Integration manual - TO136 on Linux device

TOSL

Release 4.6.4 (doc Pa/L/Ls/T/Ts)

Jul 18, 2018

Authored by *Trusted Objects*

Confidential

Contents

Confidential Information
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.

Confidential

CONFIDENTIAL INFORMATION
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.

Confidential

CONFIDENTIAL INFORMATION
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.

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.

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

Confidential

CONFIDENTIAL INFORMATION
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.

1. libTO overview

The libTO is to be integrated as part of your software to provide to your application an interface to easily deal with Secure Element features. It aims to help developers to work with TO, 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 hardware. Dynamic allocation is not used by the library, and it tries to use standard C APIs.

1.1 Overall architecture

Below is detailed the library architecture.

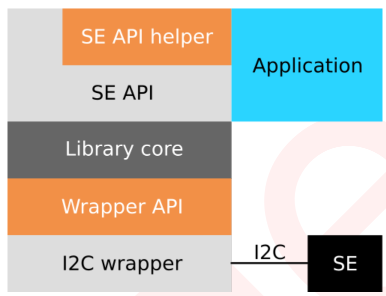

Fig. 1.1: Library architecture

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

These APIs are using library internal mechanisms to abstract TO 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*.

1.2 Library files tree

The library files tree structure is the following:

- **/include:** headers providing library APIs, see *Provided APIs*
- **/src:** library sources
- **/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

Confidential

CONFIDENTIAL INFORMATION
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.

1.3 Limitations

1.3.1 Multi-process environments

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.

If you need to use the library from different processes or execution threads, we recommend to embed the library into a dedicated process to handle concurrency, on which the other ones rely.

Confidential

CONFIDENTIAL INFORMATION
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.

2. Library setup and configuration

2.1 Linux installation instructions

In order to work with Secure Element from a Linux PC, please follow the installation instructions below.

Note: The following prerequisites are expected in this article:

- a StarterKit or a Secure Element soldered onto a development board
- an I2C device master connected to the TO
- the ability to build C code for the target hardware

2.1.1 I2C adapter

The library relies on an *I2C wrapper* to interact with the underlying I2C hardware.

For Linux, you have several I2C wrappers already available with the library:

- if you want to use a generic Linux I2C adapter, read *Use Linux generic I2C wrapper*
- if you want to use the CP2112 I2C master, read *Use CP2112 I2C adapter on Linux*, this is the adapter used by the StarterKit
- if you want to use RaspberryPi I2C, read *RaspberryPi (Raspbian) I2C configuration instructions*

for these ones, just use the appropriate "*i2c=*" parameter with the *configure* script, at the *Library configuration* step below.

If you want to use another adapter, you have to implement its support in the library, read *I2C wrapper implementation guidelines*.

Warning: A functional I2C wrapper is mandatory to use the library on your platform.

2.1.2 Library configuration

First, prepare autotools from the library directory:

```
autoreconf -fi
mkdir build && cd build
```

Configure the project:

```
../configure
```

configure script accepts several settings parameters, for details read *Library configuration with autotools*. At this step you should define which I2C wrapper you want to use. For example:

```
../configure i2c=linux_generic i2c-dev=/dev/i2c-0
```


CONFIDENTIAL INFORMATION
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.

to use the generic Linux I2C wrapper on the I2C-0 device.

You can also specify the location where the library has to be installed:

```
../configure i2c=... --prefix=/usr
```

in this example to install the library into your standard system paths instead of */usr/local* (default).

2.1.3 Build and install

Still from the same *build* directory, build the library:

```
make
```

and install:

```
sudo make install
```

2.1.4 Environment variables

By default, if you have not used the *–prefix* configure argument, everything is installed into */usr/local* subdirectories. In this case, be sure the following variables are defined:

```
export PYTHONPATH="$PYTHONPATH:/usr/local/lib/pythonX.X/site-packages/"
export LD_LIBRARY_PATH="$LD_LIBRARY_PATH:/usr/local/lib"
export PATH="$PATH:/usr/local/bin"
```

consider adding this to your *~/.bashrc*.

2.1.5 Test the library

Now you can use the *get_sn* example or *TOsh.py* shell with *get_sn* command to check if the library and its I2C wrapper are setup correctly.

With *get_sn* example program:

```
$ get_sn
Secure Element initialized
Secure Element serial number: 00 00 01 00 00 00 01 A0
```

and with *TOsh.py*:

```
$ TOsh.py
Welcome to the Secure Element shell.
Type help or ? to list commands.
Secure Element % get_sn
00000100000001a0
```

Confidential

CONFIDENTIAL INFORMATION
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.

2.2 Windows installation instructions (MSYS2)

In order to work with a Secure Element from a Windows PC using MSYS2, please follow the installation instructions below.

Note: The following prerequisites are expected in this article:

- a Secure Element StarterKit or a Secure Element soldered onto a development board
- an I2C device master connected to the TO

2.2.1 MSYS2

Download and install the 32-bits version of MSYS2 from msys2.github.io. Once installation is finished, open *MSYS2 MinGW 32-bit* shell, and run the following commands to install needed additional packages:

```
pacman -S mingw-w64-i686-toolchain mingw-w64-i686-libtool
pacman -S autoconf automake make
pacman -S mingw-w64-i686-python3
```

2.2.2 I2C adapter

The library relies on an *I2C wrapper* to interact with the I2C master device.

For Windows, you have a CP2112 I2C wrapper already available with the library, this is the adapter used by the Secure Element StarterKit. To use it, just use the *i2c=cp2112* parameter with the *configure* script, at the *Library configuration* step below.

If you want to use another adapter, you have to implement its support in the library, read *I2C wrapper implementation guidelines*.

Warning: A functional I2C wrapper is mandatory to use the library.

2.2.3 Library configuration

From MSYS shell, prepare autotools:

```
autoreconf -fi
mkdir build && cd build
```

Configure the project (here with CP2112 I2C wrapper):

```
../configure i2c=cp2112
```

configure script accepts several settings parameters, for details read *Library configuration with autotools*.

Confidential Information
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.

2.2.4 Build and install

Build the project from the previously created *build* directory:

```
make -j 5
```

and install:

```
make install
```

2.2.5 Environment variables

By default, everything is installed into */mingw32* subdirectories, be sure to define the following:

```
export PYTHONPATH="/mingw32/lib/site-packages"
export PATH="/mingw32/lib/:$PATH"
```

and consider adding this to your *~/.bashrc*.

2.2.6 Test the library

With a Secure Element connected to the PC, through a Secure Element StarterKit (CP2112) for example, run *TOsh.py* shell with *get_sn* or the *get_sn* example from MSYS2 MinGW 32-bits shell.

With *get_sn* example program:

```
$ get_sn
Secure Element initialized
Secure Element serial number: 00 00 01 00 00 00 01 A0
```

and with *TOsh.py*:

```
$ TOsh.py
Welcome to the Secure Element shell.
Type help or ? to list commands.
Secure Element % get_sn
00000100000001a0
```

2.3 Windows installation instructions (MinGW)

In order to work with a Secure Element from a Windows PC using MinGW, please follow the installation instructions below.

Warning: The recommended Windows installation environment is MSYS2, read *Windows installation instructions (MSYS2)*. Continue with this guide only if you really want to use MinGW.

Note: The following prerequisites are expected in this article:

CONFIDENTIAL INFORMATION
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.

- a StarterKit or a Secure Element soldered onto a development board
- an I2C device master connected to the TO

2.3.1 MinGW

Download and install MinGW from mingw.org. You need at least to select from *Basic Setup*: *mingw-developer-toolkit*, *mingw32-base* and *msys-base*.

Download pkg-config-lite and install it into your MinGW directory.

Download and install Python from python.org, choose custom installation, ensure the installer defines environment variables and includes binaries into the *PATH*, and set installation path to *C:\MinGW\opt\python3*.

2.3.2 I2C adapter

The library relies on an *I2C wrapper* to interact with the I2C master device.

For Windows, you have a CP2112 I2C wrapper already available with the library, this is the adapter used by the StarterKit. To use it, just use the *i2c=cp2112* parameter with the *configure* script, at the *Library configuration* step below.

If you want to use another adapter, you have to implement its support in the library, read *I2C wrapper implementation guidelines*.

> **Warning:** A functional I2C wrapper is mandatory to use the library.

2.3.3 Library configuration

From MSYS shell, prepare autotools:

```
autoreconf -fi
mkdir build && cd build
```

Configure the project:

```
../configure i2c=cp2112
```

configure script accepts several settings parameters, for details read *Library configuration with autotools*.

You can also use:

```
../configure --prefix=/usr
```

if you want to install into your standard system paths instead of into */usr/local*.

2.3.4 Build and install

Build the project from the previously created *build* directory:

CONFIDENTIAL INFORMATION
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.

```
make -j 5
```

and install:

```
make install
```

2.3.5 Environment variables

By default, everything is installed into */usr/local* MinGW subdirectories, if you have not set the *–prefix* configure argument. In this case, be sure the following variables are defined:

- *PYTHONPATH* should contain */usr/local/lib/site-packages/*
- *PATH* should contain */usr/local/bin* and */usr/local/lib*

or, if you used *–prefix=/usr*:

- *PYTHONPATH* should contain */usr/lib/site-packages/*
- *PATH* should contain */usr/bin* and */usr/lib*

consider adding this to your *~/.bashrc*.

2.3.6 Test the library

With a Secure Element connected to the PC, through a StarterKit (CP2112) for example, run *TOsh.py* shell with *get_sn* or the *get_sn* example from MSYS2 MinGW 32-bits shell.

With *get_sn* example program:

```
$ get_sn
Secure Element initialized
Secure Element serial number: 00 00 01 00 00 00 01 A0
```

and with *TOsh.py*:

```
$ TOsh.py
Welcome to the Secure Element shell.
Type help or ? to list commands.
Secure Element % get_sn
00000100000001a0
```

2.4 Library configuration with autotools

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

Note: Below it is assumed you have read the appropriate libTO installation guide.

CONFIDENTIAL INFORMATION
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.

2.4.1 Global settings

The *configure* script accepts the following parameters:

Flag	Description
i2c=	Select the I2C wrapper to use: cp2112, raspberrypi, linux_generic, net_bridge (default)
endian=	Force endianness: big, little
seclink=	Secure link engine to use: arc4, aeshmac, none (default)
–enable-debug	Library debug mode (default: disabled)
i2c_dev=	**ONLY FOR linux_generic WRAPPER** I2C device to use (*/dev/i2c-0* for example)
io_buffer_size=	(expert) Customize internal I/O buffer size
cmd_max_params_nb=	(expert) Customize maximum number of parameters taken by commands, for internal library use
tls_io_buffer_size=	(expert) Customize internal TLS I/O buffer size
tls_flight_buffer_size=	(expert) Customize internal TLS flight buffer size

2.4.1.1 Endianness

The *configure* script should automatically detect if your target system has the *endian.h* header file. Else, endianness settings may be got from preprocessor pre-defined macros if available.

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

In all cases, if you know your target endianness, you can force it by using the *endian* configure option presented above, example:

```
./configure endian=big ...
```

or:

```
./configure endian=little ...
```

2.4.2 Features settings

It may be interesting to only enable features required by the projet needs, in order to minimize library memory usage.

2.4.2.1 Macro. settings

These settings are used to enable or disable large sets of features (macroscopic settings). There are two kinds of features:

- the ones disabled by default, then define the relevant flag to enable
- the ones enabled by default, disabled by defining a flag

Confidential Information
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.

The *configure* script accepts the following parameters:

Flag	Description
–enable-lora	LoRa APIs (default: disabled)
–disable-lora-optimized	LoRa optimized API (default: enabled)
–enable-tls	TLS standard APIs (default: disabled)
–disable-tls-helper	TLS handshake helper (default: enabled)
–disable-tls-optimized	TLS optimized APIs (default: enabled)
–enable-dtls	DTLS APIs (default: disabled)
–disable-ecies-helper	ECIES sequence helper (default: enabled)
–disable-TO-info	Secure Element informations APIs (get_sn, get_pn, . . .) (default: enabled)
–disable-get-random	Random number generator API (default: enabled)
–disable-cert	Certificate management APIs (default: enabled)
–disable-signing	Signing and verification APIs (default: enabled)
–disable-aes-encrypt	AES encryption/decryption APIs (default: enabled)
–disable-sec-msg	Secure messaging APIs (default: enabled)
–disable-sha256	SHA256 hash APIs (default: enabled)
–disable-keys	Keys management APIs (default: enabled)
–disable-fingerprint	Fingerprint APIs (default: disabled)
–disable-hmac	HMAC computation/verification APIs (default: enabled)
–disable-cmac	CMAC computation/verification APIs (default: enabled)
–disable-nvm	NVM secure storage APIs (default: enabled)
–disable-status-pio-config	Secure Element status PIO settings API

2.4.2.2 Micro. 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 be defined 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.

