_CLIENT\_AUTH (C enumerator), 122

TO\_tls\_state\_e.TO\_TLS\_STATE\_FLIGHT\_6 (C enumerator), 122

TO\_tls\_state\_e.TO\_TLS\_STATE\_FLIGHT\_6\_INIT (C enumerator), 122

TO\_tls\_state\_e.TO\_TLS\_STATE\_HANDSHAKE\_DONE (C enumerator), 122

TO\_tls\_state\_e.TO\_TLS\_STATE\_HANDSHAKE\_FAILED (C enumerator), 123

TO\_tls\_state\_e.TO\_TLS\_STATE\_HANDSHAKE\_START (C enumerator), 121

TO\_tls\_state\_e.TO\_TLS\_STATE\_MEDIATOR\_CERTIFICATE (C enumerator), 122

TO\_tls\_state\_e.TO\_TLS\_STATE\_SERVER\_CERTIFICATE (C enumerator), 122

TO\_tls\_state\_e.TO\_TLS\_STATE\_SERVER\_CERTIFICATE\_REQUEST (C enumerator), 122

TO\_tls\_state\_e.TO\_TLS\_STATE\_SERVER\_CHANGE\_CIPHER\_SPEC (C enumerator), 122

TO\_tls\_state\_e.TO\_TLS\_STATE\_SERVER\_FINISHED (C enumerator), 122

TO\_tls\_state\_e.TO\_TLS\_STATE\_SERVER\_HELLO (C enumerator), 122

TO\_tls\_state\_e.TO\_TLS\_STATE\_SERVER\_HELLO\_DONE (C enumerator), 122

TO\_tls\_state\_e.TO\_TLS\_STATE\_SERVER\_HELLO\_VERIFY\_REQUEST (C enumerator), 121

