i.MX8 SHE API

Revision_1.1

Generated by Doxygen 1.8.15

1 SHE API

This document is a software referece description of the API provided by the i.MX8 SHE solutions.

2 Revision History

Revision	date	description
0.5	Mai 03 2019	first draf
1.0	June 28 2019	complete functions definition
1.1	December 20 2019	add she_cmd_load_key_ext API

3 General concepts related to the API

3.1 Session

The API must be initialized by a potential requestor by opening a session.
The session establishes a route (MU, DomainID...) between the requester and the SHE module, and grants the usage of a specified key store. When a session is opened, the SHE module returns a handle identifying the session to the requester.

4 Module Index

4.1 Modules

Here is a list of all modules:

5 Module Documentation

5.1 Error codes

Enumerations

- enum she_err_t {
  ERC_NO_ERROR = 0x0,
  ERC_SEQUENCE_ERROR = 0x1,
  ERC_KEY_NOT_AVAILABLE = 0x2,
  ERC_KEY_INVALID = 0x3,
  ERC_KEY_EMPTY = 0x4,
  ERC_NO_SECURE_BOOT = 0x5,
  ERC_KEY_WRITE_PROTECTED = 0x6,
  ERC_KEY_UPDATE_ERROR = 0x7,
  ERC_RNG_SEED = 0x8,
  ERC_NO_DEBUGGING = 0x9,
  ERC_BUSY = 0xA,
  ERC_MEMORY_FAILURE = 0xB,
  ERC_GENERAL_ERROR = 0xC }

5.1.1 Detailed Description

Error codes returned by SHE functions.

5.1.2 Enumeration Type Documentation

5.1.2.1 she_err_t

```
enum she_err_t
```

Enumerator

ERC_NO_ERROR	Success.
ERC_SEQUENCE_ERROR	Invalid sequence of commands.
ERC_KEY_NOT_AVAILABLE	Key is locked.
ERC_KEY_INVALID	Key not allowed for the given operation.
ERC_KEY_EMPTY	Key has not beed initialized yet.
ERC_NO_SECURE_BOOT	Conditions for a secure boot process are not met.
ERC_KEY_WRITE_PROTECTED	Memory slot for this key has been write-protected.
ERC_KEY_UPDATE_ERROR	Key update did not succeed due to errors in verification of the messages.
ERC_RNG_SEED	The seed has not been initialized.
ERC_NO_DEBUGGING	Internal debugging is not possible.
ERC_BUSY	A function of SHE is called while another function is still processing.
ERC_MEMORY_FAILURE	Memory error (e.g. flipped bits)
ERC_GENERAL_ERROR	Error not covered by other codes occured.

5.2 SHE keys

Macros

- #define **SHE_KEY_1** (0x04)
- #define **SHE_KEY_2** (0x05)
- #define **SHE_KEY_3** (0x06)
- #define **SHE_KEY_4** (0x07)
- #define **SHE_KEY_5** (0x08)
- #define **SHE_KEY_6** (0x09)
- #define **SHE_KEY_7** (0x0a)
- #define **SHE_KEY_8** (0x0b)
- #define **SHE_KEY_9** (0x0c)
- #define **SHE_KEY_10** (0x0d)
- #define **SHE_RAM_KEY** (0x0e)

5.2.1 Detailed Description

Identifiers for SHE keys.

5.3 SHE+ key extension

Macros

- #define SHE_KEY_DEFAULT (0x00)

  no key extension: keys from 0 to 10 as defined in SHE specification.
- #define SHE_KEY_N_EXT_1 (0x10)

  keys 11 to 20.
- #define SHE_KEY_N_EXT_2 (0x20)

  keys 21 to 30.
- #define SHE_KEY_N_EXT_3 (0x30)

  keys 31 to 40.
- #define SHE_KEY_N_EXT_4 (0x40)

  keys 41 to 50.

5.3.1 Detailed Description

Identifiers for the SHE key extension.