There are the following exceptions which can not be disabled with a per-API granularity because it makes no sense:

- ***_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**
- **LoRa** APIs
- **TLS** APIs

These flags can be used with *configure* script as in the following example:

CONFIDENTIAL INFORMATION
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.

```
./configure ... CFLAGS='-DTO_DISABLE_API_GET_RANDOM'
```

here to disable the random number generator API.

Confidential

Confidential Information
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.

3. I2C wrapper

3.1 I2C wrapper

To be able to communicate with TO, 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 TO, 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).

3.1.1 Available wrappers

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

- **cp2112.c**: Linux Silicon Labs CP2112 wrapper, *Use CP2112 I2C adapter on Linux*
- **cp2112-win.c**: Windows Silicon Labs CP2112 wrapper
- **raspberrypi.c**: RaspberryPi (Raspbian) I2C wrapper, *RaspberryPi (Raspbian) I2C configuration instructions*
- **linux_generic.c**: Linux generic I2C wrapper, *Use Linux generic I2C wrapper*

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

3.2 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 TO.

3.2.1 Timeout

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

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.

3.2.2 Library debug mode

You may want to enable libTO debug mode 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_DEBUG** preprocessor flag can be defined to enable debug mode. If you are building the library with Autotools, use *./configure* with *–enable-debug* option.

CONFIDENTIAL INFORMATION
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.

3.2.3 I2C wrapper integration

3.2.3.1 Autotools

Details below are is interesting for you only if you want to integrate your wrapper with library Autotools (Unix or Windows platforms build). If it is not the case because you are working with an MCU, skip this section.

First of all, your I2C wrapper implementation should be included into the *wrapper* directory.

Add support for your I2C wrapper into the *configure.ac* file by adding a new line after CP2112, like the following:

```
AM_CONDITIONAL(ENABLE_I2C_MYWRAPPER, test x$I2C = mywrapper)
```

Add into the *wrapper/Makefile.am* an entry with the following form:

```
if ENABLE_I2C_MYWRAPPER
libi2c_wrapper_la_SOURCES = mywrapper.c
endif
```

Do autoreconf and prepare build:

```
autoreconf -fi
mkdir build && cd build
```

Configure, and select your own wrapper before building:

```
../configure i2c=mywrapper
make
```

And you can check the communications is OK by running:

```
./examples/get_sn
```

which should return the Secure Element serial number.

3.3 Use CP2112 I2C adapter on Linux

In this article are detailed instructions to make CP2112 I2C adapter working on Linux.

3.3.1 Make hid_cp2112 kernel module compatible with TO

The cp2112 I2C wrapper is using *hid_cp2112* Linux kernel module for TO communications. By default, the *hid_cp2112* driver hardcodes two values:

- the number of times to request transfer status before giving up waiting for transfer completion (set to 10)
- the time in milliseconds to wait for reading a response or a transfer status response (set to 50)

These hardcoded values does not fit Secure Element communication needs.

The attached patch **0001-drivers-hid-hid_cp2112-transfer-status-retries-and-r.patch** has to be applied to the hid_cp2112 kernel module, and the module has to be rebuilt. This patch allows to change module hardcoded values from *sysfs*.

CONFIDENTIAL INFORMATION
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.

3.3.1.1 Download kernel sources

From a terminal, run

```
uname -r
```

to know your kernel version.

Download the right kernel sources, for example:

```
wget https://www.kernel.org/pub/linux/kernel/v4.x/linux-4.7.2.tar.xz
```

3.3.1.2 Prepare module rebuild

Extract the downloaded archive and go to the sources directory.

Run:

```
make mrproper
```

and retrieve your current kernel configuration:

```
cp /lib/modules/`uname -r`/build/.config ./
cp /lib/modules/`uname -r`/build/Module.symvers ./
```

then do:

```
make prepare && make scripts
```

Finally, apply 0001-drivers-hid-hid_cp2112-transfer-status-retries-and-r.patch:

```
patch -p1 < 0001-drivers-hid-hid_cp2112-transfer-status-retries-and-r.patch
```

The attached patch **0002-drivers-hid-hid-cp2112-add-parameters-for-specials-gpios.patch** can be applied to enable CP2112 special GPIOs functions (clock output & RX/TX LEDs blink on transfers). This patch is optional.

3.3.1.3 Build hid-cp2112

Just run:

```
make M=drivers/hid
```

3.3.2 CP2112 needed kernel modules setup

We are going to properly configure modules needed by CP2112 I2C adapter.

3.3.2.1 Load modules

If your system uses gziped modules (see if you have *.ko.gz* files into */lib/modules/'uname -r'/kernel/drivers/hid/*), do the following:

TRUSTED OBJECTS
CONFIDENTIAL INFORMATION
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.
Confidential

```
gzip drivers/hid/hid-cp2112.ko
sudo cp drivers/hid/hid-cp2112.ko.gz /lib/modules/`uname -r`/kernel/drivers/hid/hid-
↪cp2112.ko.gz
```

else, if your system doesn't uses gziped modules, do:

```
sudo cp drivers/hid/hid-cp2112.ko /lib/modules/`uname -r`/kernel/drivers/hid/hid-
↪cp2112.ko
```

Reload the module:

```
sudo rmmod hid_cp2112
sudo modprobe hid_cp2112
```

Also ensure the **i2c_dev** module is loaded:

```
lsmod|grep i2c_dev
```

if the module is not present, do:

```
sudo modprobe i2c_dev
```

3.3.2.2 Udev rules

Copy the attached **50-cp2112.rules** udev rules file in the */etc/udev/rules.d* directory, and run:

```
sudo udevadm control --reload
```

These udev rules allows:

- every user to access read/write to the CP2112 device
- every user to access read/write to the hid-cp2112 driver *sysfs* settings (read *Module settings*)

3.3.2.3 Module settings

Now the *hid_cp2112* module allows to set/get previously hardcoded values from *sysfs*:

- */sys/module/hid_cp2112/parameters/xfer_status_retries*
- */sys/module/hid_cp2112/parameters/response_timeout*

these two parameters are set by the CP2112 wrapper, and they should be set to a big value (10000 for example).

For CP2112 LEDs:

- */sys/module/hid_cp2112/parameters/enable_special_rx*
- */sys/module/hid_cp2112/parameters/enable_special_tx*

these two parameters are disabled by default and can be enabled (set to 1) to enable rx/tx LEDs (only if the appropriate patch has been applied). The CP2112 module has to be disconnected then connected again to have these settings taken into account.

Confidential Information
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.

3.3.3 libTO CP2112 wrapper

The CP2112 wrapper is enabled by *configure* with the *i2c=cp2112* option. Then, configure the library build with:

```
../configure ... i2c=cp2112
```

This wrapper depends on *libudev* to automatically detect the HID/I2C device to use.

3.4 Use Linux generic I2C wrapper

The Linux generic I2C wrapper is based on Linux *i2c_dev* devices, having devices nodes accessible from **/dev/i2c-***.

If your I2C driver is correctly loaded, please ensure to load *i2c_dev* kernel module in order to have a device node from **/dev/i2c-***:

```
sudo modprobe i2c_dev
```

Note: The following prerequisites are expected in this article for the target system:

- it is running a Linux OS
- it has an I2C master device available from **/dev/i2c-***

3.4.1 Installation with autotools (recommended)

Just follow the *Linux installation instructions*, but at the *configure* time use the following parameters:

```
../configure i2c=linux_generic i2c_dev=/dev/i2c-0
```

replace **/dev/i2c-0** with the appropriate device node path.

3.4.2 Installation without autotools

It is assumed the TO library is already integrated into your development tool. Then you have to define the following for the project:

- ENABLE_I2C_LINUX_GENERIC
- TO_I2C_DEVICE set to "/dev/i2c-0"

replace **/dev/i2c-0** with the appropriate device node path.

3.4.3 Footnotes

Maybe this generic wrapper will not fit your I2C master device needs, and then it will be needed to fix it according to this device. The sources of this wrapper are available from libTO source tree, *wrapper/linux_generic.c*.

CONFIDENTIAL INFORMATION
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.

3.5 RaspberryPi (Raspbian) I2C configuration instructions

In order to use a Secure Element from a RaspberryPi, please follow the installation instructions below.

Note: This article explains how to use Linux I2C bitbanging with TO, not RaspberryPi hardware I2C as an internal clock stretching issue is present on it and causes troubles with TO.

3.5.1 TO library

Follow the *Linux installation instructions*, but at the *configure* time use the following parameters:

```
../configure i2c=raspberrypi
```

Note: This wrapper is able to control Secure Element power supply.

3.5.2 I2C bitbanging configuration

On your RaspberryPi, ensure your */boot/config.txt* file contains the following:

```
dtparam=i2c_arm=off
dtoverlay=i2c-gpio
```

Then copy the attached **i2c-gpio-overlay.dts** RaspberryPi GPIO overlay file to your RaspberryPi SD card.

Once logged-in on the RaspberryPi, run the following command:

```
dtc -@ -I dts -O dtb -o i2c-gpio.dtbo /path/to/i2c-gpio-overlay.dts
```

and copy the generated **i2c-gpio.dtbo** file to */boot/overlays/i2c-gpio.dtbo* (replace the existing file).

Edit */etc/modules* and add the following:

```
i2c-gpio
i2c-dev
```

After rebooting the RaspberryPi you should have something like the following output by running *dmesg|grep i2c*:

```
[    3.169346] i2c-gpio i2c@0: using pins 23 (SDA) and 24 (SCL)
[    3.176507] i2c /dev entries driver
```

and you should have a */dev/i2c-3* device present.

3.5.3 Connect Secure Element on the I2C bus

The Secure Element must be connected to the RaspberryPi as detailed on the following figure:

CONFIDENTIAL INFORMATION
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.

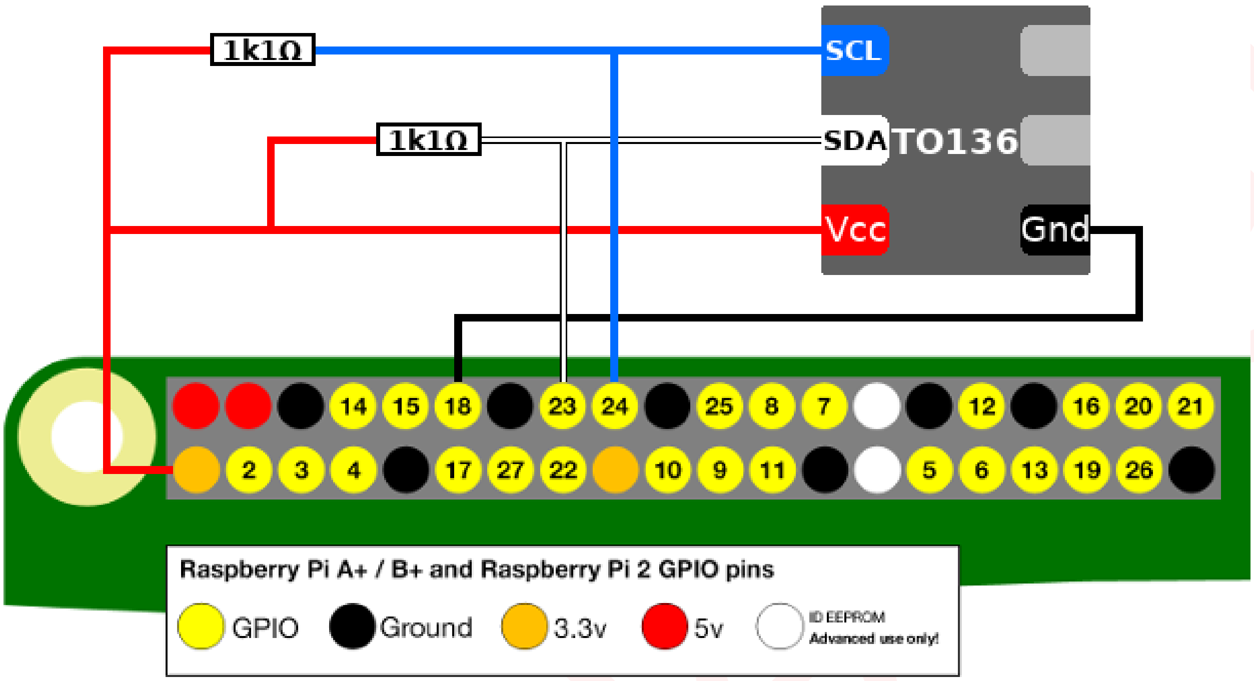

Fig. 3.1: Secure Element RaspberryPI wiring

Secure Element **Gnd** pin is connected to RaspberryPi pin 18, which is a GPIO. This allows the library I2C wrapper to control Secure Element power ON/OFF. This can be changed by editing the RaspberryPi I2C wrapper source file, from the library source tree, *wrapper/raspberrypi.c*.

I2C bitbanging is configured on the RaspberryPi pins 23 and 24, respectively connected to Secure Element **SDA** and **SCL**. This can be changed by editing the **i2c-gpio-overlay.dts** file previously used to configure bitbanging.

Secure Element **Vcc** pin is connected to a 3.3v RaspberryPi pin.

There are 1.1 kOhm resistors between **SCL/SDA** and 3.3v **Vcc** line.

