Integration manual - Administration commands

TODO003-A

Release 6.3.9

January 27, 2023

Authored by *Trusted Objects*

Contents

1. Legal

1.1 Disclaimer

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

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

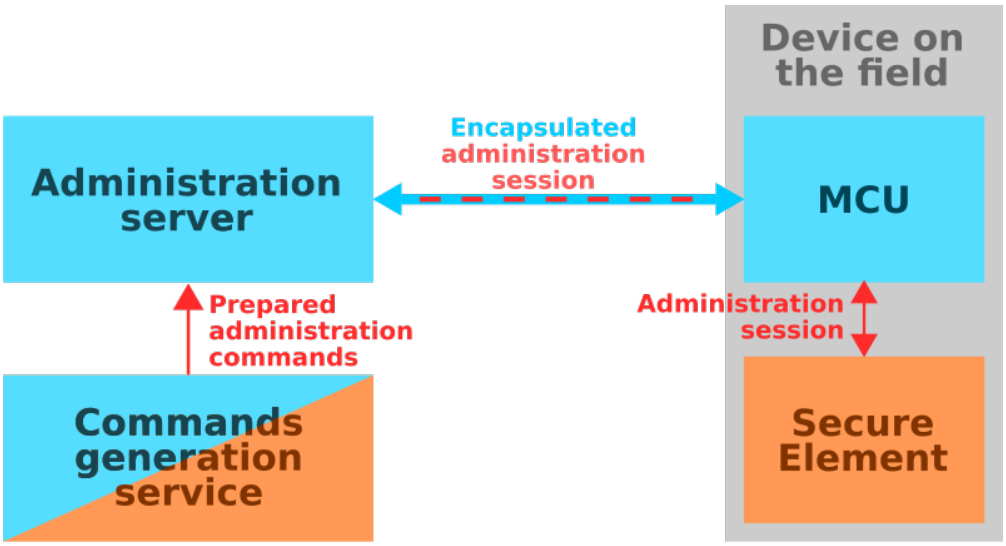
2. Introduction

Administration commands allow to securely update some Secure Elements settings on the field. These commands rely on an administration session designed to ensure the security of these operations.

The administration session is initiated from a remote server on a target Secure Element on the field.

The protocol detailed in this document is inspired by GlobalPlatform Card Technology - Secure Channel Protocol 03, see [1].

3. Overview



The orange parts are managed or defined by Trusted-Objects, the blue parts are managed by the customer.

The commands generation service is provided by a secret provider, whom can be the customer or Trusted-Objects, it aims to generate administration commands readable by any customers Secure Element, according to its administration needs.

The administration server is managed by the customer, its role is to manage devices and Secure Elements for administration tasks, by handling the administration session. It is responsible of the administration campaigns deployment.

The MCU embedded on the device receives administration requests encapsulated in a customer channel. It has to decapsulate the administration commands and forward them to the Secure Element. The MCU is not dealing with security concerns as data confidentiality and integrity, these aspects are handled by the administration server and the Secure Element.

The Secure Element checks the administration session integrity and applies the requested commands.

The administration session ensures the following security rules:

- **Data confidentiality, integrity and authenticity:** the administration commands applied by Secure Elements and their data can not be known or modified by an attacker, and the origin of the data is trusted.
- **Commands flow preservation:** for a given administration session, the administration commands flow order can not be altered, the Secure Element can only execute commands in the intended order.
- **Session completion:** the remote server is able to know if the session has been fully completed. If the session is complete, all the commands has been applied in the intended order and the Secure Element is updated as expected.

4. Involved keys

The following keys are involved in the protocol:

- KrootENC: root encryption key, used for key derivation - [handled by customer]
- KrootMAC: root MAC key, used for key derivation - [handled by customer]
- Kenc: used to generate session keys for AES encryption / decryption, this key is derived from KrootENC (see *Root keys derivation*) - [handled by customer]
- Kmac: used for session authentication, and to generate session keys for CMAC computation / verification, this key is derived from KrootMAC (see *Root keys derivation*) - [handled by customer]
- Senc: session key for encryption / decryption for administration session confidentiality
- Smac: session key for CMAC computation / verification for administration session integrity and authenticity
- Srmac: session key for CMAC computation / verification for responses integrity and authenticity
- Kdec: used to encrypt / decrypt pre-encrypted administration commands - [handled by a secret provider, whom can be the customer or Trusted-Objects]

4.1 Root keys derivation

Kenc and Kmac are computed as follows:

- Kenc = CMAC(KrootENC, Pad((SHA256(diversification_data)))
- Kmac = CMAC(KrootMAC, Pad((SHA256(diversification_data)))

The diversification data, which is the Secure Element serial number, is obtained from the Secure Element at the *Initialization* step of administration session.