TO\_tls\_state\_e.TO\_TLS\_STATE\_SERVER\_KEY\_EXCHANGE (C enumerator), 122

TO\_tls\_state\_t (C type), 118

TOCERTF\_NONE (C macro), 93

TOCERTF\_SHORT (C macro), 93

TOCERTF\_SHORT\_V2 (C macro), 93

TOCERTF\_STANDALONE (C macro), 93

TOCERTF\_SUBJECT\_NAME\_SIZE (C macro), 93

TOCERTF\_UNKNOWN (*C macro*), 93  
 TOCERTF\_VALIDITY\_DATE\_SIZE (*C macro*), 93  
 TOCERTF\_X509 (*C macro*), 93  
 TODRV\_HSE\_CMD\_ACCESS\_DUMMY\_DATA (*C macro*), 99  
 TODRV\_HSE\_CMD\_ADMIN\_COMMAND (*C macro*), 107  
 TODRV\_HSE\_CMD\_ADMIN\_COMMAND\_WITH\_RESPONSE (*C macro*), 107  
 TODRV\_HSE\_CMD\_ADMIN\_SET\_SLOT (*C macro*), 107  
 TODRV\_HSE\_CMD\_AES128CBC\_CMIC\_SECURE\_MESSAGE (*C macro*), 102  
 TODRV\_HSE\_CMD\_AES128CBC\_CMIC\_UNSECURE\_MESSAGE (*C macro*), 102  
 TODRV\_HSE\_CMD\_AES128CBC\_DECRYPT (*C macro*), 101  
 TODRV\_HSE\_CMD\_AES128CBC\_ENCRYPT (*C macro*), 101  
 TODRV\_HSE\_CMD\_AES128CBC\_HMAC\_SECURE\_MESSAGE (*C macro*), 102  
 TODRV\_HSE\_CMD\_AES128CBC\_HMAC\_UNSECURE\_MESSAGE (*C macro*), 102  
 TODRV\_HSE\_CMD\_AES128CBC\_IV\_ENCRYPT (*C macro*), 101  
 TODRV\_HSE\_CMD\_AES128CCM\_DECRYPT (*C macro*), 101  
 TODRV\_HSE\_CMD\_AES128CCM\_ENCRYPT (*C macro*), 101  
 TODRV\_HSE\_CMD\_AES128ECB\_DECRYPT (*C macro*), 101  
 TODRV\_HSE\_CMD\_AES128ECB\_ENCRYPT (*C macro*), 101  
 TODRV\_HSE\_CMD\_AES128GCM\_DECRYPT (*C macro*), 101  
 TODRV\_HSE\_CMD\_AES128GCM\_ENCRYPT (*C macro*), 101  
 TODRV\_HSE\_CMD\_AUTH\_ADMIN\_SESSION (*C macro*), 107  
 TODRV\_HSE\_CMD\_COMMIT\_NEW\_FW\_HASH (*C macro*), 108  
 TODRV\_HSE\_CMD\_COMPUTE\_CMIC (*C macro*), 102  
 TODRV\_HSE\_CMD\_COMPUTE\_HMAC (*C macro*), 101  
 TODRV\_HSE\_CMD\_COMPUTE\_HMAC\_FINAL (*C macro*), 101  
 TODRV\_HSE\_CMD\_COMPUTE\_HMAC\_INIT (*C macro*), 101  
 TODRV\_HSE\_CMD\_COMPUTE\_HMAC\_UPDATE (*C macro*), 101  
 TODRV\_HSE\_CMD\_DATA\_MIGRATION (*C macro*), 108  
 TODRV\_HSE\_CMD\_ECHO (*C macro*), 99  
 TODRV\_HSE\_CMD\_FINI\_ADMIN\_SESSION (*C macro*), 107  
 TODRV\_HSE\_CMD\_GET\_BOOT\_MEASUREMENT (*C macro*), 108  
 TODRV\_HSE\_CMD\_GET\_CERTIFICATE (*C macro*), 99  