5.4 Key store provisioning

Macros

- #define SHE_STORAGE_CREATE_SUCCESS 0u

  New storage created succesfully.
- #define SHE_STORAGE_CREATE_WARNING 1u

  New storage created but its usage is restricted to a limited security state of the chip.
- #define SHE_STORAGE_CREATE_UNAUTHORIZED 2u

  Creation of the storage is not authorized.
- #define SHE_STORAGE_CREATE_FAIL 3u

  Creation of the storage failed for any other reason.
- #define SHE_STORAGE_NUMBER_UPDATES_DEFAULT 300u

  default number of maximum number of updated for SHE storage.

Functions

- uint32_t she_storage_create (uint32_t key_storage_identifier, uint32_t authentication_nonce, uint16_t max↩ _updates_number, uint8_t ∗signed_message, uint32_t msg_len)

5.4.1 Detailed Description

5.4.2 Function Documentation

5.4.2.1 she_storage_create()

```
uint32_t she_storage_create (
             uint32_t key_storage_identifier,
             uint32_t authentication_nonce,
             uint16_t max_updates_number,
             uint8_t * signed_message,
             uint32_t msg_len )
```

Creates an empty SHE storage.

Must be called at least once on every device before using any other SHE API.
A signed message must be provided to replace an existing key store. This message is not necessary under some conditions related to chip's lifecycle.

Parameters

key_storage_identifier	key store identifier
authentication_nonce	user defined nonce to be used as authentication proof for accesing the key store.
max_updates_number	maximum number of updates authorized on this new storage. This parameter has the goal to limit the occupation of the monotonic counter used as anti-rollback protection. If the maximum number of updates is reached, SHE still allows key store updates but without updating the monotonic counter giving the opportunity for rollback attacks. Always forced to 300 in the current release.
signed_message	pointer to a signed message authorizing the operation (NULL if no signed message to be used)
msg_len	length in bytes of the signed message

Returns

error code

5.5 Session

Functions

- struct she_hdl_s * she_open_session (uint32_t key_storage_identifier, uint32_t authentication_nonce, void(*async_cb)(void *priv, she_err_t err), void *priv)
- void she_close_session (struct she_hdl_s *hdl)

5.5.1 Detailed Description

5.5.2 Function Documentation

5.5.2.1 she_open_session()

```
struct she_hdl_s* she_open_session (
            uint32_t key_storage_identifier,
            uint32_t authentication_nonce,
            void(*)(void *priv, she_err_t err) async_cb,
            void * priv )
```

Initiate a SHE session. The returned session handle pointer is typed with the struct "she_hdl_s".
The user doesn't need to know or to access the fields of this struct.
It only needs to store this pointer and pass it to every calls to other APIs within the same SHE session.

Note that asynchronous API is currently not supported. async_cb and priv pointers must be set to NULL.

Parameters

key_storage_identifier	key store identifier
authentication_nonce	user defined nonce used as authentication proof for accesing the key store..
async_cb	user callback to be called on completion of a SHE operation
priv	user pointer to be passed to the callback

Returns

pointer to the session handle.

5.5.2.2 she_close_session()

```
void she_close_session (
            struct she_hdl_s * hdl )
```

Terminate a previously opened SHE session

Parameters

hdl	pointer to the session handler to be closed.

5.6 SHE commands

Modules

- CMD_GENERATE_MAC
- CMD_VERIFY_MAC
- CMD_ENC_CBC
- CMD_DEC_CBC
- CMD_ENC_ECB
- CMD_DEC_ECB
- CMD_LOAD_KEY
- CMD_LOAD_PLAIN_KEY
- CMD_EXPORT_RAM_KEY
- CMD_INIT_RNG
- CMD_EXTEND_SEED
- CMD_RND
- CMD_GET_STATUS
- CMD_GET_ID
- CMD_CANCEL
- last rating code

5.6.1 Detailed Description

5.7 CMD_GENERATE_MAC

Macros

- #define SHE_MAC_SIZE 16u

  size of the MAC generated is 128bits.

Functions

- she_err_t she_cmd_generate_mac (struct she_hdl_s *hdl, uint8_t key_ext, uint8_t key_id, uint16_t message_length, uint8_t *message, uint8_t *mac)

5.7.1 Detailed Description

5.7.2 Function Documentation

5.7.2.1 she_cmd_generate_mac()

```
she_err_t she_cmd_generate_mac (
            struct she_hdl_s * hdl,
            uint8_t key_ext,
            uint8_t key_id,
            uint16_t message_length,
            uint8_t * message,
            uint8_t * mac )
```

Generates a MAC of a given message with the help of a key identified by key_id.