The padding (Pad) is specified by PKCS#7, RFC 5652 section 6.3, see [2]. It is added to the trailing of the data, where 16-(L mod 16) bytes are appended having all the value 16-(L mod 16), L is the length of the data.

Examples of padding to be appended at the trailing of the data according to its size L:

- 01: if L mod 16 = 15
- 02 02: if L mod 16 = 14
-
- 0A 0A 0A 0A 0A 0A 0A 0A 0A 0A: if L mod 16 = 6
-
- 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10: if L mod 16 = 0

padding values above are expressed in HEX.

5. Administration session protocol

Below is detailed the administration session protocol to be used to run administration commands on the Secure Element.

5.1 Session context

The session context should contain the following fields:

- Senc and Smac: session encryption and CMAC keys
- CMAC chaing value (CMAC size), to be able to chain administration commands and ensure no one is missing in the flow
- EncCnt: encryption counter (2 bytes), used for Initial Chaining Vector (ICV) computation
- Administration server and Secure Element challenges, for the *Initialization* and *Authentication* steps.

5.2 Session steps

This section describes the different session steps:

- *Slot selection*: select an administration slot (each slot is personalized with dedicated keys and permissions
- *Initialization*: initiates an administration session, the Secure element is authenticated by the administration server
- *Authentication*: the administration server is authenticated by the Secure Element
- *Commands*: administration commands are sent from the administration server to the Secure Element
- *Finish*: closes the administration session

Below, the Secure Element requests and responses headers are removed, only the command code is given.

Warning: The Secure Element must not be powered-off from *Initialization* step to *Finish* step, as the session is stored in non-persistend memory. If it happens, the administration session has to be restarted from the begining.

5.2.1 Slot selection

This step selects the administration slot to use. It must be called each time we need to switch from a slot to another. A slot stays active even if session is reset. Changing active slot will destroy active session.

Secure Element command code: TODRV_HSE_CMD_ADMIN_SET_SLOT (0x0053). See also *TOSE_admin_set_slot()*.

Continue with *Initialization* step to initiate the Secure Element administration session.

5.2.2 Initialization

This step initiates an administration session between the administration server and the Secure Element. A server challenge is provided to the Secure Element, which returns its own challenge, its cryptogram, the diversification data and some protocol information. At the end of this step, the Secure Element is authenticated against the administration server.

Secure Element command code: TODRV_HSE_CMD_INIT_ADMIN_SESSION (0x0049). See also *TOSE_admin_session_init()*.

The following data is sent to the Secure Element:

Server challenge
8 bytes

where the server challenge is a random number.

The response, in case of success, is:

Diversification data	Protocol info.	SE challenge	SE cryptogram
8 bytes	4 bytes	8 bytes	8 bytes

At this step, the server is able to derivate session keys and to authenticate Secure Element, as explained below.

First of all, the administration server has to perform *Root keys derivation*, in order to get Kmac and Kenc.

Then, to get Smac, compute CMAC using Kmac on the following derivation data:

0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00
0x00 0x00 0x00 0x06 0x00 0x00 0x80 0x01
Server challenge (8 bytes)
Secure Element challenge (8 bytes)

the result is Smac.

To get Senc, compute CMAC using Kenc on the following derivation data:

0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00
0x00 0x00 0x00 0x04 0x00 0x00 0x80 0x01
Server challenge (8 bytes)
Secure Element challenge (8 bytes)

the result is Senc.

And, to get Srmac, compute CMAC using Kmac on the following derivation data:

0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00
0x00 0x00 0x00 0x07 0x00 0x00 0x80 0x01
Server challenge (8 bytes)
Secure Element challenge (8 bytes)

the result is Srmac.

The server is now able to verify the Secure Element cryptogram. To do this, compute CMAC using Smac on the following derivation data:

0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00
0x00 0x00 0x00 0x00 0x00 0x00 0x40 0x01
Server challenge (8 bytes)
Secure Element challenge (8 bytes)

If the result first 8 bytes are equal to the Secure Element cryptogram, it is now authenticated against the server.

Continue with *Authentication* step to authenticate server against the Secure Element.

5.2.3 Authentication

This step authenticates the server against the Secure Element, once completed the server is able to send administration commands to the Secure Element.

Secure Element command code: TODRV_HSE_CMD_AUTH_ADMIN_SESSION (0x004A). See also *TOSE_admin_session_auth_server()*.

The following data is sent to the Secure Element:

Options	Server cryptogram	CMAC
2 bytes	8 bytes	8 bytes

The options field must be set to 0.

The server cryptogram is the first 8 bytes in the result of a CMAC computation using Smac on the following derivation data:

0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00
0x00 0x00 0x00 0x01 0x00 0x00 0x40 0x01
Server challenge (8 bytes)
Secure Element challenge (8 bytes)

The command CMAC is computed on the options and the server cryptogram concatenated, using Smac. Only the 8 first bytes of the result are kept in the command CMAC.