 TODRV\_HSE\_CMD\_GET\_CERTIFICATE\_AND\_SIGN (*C macro*), 100  
 TODRV\_HSE\_CMD\_GET\_CERTIFICATE\_FINAL (*C macro*), 100  
 TODRV\_HSE\_CMD\_GET\_CERTIFICATE\_INIT (*C macro*), 100  
 TODRV\_HSE\_CMD\_GET\_CERTIFICATE\_SIGNING\_REQUEST (*C macro*), 99  
 TODRV\_HSE\_CMD\_GET\_CERTIFICATE\_SUBJECT\_CN (*C macro*), 99  
 TODRV\_HSE\_CMD\_GET\_CERTIFICATE\_UPDATE (*C macro*), 100  
 TODRV\_HSE\_CMD\_GET\_CERTIFICATE\_X509 (*C macro*), 100  
 TODRV\_HSE\_CMD\_GET\_CHALLENGE\_AND\_STORE (*C macro*), 100  
 TODRV\_HSE\_CMD\_GET\_HW\_SN (*C macro*), 98  
 TODRV\_HSE\_CMD\_GET\_HW\_VERSION (*C macro*), 98  
 TODRV\_HSE\_CMD\_GET\_KEY\_FINGERPRINT (*C macro*), 102  
 TODRV\_HSE\_CMD\_GET\_NVM\_SIZE (*C macro*), 99  
 TODRV\_HSE\_CMD\_GET\_PN (*C macro*), 98  
 TODRV\_HSE\_CMD\_GET\_PRODUCT\_ID (*C macro*), 99  
 TODRV\_HSE\_CMD\_GET\_PUBLIC\_KEY (*C macro*), 102  
 TODRV\_HSE\_CMD\_GET\_RANDOM (*C macro*), 99  
 TODRV\_HSE\_CMD\_GET\_SE\_MEASUREMENT (*C macro*), 108  
 TODRV\_HSE\_CMD\_GET\_SN (*C macro*), 98  
 TODRV\_HSE\_CMD\_GET\_STATE (*C macro*), 107  
 TODRV\_HSE\_CMD\_GET\_STATUS\_PIO\_CONFIG (*C macro*), 99  
 TODRV\_HSE\_CMD\_GET\_SW\_VERSION (*C macro*), 99  
 TODRV\_HSE\_CMD\_GET\_UNSIGN\_PUBLIC\_KEY (*C macro*), 102  
 TODRV\_HSE\_CMD\_INIT (*C macro*), 98  
 TODRV\_HSE\_CMD\_INIT\_ADMIN\_SESSION (*C macro*), 107  
 TODRV\_HSE\_CMD\_INVALIDATE\_NEW\_HASH (*C macro*), 108  
 TODRV\_HSE\_CMD\_LOADER\_BCAST\_COMMIT\_RELEASE (*C macro*), 108  
 TODRV\_HSE\_CMD\_LOADER\_BCAST\_GET\_INFO (*C macro*), 107  
 TODRV\_HSE\_CMD\_LOADER\_BCAST\_INITIALIZE\_UPGRADE (*C macro*), 108  
 TODRV\_HSE\_CMD\_LOADER\_BCAST\_RESTORE (*C macro*), 108  
 TODRV\_HSE\_CMD\_LOADER\_BCAST\_WRITE\_DATA (*C macro*), 108  
 TODRV\_HSE\_CMD\_LOCK (*C macro*), 107  
 TODRV\_HSE\_CMD\_LORA\_COMPUTE\_MIC (*C macro*), 106  
 TODRV\_HSE\_CMD\_LORA\_COMPUTE\_SHARED\_KEYS (*C macro*), 106  
 TODRV\_HSE\_CMD\_LORA\_DECRYPT\_JOIN (*C macro*), 106  
 TODRV\_HSE\_CMD\_LORA\_ENCRYPT\_PAYLOAD (*C macro*), 106  
 TODRV\_HSE\_CMD\_LORA\_GET\_APPEUI (*C macro*), 106  
 TODRV\_HSE\_CMD\_LORA\_GET\_DEVADDR (*C macro*), 106