Parameters

hdl	pointer to the SHE session handler
key_ext	identifier of the key extension to be used for the operation
key_id	identifier of the key to be used for the operation
message_length	lenght in bytes of the input message. The message is padded to be a multiple of 128 bits by SHE.
message	pointer to the message to be processed
mac	pointer to where the output MAC should be written (128bits should be allocated there)

Returns

error code

5.8 CMD_VERIFY_MAC

Macros

- #define SHE_MAC_VERIFICATION_SUCCESS 0u

  indication of mac verification success
- #define SHE_MAC_VERIFICATION_FAILED 1u

  indication of mac verification failure

Functions

- she_err_t she_cmd_verify_mac (struct she_hdl_s ∗hdl, uint8_t key_ext, uint8_t key_id, uint16_t message_↩ length, uint8_t ∗message, uint8_t ∗mac, uint8_t mac_length, uint8_t ∗verification_status)

5.8.1 Detailed Description

5.8.2 Function Documentation

5.8.2.1 she_cmd_verify_mac()

```
she_err_t she_cmd_verify_mac (
            struct she_hdl_s * hdl,
            uint8_t key_ext,
            uint8_t key_id,
            uint16_t message_length,
            uint8_t * message,
            uint8_t * mac,
            uint8_t mac_length,
            uint8_t * verification_status )
```

Verifies the MAC of a given message with the help of a key identified by key_id.

Parameters

hdl	pointer to the SHE session handler
key_ext	identifier of the key extension to be used for the operation
key_id	identifier of the key to be used for the operation
message_length	lenght in bytes of the input message. The message is padded to be a multiple of 128 bits by SHE.
message	pointer to the message to be processed
mac	pointer to the MAC to be compared (implicitely 128 bits)
mac_length	number of bytes to compare (must be at least 4)
verification_status	pointer to where write the result of the MAC comparison

Returns

error code

5.9 CMD_ENC_CBC

Macros

- #define SHE_AES_BLOCK_SIZE_128 16u

  size in bytes of a 128bits CBC block

Functions

- she_err_t she_cmd_enc_cbc (struct she_hdl_s ∗hdl, uint8_t key_ext, uint8_t key_id, uint32_t data_length, uint8_t ∗iv, uint8_t ∗plaintext, uint8_t ∗ciphertext)

5.9.1 Detailed Description

5.9.2 Function Documentation

5.9.2.1 she_cmd_enc_cbc()

```
she_err_t she_cmd_enc_cbc (
            struct she_hdl_s * hdl,
            uint8_t key_ext,
            uint8_t key_id,
            uint32_t data_length,
            uint8_t * iv,
            uint8_t * plaintext,
            uint8_t * ciphertext )
```

CBC encryption of a given plaintext with the key identified by key_id.

Parameters

hdl	pointer to the SHE session handler
key_ext	identifier of the key extension to be used for the operation
key_id	identifier of the key to be used for the operation
data_length	lenght in bytes of the plaintext and the cyphertext. Must be a multiple of 128bits.
iv	pointer to the 128bits IV to use for the encryption.
plaintext	pointer to the message to be encrypted.
ciphertext	pointer to ciphertext output area.

Returns

error code

5.10 CMD_DEC_CBC

Functions

- she_err_t she_cmd_dec_cbc (struct she_hdl_s ∗hdl, uint8_t key_ext, uint8_t key_id, uint32_t data_length, uint8_t ∗iv, uint8_t ∗ciphertext, uint8_t ∗plaintext)

5.10.1 Detailed Description

5.10.2 Function Documentation

5.10.2.1 she_cmd_dec_cbc()

```
she_err_t she_cmd_dec_cbc (
            struct she_hdl_s * hdl,
            uint8_t key_ext,
            uint8_t key_id,
            uint32_t data_length,
            uint8_t * iv,
            uint8_t * ciphertext,
            uint8_t * plaintext )
```

CBC decryption of a given ciphertext with the key identified by key_id.

