ARC4/CRC16-CCITT Secure Link with Trusted Objects Secure Element

Release 6.1.2

March 16, 2022

Authored by *Trusted Objects*

Contents

Note: ARC4 with CRC16-CCITT secure link protocol must be enabled in your Secure Element release if you want to use this feature.

Note: The following is to be considered only if secure link is required by project needs.

Secure link is an optional Secure Element feature, which aims to encapsulate standard Secure Element commands to provide security mechanisms as encryption, authentication, or anti-replay, in order to have a secure link between your application and the Secure Element.

ARC4/CRC16-CCITT secure link provide weak security layer with low CPU and memory usage.

Warning: ARC4 is a weak security algorithm. Moreover, only encryption and anti-replay are ensured by this secure link, not messages integrity. It has the advantage to be simple enough to dramatically limit MCU ressources usage, but if your MCU can support stronger algorithms, consider using AES secure link protocol.

1. Legal

1.1 Disclaimer

2. How secure link works in libTO?

Secure link is made transparent by libTO.

Every command is constructed, provided to the libTO secure link engine to add security layer, and sent to Secure Element. When the response is received from Secure Element, it is unsecured by libTO secure link engine and then interpreted.

Secure link needs shared keys between client application and Secure Element, used to secure exchanged data.

A keys renewal mechanism is used to periodically renew secure link keys, internally handled by libTO. The keys storage is to be managed by client application. See `TODRV_HSE_seclink_store_keys_cb()` and `TODRV_HSE_seclink_load_keys_cb()` Secure Link APIs, and refer to *seclink.c* example from the library sources tree.

2.1 Configure libTO to use secure link with your MCU

Add *seclink_arc4.c* to your build process. Make sure to not build *seclink_none.c* file.

3. Secure link usage

Below are some aspects to consider in order to use secure link.

3.1 Key renewal process

Secure link protocol relies on keys, used to secure communications with Secure Element. These keys have to be managed by the client application code, and a keys renewal mechanism is used to periodically renew these keys.

The following callbacks have to be implemented in the application code:

`TODRV_HSE_seclink_store_keys_cb()`

`TODRV_HSE_seclink_load_keys_cb()`

allowing libTO to access the keys when needed, and Secure Element to request keys renewal procedure. The application is then responsible to manage the keys storage into a NVM.

The callbacks above are to be set just after TOSE_init() call, using the following functions:

`TODRV_HSE_seclink_set_store_keys_cb()`

`TODRV_HSE_seclink_set_load_keys_cb()`

See *seclink.c* example from the library sources tree.

The *Secure link API* chapter below gives more details about these APIs.

3.2 What to do on communication error?

If a communication error occurs, Secure Element and libTO secure link protocol internal states may be desynchronized.

The recommended behavior in this case is to use the following API to redo secure link initialization:

`TODRV_HSE_seclink_reset()`

4. Secure link API

This section details the libTO secure link API, to be used for secure link integration.

`TO_SECLINKAPI TO_lib_ret_t TODRV_HSE_seclink_reset (void)`

Reset secure link.

This function can be used to initialize secure link, after each successful TO_init() calls. If not called manually after TO_init(), it is automatically called on first command.

According to secure link protocol, this function may reset some internal state, request an initial vector from Secure Element, etc

Returns TO_OK on reset success, secure link is ready to be used.

`TO_SECLINKAPI void TODRV_HSE_seclink_set_store_keys_cb (TODRV_HSE_seclink_store_keys_cb cb)`

Set secure link keys storage callback.

This function is used to set secure link keys storage callback. The callback function will be used by the library to allow user to store new keys in remplacement of the old ones in cases of a secure link keys renewal procedure.

This function has to be called just after TO_init() if secure link is used by the project with a keys renewal mechanism enabled. In this case, do not use Secure Element APIs before having defined and set this callback, or you may miss keys storage notifications if a keys renewal procedure occurs.

Parameters

- **`cb`** – Callback function pointer, see TODRV_HSE_seclink_store_keys_cb.

`TO_SECLINKAPI void TODRV_HSE_seclink_set_load_keys_cb (TODRV_HSE_seclink_load_keys_cb cb)`

Set secure link callback to load keys.

This function is used to set secure link callback used by the library to load keys. The callback function will be called later by the library.

This function has to be called just after TO_init().

Parameters

- **`cb`** – Callback function pointer, see TODRV_HSE_seclink_load_keys_cb.

`TO_SECLINKAPI TO_lib_ret_t TODRV_HSE_seclink_request_renewed_keys (void)`

Get secure link renewed keys.

This function can only be used if you have the old keys. When using this function, it calls the configured secure link key renewal callback, allowing user to store the new key.

See TODRV_HSE_seclink_set_key_renewal_cb() and TODRV_HSE_seclink_keys_renewal_cb.

`TO_SECLINKAPI int TODRV_HSE_seclink_bypass (int bypass)`

Bypass Secure Element secure link and use clear text ones.

If called just after TO_init(), *TODRV_HSE_seclink_reset()* will not be called automatically. According to Secure Element settings, bypassing secure link may be impossible.

Parameters

- `bypass` – Set to 1 to bypass secure link, set to 0 to use secure commands.

Returns Previous secure link bypassing state.

5. Secure link protocol

This section describes how ARC4/CRC16-CCITT secure link is working. Everything is abstracted by libTO and the provided implementation *seclink_arc4.c*.