TODRV_HSE_CMD_LORA_GET_DEVEUI ( <i>C macro</i> ), 106	TODRV_HSE_CMD_SET_STATUS_PIO_CONFIG ( <i>C macro</i> ), 99
TODRV_HSE_CMD_LORA_GET_JOIN_REQUEST ( <i>C macro</i> ), 106	TODRV_HSE_CMD_SHA256 ( <i>C macro</i> ), 102
TODRV_HSE_CMD_LORA_HANDLE_JOIN_ACCEPT ( <i>C macro</i> ), 106	TODRV_HSE_CMD_SHA256_FINAL ( <i>C macro</i> ), 102
TODRV_HSE_CMD_LORA_SECURE_PHYPAYLOAD ( <i>C macro</i> ), 106	TODRV_HSE_CMD_SHA256_INIT ( <i>C macro</i> ), 102
TODRV_HSE_CMD_LORA_UNSECURE_PHYPAYLOAD ( <i>C macro</i> ), 106	TODRV_HSE_CMD_SHA256_UPDATE ( <i>C macro</i> ), 102
TODRV_HSE_cmd_name_from_number ( <i>C function</i> ), 88	TODRV_HSE_CMD_SIGN ( <i>C macro</i> ), 100
TODRV_HSE_CMD_READ_NVM ( <i>C macro</i> ), 99	TODRV_HSE_CMD_SIGN_HASH ( <i>C macro</i> ), 100
TODRV_HSE_CMD_RENEW_ECC_KEYS ( <i>C macro</i> ), 102	TODRV_HSE_CMD_SLEEP ( <i>C macro</i> ), 99
TODRV_HSE_CMD_RENEW_SHARED_KEYS ( <i>C macro</i> ), 102	TODRV_HSE_CMD_STORE_NEW_TRUSTED_FW_HASH ( <i>C macro</i> ), 108
TODRV_HSE_CMD_RES ( <i>C macro</i> ), 98	TODRV_HSE_CMD_TLS_CALCULATE_FINISHED ( <i>C macro</i> ), 103
TODRV_HSE_CMD_SECLINK_AESMAC ( <i>C macro</i> ), 107	TODRV_HSE_CMD_TLS_GET_CERTIFICATE ( <i>C macro</i> ), 104
TODRV_HSE_CMD_SECLINK_AESMAC_GET_IV ( <i>C macro</i> ), 107	TODRV_HSE_CMD_TLS_GET_CERTIFICATE_FINAL ( <i>C macro</i> ), 104
TODRV_HSE_CMD_SECLINK_AESMAC_GET_NEW_KEYS ( <i>C macro</i> ), 107	TODRV_HSE_CMD_TLS_GET_CERTIFICATE_INIT ( <i>C macro</i> ), 104
TODRV_HSE_CMD_SECLINK_ARC4 ( <i>C macro</i> ), 107	TODRV_HSE_CMD_TLS_GET_CERTIFICATE_SLOT ( <i>C macro</i> ), 105
TODRV_HSE_CMD_SECLINK_ARC4_GET_IV ( <i>C macro</i> ), 107	TODRV_HSE_CMD_TLS_GET_CERTIFICATE_UPDATE ( <i>C macro</i> ), 104
TODRV_HSE_CMD_SECLINK_ARC4_GET_NEW_KEY ( <i>C macro</i> ), 107	TODRV_HSE_CMD_TLS_GET_CERTIFICATE_VERIFY ( <i>C macro</i> ), 105
TODRV_HSE_CMD_SECURE_MESSAGE ( <i>C macro</i> ), 105	TODRV_HSE_CMD_TLS_GET_CHANGE_CIPHER_SPEC ( <i>C macro</i> ), 105
TODRV_HSE_CMD_SECURE_MESSAGE_FINAL ( <i>C macro</i> ), 106	TODRV_HSE_CMD_TLS_GET_CLIENT_HELLO ( <i>C macro</i> ), 103
TODRV_HSE_CMD_SECURE_MESSAGE_INIT ( <i>C macro</i> ), 105	TODRV_HSE_CMD_TLS_GET_CLIENT_HELLO_FINAL ( <i>C macro</i> ), 103
TODRV_HSE_CMD_SECURE_MESSAGE_UPDATE ( <i>C macro</i> ), 106	TODRV_HSE_CMD_TLS_GET_CLIENT_HELLO_INIT ( <i>C macro</i> ), 103
TODRV_HSE_CMD_SET_AES_KEY ( <i>C macro</i> ), 107	TODRV_HSE_CMD_TLS_GET_CLIENT_HELLO_UPDATE ( <i>C macro</i> ), 103
TODRV_HSE_CMD_SET_CERTIFICATE ( <i>C macro</i> ), 99	TODRV_HSE_CMD_TLS_GET_CLIENT_KEY_EXCHANGE ( <i>C macro</i> ), 105
TODRV_HSE_CMD_SET_CERTIFICATE_FINAL ( <i>C macro</i> ), 99	TODRV_HSE_CMD_TLS_GET_FINISHED ( <i>C macro</i> ), 105
TODRV_HSE_CMD_SET_CERTIFICATE_INIT ( <i>C macro</i> ), 99	TODRV_HSE_CMD_TLS_GET_MASTER_SECRET ( <i>C macro</i> ), 102
TODRV_HSE_CMD_SET_CERTIFICATE_SIGNING_REQUEST ( <i>C macro</i> ), 99	TODRV_HSE_CMD_TLS_GET_MASTER_SECRET_DERIVED_KEYS ( <i>C macro</i> ), 103
TODRV_HSE_CMD_SET_CERTIFICATE_UPDATE ( <i>C macro</i> ), 99	TODRV_HSE_CMD_TLS_GET_RANDOM_AND_STORE ( <i>C macro</i> ), 102
TODRV_HSE_CMD_SET_CERTIFICATE_X509 ( <i>C macro</i> ), 100	TODRV_HSE_CMD_TLS_HANDLE_CERTIFICATE_REQUEST ( <i>C macro</i> ), 104
TODRV_HSE_CMD_SET_CMAC_KEY ( <i>C macro</i> ), 107	TODRV_HSE_CMD_TLS_HANDLE_CHANGE_CIPHER_SPEC ( <i>C macro</i> ), 105
TODRV_HSE_CMD_SET_HMAC_KEY ( <i>C macro</i> ), 107	TODRV_HSE_CMD_TLS_HANDLE_FINISHED ( <i>C macro</i> ), 105
TODRV_HSE_CMD_SET_MEASURE_BOOT ( <i>C macro</i> ), 108	TODRV_HSE_CMD_TLS_HANDLE_HELLO_VERIFY_REQUEST
TODRV_HSE_CMD_SET_NEXT_STATE ( <i>C macro</i> ), 106	
TODRV_HSE_CMD_SET_PRE_PERSONALIZATION_DATA ( <i>C macro</i> ), 106	
TODRV_HSE_CMD_SET_REMOTE_PUBLIC_KEY ( <i>C macro</i> ), 102	