Parameters

hdl	pointer to the SHE session handler
key_ext	identifier of the key extension to be used for the operation
key_id	identifier of the key to be used for the operation
data_length	lenght in bytes of the plaintext and the cyphertext. Must be a multiple of 128bits.
iv	pointer to the 128bits IV to use for the decryption.
ciphertext	pointer to ciphertext to be decrypted.
plaintext	pointer to the plaintext output area.

Returns

error code

5.11 CMD_ENC_ECB

Functions

- she_err_t she_cmd_enc_ecb (struct she_hdl_s *hdl, uint8_t key_ext, uint8_t key_id, uint8_t *plaintext, uint8_t *ciphertext)

5.11.1 Detailed Description

5.11.2 Function Documentation

5.11.2.1 she_cmd_enc_ecb()

```
she_err_t she_cmd_enc_ecb (
            struct she_hdl_s * hdl,
            uint8_t key_ext,
            uint8_t key_id,
            uint8_t * plaintext,
            uint8_t * ciphertext )
```

ECB encryption of a given plaintext with the key identified by key_id.

Parameters

hdl	pointer to the SHE session handler
key_ext	identifier of the key extension to be used for the operation
key_id	identifier of the key to be used for the operation
plaintext	pointer to the 128bits message to be encrypted.
ciphertext	pointer to ciphertext output area (128bits).

Returns

error code

5.12 CMD_DEC_ECB

Functions

- she_err_t she_cmd_dec_ecb (struct she_hdl_s ∗hdl, uint8_t key_ext, uint8_t key_id, uint8_t ∗ciphertext, uint8_t ∗plaintext)

5.12.1 Detailed Description

5.12.2 Function Documentation

5.12.2.1 she_cmd_dec_ecb()

```
she_err_t she_cmd_dec_ecb (
            struct she_hdl_s * hdl,
            uint8_t key_ext,
            uint8_t key_id,
            uint8_t * ciphertext,
            uint8_t * plaintext )
```

ECB decryption of a given ciphertext with the key identified by key_id.

Parameters

hdl	pointer to the SHE session handler
key_ext	identifier of the key extension to be used for the operation
key_id	identifier of the key to be used for the operation
ciphertext	pointer to 128bits ciphertext to be decrypted.
plaintext	pointer to the plaintext output area (128bits).

Returns

error code

5.13 CMD_LOAD_KEY

Macros

- #define SHE_LOAD_KEY_EXT_FLAGS_STRICT_OPERATION ((she_cmd_load_key_ext_flags_t)(1 << 7))

  The request is completed only when the key store is written in the NVM and the monotonic counter is incremented.
- #define SHE_KEY_SIZE 16u

  SHE keys are 128 bits (16 bytes) long.

Typedefs

- typedef uint8_t **she_cmd_load_key_ext_flags_t**

Functions

- she_err_t she_cmd_load_key (struct she_hdl_s *hdl, uint8_t key_ext, uint8_t key_id, uint8_t *m1, uint8_t *m2, uint8_t *m3, uint8_t *m4, uint8_t *m5)
- she_err_t she_cmd_load_key_ext (struct she_hdl_s *hdl, uint8_t key_ext, uint8_t key_id, uint8_t *m1, uint8_t *m2, uint8_t *m3, uint8_t *m4, uint8_t *m5, she_cmd_load_key_ext_flags_t flags)

5.13.1 Detailed Description

5.13.2 Function Documentation

5.13.2.1 she_cmd_load_key()

```
she_err_t she_cmd_load_key (
            struct she_hdl_s * hdl,
            uint8_t key_ext,
            uint8_t key_id,
            uint8_t * m1,
            uint8_t * m2,
            uint8_t * m3,
            uint8_t * m4,
            uint8_t * m5 )
```

Update an internal key of SHE with the protocol specified by SHE. The request is completed only when the new key has been written in the NVM. The monotonic counter is incremented for aach successful update.