On Secure Element success response, the server is authenticated against the Secure Element.

The following server session fields are to be set:

- EncCnt = 1
- CMAC chaining value set to 00 00 00

and the server can continue with *Commands* step.

5.2.4 Commands

The customer receives administration commands pre-encrypted by the secret provider, pre-encrypted commands format knowledge is not required for standard customer authentication server implementation, but more details are in *Pre-encrypted commands* section.

Note: If EncCnt loops to 0, the session will be reset by the Secure Element on an administration command attempt. As EncCnt is coded on 2 bytes, starting from 1, a maximum of 65535 administration commands can be executed per session. In such case, go to *Finish* step.

The initial chaining vector has to be computed by AES128 (**NOTE:** AES128, not AES128-CBC !) encrypting the following block using Senc:

0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00	EncCnt
14 bytes	2 bytes

after this, EncCnt has to be incremented by 1.

Then the pre-encrypted command has to be AES128-CBC encrypted using the initial chaining vector and Senc, the double-encrypted command is obtained: encrypted by the secret provider (Kdec) and by Senc.

The command CMAC is computed on the following data using Smac:

Session CMAC chaining value	Double-encrypted command
16 bytes	N-bytes

The session CMAC chaining value is replaced by the result of this computation.

Finally, the command can be sent to the Secure Element. Secure Element command code: TODRV_HSE_CMD_ADMIN_COMMAND (0x0014). See also `TOSE_admin_command()`.

The following data is sent to the Secure Element:

CMAC	Double-encrypted command
8 bytes	N bytes

only the first 8 bytes of the computed CMAC are kept in the request.

On success, we can suppose the command has been applied by the Secure Element. It is possible now to continue with other *Commands*, and when all the commands have been sent, to be sure the session has been fully completed, continue with *Finish* step.

Note: The administration session is reset on any command error (malformed request, wrong MAC,)

5.2.5 Commands with response

This particular kind of commands are built and sent like administration commands (see *Commands*), except that they have to be sent to the Secure Element using the following Secure Element command code: TODRV_HSE_CMD_ADMIN_COMMAND_WITH_RESPONSE (0x0051). See also *TOSE_admin_command_with_response2()*.

And, as a response is expected, it is needed to handle it as follows.

The response length depends on the administration command used. The response includes encrypted and padded data, and its CMAC, which have to be taken into account to request the right response size.

First of all, the response CMAC has to be verified. The response CMAC is computed on the following data using Srmac:

Session CMAC chaining value	Encrypted response
16 bytes	N-bytes

Compare the first 8 bytes of the returned CMAC with the last 8 bytes of the response.

The initial chaining vector has to be computed by AES128 encrypting the following block using Senc:

0x80 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00	EncCnt - 1
14 bytes	2 bytes

Note: EncCnt is used decremented by 1 here because it is supposed to have been incremented when building the administration command frame. The command and the response must use the same EncCnt value for ICV.

Then the response has to be AES128-CBC decrypted using the initial chaining vector and Senc, the clear response (with padding) is obtained. Check and remove the padding before using the response data (see *Data padding*).

5.2.6 Finish

This step closes the administration session and allows the server to verify the session completion.

Secure Element command code: TODRV_HSE_CMD_ADMIN_COMMAND (0x0014), no attached data. See also *TOSE_admin_session_fini()*.

On success, the response is:

Session CMAC
8 bytes

The server has to check this session CMAC value by computing a CMAC on the session CMAC chaining value, using Smac key. If the first computed 8 bytes match, the administration session on this device can be considered as complete by the server, with the guarantee that the Secure Element has executed all the commands in the intended order. In the other hand, if the CMAC does not match, the Secure Element

has to be considered as not correctly updated, left in an intermediate state, and a successful administration session has to be performed before continuing to work with the device in a trusted context.

At this step, the administration server should clear all the session data.

6. Library APIs

The following APIs are provided by libTO to help to send administration commands to the Secure Element.

TO_ret_t **TOSE_admin_session_init**(TOSE_ctx_t *ctx, const uint8_t server_challenge[TO_ADMIN_CHALLENGE_SIZE], uint8_t se_challenge[TO_ADMIN_CHALLENGE_SIZE], uint8_t se_cryptogram[TO_ADMIN_CRYPTOGRAM_SIZE], uint8_t diversification_data[TO_ADMIN_DIVERS_DATA_SIZE], uint8_t protocol_info[TO_ADMIN_PROTO_INFO_SIZE])

Initialize administration session.

This function initializes a new administration session between a server and the Secure Element.