(C macro), 103  
 TODRV\_HSE\_CMD\_TLS\_HANDLE\_MEDIATOR\_CERTIFICATE (C macro), 104  
 TODRV\_HSE\_CMD\_TLS\_HANDLE\_SERVER\_CERTIFICATE (C macro), 104  
 TODRV\_HSE\_CMD\_TLS\_HANDLE\_SERVER\_CERTIFICATE\_FINAL (C macro), 104  
 TODRV\_HSE\_CMD\_TLS\_HANDLE\_SERVER\_CERTIFICATE\_INIT (C macro), 104  
 TODRV\_HSE\_CMD\_TLS\_HANDLE\_SERVER\_CERTIFICATE\_UPDATE (C macro), 104  
 TODRV\_HSE\_CMD\_TLS\_HANDLE\_SERVER\_HELLO (C macro), 103  
 TODRV\_HSE\_CMD\_TLS\_HANDLE\_SERVER\_HELLO\_DONE (C macro), 104  
 TODRV\_HSE\_CMD\_TLS\_HANDLE\_SERVER\_HELLO\_FINAL (C macro), 104  
 TODRV\_HSE\_CMD\_TLS\_HANDLE\_SERVER\_HELLO\_INIT (C macro), 103  
 TODRV\_HSE\_CMD\_TLS\_HANDLE\_SERVER\_HELLO\_UPDATE (C macro), 103  
 TODRV\_HSE\_CMD\_TLS\_HANDLE\_SERVER\_KEY\_EXCHANGE (C macro), 104  
 TODRV\_HSE\_CMD\_TLS\_HANDLE\_SERVER\_KEY\_EXCHANGE\_FINAL (C macro), 104  
 TODRV\_HSE\_CMD\_TLS\_HANDLE\_SERVER\_KEY\_EXCHANGE\_INIT (C macro), 104  
 TODRV\_HSE\_CMD\_TLS\_HANDLE\_SERVER\_KEY\_EXCHANGE\_UPDATE (C macro), 104  
 TODRV\_HSE\_CMD\_TLS\_RENEW\_KEYS (C macro), 102  
 TODRV\_HSE\_CMD\_TLS\_RENEW\_KEYS\_ECDHE (C macro), 103  
 TODRV\_HSE\_CMD\_TLS\_RESET (C macro), 103  
 TODRV\_HSE\_CMD\_TLS\_SECURE\_MESSAGE (C macro), 105  
 TODRV\_HSE\_CMD\_TLS\_SECURE\_MESSAGE\_FINAL (C macro), 105  
 TODRV\_HSE\_CMD\_TLS\_SECURE\_MESSAGE\_INIT (C macro), 105  
 TODRV\_HSE\_CMD\_TLS\_SECURE\_MESSAGE\_UPDATE (C macro), 105  
 TODRV\_HSE\_CMD\_TLS\_SET\_CONFIG (C macro), 103  
 TODRV\_HSE\_CMD\_TLS\_SET\_CONNECTION\_ID\_EXT\_ID (C macro), 103  
 TODRV\_HSE\_CMD\_TLS\_SET\_MODE (C macro), 103  
 TODRV\_HSE\_CMD\_TLS\_SET\_SERVER\_EPUBLIC\_KEY (C macro), 103  
 TODRV\_HSE\_CMD\_TLS\_SET\_SERVER\_RANDOM (C macro), 103  
 TODRV\_HSE\_CMD\_TLS\_SET\_SESSION (C macro), 103  
 TODRV\_HSE\_CMD\_TLS\_UNSECURE\_MESSAGE (C macro), 105  
 TODRV\_HSE\_CMD\_TLS\_UNSECURE\_MESSAGE\_FINAL (C macro), 105  