CONFIDENTIAL INFORMATION
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.

4. Provided APIs

4.1 Secure Element API

These APIs are used to setup I2C communication and then send basic commands to Secure Element.

```
#include <TO.h>
```

4.1.1 I2C communication

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

4.1.1.1 I2C setup

Functions to manage connection with Secure Element.

int **TO_init** (void)

Initialize Secure Element communication.

If endianness is not explicitely defined through project settings macros, this function performs an automatic endianness detection.

Return TO_OK if initialization was successful.

int **TO_fini** (void)

Finish Secure Element communication.

Return TO_OK if finalization was successful.

int **TO_config** (unsigned char *i2c_addr*, unsigned char *misc_settings*)

Configure Secure Element communication.

See *TO_data_config()* for more details.

Parameters

- `i2c_addr`: I2C address to use
- `misc_settings`: Misc. settings byte. It have 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.

Return TO_OK if configuration was successful.

4.1.1.2 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*.

TRUSTED OBJECTS
CONFIDENTIAL INFORMATION
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.
Confidential

Warning: I2C must be initialized, see *TO_init ()*.

int **TO_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

Return

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

int **TO_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 recieved data
- `length`: Amount of data to read in bytes

Return

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

int **TO_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.

Parameters

- `duration`: Pointer to store last command duration in microseconds

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.

Return

- TO_OK if data has been read sucessfully

CONFIDENTIAL INFORMATION
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.
Confidential

- TO_ERROR if an internal error has occured

4.1.2 Secure Element functions

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

The following APIs are directly based on Secure Element APIs.

4.1.2.1 System

Misc. system functions.

int **`TO_get_serial_number`** (uint8_t *serial_number[TO_SN_SIZE]*)

Returns the unique Secure Element Serial Number.

Serial Number data are encoded on 8 bytes. The first 3 bytes identify Certificate Authority (CA), or the Factory if CA is not relevant. The last 5 bytes are the chip ID. Each Secure Element has an unique serial number.

Parameters

- `serial_number`: Returned device serial number

Return

- 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

int **`TO_get_product_number`** (uint8_t *product_number[TO_PN_SIZE]*)

Returns the Product Number of the TO.

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

Parameters

- `product_number`: Returned device product number

Return

- 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

CONFIDENTIAL INFORMATION
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.

- TO_ERROR: generic error

int **TO_get_hardware_version** (uint8_t *hardware_version[TO_HW_VERSION_SIZE]*)

Returns the Hardware Version of the TO.

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

- 00: reserved
- 01: SCB136i

Parameters

- `hardware_version`: Returned device hardware version

Return

- 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

int **TO_get_software_version** (uint8_t * *major*, uint8_t * *minor*, uint8_t * *revision*)

Returns the Software Version of the TO.

Software version major number is incremented on API change, minor number is incremented when there are changes in features without breaking the API, revision number is incremented for each new build (without major change, and with no API break).

Parameters

- `major`: Major number
- `minor`: Minor number
- `revision`: Revision number

Return

- 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

int **TO_get_random** (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.

CONFIDENTIAL INFORMATION
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.

Confidential

Parameters

- `random_length`: Requested random length
- `random`: Returned random number

Return

- TORSP_SUCCESS on success
- TORSP_NOT_AVAILABLE: random length out of range
- 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

4.1.2.2 Hashes

Hashing functions.

int **`TO_sha256`** (const uint8_t * *data*, const uint16_t *data_length*, uint8_t * *sha256*)

SHA256 computation.

Compute SHA256 hash on the given data.

Parameters

- `data`: Data to compute SHA256 on
- `data_length`: Data length, max. 512 bytes
- `sha256`: returned computed SHA256

Return

- 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

int **`TO_sha256_init`** (void)

Compute SHA256 on more than 512 bytes of data.

This function must be followed by calls to Secure Element_sha256_update() and *TO_sha256_final()*.

Return

- TORSP_SUCCESS on success
- TO_DEVICE_WRITE_ERROR: error writing data to Secure Element

CONFIDENTIAL INFORMATION
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.

- 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

int **`TO_sha256_update`** (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 *TO_sha256_init()*.

Parameters

- `data`: Data to compute SHA256 on
- `length`: Data length, max. 512 bytes

Return

- TORSP_SUCCESS on success
- TORSP_COND_OF_USE_NOT_SATISFIED if not called after *TO_sha256_init()* or *TO_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

int **`TO_sha256_final`** (uint8_t * *sha256*)
Returns the SHA256 hash of the data previously given.

This function must be called after *TO_sha256_init()* and *TO_sha256_update()*.

Parameters

- `sha256`: returned computed SHA256

Return

- TORSP_SUCCESS on success
- TORSP_COND_OF_USE_NOT_SATISFIED if not called after *TO_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

CONFIDENTIAL INFORMATION
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.
Confidential

4.1.2.3 Keys

Keys management functions.

int **TO_set_remote_public_key** (const uint8_t *key_index*, const uint8_t *public_key[TO_ECC_PUB_KEYSIZE]*, const uint8_t *signature[TO_SIGNATURE_SIZE]*)

Set remote public key.

This command requests the Secure Element to store, at the given index, a public key to be used in the ECIES process.

Parameters

- `key_index`: Index of the key to be set, starting from 0
- `public_key`: Key to set
- `signature`: Public key signature with the certificate previously sent with verify_certificate_and_store()

A signature is attached to the new public key and must be verified with the certificate previously sent using verify_certificate_and_store(). This command is disabled if public key is configured as non-writable during (pre-)personalization.

A CA signed certificate is first sent to the Secure Element using verify_certificate_and_store(), get_challenge_and_store(), and verify_challenge_signature() commands (remote authentication). If the Certificate Authority signature of the certificate is validated, the public key of the certificate is stored. Then, this certificate is used to verify the signature of any ephemeral public key sent using set_remote_public_key(). The signature is calculated on all bytes of the New Remote Public Key. If the signature verification failed, Secure Element will not store the public key. Please refer to Secure Element Datasheet - "Chain of Trust between Authentication and Secure Messaging" chapter for more details.

Return

- TORSP_SUCCESS on success
- TORSP_BAD_SIGNATURE: invalid signature
- 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

int **TO_renew_ecc_keys** (const uint8_t *key_index*)

Renew ECC keys pair.

Renews Elliptic Curve key pair for the corresponding index.

Parameters

- `key_index`: Index of the ECC key pair to renew, starting from 0

Return

- TORSP_SUCCESS on success
- TORSP_ARG_OUT_OF_RANGE: invalid key index

CONFIDENTIAL INFORMATION
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.

- 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

int **TO_get_public_key**(const uint8_t *key_index*, uint8_t *public_key[TO_ECC_PUB_KEYSIZE]*, uint8_t *signature[TO_SIGNATURE_SIZE]*)

Get the public key corresponding to the given index, and the signature of this public key.

Signature can be verified using the public key of the certificate returned by get_certificate().

Parameters

- `key_index`: Public key index
- `public_key`: The requested public key
- `signature`: Public key signature, can be verified using the public key of the certificate returned by GET_CERTIFICATE

This signature is calculated on all bytes of the Public Key in the TO response. Key pair used to generate and verify this signature is the one associated to certificate sent by the Secure Element in get_certificate() or get_certificate_and_sign() commands. Please refer to Secure Element Datasheet - "Chain of Trust between Authentication and Secure Messaging" chapter for more details.

Return

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

int **TO_get_unsigned_public_key**(const uint8_t *key_index*, uint8_t *public_key[TO_ECC_PUB_KEYSIZE]*)

Get the public key corresponding to the given index.

Return

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

Parameters

CONFIDENTIAL INFORMATION
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.

- `key_index`: Public key index
- `public_key`: The requested public key

int **`TO_renew_shared_keys`** (const uint8_t *key_index*, const uint8_t *public_key_index*)

Renew shared keys.

Renews shared keys (AES and HMAC), stored at the same index as Secure Element ephemeral public/private key pair.

Parameters

- `key_index`: Index of the Secure Element ephemeral public/private key pair, starting from 0
- `public_key_index`: Index where the remote public key is stored in the Secure Element, starting from 0.

Return

- 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

int **`TO_get_key_fingerprint`** (*TO_key_type_t key_type*, uint8_t *key_index*, uint8_t * *fingerprint[TO_KEY_FINGERPRINT_SIZE]*)

Get key fingerprint.

Retrieve the 3 bytes fingerprint of the key corresponding to given type and index.

Parameters

- `key_type`: Type of key
- `key_index`: Index of the key for given type starting from 0
- `fingerprint`: 3 bytes fingerprint of the key

See Secure Element Datasheet - "GET_KEY_FINGERPRINT" chapter for defails about fingerprint computation.

This function is available only for fixed keys.

Note: all first keys of the same type have the same index. For example, the first AES key and the first Public Key have both index 0.

Return

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

TRUSTED OBJECTS

CONFIDENTIAL INFORMATION
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.

Confidential

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

4.1.2.4 Encryption

Ciphered messaging functions.

int **TO_aes_encrypt** (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 TO, 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 TO.

Parameters

- `key_index`: Index of the key to use for data encryption, starting from 0
- `data`: Data to encrypt
- `data_length`: Length of the data to encrypt
- `initial_vector`: Initial vector
- `cryptogram`: Cryptogram

Return

- 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

int **TO_aes_iv_encrypt** (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 generates Initial Vector by using ENCRYPT command when possible.

Return

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

CONFIDENTIAL INFORMATION
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.

- 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

Parameters

- `key_index`: Index of the key to use for data encryption, starting from 0
- `initial_vector`: Random data (16 bytes)
- `data`: Data to encrypt
- `data_length`:
- `cryptogram`: Returned encrypted data

int **`TO_aes_decrypt`** (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*)

Reverse operation of encrypt().

Requires the initial vector provided by the encryption function.

Parameters

- `key_index`: Index of the key to use for data decryption, starting from 0
- `initial_vector`: Random data (16 bytes) generated by encrypt function
- `cryptogram`: Data to decrypt
- `cryptogram_length`: Cryptogram length, less or equal to 512 bytes
- `data`: returned decrypted data

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.

Return

- 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

4.1.2.5 MAC

Message Authentication Code functions (HMAC and CMAC).

Confidential

CONFIDENTIAL INFORMATION
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.

int **TO_compute_hmac** (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 compute_hmac_init(), compute_hmac_update(), . . . , compute_hmac_final().

Parameters

- `key_index`: Index of the key to use for HMAC calculation, starting from 0
- `data`: Data to compute HMAC on
- `data_length`:
- `hmac_data`: Computed HMAC

Return

- 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

int **TO_compute_hmac_init** (uint8_t *key_index*)

Compute HMAC on more than 512 bytes of data.

This is the first command of the sequence compute_hmac_init(), compute_hmac_update(), . . . , compute_hmac_final(). It is used to Secure Element send Key_index.

Parameters

- `key_index`: Index of the key to use for HMAC calculation, starting from 0

Return

- 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

int **TO_compute_hmac_update** (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.

CONFIDENTIAL INFORMATION
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.

Parameters

- `data`: Data to compute HMAC on
- `length`: Data length

Return

- TORSP_SUCCESS on success
- TORSP_COND_OF_USE_NOT_SATISFIED: need to call 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

int **`TO_compute_hmac_final`** (uint8_t *hmac[TO_HMAC_SIZE]*)

Returns computed HMAC.

This is the last command of the sequence compute_hmac_init(), compute_hmac_update(), …, compute_hmac_final().

Parameters

- `hmac`: Returned computed HMAC

Return

- TORSP_SUCCESS on success
- TORSP_COND_OF_USE_NOT_SATISFIED: need to call compute_hmac_init() and 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

int **`TO_verify_hmac`** (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 verify_hmac_init(), verify_hmac_update(), …, verify_hmac_final()

Parameters

- `key_index`: Index of the key to use for HMAC calculation, starting from 0
- `data`: Data to verify HMAC on
- `data_length`:

TRUSTED OBJECTS

CONFIDENTIAL INFORMATION
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.

- `hmac_data`: returned computed HMAC

Return

- 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

int **`TO_verify_hmac_init`** (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

- `key_index`: Index of the key to use for HMAC calculation, starting from 0

Return

- 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

int **`TO_verify_hmac_update`** (const uint8_t * *data*, uint16_t *length*)
Used to send data to verify HMAC on.

After calling verify_hmac_init() to provide key index, you can call 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 verify_hmac_final.

Parameters

- `data`: Data to verify HMAC on
- `length`: Data length

Return

- TORSP_SUCCESS on success

CONFIDENTIAL INFORMATION
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.

- 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

int **`TO_verify_hmac_final`** (const uint8_t *hmac[TO_HMAC_SIZE]*)

This command is used to send HMAC to verify.

Data was previously sent by the sequence verify_hmac_init(), verify_hmac_update(), . . . , verify_hmac_final(). This command succeed if the HMAC is correct for the given data.