Note: ARC4 secure link uses CRC codes which are CRC16 CCITT reflected, on 2 bytes.

5.1 Get initial vector from Secure Element

The following command must be called after every Secure Element power on to allow ARC4 secure link encapsulation usage. It may be called again in case of communication error to resynchronize ARC4 internal states between MCU and the Secure Element. This command returns an initial vector.

Command details

Code	Value
CMD	0xFF01
Lc	0x0000
Res	0x00

Response details on success

Value	Description
Lr	0x0010
St	0x90 = SUCCESS
Res	0x00
IV0	Initial vector MSB
IV15	Initial vector LSB

Responses in case of error

In this case there is no response data.

St	Description
0xFE	Internal error

MCU-side initialization

As the Secure Element, the MCU needs to initialize its ARC4 algorithm internal state. The initial vector, and a secret shared key are needed. This phase is called ARC4 Key Scheduling Algorithm (KSA).

Initialization pseudo-code:

```
IVK = concat(IV, K)
for i from 0 to 255
        S[i] = i
endfor
j = 0
for i from 0 to 255
        j = (j + S[i] + IVK[i mod length(IVK)]) mod 256
        swap S[i] and S[j]
endfor
i = 0
j = 0
```

where:

- S is the internal secret state
- i and j are internal secret indexes
- K is the secret shared key
- IV is the initial vector

Finally, encrypt and drop 256 bytes of data (see TOCMD_SECLINK_ARC4 for encryption pseudo-code).

S, i and j must be kept into a secret state for following encryption and decryption operations.

5.2 Write secure link and read secure response

Encapsulate and encrypt a classic Secure Element command using RC4 cipher algorithm.

Command details

Field	Value
CMD	0xFF00
Lc	Encapsulated command size
Res	0x00
Encapsulated data	Secure link ARC4 encrypted data (see table below)

The encapsulated data in the table above is ARC4 encrypted, and formatted like this:

Field	Value
CMD	Encapsulated command code
Lc	Encapsulated command data size
Res	0x00
Data	Encapsulated command data
CRC	Encapsulated command CRC16

Response details on success

Field	Value
Lr	Encapsulated response size
ST	0x90 = SUCCESS
Res	0x00
Encapsulated response	Secure link ARC4 encrypted response (see table below)

The encapsulated response in the table above is ARC4 encrypted, and formatted like this:

Field	Value
Lr	Encapsulated response size
ST	Encapsulated response status
Res	0x00
Data	Encapsulated response data
CRC	Encapsulated response CRC16

Responses in case of error

In this case there is no response data (no encapsulated command response).

St	Description
0x85	Initial vector has not been requested yet
0xFD	Key has to be renewed (see *Key renewal*)

5.3 Key renewal

When a secure link ARC4 command fails with the key renewal error code, the following command is used to get the new key.

Code	Value
CMD	0xFF04
Lc	0x0000
Res	0x00

Response details on success

Value	Description
Lr	0x0022
St	0x90 = SUCCESS
Res	0x00
IV0	Initial vector MSB
IV15	Initial vector LSB
Key0	New key MSB
Key15	New key LSB
CRC	New key CRC16

Note: In the response details above, the new key and its CRC are encrypted using the old ARC4 key and the given initial vector.

Response usage:

1. Check length and status fields.
2. Initialise ARC4 using the old key and the provided initial vector (see the initialisation pseudo-code described earlier in this document).
3. Decrypt the new key and its CRC.
4. Check the new key CRC, if it is wrong, retry the command, maybe a communication error occured.
5. Save the new key into a NVM.
6. *Get initial vector from Secure Element* and use it to initialise ARC4, with the new key

Responses in case of error

In this case there is no response data.

St	Description
0x85	No new key available

5.4 MCU-side encryption / decryption

The same algorithm below is used to encrypt or decrypt data. It has to be used to encapsulate Secure Element command into the secure link, and to decapsulate the Secure Element response from secure response.

Pseudo-code:

```
while databyte to proceed:
        i = (i + 1) mod 256
        j = (j + S[i]) mod 256
        swap S[i] and S[j]
        x = S[(S[i] + S[j]) mod 256]
        output x XOR databyte
endwhile
```

where:

- S is the internal secret state
- i and j are internal secret indexes
- K is the secret shared key
- x is a keystream value

If the input is clear text, the output is encrypted, if the input is encrypted, the output is clear text.

6. Tips

6.1 Initialization

Secure link protocols may need to be initialized before use. It is automatically done on the first command write, or it can be enforced manually using the following API:

`TODRV_HSE_seclink_reset()`

6.2 Bypass secure link

With secure link enabled, if it is needed to bypass secure link and use clear text commands and responses, the following API can be used:

`TODRV_HSE_seclink_bypass()`

Note: It is not possible to bypass secure link if Secure Element settings enforce secure link usage.

Note: TODRV_HSE_trp_write() and TODRV_HSE_trp_read() are never using secure link, because these functions are designed to write and read data directly on I2C.

7. Write a secure link engine

If the secure link protocol reference implementation does not fit your needs, it is possible to write a custom implementation.

To do this, read the documentation of the *Secure link protocol*, and implement the API from *seclink.h* (present in libTO source tree) in a new C file. Then, just add this new file to your build process to use your custom implementation.