 TODRV\_HSE\_CMD\_TLS\_UNSECURE\_MESSAGE\_INIT (C macro), 105  
 TODRV\_HSE\_CMD\_TLS\_UNSECURE\_MESSAGE\_UPDATE (C macro), 105  
 TODRV\_HSE\_CMD\_UNLOCK (C macro), 107  
 TODRV\_HSE\_CMD\_UNSECURE\_MESSAGE (C macro), 106  
 TODRV\_HSE\_CMD\_UNSECURE\_MESSAGE\_FINAL (C macro), 106  
 TODRV\_HSE\_CMD\_UNSECURE\_MESSAGE\_INIT (C macro), 106  
 TODRV\_HSE\_CMD\_UNSECURE\_MESSAGE\_UPDATE (C macro), 106  
 TODRV\_HSE\_CMD\_VALIDATE\_NEW\_FW\_HASH (C macro), 108  
 TODRV\_HSE\_CMD\_VERIFY (C macro), 100  
 TODRV\_HSE\_CMD\_VERIFY\_CA\_CERTIFICATE\_AND\_STORE (C macro), 100  
 TODRV\_HSE\_CMD\_VERIFY\_CERTIFICATE\_AND\_STORE (C macro), 100  
 TODRV\_HSE\_CMD\_VERIFY\_CHAIN\_CA\_CERTIFICATE\_AND\_STORE\_FINAL (C macro), 101  
 TODRV\_HSE\_CMD\_VERIFY\_CHAIN\_CA\_CERTIFICATE\_AND\_STORE\_INIT (C macro), 100  
 TODRV\_HSE\_CMD\_VERIFY\_CHAIN\_CA\_CERTIFICATE\_AND\_STORE\_UPDATE (C macro), 101  
 TODRV\_HSE\_CMD\_VERIFY\_CHAIN\_CERTIFICATE\_AND\_STORE\_FINAL (C macro), 100  
 TODRV\_HSE\_CMD\_VERIFY\_CHAIN\_CERTIFICATE\_AND\_STORE\_INIT (C macro), 100  
 TODRV\_HSE\_CMD\_VERIFY\_CHAIN\_CERTIFICATE\_AND\_STORE\_UPDATE (C macro), 100  
 TODRV\_HSE\_CMD\_VERIFY\_CHALLENGE\_SIGNATURE (C macro), 100  
 TODRV\_HSE\_CMD\_VERIFY\_CMAC (C macro), 102  
 TODRV\_HSE\_CMD\_VERIFY\_HASH\_SIGNATURE (C macro), 100  
 TODRV\_HSE\_CMD\_VERIFY\_HMAC (C macro), 101  
 TODRV\_HSE\_CMD\_VERIFY\_HMAC\_FINAL (C macro), 101  
 TODRV\_HSE\_CMD\_VERIFY\_HMAC\_INIT (C macro), 101  
 TODRV\_HSE\_CMD\_VERIFY\_HMAC\_UPDATE (C macro), 101  
 TODRV\_HSE\_CMD\_WRITE\_NVM (C macro), 99  
 TODRV\_HSE\_command\_data (C var), 84  
 TODRV\_HSE\_get\_ctx (C function), 37  
 TODRV\_HSE\_post\_command\_hook (C type), 87  
 TODRV\_HSE\_post\_write\_hook (C type), 87  
 TODRV\_HSE\_pre\_command\_hook (C type), 86  
 TODRV\_HSE\_prepare\_command\_data (C function), 85