Parameters

- `hmac`: HMAC to verify

Return

- TORSP_SUCCESS on success
- TORSP_BAD_SIGNATURE: verification failed
- TORSP_COND_OF_USE_NOT_SATISFIED: verify_hmac_init() or 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

int **`TO_compute_cmac`** (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

- `key_index`: Index of the key to use for CMAC calculation, starting from 0
- `data`: Data to compute CMAC on
- `data_length`:
- `cmac_data`: Returned computed CMAC

Return

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

Confidential Information
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.
Confidential

- 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

int **TO_verify_cmac** (const uint8_t *key_index*, const uint8_t * *data*, const uint16_t *data_length*, uint8_t *cmac_data[TO_CMAC_SIZE]*)

Verify CMAC.

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

Parameters

- `key_index`: Index of the key to use to compute the CMAC tag, starting from 0
- `data`: Data to verify CMAC on
- `data_length`:
- `cmac_data`: expected CMAC

Return

- 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

4.1.2.6 Secure messaging

Secure messaging functions, using AES128-CBC encryption and HMAC.

int **TO_secure_message** (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]*)

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

It is equivalent to call encrypt() command, then 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 TO, the TO's 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).

Parameters

- `aes_key_index`: Index of the key to use for data encryption, starting from 0
- `hmac_key_index`: Index of the key to use for HMAC, starting from 0
- `data`: Message to be secured

TRUSTED OBJECTS
CONFIDENTIAL INFORMATION
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.
Confidential

- `data_length`:
- `initial_vector`: Block of 16 random bytes generated by the Secure Element and required to decrypt the data
- `cryptogram`: Message cryptogram (same size as data)
- `hmac`: Message HMAC

Note: As padding is not handled by the TO, 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

Return

- 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

int **`TO_unsecure_message`** (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*)

Reverse operation of secure_message()

Data are decrypted only if the HMAC tag is valid.

Parameters

- `aes_key_index`: Index of the key to use for data decryption, starting from 0
- `hmac_key_index`: Index of the key to use for HMAC verification, starting from 0
- `initial_vector`: Initial vector for decryption
- `cryptogram`: Message cryptogram
- `cryptogram_length`:
- `hmac`: Expected HMAC
- `data`: Decrypted data

Return

- 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

CONFIDENTIAL INFORMATION
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.

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

4.1.2.7 Authentication

Certificates management and signature functions.

int **TO_sign** (const uint8_t *key_index*, uint8_t * *challenge*, const uint16_t *challenge_length*, uint8_t * *signature*)

Returns the Elliptic Curve Digital Signature of the given data.

Signature Size is twice the size of the ECC key in bytes.

Parameters

- `key_index`: Key index to use for signature
- `challenge`: Challenge to be signed
- `challenge_length`:
- `signature`: Returned challenge signature

Return

- 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

int **TO_verify** (const uint8_t *key_index*, uint8_t * *data*, const uint16_t *data_length*, const uint8_t * *signature*)

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 SET_REMOTE_PUBLIC_KEY command.

Parameters

- `key_index`: Key index to use for verification
- `data`: Data to verify signature on
- `data_length`:
- `signature`: Expected data signature

Return

- TORSP_SUCCESS on success
- TORSP_ARG_OUT_OF_RANGE: invalid key index
- TORSP_BAD_SIGNATURE: invalid signature

Confidential Information
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.
Confidential

- 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

int **`TO_sign_hash`** (const uint8_t *key_index*, const uint8_t *hash[TO_HASH_SIZE]*, uint8_t * *signature*)

Returns the Elliptic Curve Digital Signature of the given hash.

Signature Size is twice the size of the ECC key in bytes.

Parameters

- `key_index`: Key index to use for signature
- `hash`: Hash to be signed
- `signature`: Returned hash signature

Return

- 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

int **`TO_verify_hash_signature`** (const uint8_t *key_index*, const uint8_t *hash[TO_HASH_SIZE]*, const uint8_t * *signature*)

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

The public key used for the signature verification must be previously provided using the SET_REMOTE_PUBLIC_KEY command.

Parameters

- `key_index`: Key index to use for verification
- `hash`: Hash to verify signature on
- `signature`: Expected hash signature

Return

- 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

CONFIDENTIAL INFORMATION
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.
Confidential

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

int **TO_get_certificate_subject_cn** (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

- `certificate_index`: Requested certificate index
- `subject_cn`: Returned certificate subject common name null terminated string

Return

- 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

int **TO_get_certificate** (const uint8_t *certificate_index*, const *TO_certificate_format_t* *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

- `certificate_index`: Requested certificate index
- `format`: Requested certificate format
- `certificate`: Certificate, size depends on the certificate type (see TO_cert_*_t)

Return

- 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

CONFIDENTIAL INFORMATION
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.

- TO_ERROR: generic error

int **TO_get_certificate_x509** (const uint8_t *certificate_index*, uint8_t * *certificate*, uint16_t * *size*)

Returns one of the Secure Element certificates, x509 DER formated.

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

Parameters

- `certificate_index`: Requested certificate index
- `certificate`: Returned certificate data, this buffer must be at least TO_MAXSIZE
- `size`: Returned certificate real size (which is less or equal to 512 bytes)

Return

- 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

int **TO_get_certificate_and_sign** (const uint8_t *certificate_index*, const *TO_certificate_format_t format*, uint8_t * *challenge*, const uint16_t *challenge_length*, uint8_t * *certificate*, uint8_t * *signature*)

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

This command is equivalent to GET_CERTIFICATE and SIGN commands in only 1 message.

Parameters

- `certificate_index`: Index of the certificate to return, starting from 0
- `format`: Format of the TO's certificate, read the Secure Element Datasheet, "Certificates description" chapter
- `challenge`: Challenge to be signed
- `challenge_length`: Length of the challenge to be signed
- `certificate`: Certificate, size depends on the certificate type (see TO_cert_*_t)
- `signature`: Returned signature

Return

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

CONFIDENTIAL INFORMATION
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.

- TO_DEVICE_WRITE_ERROR: error writing data to Secure Element
- TO_DEVICE_READ_ERROR: error reading data from Secure Element
- TO_ERROR: generic error

int **TO_get_certificate_x509_and_sign** (const uint8_t *certificate_index*, uint8_t * *challenge*, const uint16_t *challenge_length*, uint8_t * *certificate*, uint16_t * *size*, uint8_t * *signature*)

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

This command is equivalent to GET_CERTIFICATE and SIGN commands in only 1 message.

Parameters

- `certificate_index`: Index of the certificate to return, starting from 0
- `challenge`: Challenge to be signed
- `challenge_length`: Length of the challenge to be signed
- `certificate`: Returned certificate data, this buffer must be at least TO_MAXSIZE
- `size`: Returned certificate real size (which is less or equal to 512 bytes)
- `signature`: Returned signature

Return

- 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

int **TO_verify_certificate_and_store** (const uint8_t *ca_key_id*, const *TO_certificate_format_t format*, uint8_t * *certificate*)

Requests to verify Certificate Authority Signature of the given certificate, if verification succeeds, this certificate is stored into Secure Element Memory.

This command is required before using GET_CHALLENGE_AND_STORE and VERIFY_CHALLENGE_SIGNATURE.

Parameters

- `ca_key_id`: Index of the Certificate Authority public Key
- `format`: Format of the certificate
- `certificate`: Certificate to be verified and stored

Return

- TORSP_SUCCESS on success

Confidential Information
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.

Confidential

- TORSP_NOT_AVAILABLE: certificate Format not supported
- 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

int **`TO_verify_ca_certificate_and_store`** (const uint8_t *ca_key_index*, const uint8_t *subca_key_index*, const uint8_t * *certificate*, const uint16_t *certificate_len*)

Requests to verify CA Certificate Authority Signature of the given certificate, if verification succeeds, this certificate is stored into Secure Element Memory.

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

Parameters

- `ca_key_index`: CA index to verify subCA
- `subca_key_index`: subCA index to store subCA
- `certificate`: Certificate to be verified and stored
- `certificate_len`: Certificate length

Return

- 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

int **`TO_get_challenge_and_store`** (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 VERIFY_CHALLENGE_SIGNATURE.

Parameters

- `challenge`: Returned challenge

Return

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

CONFIDENTIAL INFORMATION
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.

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

int **`TO_verify_challenge_signature`** (const uint8_t *signature[TO_SIGNATURE_SIZE]*)
Verifies if the given signature matches with the signature of the challenge previously sent by GET_CHALLENGE_AND_STORE, using the public key of the certificate previously sent by VERIFY_CERTIFICATE_AND_STORE.

Note: VERIFY_CERTIFICATE_AND_STORE must be called before this command. GET_CHALLENGE_AND_STORE must be called before this command.

Parameters

- `signature`: Challenge signature to verify

Return

- TORSP_SUCCESS on success
- TORSP_BAD_SIGNATURE: verification failed
- TORSP_COND_OF_USE_NOT_SATISFIED: VERIFY_CERTIFICATE_AND_STORE and GET_CHALLENGE_AND_STORE 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

int **`TO_verify_chain_certificate_and_store_init`** (const uint8_t *ca_key_index*)
Initialize certificate chain verification.

This command is required before using VERIFY_CHAIN_CERTIFICATE_AND_STORE_UPDATE.

Parameters

- `ca_key_index`: CA key index (use TO_CA_IDX_AUTO to enable Authority Key Identifier based CA detection)

Return

- 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

CONFIDENTIAL INFORMATION
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.

int **TO_verify_chain_certificate_and_store_update** (const uint8_t * *chain_certificate*, const uint16_t *chain_certificate_length*)

Update certificate chain verification with certificate chain data.

This command must be used after VERIFY_CHAIN_CERTIFICATE_AND_STORE_UPDATE_INIT and is required before using VERIFY_CHAIN_CERTIFICATE_AND_STORE_UPDATE_FINAL and can be repeated to deal with certificate chains longer than 512 bytes.

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)

Certificate chain can be cut anywhere.

Return

- TORSP_SUCCESS on success
- 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

int **TO_verify_chain_certificate_and_store_final** (void)

Finalize certificate chain verification.

This command must be used after VERIFY_CHAIN_CERTIFICATE_AND_STORE_UPDATE_UPDATE to verify last certificate and store final certificate.

Return

- TORSP_SUCCESS on success
- 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

int **TO_verify_chain_ca_certificate_and_store_init** (const uint8_t *ca_key_index*, const uint8_t *subca_key_index*)

Initialize CA certificate chain verification.

This command is required before using VERIFY_CHAIN_CA_CERTIFICATE_AND_STORE_UPDATE.

Parameters

CONFIDENTIAL INFORMATION
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.
Confidential

- `ca_key_index`: CA key index (use TO_CA_IDX_AUTO to enable Authority Key Identifier based CA detection)
- `subca_key_index`: subCA index to store subCA

Return

- 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

int **`TO_verify_chain_ca_certificate_and_store_update`** (const uint8_t * *chain_certificate*, const uint16_t *chain_certificate_length*)

Update CA certificate chain verification with certificate chain data.

This command must be used after VERIFY_CHAIN_CA_CERTIFICATE_AND_STORE_UPDATE_INIT and is required before using VERIFY_CHAIN_CA_CERTIFICATE_AND_STORE_UPDATE_FINAL and can be repeated to deal with certificate chains longer than 512 bytes.

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)

Certificate chain can be cut anywhere.

Return

- TORSP_SUCCESS on success
- 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

int **`TO_verify_chain_ca_certificate_and_store_final`** (void)

Finalize certificate chain verification.

This command must be used after VERIFY_CHAIN_CA_CERTIFICATE_AND_STORE_UPDATE_UPDATE to verify last certificate and store first intermediate CA certificate.

Return

- TORSP_SUCCESS on success
- TORSP_BAD_SIGNATURE: invalid signature

CONFIDENTIAL INFORMATION
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.

- 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

4.1.2.8 NVM

Functions to use Secure Element secure data storage.

int **`TO_write_nvm`**(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.

Return TO_OK if data has been written sucessfully

- TO_DEVICE_WRITE_ERROR: error writing data to Secure Element
- TO_ERROR if an internal error has occured

Parameters

- `offset`: Offset in zone to write data
- `data`: Buffer containing data to send
- `length`: Amount of data to send in bytes (512 bytes max.)
- `key`: Key used to read/write previous data

int **`TO_read_nvm`**(const uint16_t *offset*, void * *data*, unsigned int *length*, const uint8_t *key[TO_AES_KEYSIZE]*)

Read data from Secure Element NVM reserved zone.

Return TO_OK if data has been written sucessfully

- TO_DEVICE_WRITE_ERROR: error writing data to Secure Element
- TO_ERROR if an internal error has occured

Parameters

- `offset`: Offset in zone to read data
- `data`: Buffer to store data
- `length`: Amount of data to read in bytes (512 bytes max.)
- `key`: Key used to write data

int **`TO_get_nvm_size`**(uint16_t * *size*)

Get NVM reserved zone available size.

Return TO_OK if size has been retrieved sucessfully

- TO_DEVICE_WRITE_ERROR: error writing data to Secure Element
- TO_ERROR if an internal error has occured

CONFIDENTIAL INFORMATION
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.