Parameters

- **ctx** – **[in]** Pointer to the SE context
- **server_challenge** – **[in]** Server challenge, coming from the server
- **se_challenge** – **[out]** Returned Secure Element challenge
- **se_cryptogram** – **[out]** Returned Secure Element cryptogram
- **diversification_data** – **[out]** Returned diversification data
- **protocol_info** – **[out]** Returned protocol info

TO_ret_t **TOSE_admin_session_auth_server**(TOSE_ctx_t *ctx, const uint8_t options[TO_ADMIN_OPTIONS_SIZE], const uint8_t server_cryptogram[TO_ADMIN_CRYPTOGRAM_SIZE], const uint8_t mac[TO_ADMIN_MAC_SIZE])

Activate administration session by authenticating server.

This function allows the server to authenticate against the Secure Element, in order to activate authentication session.

Parameters

- **ctx** – **[in]** Pointer to the SE context
- **options** – **[in]** Administration session options
- **server_cryptogram** – **[in]** Server cryptogram, coming from the server
- **mac** – **[in]** MAC computed on options and server cryptogram

TO_ret_t **TOSE_admin_command**(TOSE_ctx_t *ctx, const uint8_t *command, uint16_t length)

Executes an authenticated administrative command.

Parameters

- **ctx** – **[in]** Pointer to the SE context
- **command** – **[in]** Buffer containing the MACed command to be executed. This buffer length is length + 8 bytes. It contains :
  - 8 bytes for the AES-Cmac (using the admin session KMac key)

  - N bytes of command, encrypted using SCP03 and KMAC (using ICV and AES-CBC). Once decrypted, those contain :
    * N-16 bytes of command, encrypted with KENC (using the following IV and AES-CBC)
    * 16 bytes of IV, used for the second layer of encryption
- **length** – [**in**] Expresses the length of the Command, excluding the 8 bytes of CMAC at the start

TO_ret_t **TOSE_admin_command_with_response**(TOSE_ctx_t *ctx, const uint8_t *command, uint16_t length, uint8_t *response, uint16_t response_length)

Administration command with response data.

Deprecated:

This function is deprecated and may disappear in future releases, use *TOSE_admin_command_with_response2()* instead.

Parameters

- **ctx** – [**in**] Pointer to the SE context
- **command** – [**in**] The command
- **length** – [**in**] The command length
- **response** – [**out**] Buffer to store response
- **response_length** – [**in**] Expected response length

TO_ret_t **TOSE_admin_command_with_response2**(TOSE_ctx_t *ctx, const uint8_t *command, uint16_t length, uint8_t *response, uint16_t *response_length)

Administration command with response data with variable length.

Parameters

- **ctx** – [**in**] Pointer to the SE context
- **command** – [**in**] The command
- **length** – [**in**] The command length
- **response** – [**out**] Buffer to store response
- **response_length** – [**inout**] Buffer length (input), response length (output)

TO_ret_t **TOSE_admin_session_fini**(TOSE_ctx_t *ctx, uint8_t mac[TO_ADMIN_MAC_SIZE])

Terminates administration session.

Parameters

- **ctx** – [**in**] Pointer to the SE context
- **mac** – [**out**] The session MAC returned by the Secure Element

TO_ret_t **TOSE_admin_set_slot**(TOSE_ctx_t *ctx, const uint8_t index)

Set administration slot to use from now on.

Parameters

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

7. Appendices

7.1 Data padding

Data block padding is appended according to NIST SP 800-38A, appendix A, see [3]:

- append a 0x80 byte to the data
- if needed, append 0x00 bytes until it is block size aligned

7.2 Pre-encrypted commands

The customer receives administration commands pre-encrypted by a secret provider. Secret provider can be the customer himself, or Trusted-Objects. Below is detailed how a pre-encrypted command is formed.

A raw command block is formatted like this:

Index	Sub-index	Data	CRC-16 CCITT
4 bytes	1 bytes	n bytes	2 bytes

The command CRC is a CRC-16 CCITT with following characteristics:

- **0xFFFF initial value**,
- the polynomial is **0x1021**,
- data and output are **reflected**.
- it is computed on the command index, sub-index, and data, and is here to prevent transfer errors,
- it is appended to the command data block in big endian form.

To double check your CRC implementation, you should verify that an input pattern 123456789 gives a CRC value of 0x6F91.

Padding has to be appended to command data block, see *Data padding*.

Padded data is AES128-CBC encrypted using a random initial vector and Kdec.

The customer receives a pre-encrypted command block formatted like this:

Command cryptogram	Initial vector

These commands can be decrypted by any customers Secure Element.

8. References

[**1**] GlobalPlatform Card Technology - Secure Channel Protocol 03 - Card Specification v2.2 Amendment D - Version 1.1.1 - GPC_SPE_014 (2014)

[**2**] PKCS#7 - RFC-5652 - Section 6.3 Content-encryption Process (2009)

[**3**] NIST Special Publication 800-38A - Recommendation for Block Cipher Modes of Operation - Appendix A (2001)

Index

T