- TODRV\_HSE\_prepare\_command\_data\_byte (*C function*), 85
- TODRV\_HSE\_reset\_command\_data (*C function*), 86
- TODRV\_HSE\_response\_data (*C var*), 84
- TODRV\_HSE\_send\_command (*C function*), 86
- TODRV\_HSE\_set\_command\_data (*C function*), 85
- TODRV\_HSE\_set\_lib\_hook\_post\_command (*C function*), 88
- TODRV\_HSE\_set\_lib\_hook\_post\_write (*C function*), 87
- TODRV\_HSE\_set\_lib\_hook\_pre\_command (*C function*), 87
- TODRV\_HSE\_trp\_config (*C function*), 33
- TODRV\_HSE\_trp\_last\_command\_duration (*C function*), 34
- TODRV\_HSE\_trp\_last\_command\_stack\_usage (*C function*), 35
- TODRV\_HSE\_trp\_read (*C function*), 34
- TODRV\_HSE\_trp\_write (*C function*), 34
- TODRV\_SSE\_get\_ctx (*C function*), 37
- TOSE\_access\_dummy\_data (*C function*), 39
- TOSE\_aes128cbc\_decrypt (*C function*), 52
- TOSE\_aes128cbc\_encrypt (*C function*), 51
- TOSE\_aes128cbc\_iv\_encrypt (*C function*), 52
- TOSE\_aes128ccm\_decrypt (*C function*), 55
- TOSE\_aes128ccm\_encrypt (*C function*), 54
- TOSE\_aes128ecb\_decrypt (*C function*), 56
- TOSE\_aes128ecb\_encrypt (*C function*), 56
- TOSE\_aes128gcm\_decrypt (*C function*), 54
- TOSE\_aes128gcm\_encrypt (*C function*), 53
- TOSE\_compute\_cmacc (*C function*), 61
- TOSE\_compute\_hmac (*C function*), 57
- TOSE\_compute\_hmac\_final (*C function*), 58
- TOSE\_compute\_hmac\_init (*C function*), 58
- TOSE\_compute\_hmac\_update (*C function*), 58
- TOSE\_ctx\_s (*C struct*), 109
- TOSE\_ctx\_s.drv (*C var*), 109
- TOSE\_ctx\_s.initialized (*C var*), 109
- TOSE\_ctx\_t (*C type*), 108
- TOSE\_drv\_ctx\_s (*C struct*), 108
- TOSE\_drv\_ctx\_s.api (*C var*), 109
- TOSE\_drv\_ctx\_s.func\_offset (*C var*), 109
- TOSE\_drv\_ctx\_s.log\_ctx (*C var*), 109
- TOSE\_drv\_ctx\_s.priv\_ctx (*C var*), 109
- TOSE\_drv\_ctx\_t (*C type*), 108
- TOSE\_fini (*C function*), 37
- TOSE\_get\_certificate (*C function*), 47
- TOSE\_get\_certificate\_signing\_request (*C function*), 44
- TOSE\_get\_certificate\_subject\_cn (*C function*), 43
- TOSE\_get\_certificate\_x509 (*C function*), 47
- TOSE\_get\_challenge\_and\_store (*C function*), 49
- TOSE\_get\_hardware\_serial\_number (*C function*), 37
- TOSE\_get\_hardware\_version (*C function*), 38
- TOSE\_get\_nvmm\_size (*C function*), 36
- TOSE\_get\_product\_id (*C function*), 38
- TOSE\_get\_product\_number (*C function*), 37
- TOSE\_get\_random (*C function*), 38
- TOSE\_get\_serial\_number (*C function*), 37
- TOSE\_get\_software\_version (*C function*), 38
- TOSE\_helper\_ecies\_seq\_auth\_remote\_1 (*C function*), 16
- TOSE\_helper\_ecies\_seq\_auth\_remote\_2 (*C function*), 17
- TOSE\_helper\_ecies\_seq\_auth\_TO (*C function*), 15
- TOSE\_helper\_ecies\_seq\_secure\_messaging (*C function*), 17
- TOSE\_helper\_get\_certificate\_x509\_and\_sign (*C function*), 22
- TOSE\_helper\_secure\_payload (*C function*), 18
- TOSE\_HELPER\_SECURE\_PAYLOAD\_GET\_CRYPTOGRAM (*C macro*), 20
- TOSE\_HELPER\_SECURE\_PAYLOAD\_GET\_CRYPTOGRAM\_LEN (*C macro*), 21
- TOSE\_HELPER\_SECURE\_PAYLOAD\_GET\_INITIAL\_VECTOR (*C macro*), 20
- TOSE\_HELPER\_SECURE\_PAYLOAD\_GET\_INITIAL\_VECTOR\_LEN (*C macro*), 20
- TOSE\_HELPER\_SECURE\_PAYLOAD\_GET\_SEQUENCE (*C macro*), 20
- TOSE\_HELPER\_SECURE\_PAYLOAD\_GET\_SEQUENCE\_LEN (*C macro*), 20
- TOSE\_helper\_set\_certificate\_x509 (*C function*), 22
- TOSE\_helper\_tls\_cleanup (*C function*), 27
- TOSE\_helper\_tls\_close (*C function*), 27
- TOSE\_helper\_tls\_ctx\_t (*C type*), 26
- TOSE\_helper\_tls\_do\_handshake (*C function*), 31
- TOSE\_helper\_tls\_do\_handshake\_step (*C function*), 31
- TOSE\_helper\_tls\_fini (*C function*), 27
- TOSE\_helper\_tls\_get\_certificate\_slot (*C function*), 31
- TOSE\_helper\_tls\_init\_session (*C function*), 26
- TOSE\_helper\_tls\_receive (*C function*), 32
- TOSE\_helper\_tls\_receive\_func (*C type*), 23
- TOSE\_helper\_tls\_recv (*C function*), 32
- TOSE\_helper\_tls\_secure\_record (*C type*), 25
- TOSE\_helper\_tls\_send (*C function*), 32
- TOSE\_helper\_tls\_send\_func (*C type*), 23