Parameters

- `size`: NVM size

4.1.2.9 TLS

int **`TO_set_tls_server_random`** (uint8_t *random[TO_TLS_RANDOM_SIZE]*)

Set TLS server random.

Send TLS server random to Secure Element.

Parameters

- `random`: Server random including a timestamp as prefix

Return

- 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

int **`TO_set_tls_server_eph_pub_key`** (uint8_t *key_index*, uint8_t *ecc_params[TO_TLS_SERVER_PARAMS_SIZE]*, uint8_t *signature[TO_SIGNATURE_SIZE]*)

Set TLS server ephemeral public key.

Send TLS server ephemeral public key to Secure Element.

Parameters

- `key_index`: Index of the public key to update
- `ecc_params`: Includes curve type, format and name, length of the public key concatenated with the uncompression tag (0x04)
- `signature`: Signature of the concatenation of 'client_random', 'server_random' and 'ecc_params'

Return

- 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

CONFIDENTIAL INFORMATION
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.

int **TO_get_tls_random_and_store** (uint8_t *timestamp[TO_TIMESTAMP_SIZE]*, uint8_t *random[TO_TLS_RANDOM_SIZE]*)

Get TLS random.

Get TLS random from Secure Element.

Parameters

- `timestamp`: POSIX timestamp (seconds since January 1st 1970 00:00:00 UTC)
- `random`: Returned random challenge

Return

- 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

int **TO_get_tls_master_secret** (uint8_t *master_secret[TO_TLS_MASTER_SECRET_SIZE]*)

Get TLS master secret.

Request TLS master secret to Secure Element.

Parameters

- `master_secret`: returned master secret

Return

- TORSP_SUCCESS on success
- TORSP_ARG_OUT_OF_RANGE: invalid certificate index
- TO_INVALID_RESPONSE_LENGTH: invalid response length
- TO_DEVICE_WRITE_ERROR: error writing data to Secure Element
- TO_DEVICE_READ_ERROR: error reading data from Secure Element
- TO_ERROR: generic error

int **TO_renew_tls_keys** (const uint8_t *key_index*, const uint8_t *enc_key_index*, const uint8_t *dec_key_index*)

Renew TLS keys.

Renew TLS keys with a master secret derivation.

Parameters

- `key_index`: Index of TLS keys to renew
- `enc_key_index`: Index to store encryption AES/HMAC keys
- `dec_key_index`: Index to store decryption AES/HMAC keys

Return

TRUSTED OBJECTS
CONFIDENTIAL INFORMATION
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.
Confidential

- TORSP_SUCCESS on success
- TORSP_ARG_OUT_OF_RANGE: invalid certificate 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

int **`TO_renew_tls_keys_ecdhe`** (const uint8_t *kpriv_index*, const uint8_t *kpub_index*, const uint8_t *enc_key_index*, const uint8_t *dec_key_index*)

Derive master secret.

ECDHE method.

Parameters

- `kpriv_index`: Index of the private key to use
- `kpub_index`: Index of the remote public key to use
- `enc_key_index`: Index to store encryption AES/HMAC keys
- `dec_key_index`: Index to store decryption AES/HMAC keys

Return

- TORSP_SUCCESS on success
- TORSP_ARG_OUT_OF_RANGE: invalid certificate 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

int **`TO_tls_calculate_finished`** (const int *from*, const uint8_t *handshake_hash[TO_HASH_SIZE]*, uint8_t *finished[TO_TLS_FINISHED_SIZE]*)

Calculate finished.

Return

- TORSP_SUCCESS on success
- TO_DEVICE_WRITE_ERROR: error writing data to Secure Element136
- TO_DEVICE_READ_ERROR: error reading data from Secure Element136
- TO_ERROR: generic error

Parameters

- `from`: 0 if message is from client, 1 if it is from server
- `handshake_hash`: Hash of all handshake messages

Confidential Information
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.
Confidential

- `finished`: Result

4.1.2.10 TLS optimized

These APIs provides an easier way to use TLS.

int **TO_tls_reset** (void)
Reset TLS session.

Return

- 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_ERROR: generic error

int **TO_tls_set_mode** (const *TO_tls_mode_t* *mode*)
Set TLS mode (version and TLS/DTLS) (resets TLS handshake in case of change).

Return

- 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_ERROR: generic error

Parameters

- `mode`: TLS mode

int **TO_tls_get_client_hello** (const uint8_t *timestamp[TO_TIMESTAMP_SIZE]*, uint8_t * *client_hello*, uint16_t * *client_hello_len*)
Get TLS ClientHello.

Return

- 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

Parameters

- `timestamp`: Timestamp (seconds since epoch)
- `client_hello`: ClientHello payload
- `client_hello_len`: ClientHello payload length

Confidential Information
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.

int **TO_tls_handle_hello_verify_request** (const uint8_t * *hello_verify_request*, const uint32_t *hello_verify_request_len*)

Handle TLS HelloVerifyRequest.

Return

- 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

Parameters

- `hello_verify_request`: HelloVerifyRequest payload
- `hello_verify_request_len`: HelloVerifyRequest payload length

int **TO_tls_handle_server_hello** (const uint8_t * *server_hello*, const uint32_t *server_hello_len*)

Handle TLS ServerHello.

Return

- 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

Parameters

- `server_hello`: ServertHello payload
- `server_hello_len`: ServertHello payload length

int **TO_tls_handle_server_certificate_init** (const uint8_t *server_certificate_init[TO_TLS_SERVER_CERTIFICATE_IN*

Handle TLS Server Certificate header.

Return

- 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

Confidential Information
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.

Parameters

- `server_certificate_init`: Certificate payload header

int **`TO_tls_handle_server_certificate_update`** (const uint8_t * *server_certificate_update*, const uint32_t *server_certificate_update_len*)

Handle TLS Server Certificate partial payload.

Return

- 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

Parameters

- `server_certificate_update`: Certificate partial payload
- `server_certificate_update_len`: Certificate partial payload length

int **`TO_tls_handle_server_certificate_final`** (void)

Finish TLS Server Certificate handling.

Return

- 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

int **`TO_tls_handle_server_key_exchange`** (const uint8_t * *server_key_exchange*, const uint32_t *server_key_exchange_len*)

Handle TLS ServerKeyExchange.

Return

- 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

CONFIDENTIAL INFORMATION
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.

Parameters

- `server_key_exchange`: ServerKeyExchange payload
- `server_key_exchange_len`: ServerKeyExchange payload length

int **`TO_tls_handle_certificate_request`** (const uint8_t * *certificate_request*, const uint32_t *certificate_request_len*)

Handle TLS CertificateRequest.

Return

- 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

Parameters

- `certificate_request`: CertificateRequest payload
- `certificate_request_len`: CertificateRequest payload length

int **`TO_tls_handle_server_hello_done`** (const uint8_t *server_hello_done[TO_TLS_SERVER_HELLO_DONE_SIZE]*)

Handle TLS ServerHelloDone.

Return

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

Parameters

- `server_hello_done`: ServerHelloDone payload

int **`TO_tls_get_certificate`** (uint8_t * *certificate*, uint16_t * *certificate_len*)

Get TLS Certificate.

Return

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

TRUSTED OBJECTS

CONFIDENTIAL INFORMATION
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.

Confidential

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

Parameters

- `certificate`: Certificate payload
- `certificate_len`: Certificate payload length

int **`TO_tls_get_certificate_init`** (uint8_t *certificate[TO_TLS_CLIENT_CERTIFICATE_INIT_SIZE]*)
Get TLS Certificate initialization.

Return

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

Parameters

- `certificate`: Certificate payload

int **`TO_tls_get_certificate_update`** (uint8_t * *certificate*, uint16_t * *certificate_len*)
Get TLS Certificate update.

Return

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

Parameters

- `certificate`: Certificate payload
- `certificate_len`: Certificate payload length

int **`TO_tls_get_certificate_final`** (void)
Get TLS Certificate finalize.

Return

- 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_MEMORY_ERROR: internal I/O buffer overflow

CONFIDENTIAL INFORMATION
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.
Confidential

- TO_ERROR: generic error

int **TO_tls_get_client_key_exchange** (uint8_t * *client_key_exchange*, uint16_t * *client_key_exchange_len*)

Get TLS ClientKeyExchange.

Return

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

Parameters

- `client_key_exchange`: ClientKeyExchange payload
- `client_key_exchange_len`: ClientKeyExchange payload length

int **TO_tls_get_certificate_verify** (uint8_t *certificate_verify[TO_TLS_CERTIFICATE_VERIFY_MAXSIZE]*, uint16_t * *certificate_verify_len*)

Get TLS CertificateVerify.

Return

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

Parameters

- `certificate_verify`: CertificateVerify payload
- `certificate_verify_len`: CertificateVerify payload length

int **TO_tls_get_change_cipher_spec** (uint8_t *change_cipher_spec[TO_TLS_CHANGE_CIPHER_SPEC_SIZE]*)

Get TLS ChangeCipherSpec.

Return

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

Parameters

CONFIDENTIAL INFORMATION
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.

- `change_cipher_spec`: ChangeCipherSpec payload

int **`TO_tls_get_finished`** (uint8_t *finished[TO_TLS_FINISHED_PAYLOAD_SIZE]*)
Get TLS Finished.

Return

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

Parameters

- `finished`: Finish payload

int **`TO_tls_handle_change_cipher_spec`** (const uint8_t *change_cipher_spec[TO_TLS_CHANGE_CIPHER_SPEC_SIZE]*)
Handle TLS ChangeCipherSpec.

Return

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

Parameters

- `change_cipher_spec`: ChangeCipherSpec payload

int **`TO_tls_handle_finished`** (const uint8_t *finished[TO_TLS_FINISHED_PAYLOAD_SIZE]*)
Handle TLS Finished.

Return

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

CONFIDENTIAL INFORMATION
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.
Confidential

Parameters

- `finished`: Finished payload

int **`TO_tls_secure_message`** (const uint8_t *header[TO_TLS_HEADER_SIZE]*, const uint8_t * *data*, const uint16_t *data_len*, uint8_t *initial_vector[TO_INITIALVECTOR_SIZE]*, uint8_t * *cryptogram*, uint16_t * *cryptogram_len*)

Secure message with TLS.

Return

- 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

Parameters

- `header`: TLS header
- `data`: TLS data
- `data_len`: TLS data length
- `initial_vector`: Initial vector used to encrypt
- `cryptogram`: Securized message (without header)
- `cryptogram_len`: Securized message (without header) length

int **`TO_tls_secure_message_init`** (const uint8_t *header[TO_TLS_HEADER_SIZE]*, uint8_t *initial_vector[TO_INITIALVECTOR_SIZE]*)

Secure message with TLS initialization.

Return

- 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

Parameters

- `header`: TLS header
- `initial_vector`: Initial vector used to encrypt

CONFIDENTIAL INFORMATION
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.
Confidential

int **TO_tls_secure_message_update** (const uint8_t * *data*, const uint16_t *data_len*, uint8_t * *cryptogram*)

Update secure message data to secure message with TLS.

Return

- 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

Parameters

- `data`: TLS data
- `data_len`: TLS data length (must be 16 bytes aligned, last unaligned bytes must be sent with `TO_tls_secure_message_final`
- `cryptogram`: Securized data

int **TO_tls_secure_message_final** (const uint8_t * *data*, const uint16_t *data_len*, uint8_t * *cryptogram*, uint16_t * *cryptogram_len*)

Secure message with TLS finalization.

Return

- 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

Parameters

- `data`: TLS end data
- `data_len`: TLS end data length (must be less than 16 bytes)
- `cryptogram`: Securized message last blocks
- `cryptogram_len`: Securized message last blocks length

int **TO_tls_unsecure_message** (const uint8_t *header[TO_TLS_HEADER_SIZE]*, const uint8_t *initial_vector[TO_INITIALVECTOR_SIZE]*, const uint8_t * *cryptogram*, const uint16_t *cryptogram_len*, uint8_t * *data*, uint16_t * *data_len*)

Unsecure message with TLS.

Return

- TORSP_SUCCESS on success

Confidential Information
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.
Confidential

- 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

Parameters

- `header`: TLS header
- `initial_vector`: Initial vector used to encrypt
- `cryptogram`: Securized message (without header)
- `cryptogram_len`: Securized message (without header) length
- `data`: TLS data
- `data_len`: TLS data length

int **`TO_tls_unsecure_message_init`** (const uint16_t *cryptogram_len*, const uint8_t *header[TO_TLS_HEADER_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]*)

Unsecure message with TLS initialization.

Return

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

Parameters

- `cryptogram_len`: Cryptogram length
- `header`: TLS header
- `initial_vector`: Initial vector used to encrypt
- `last_block_iv`: Last AES block initial vector (penultimate block)
- `last_block`: Last AES block

int **`TO_tls_unsecure_message_update`** (const uint8_t * *cryptogram*, const uint16_t *cryptogram_len*, uint8_t * *data*, uint16_t * *data_len*)

Update unsecure message data to unsecure message with TLS.

Return

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

Confidential Information
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.

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

Parameters

- `cryptogram`: Securized message (without header and initial vector)
- `cryptogram_len`: Securized message (without header and initial vector) length
- `data`: TLS clear data
- `data_len`: TLS clear data length

int **`TO_tls_unsecure_message_final`** (void)

Unsecure message with TLS finalization.

Return

- TORSP_SUCCESS on success
- TORSP_BAD_SIGNATURE: invalid HMAC
- 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

4.1.2.11 LoRa

int **`TO_lora_compute_mic`** (const uint8_t * *data*, uint16_t *data_length*, uint32_t *address*, uint8_t *direction*, uint32_t *seq_counter*, uint8_t *mic[TO_LORA_MIC_SIZE]*)

Computes the LoRaMAC frame MIC field.

Return

- TORSP_SUCCESS on success
- TORSP_*: for any error occured while handling command
- TO_DEVICE_WRITE_ERROR: error writing data to Secure Element
- TO_DEVICE_READ_ERROR: error reading data from Secure Element
- TO_ERROR: generic error

Parameters

- `data`: Data buffer
- `data_length`: Data buffer size
- `address`: Frame address
- `direction`: Frame direction [0: uplink, 1 downlink]
- `seq_counter`: Frame sequence counter
- `mic`: Computed MIC field

CONFIDENTIAL INFORMATION
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.

Confidential

int **TO_lora_encrypt_payload** (const uint8_t * *data*, uint16_t *data_length*, const uint8_t * *fport*, uint32_t *address*, uint8_t *direction*, uint32_t *seq_counter*, uint8_t * *enc_buffer*)

Computes the LoRaMAC payload encryption.

Return

- TORSP_SUCCESS on success
- TORSP_*: for any error occured while handling command
- TO_DEVICE_WRITE_ERROR: error writing data to Secure Element
- TO_DEVICE_READ_ERROR: error reading data from Secure Element
- TO_ERROR: generic error

Parameters

- `data`: Data buffer
- `data_length`: Data buffer size
- `fport`: Frame port (as pointer to keep retrocompatibility)
- `address`: Frame address
- `direction`: Frame direction [0: uplink, 1 downlink]
- `seq_counter`: Frame sequence counter
- `enc_buffer`: Encrypted buffer

int **TO_lora_join_compute_mic** (const uint8_t * *data*, uint16_t *data_length*, uint8_t *mic[TO_LORA_MIC_SIZE]*)

Computes the LoRaMAC Join Request frame MIC field.

Return

- TORSP_SUCCESS on success
- TORSP_*: for any error occured while handling command
- TO_DEVICE_WRITE_ERROR: error writing data to Secure Element
- TO_DEVICE_READ_ERROR: error reading data from Secure Element
- TO_ERROR: generic error

Parameters

- `data`: Data buffer
- `data_length`: Data buffer size
- `mic`: Computed MIC field

int **TO_lora_decrypt_join** (const uint8_t * *data*, uint16_t *data_length*, uint8_t * *dec_buffer*)

Computes the LoRaMAC join frame decryption MIC field.

Return

- TORSP_SUCCESS on success

Confidential Information
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.
Confidential

- TORSP_*: for any error occured while handling command
- TO_DEVICE_WRITE_ERROR: error writing data to Secure Element
- TO_DEVICE_READ_ERROR: error reading data from Secure Element
- TO_ERROR: generic error

Parameters

- `data`: Data buffer
- `data_length`: Data buffer size
- `dec_buffer`: Decrypted buffer

int **`TO_lora_compute_shared_keys`** (const uint8_t * *app_nonce*, const uint8_t * *net_id*, uint16_t *dev_nonce*)

Computes the LoRaMAC join frame decryption.

Return

- TORSP_SUCCESS on success
- TORSP_*: for any error occured while handling command
- TO_DEVICE_WRITE_ERROR: error writing data to Secure Element
- TO_DEVICE_READ_ERROR: error reading data from Secure Element
- TO_ERROR: generic error

Parameters

- `app_nonce`: Application nonce
- `net_id`: Network ID
- `dev_nonce`: Device nonce

int **`TO_lora_get_app_eui`** (uint8_t *app_eui[TO_LORA_APPEUI_SIZE]*)

Get AppEUI.

Return

- TORSP_SUCCESS on success
- TORSP_*: for any error occured while handling command
- TO_DEVICE_WRITE_ERROR: error writing data to Secure Element
- TO_DEVICE_READ_ERROR: error reading data from Secure Element
- TO_ERROR: generic error

Parameters

- `app_eui`: Application EUI

int **`TO_lora_get_dev_eui`** (uint8_t *dev_eui[TO_LORA_DEVEUI_SIZE]*)

Get DevEUI.

Return

Confidential Information
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.
Confidential

- TORSP_SUCCESS on success
- TORSP_*: for any error occured while handling command
- TO_DEVICE_WRITE_ERROR: error writing data to Secure Element
- TO_DEVICE_READ_ERROR: error reading data from Secure Element
- TO_ERROR: generic error

Parameters

- `dev_eui`: Device EUI

4.1.2.12 LoRa optimized

These APIs provides an easier way to use LoRa.

int **`TO_lora_get_join_request_phypayload`** (uint8_t *data[TO_LORA_JOINREQUEST_SIZE]*)
Get encrypted join request payload.

Return

- TORSP_SUCCESS on success
- TORSP_*: for any error occured while handling command
- TO_DEVICE_WRITE_ERROR: error writing data to Secure Element
- TO_DEVICE_READ_ERROR: error reading data from Secure Element
- TO_ERROR: generic error

Parameters

- `data`: Join request payload

int **`TO_lora_handle_join_accept_phypayload`** (const uint8_t * *data*, const uint16_t *data_length*, uint8_t * *dec_buffer*)
Handle encrypted join accept payload.

Return

- TORSP_SUCCESS on success
- TORSP_*: for any error occured while handling command
- TO_DEVICE_WRITE_ERROR: error writing data to Secure Element
- TO_DEVICE_READ_ERROR: error reading data from Secure Element
- TO_ERROR: generic error

Parameters

- `data`: Join accept payload (MHDR + payload + MIC)
- `data_length`: Join accept payload size
- `dec_buffer`: Decrypted join accept payload

CONFIDENTIAL INFORMATION
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.

int **TO_lora_secure_phypayload** (const uint8_t *mhdr*, const uint8_t *fctrl*, const uint8_t * *fopts*, const uint8_t *fport*, const uint8_t * *payload*, const int *payload_size*, uint8_t * *enc_buffer*)

Encrypt PHYPayload.

Return

- TORSP_SUCCESS on success
- TORSP_*: for any error occured while handling command
- TO_DEVICE_WRITE_ERROR: error writing data to Secure Element
- TO_DEVICE_READ_ERROR: error reading data from Secure Element
- TO_ERROR: generic error

Parameters

- `mhdr`: MHDR
- `fctrl`: Frame control
- `fopts`: Frame options (optional, FCtrl FOptsLen part must be 0 if missing)
- `fport`: Frame port (optional, must be present if payload_size > 0)
- `payload`: payload to encrypt (optional)
- `payload_size`: payload size (must be 0 if payload is null)
- `enc_buffer`: Encrypted PHYPayload (size TO_LORA_MHDR_SIZE + TO_LORA_DEVADDR_SIZE + TO_LORA_FCTRL_SIZE + TO_LORA_FCNT_SIZE / 2 + FOptLen + (payload_size ? payload_size + 1 : 0) + TO_LORA_MIC_SIZE)

int **TO_lora_unsecure_phypayload** (const uint8_t * *data*, const uint16_t *data_length*, uint8_t * *dec_buffer*)

Decrypt PHYPayload.

Return

- TORSP_SUCCESS on success
- TORSP_*: for any error occured while handling command
- TO_DEVICE_WRITE_ERROR: error writing data to Secure Element
- TO_DEVICE_READ_ERROR: error reading data from Secure Element
- TO_ERROR: generic error

Parameters

- `data`: PHYPayload to decrypt
- `data_length`: PHYPayload size
- `dec_buffer`: Decrypted PHYPayload (size data_length - TO_LORA_MIC_SIZE)

CONFIDENTIAL INFORMATION
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.
Confidential

4.2 Helper API

These APIs are designed to make some complex Secure Element operations simpler.

```
#include <TO_helper.h>
```

4.2.1 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 TO
- authenticate remote device against TO
- prepare secure messaging

The two first steps are for mutual authentication between remote device and TO, 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.

4.2.1.1 Authenticate TO

int **`TO_helper_ecies_seq_auth_TO`** (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.

Parameters

- `certificate_index`: Index of the Secure Element certificate to use
- `challenge`: Challenge (randomly generated) to be provided to the Secure Element
- `TO_certificate`: Short certificate returned by Secure Element
- `challenge_signature`: Signature of the challenge by Secure Element

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

CONFIDENTIAL INFORMATION
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.

- verify challenge signature using Secure Element certificate public key if previous steps are validated, continue with the next ECIES step: *TO_helper_ecies_seq_auth_remote_1()* to authenticate the remote device.

Return TO_OK if this step is passed successfully.

4.2.1.2 Authenticate remote

int **`TO_helper_ecies_seq_auth_remote_1`** (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.

Parameters

- `ca_pubkey_index`: Index of Certificate Authority public key
- `remote_certificate`: Remote device standalone certificate
- `challenge`: Challenge returned by Secure Element 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 *TO_helper_ecies_seq_auth_remote_2()* to finalize remote device authentication.

Return TO_OK if this step is passed successfully, else:

- TORSP_BAD_SIGNATURE: the remote device certificate CA signature is invalid

int **`TO_helper_ecies_seq_auth_remote_2`** (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.

Parameters

- `challenge_signature`: 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

TRUSTED OBJECTS
CONFIDENTIAL INFORMATION
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.
Confidential

- compute the challenge signature After call you need to:
- check return value (see below) if previous steps are validated, continue with *TO_helper_ecies_seq_secure_messaging()*.

Return TO_OK if this step is passed successfully, else:

- TORSP_BAD_SIGNATURE: the challenge signature is invalid

4.2.1.3 Secure messaging

int **TO_helper_ecies_seq_secure_messaging** (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.

Parameters

- `remote_pubkey_index`: Index where the public key will be stored
- `ecc_keypair_index`: Index of the ECC key pair to renew
- `remote_eph_pubkey`: Remote device ephemeral public key
- `remote_eph_pubkey_signature`: Remote device ephemeral public key signature
- `TO_eph_pubkey`: Returned Secure Element ephemeral public key
- `TO_eph_pubkey_signature`: Secure Element ephemeral public key signature

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.

Return TO_OK if this step is passed successfully, else:

Confidential Information
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.

- TORSP_BAD_SIGNATURE: the remote device public key signature is invalid

4.2.2 TLS handshake

The following function is an easy-to-use TLS handshake abstraction.
It only needs a function to send, and a function to receive data.
Calling this function will do all the steps of the TLS handshake.

4.2.2.1 Handshake

int **TO_helper_tls_handshake_init** (void)

Initialize TLS handshake.

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

Return TO_OK if initialization succeed, else TO_ERROR

int **TO_helper_tls_handshake** (void * *ctx*, *TO_helper_tls_handshake_send_func* *send_func*, *TO_helper_tls_handshake_receive_func* *receive_func*)

Do TLS handshake.

This function does all the steps of a TLS handshake. It encapsulates TO payloads from optimized API in a TLS record, and send it on the network through given function. It decapsulates TLS records received from the network and send it to TO. This function uses `TO_helper_tls_handshake_init` and `TO_helper_tls_handshake_step`.

Parameters

- `ctx`: Opaque context to forward to given functions
- `send_func`: Function to send on network
- `receive_func`: Function to receive from network

Return TO_OK if data has been sent successfully, else TO_ERROR

int **TO_helper_tls_handshake_step** (void * *ctx*, *TO_helper_tls_handshake_send_func* *send_func*, *TO_helper_tls_handshake_receive_func* *receive_func*)

Do TLS handshake step.

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

Parameters

- `ctx`: Opaque context to forward to given functions
- `send_func`: Function to send on network
- `receive_func`: Function to receive from network

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

CONFIDENTIAL INFORMATION
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.

Confidential

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

4.2.2.2 Send message

int **TO_helper_tls_send_message** (uint8_t * *msg*, uint32_t *msg_len*, void * *ctx*, *TO_helper_tls_handshake_send_func* *send_func*)

Send TLS encrypted message.

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

Parameters

- `msg`: Message
- `msg_len`: Message length
- `ctx`: Opaque context to forward to given functions
- `send_func`: Function to send on network

Return TO_OK if message has been sent successfully, else TO_ERROR

4.2.2.3 Send callback

typedef int **(* TO_helper_tls_handshake_send_func)** (void **ctx*, const uint8_t **data*, const uint32_t *len*)

Handshake helper network send function.

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

Parameters

- `ctx`: Opaque context given to "TO_helper_tls_handshake"
- `data`: Data to send
- `len`: Length of data

Return TO_OK if data has been sent successfully, else TO_ERROR

4.2.2.4 Receive message

int **TO_helper_tls_receive_message** (uint8_t * *msg*, uint32_t *max_msg_len*, uint32_t * *msg_len*, void * *ctx*, *TO_helper_tls_handshake_receive_func* *receive_func*)

Receive TLS encrypted message.