---

TOSE\_helper\_tls\_set\_cipher\_suites (C function), 28  
 TOSE\_helper\_tls\_set\_config\_certificate\_slot (C function), 29  
 TOSE\_helper\_tls\_set\_config\_mode (C function), 28  
 TOSE\_helper\_tls\_set\_fragment\_max\_size (C function), 28  
 TOSE\_helper\_tls\_set\_mode\_handshake\_only (C function), 29  
 TOSE\_helper\_tls\_set\_retransmission\_max (C function), 28  
 TOSE\_helper\_tls\_set\_retransmission\_timeout (C function), 27  
 TOSE\_helper\_tls\_set\_server\_name (C function), 29  
 TOSE\_helper\_tls\_setup\_cipher\_ctx (C type), 25  
 TOSE\_helper\_tls\_unsecure\_record (C type), 24  
 TOSE\_helper\_unsecure\_payload (C function), 19  
 TOSE\_helper\_verify\_chain\_ca\_certificate\_and\_store (C function), 21  
 TOSE\_helper\_verify\_chain\_certificate\_and\_store (C function), 21  
 TOSE\_init (C function), 37  
 TOSE\_read\_nvram (C function), 36  
 TOSE\_secure\_payload\_final (C function), 63  
 TOSE\_secure\_payload\_init (C function), 62  
 TOSE\_secure\_payload\_update (C function), 63  
 TOSE\_set\_certificate\_signing\_request\_dn (C function), 44  
 TOSE\_set\_certificate\_x509 (C function), 45  
 TOSE\_set\_certificate\_x509\_final (C function), 46  
 TOSE\_set\_certificate\_x509\_init (C function), 46  
 TOSE\_set\_certificate\_x509\_update (C function), 46  
 TOSE\_sha256 (C function), 39  
 TOSE\_sha256\_final (C function), 40  
 TOSE\_sha256\_init (C function), 39  
 TOSE\_sha256\_update (C function), 40  
 TOSE\_sign (C function), 41  
 TOSE\_sign\_hash (C function), 42  
 TOSE\_tls\_get\_certificate (C function), 77  
 TOSE\_tls\_get\_certificate\_final (C function), 78  
 TOSE\_tls\_get\_certificate\_init (C function), 78  
 TOSE\_tls\_get\_certificate\_slot (C function), 80  
 TOSE\_tls\_get\_certificate\_update (C function), 78  
 TOSE\_tls\_get\_certificate\_verify (C function), 79  
 TOSE\_tls\_get\_change\_cipher\_spec (C function), 79  
 TOSE\_tls\_get\_client\_hello (C function), 71  
 TOSE\_tls\_get\_client\_hello\_ext (C function), 71  
 TOSE\_tls\_get\_client\_hello\_final (C function), 73  
 TOSE\_tls\_get\_client\_hello\_init (C function), 72  
 TOSE\_tls\_get\_client\_hello\_update (C function), 72  
 TOSE\_tls\_get\_client\_key\_exchange (C function), 78  
 TOSE\_tls\_get\_finished (C function), 79  
 TOSE\_tls\_handle\_certificate\_request (C function), 77  
 TOSE\_tls\_handle\_change\_cipher\_spec (C function), 79  
 TOSE\_tls\_handle\_finished (C function), 80  
 TOSE\_tls\_handle\_hello\_verify\_request (C function), 73  
 TOSE\_tls\_handle\_mediator\_certificate (C function), 81  
 TOSE\_tls\_handle\_server\_certificate (C function), 75  
 TOSE\_tls\_handle\_server\_certificate\_final (C function), 76  
 TOSE\_tls\_handle\_server\_certificate\_init (C function), 75  
 TOSE\_tls\_handle\_server\_certificate\_update (C function), 76  
 TOSE\_tls\_handle\_server\_hello (C function), 73  
 TOSE\_tls\_handle\_server\_hello\_done (C function), 77  
 TOSE\_tls\_handle\_server\_hello\_final (C function), 75  
 TOSE\_tls\_handle\_server\_hello\_init (C function), 74  
 TOSE\_tls\_handle\_server\_hello\_update (C function), 74  
 TOSE\_tls\_handle\_server\_key\_exchange (C function), 76  
 TOSE\_tls\_handle\_server\_key\_exchange\_final (C function), 77  
 TOSE\_tls\_handle\_server\_key\_exchange\_init (C function), 76  
 TOSE\_tls\_handle\_server\_key\_exchange\_update (C function), 77  
 TOSE\_tls\_reset (C function), 70  
 TOSE\_tls\_secure\_payload (C function), 80  
 TOSE\_tls\_set\_cid\_ext\_id (C function), 71  
 TOSE\_tls\_set\_config (C function), 70  
 TOSE\_tls\_set\_session (C function), 70  
 TOSE\_tls\_unsecure\_payload (C function), 81

TOSE\_unsecure\_payload\_final (*C function*), 66  
TOSE\_unsecure\_payload\_init\_aead (*C function*),  
65  
TOSE\_unsecure\_payload\_init\_cbc (*C function*), 64  
TOSE\_unsecure\_payload\_update (*C function*), 65  
TOSE\_verify (*C function*), 42  
TOSE\_verify\_ca\_certificate\_and\_store (*C func-*  
*tion*), 48  
TOSE\_verify\_cmac (*C function*), 61  
TOSE\_verify\_hash\_signature (*C function*), 43  
TOSE\_verify\_hmac (*C function*), 59  
TOSE\_verify\_hmac\_final (*C function*), 60  
TOSE\_verify\_hmac\_init (*C function*), 59  
TOSE\_verify\_hmac\_update (*C function*), 60  
TOSE\_write\_nvm (*C function*), 35