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

Parameters

- `msg`: Message output buffer
- `max_msg_len`: Message output buffer length
- `msg_len`: Receive message length

Confidential Information
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.

- `ctx`: Opaque context to forward to given functions
- `receive_func`: Function to receive from network

Return TO_OK if message has been sent successfully, else TO_ERROR

int **TO_helper_tls_receive_message_with_timeout** (uint8_t * *msg*, uint32_t *max_msg_len*, uint32_t * *msg_len*, int32_t *timeout*, void * *ctx*, *TO_helper_tls_handshake_receive_func* *receive_func*)

Receive TLS encrypted message with timeout.

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

Parameters

- `msg`: Message output buffer
- `max_msg_len`: Message output buffer length
- `msg_len`: Receive message length
- `timeout`: Receive timeout in milliseconds (-1 for no timeout)
- `ctx`: Opaque context to forward to given functions
- `receive_func`: Function to receive from network

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

4.2.2.5 Receive callback

typedef int **(* TO_helper_tls_handshake_receive_func)** (void **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 “TO_helper_tls_handshake” to receive data from the network.

Parameters

- `ctx`: Opaque context given to “TO_helper_tls_handshake”
- `data`: Data output
- `len`: Length of data to read
- `read_len`: Length of data read
- `timeout`: Receive timeout in milliseconds (-1 for no timeout)

Return TO_OK if data has been sent successfully, else:

- TO_TIMEOUT: Receive timed out
- TO_ERROR: Other error

Confidential Information
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.

4.3 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 <TO_i2c_wrapper.h>
```

4.3.1 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()*.

4.3.1.0.1 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

4.3.2 I2C bus setup

int **TO_data_init** (void)

Initialize Secure Element communication bus.

Initializes I2C bus for Secure Element communications.

Return TO_OK if initialization was successful, else TO_ERROR

int **TO_data_fini** (void)

Terminate Secure Element communication bus.

Reset (stop) I2C bus used for Secure Element communications.

Return TO_OK if reset was successful, else TO_ERROR

TRUSTED OBJECTS
CONFIDENTIAL INFORMATION
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.
Confidential

int **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.

Parameters

- `config`: I2C configuration to use

This function is optional, and even if enabled by TO_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*.

Return 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.

4.3.3 Data transfers

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

Read data from Secure Element on I2C bus.

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

Parameters

- `data`: Buffer to store recieved data
- `length`: Amount of data to read in bytes

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

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

Write data to Secure Element on I2C bus.

Writes specified amount of data to the Secure Element on I2C bus. This function returns when all data in the buffer has been written, or if an error occured. The condition start have 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

Confidential Information
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.

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

4.3.4 Miscellaneous

int **`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 TO_I2C_WRAPPER_LAST_COMMAND_DURATION in your project in order to use it through *TO_last_command_duration()* API.

Parameters

- `duration`: Pointer to store last command duration in microseconds

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.

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

4.4 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 doesn't fit your needs.

```
#include <TO_cmd.h>
```

4.4.1 Data buffers

The following buffers are accessible.

unsigned char* **`TO_command_data`**

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

unsigned char* **`TO_response_data`**

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

4.4.2 Command data preparation

The following functions are used to prepare data before sending command to TO.

int **`TO_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.

Parameters

- `offset`: Buffer offset where to insert data

CONFIDENTIAL INFORMATION
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.

- `data`: Data to be copied into the buffer
- `len`: Data length

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

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

int **`TO_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

Return 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 *TO_reset_command_data()* has been called).

int **`TO_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

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

And to reset command context:

void **`TO_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 *TO_send_command()* (if command has been aborted for example).

4.4.3 Send command

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

int **`TO_send_command`**(const uint16_t *cmd*, uint16_t *cmd_data_len*, uint16_t * *resp_data_len*, uint8_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.

CONFIDENTIAL INFORMATION
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.

Parameters

- `cmd`: Command code (see TOCMD_* 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

Return 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

4.4.4 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.

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

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

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

Parameters

- `cmd`: Command code, see Secure Element command codes
- `cmd_data_len`: Command data length

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

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

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

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

Parameters

- `cmd`: Command code, see Secure Element command codes
- `cmd_data_len`: Command data length

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

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

typedef void **(* TO_post_command_hook)** (uint16_t *cmd*, uint16_t *cmd_data_len*, uint16_t *cmd_rsp_len*, uint8_t *cmd_status*)

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

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

Parameters

CONFIDENTIAL INFORMATION
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.

- `cmd`: Command code, see Secure Element command codes
- `cmd_data_len`: Command data length
- `cmd_rsp_len`: Command response length
- `cmd_status`: Command status

void **`TO_set_lib_hook_pre_command`**(*TO_pre_command_hook hook*)
Set a pre command hook (see TO_pre_command_hook).

Parameters

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

void **`TO_set_lib_hook_post_write`**(*TO_post_write_hook hook*)
Set a post write hook (see TO_post_write_hook).

Parameters

- `hook`: Post write hook function to set (NULL to disable).

void **`TO_set_lib_hook_post_command`**(*TO_post_command_hook hook*)
Set a post cmd hook (see TO_post_command_hook).

Parameters

- `hook`: Post cmd hook function to set (NULL to disable).

4.5 Types and definitions

LibTO types and definitions.

```
#include <TO_defs.h>
```

4.5.1 Library error codes

`TO_OK` 0x0000

`TO_MEMORY_ERROR` 0x0100

`TO_DEVICE_WRITE_ERROR` 0x0200

`TO_DEVICE_READ_ERROR` 0x0400

`TO_INVALID_CA_ID` 0x1000

`TO_INVALID_CERTIFICATE_FORMAT` 0x1100

`TO_INVALID_CERTIFICATE_NUMBER` 0x1200

`TO_INVALID_RESPONSE_LENGTH` 0x2000

`TO_SECLINK_ERROR` 0x2100

`TO_TIMEOUT` 0x2200

Confidential Information
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.

TO_AGAIN 0x2400

TO_NOT_IMPLEMENTED 0x8000

TO_ERROR 0xF000

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*.

4.5.2 Secure Element error codes

TORSP_SUCCESS ((unsigned char)0x90)

TORSP_UNKNOWN_CMD ((unsigned char)0x01)

TORSP_BAD_SIGNATURE ((unsigned char)0x66)

TORSP_INVALID_LEN ((unsigned char)0x67)

TORSP_NOT_AVAILABLE ((unsigned char)0x68)

TORSP_INVALID_PADDING ((unsigned char)0x69)

TO136RSP_COM_ERROR ((unsigned char)0x72)

TORSP_NEED_AUTHENTICATION ((unsigned char)0x80)

TORSP_COND_OF_USE_NOT_SATISFIED ((unsigned char)0x85)

TORSP_ARG_OUT_OF_RANGE ((unsigned char)0x88)

TORSP_SECLINK_RENEW_KEY ((unsigned char)0xFD)

TORSP_INTERNAL_ERROR ((unsigned char)0xFE)

4.5.3 Keys types

enum keytypes::**TO_key_type_e**

Secure Element key types

Values:

KTYPE_CERT_KPUB = 0x00

KTYPE_CERT_KPRIV = 0x01

KTYPE_CA_KPUB = 0x02

KTYPE_REMOTE_KPUB = 0x03

KTYPE_ECIES_KPUB = 0x04

KTYPE_ECIES_KPRIV = 0x05

KTYPE_ECIES_KAES = 0x06

KTYPE_ECIES_KMAC = 0x07

KTYPE_LORA_KAPP = 0x08

Confidential Information
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.

KTYPE_LORA_KNET = 0x09

KTYPE_LORA_KSAPP = 0x0A

KTYPE_LORA_KSNET = 0x0B

typedef enum TO_key_type_e **TO_key_type_t**

4.5.4 Certificates

enum certs::**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:

TO_CERTIFICATE_STANDALONE = TOCERTF_STANDALONE

TO_CERTIFICATE_SHORT = TOCERTF_SHORT

TO_CERTIFICATE_X509 = TOCERTF_X509

TO_CERTIFICATE_SHORT_V2 = TOCERTF_SHORT_V2

typedef enum TO_certificate_format_e **TO_certificate_format_t**

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**

TOCERTF_STANDALONE ((unsigned char)0x00)

TOCERTF_SHORT ((unsigned char)0x01)

TOCERTF_X509 ((unsigned char)0x02)

TOCERTF_SHORT_V2 ((unsigned char)0x03)

TOCERTF_VALIDITY_DATE_SIZE 7UL

TOCERTF_SUBJECT_NAME_SIZE 15UL

TO_CA_IDX_AUTO 0xFF
CA index to enable Authority Key Identifier based CA detection

struct TO_cert_standalone_s
#include <TO_defs.h> Standalone certificate structure

struct TO_cert_short_s
#include <TO_defs.h> Short certificate structure

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

Confidential Information
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.

4.5.5 Constants

enum consts::**TO_tls_mode_e**

Values:

TO_TLS_MODE_UNKNOWN = 0

TO_TLS_MODE_TLS = 0x10

TO_TLS_MODE_TLS_1_0 = TO_TLS_MODE_TLS | 0x1

TO_TLS_MODE_TLS_1_1 = TO_TLS_MODE_TLS | 0x2

TO_TLS_MODE_TLS_1_2 = TO_TLS_MODE_TLS | 0x3

TO_TLS_MODE_DTLS = 0x20

TO_TLS_MODE_DTLS_1_0 = TO_TLS_MODE_DTLS | 0x1

TO_TLS_MODE_DTLS_1_1 = TO_TLS_MODE_DTLS | 0x2

TO_TLS_MODE_DTLS_1_2 = TO_TLS_MODE_DTLS | 0x3

typedef enum TO_tls_mode_e **TO_tls_mode_t**

TO_CMDHEAD_SIZE 5UL

TO_RSPHEAD_SIZE 4UL

TO_MAXSIZE 512UL

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_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_SHA256_HASHSIZE 32UL

TO_HASH_SIZE TO_SHA256_HASHSIZE

TO_CHALLENGE_SIZE 32UL

TO_SN_SIZE (TO_SN_CA_ID_SIZE+TO_SN_NB_SIZE)

TO_SN_CA_ID_SIZE 3UL

TO_SN_NB_SIZE 5UL

TO_PN_SIZE 12UL

CONFIDENTIAL INFORMATION
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.
Confidential

TO_HW_VERSION_SIZE 2UL

TO_HWVERSION_SCB136I 01UL

TO_HWVERSION_EMU 0xFFFFUL

TO_SW_VERSION_SIZE 3UL

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_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_FINGERPRINT_SIZE 3UL

TO_TIMESTAMP_SIZE 4UL

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_CLIENT_HELLO_MAXSIZE (TO_TLS_HANDSHAKE_HEADER_SIZE + 144UL)

TO_TLS_SERVER_HELLO_DONE_SIZE TO_TLS_HANDSHAKE_HEADER_SIZE

TO_TLS_SERVER_CERTIFICATE_INIT_SIZE (TO_TLS_HANDSHAKE_HEADER_SIZE + 3UL)

TO_TLS_CLIENT_CERTIFICATE_INIT_SIZE (TO_TLS_HANDSHAKE_HEADER_SIZE + 6UL)

TO_TLS_CLIENT_CERTIFICATE_SIZE (TO_TLS_HANDSHAKE_HEADER_SIZE + 422UL)

TO_TLS_CLIENT_KEY_EXCHANGE_MAXSIZE (TO_TLS_HANDSHAKE_HEADER_SIZE + 66UL)

TO_TLS_CERTIFICATE_VERIFY_MAXSIZE (TO_TLS_HANDSHAKE_HEADER_SIZE + 76UL)

TO_TLS_CHANGE_CIPHER_SPEC_SIZE 1UL

CONFIDENTIAL INFORMATION
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.

TO_TLS_FINISHED_PAYLOAD_SIZE (TO_TLS_HANDSHAKE_HEADER_SIZE + 12UL)

TO_TLS_HEADER_SIZE 5UL

TO_TLS_HANDSHAKE_HEADER_SIZE 4UL

TO_ARC4_KEY_SIZE 16UL

TO_ARC4_INITIALVECTOR_SIZE 16UL

TO_I2CADDR_SIZE 1UL

TO_CRC_SIZE 2UL

TO_PPERSO_ID_SIZE 4UL

TO_PPERSO_SUBID_SIZE 1UL

TO_PPERSO_TAG_SIZE 4UL

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_JOINREQUEST_SIZE (TO_LORA_MHDR_SIZE + \ TO_LORA_APPEUI_SIZE + \ TO_LORA_DEVEUI_SIZE + \ TO

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

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

TO_STATE_PREPERSO ((unsigned char)0xA3)

TO_STATE_PERSO ((unsigned char)0x52)

Confidential Information
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.

Confidential

TO_STATE_NORMAL ((unsigned char)0x00)

TO_STATE_LOCKED ((unsigned char)0xFF)

4.5.6 Secure Element commands codes

TOCMD_GET_SN ((unsigned short)0x0001)

TOCMD_RES ((unsigned short)0x0000)

TOCMD_GET_PN ((unsigned short)0x0002)

TOCMD_GET_HW_VERSION ((unsigned short)0x0003)

TOCMD_GET_SW_VERSION ((unsigned short)0x0004)

TOCMD_GET_RANDOM ((unsigned short)0x0005)

TOCMD_ECHO ((unsigned short)0x0010)

TOCMD_SLEEP ((unsigned short)0x0011)

TOCMD_READ_NVM ((unsigned short)0x0021)

TOCMD_WRITE_NVM ((unsigned short)0x0022)

TOCMD_GET_NVM_SIZE ((unsigned short)0x0050)

TOCMD_SET_STATUS_PIO_CONFIG ((unsigned short)0x00B1)

TOCMD_GET_STATUS_PIO_CONFIG ((unsigned short)0x00B2)

TOCMD_GET_CERTIFICATE_SUBJECT_CN ((unsigned short)0x0046)

TOCMD_GET_CERTIFICATE ((unsigned short)0x0006)

TOCMD_SIGN ((unsigned short)0x0007)

TOCMD_VERIFY ((unsigned short)0x0012)

TOCMD_SIGN_HASH ((unsigned short)0x001E)

TOCMD_VERIFY_HASH_SIGNATURE ((unsigned short)0x001F)

TOCMD_GET_CERTIFICATE_AND_SIGN ((unsigned short)0x0008)

TOCMD_VERIFY_CERTIFICATE_AND_STORE ((unsigned short)0x0009)

TOCMD_VERIFY_CA_CERTIFICATE_AND_STORE ((unsigned short)0x0047)

TOCMD_GET_CHALLENGE_AND_STORE ((unsigned short)0x000A)

TOCMD_VERIFY_CHALLENGE_SIGNATURE ((unsigned short)0x000B)

TOCMD_VERIFY_CHAIN_CERTIFICATE_AND_STORE_INIT ((unsigned short)0x00AD)

TOCMD_VERIFY_CHAIN_CERTIFICATE_AND_STORE_UPDATE ((unsigned short)0x00AE)

TOCMD_VERIFY_CHAIN_CERTIFICATE_AND_STORE_FINAL ((unsigned short)0x00AF)

TOCMD_VERIFY_CHAIN_CA_CERTIFICATE_AND_STORE_INIT ((unsigned short)0x00B3)

TOCMD_VERIFY_CHAIN_CA_CERTIFICATE_AND_STORE_UPDATE ((unsigned short)0x00B4)

TOCMD_VERIFY_CHAIN_CA_CERTIFICATE_AND_STORE_FINAL ((unsigned short)0x00B5)

CONFIDENTIAL INFORMATION
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.
Confidential

TOCMD_COMPUTE_HMAC ((unsigned short)0x000C)

TOCMD_COMPUTE_HMAC_INIT ((unsigned short)0x0023)

TOCMD_COMPUTE_HMAC_UPDATE ((unsigned short)0x0024)

TOCMD_COMPUTE_HMAC_FINAL ((unsigned short)0x0025)

TOCMD_VERIFY_HMAC ((unsigned short)0x000D)

TOCMD_VERIFY_HMAC_INIT ((unsigned short)0x0026)

TOCMD_VERIFY_HMAC_UPDATE ((unsigned short)0x0027)

TOCMD_VERIFY_HMAC_FINAL ((unsigned short)0x0028)

TOCMD_AESCBC_ENCRYPT ((unsigned short)0x000E)

TOCMD_AESCBC_DECRYPT ((unsigned short)0x000F)

TOCMD_AESCBC_IV_ENCRYPT ((unsigned short)0x0020)

TOCMD_COMPUTE_CMAC ((unsigned short)0x001C)

TOCMD_VERIFY_CMAC ((unsigned short)0x001D)

TOCMD_SHA256 ((unsigned short)0x00A2)

TOCMD_SHA256_INIT ((unsigned short)0x00AA)

TOCMD_SHA256_UPDATE ((unsigned short)0x00AB)

TOCMD_SHA256_FINAL ((unsigned short)0x00AC)

TOCMD_SECURE_MESSAGE ((unsigned short)0x00A0)

TOCMD_UNSECURE_MESSAGE ((unsigned short)0x00A1)

TOCMD_SET_REMOTE_PUBLIC_KEY ((unsigned short)0x00A3)

TOCMD_RENEW_ECC_KEYS ((unsigned short)0x00A4)

TOCMD_GET_PUBLIC_KEY ((unsigned short)0x00A5)

TOCMD_GET_UNSIGNED_PUBLIC_KEY ((unsigned short)0x002E)

TOCMD_RENEW_SHARED_KEYS ((unsigned short)0x00A6)

TOCMD_GET_KEY_FINGERPRINT ((unsigned short)0x0019)

TOCMD_TLS_GET_RANDOM_AND_STORE ((unsigned short)0x0029)

TOCMD_TLS_RENEW_KEYS ((unsigned short)0x002A)

TOCMD_TLS_GET_MASTER_SECRET ((unsigned short)0x002B)

TOCMD_TLS_SET_SERVER_RANDOM ((unsigned short)0x002F)

TOCMD_TLS_SET_SERVER_EPUBLIC_KEY ((unsigned short) 0x002C)

TOCMD_TLS_RENEW_KEYS_ECDHE ((unsigned short) 0x002D)

TOCMD_TLS_COMPUTE_ECDH ((unsigned short)0x0030)

TOCMD_TLS_CALCULATE_FINISHED ((unsigned short)0x0031)

TOCMD_TLS_RESET ((unsigned short)0x00B6)

CONFIDENTIAL INFORMATION
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.

TOCMD_TLS_SET_MODE ((unsigned short)0x0042)

TOCMD_TLS_GET_CLIENT_HELLO ((unsigned short)0x0032)

TOCMD_TLS_HANDLE_HELLO_VERIFY_REQUEST ((unsigned short)0x0041)

TOCMD_TLS_HANDLE_SERVER_HELLO ((unsigned short)0x0033)

TOCMD_TLS_HANDLE_SERVER_CERTIFICATE_INIT ((unsigned short)0x0043)

TOCMD_TLS_HANDLE_SERVER_CERTIFICATE_UPDATE ((unsigned short)0x0044)

TOCMD_TLS_HANDLE_SERVER_CERTIFICATE_FINAL ((unsigned short)0x0045)

TOCMD_TLS_HANDLE_SERVER_KEY_EXCHANGE ((unsigned short)0x0035)

TOCMD_TLS_HANDLE_CERTIFICATE_REQUEST ((unsigned short)0x0036)

TOCMD_TLS_HANDLE_SERVER_HELLO_DONE ((unsigned short)0x0037)

TOCMD_TLS_GET_CERTIFICATE ((unsigned short)0x0038)

TOCMD_TLS_GET_CERTIFICATE_INIT ((unsigned short)0x00BD)

TOCMD_TLS_GET_CERTIFICATE_UPDATE ((unsigned short)0x00BE)

TOCMD_TLS_GET_CERTIFICATE_FINAL ((unsigned short)0x00BF)

TOCMD_TLS_GET_CLIENT_KEY_EXCHANGE ((unsigned short)0x0039)

TOCMD_TLS_GET_CERTIFICATE_VERIFY ((unsigned short)0x003A)

TOCMD_TLS_GET_CHANGE_CIPHER_SPEC ((unsigned short)0x003B)

TOCMD_TLS_GET_FINISHED ((unsigned short)0x003C)

TOCMD_TLS_HANDLE_CHANGE_CIPHER_SPEC ((unsigned short)0x003D)

TOCMD_TLS_HANDLE_FINISHED ((unsigned short)0x003E)

TOCMD_TLS_SECURE_MESSAGE ((unsigned short)0x003F)

TOCMD_TLS_SECURE_MESSAGE_INIT ((unsigned short)0x00B7)

TOCMD_TLS_SECURE_MESSAGE_UPDATE ((unsigned short)0x00B8)

TOCMD_TLS_SECURE_MESSAGE_FINAL ((unsigned short)0x00B9)

TOCMD_TLS_UNSECURE_MESSAGE ((unsigned short)0x0040)

TOCMD_TLS_UNSECURE_MESSAGE_INIT ((unsigned short)0x00BA)

TOCMD_TLS_UNSECURE_MESSAGE_UPDATE ((unsigned short)0x00BB)

TOCMD_TLS_UNSECURE_MESSAGE_FINAL ((unsigned short)0x00BC)

TOCMD_LORA_GET_APPEUI ((unsigned short)0x0108)

TOCMD_LORA_GET_DEVEUI ((unsigned short)0x0109)

TOCMD_LORA_COMPUTE_MIC ((unsigned short)0x010A)

TOCMD_LORA_ENCRYPT_PAYLOAD ((unsigned short)0x010B)

TOCMD_LORA_DECRYPT_JOIN ((unsigned short)0x010C)

TOCMD_LORA_COMPUTE_SHARED_KEYS ((unsigned short)0x010D)

Confidential Information
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.

TOCMD_LORA_GET_DEVADDR ((unsigned short)0x0110)

TOCMD_LORA_GET_JOIN_REQUEST ((unsigned short)0x0100)

TOCMD_LORA_HANDLE_JOIN_ACCEPT ((unsigned short)0x0101)

TOCMD_LORA_SECURE_PHYPAYLOAD ((unsigned short)0x0102)

TOCMD_LORA_UNSECURE_PHYPAYLOAD ((unsigned short)0x0103)

TOCMD_SET_PRE_PERSONALIZATION_DATA ((unsigned short)0x0013)

TOCMD_SET_PERSONALIZATION_DATA ((unsigned short)0x0014)

TOCMD_SET_NEXT_STATE ((unsigned short)0x0015)

TOCMD_GET_STATE ((unsigned short)0x0016)

TOCMD_LOCK ((unsigned short)0x0017)

TOCMD_UNLOCK ((unsigned short)0x0018)

TOCMD_SET_AES_KEY ((unsigned short)0x00A7)

TOCMD_SET_HMAC_KEY ((unsigned short)0x00A8)

TOCMD_SET_CMAC_KEY ((unsigned short)0x00A9)

TOCMD_SECLINK_ARC4 ((unsigned short)0xFF00)

TOCMD_SECLINK_ARC4_GET_IV ((unsigned short)0xFF01)

TOCMD_SECLINK_ARC4_GET_NEW_KEY ((unsigned short)0xFF04)

TOCMD_SECLINK_AESHMAC ((unsigned short)0xFF02)

TOCMD_SECLINK_AESHMAC_GET_IV ((unsigned short)0xFF03)

TOCMD_SECLINK_AESHMAC_GET_NEW_KEYS ((unsigned short)0xFF05)

Confidential

CONFIDENTIAL INFORMATION
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.

5. Miscellany guides

5.1 Migration

5.1.1 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.

Standard TLS APIs have been disabled by default. Then, if you need standard TLS APIs in your project, you now have to explicitly enable these features.

5.1.1.1 Configure options (Linux project)

The following *configure* options are useless because this is now the default setting:

- –disable-tls

If required, to enable these feature for your project you can use:

- –enable-tls

See *Library configuration with autotools*. to properly configure libTO.

5.1.2 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 explicitely disable these features.

DTLS remains disabled and has to be explicitely enabled if needed.

5.1.2.1 Configure options (Linux project)

The following *configure* options are useless because this is now the default setting:

- –enable-lora
- –enable-lora-optimized
- –enable-tls-optimized
- –enable-tls-helper

If not required, to disable these feature for your project you can use:

- –disable-lora
- –disable-lora-optimized
- –disable-tls-optimized
- –disable-tls-helper

See *Library configuration with autotools*. to properly configure libTO.

Confidential Information
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.

5.1.3 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.

5.1.3.1 Changed APIs

The API `TO_tls_get_certificate()` has changed, with a new length output parameter.

5.1.4 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.

5.1.4.1 Renamed files

The library core files, *src/main.c* and *src/main.h*, has been renamed *src/core.c* and *src/core.h*.

5.1.5 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.

5.1.5.1 Renamed APIs

The following header files have been renamed:

- **include/to136.h** to **include/TO.h**
- **include/to136_helper.h** to **include/TO_helper.h**
- **include/to136_defs.h** to **TO_defs.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_...**.

5.1.5.2 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.

5.1.5.3 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.

CONFIDENTIAL INFORMATION
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.

5.1.6 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.

5.1.6.1 Headers

Include TO.h instead of TO_cli.h.

5.1.6.2 Defines

TO_I2C_WRAPPER_CONFIG replaces TO_CLI_I2C_WRAPPER_CONFIG. TO_LIB_INTERNAL_IO_BUFFER_SIZE replaces TO_CLI_INTERNAL_IO_BUFFER_SIZE.

5.1.6.3 Autotools

For Unix platforms, pkg-config file TO.pc replaces TO_client.pc.

Confidential

CONFIDENTIAL INFORMATION
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.

Index

T

CONFIDENTIAL INFORMATION
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.

CONFIDENTIAL INFORMATION
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.