Parameters

hdl	pointer to the SHE session handler
key_ext	identifier of the key extension to be used for the operation
key_id	identifier of the key to be used for the operation
m1	pointer to M1 message - 128 bits
m2	pointer to M2 message - 256 bits
m3	pointer to M3 message - 128 bits
m4	pointer to the output address for M4 message - 256 bits
m5	pointer to the output address for M5 message - 128 bits

Returns

error code

5.13.2.2 she_cmd_load_key_ext()

```
she_err_t she_cmd_load_key_ext (
            struct she_hdl_s * hdl,
            uint8_t key_ext,
            uint8_t key_id,
            uint8_t * m1,
            uint8_t * m2,
            uint8_t * m3,
            uint8_t * m4,
            uint8_t * m5,
            she_cmd_load_key_ext_flags_t flags )
```

This is an extension of the she_cmd_load_key API. The functionality of the she_cmd_load_key API is extended by adding a flag argument.

- STRICT OPERATION flag: User can use this flag to perform multiple updates before writing the key store into the NVM and incrementing the monotonic counter. The updates to the key store must be considered as effective only after an operation specifying the flag "STRICT OPERATION" is aknowledged by SHE.

Parameters

hdl	pointer to the SHE session handler
key_ext	identifier of the key extension to be used for the operation
key_id	identifier of the key to be used for the operation
m1	pointer to M1 message - 128 bits
m2	pointer to M2 message - 256 bits
m3	pointer to M3 message - 128 bits
m4	pointer to the output address for M4 message - 256 bits
m5	pointer to the output address for M5 message - 128 bits
flags	bitmap specifying the operation properties.

Returns

error code

5.14 CMD_LOAD_PLAIN_KEY

Functions

- she_err_t she_cmd_load_plain_key (struct she_hdl_s ∗hdl, uint8_t ∗key)

5.14.1 Detailed Description

5.14.2 Function Documentation

5.14.2.1 she_cmd_load_plain_key()

```
she_err_t she_cmd_load_plain_key (
            struct she_hdl_s * hdl,
            uint8_t * key )
```

Load a key as plaintext to the RAM_KEY slot without encryption and verification.

Parameters

hdl	pointer to the SHE session handler
key	pointer to the plaintext key to be loaded - 128bits

Returns

error code

5.15 CMD_EXPORT_RAM_KEY

Functions

- she_err_t she_cmd_export_ram_key (struct she_hdl_s ∗hdl, uint8_t ∗m1, uint8_t ∗m2, uint8_t ∗m3, uint8_t ∗m4, uint8_t ∗m5)

5.15.1 Detailed Description

5.15.2 Function Documentation

5.15.2.1 she_cmd_export_ram_key()

```
she_err_t she_cmd_export_ram_key (
            struct she_hdl_s * hdl,
            uint8_t * m1,
            uint8_t * m2,
            uint8_t * m3,
            uint8_t * m4,
            uint8_t * m5 )
```

exports the RAM_KEY into a format protected by SECRET_KEY.

Parameters

hdl	pointer to the SHE session handler
m1	pointer to the output address for M1 message - 128 bits
m2	pointer to the output address for M2 message - 256 bits
m3	pointer to the output address for M3 message - 128 bits
m4	pointer to the output address for M4 message - 256 bits
m5	pointer to the output address for M5 message - 128 bits

Returns

error code

5.16 CMD_INIT_RNG

Functions

- she_err_t she_cmd_init_rng (struct she_hdl_s *hdl)

5.16.1 Detailed Description

5.16.2 Function Documentation

5.16.2.1 she_cmd_init_rng()

```
she_err_t she_cmd_init_rng (
            struct she_hdl_s * hdl )
```

initializes the seed and derives a key for the PRNG. The function must be called before CMD_RND after every power cycle/reset.

Parameters

hdl	pointer to the SHE session handler

Returns

error code

5.17 CMD_EXTEND_SEED

Macros

- #define **SHE_ENTROPY_SIZE** 16u

Functions

- she_err_t she_cmd_extend_seed (struct she_hdl_s ∗hdl, uint8_t ∗entropy)

5.17.1 Detailed Description

5.17.2 Function Documentation

5.17.2.1 she_cmd_extend_seed()

```
she_err_t she_cmd_extend_seed (
            struct she_hdl_s * hdl,
            uint8_t * entropy )
```

extends the seed of the PRNG by compressing the former seed value and the supplied entropy into a new seed which will be used to generate the following random numbers. The random number generator has to be initialized by CMD_INIT_RNG before the seed can be extended.

Parameters

hdl	pointer to the SHE session handler
entropy	pointer to the entropy vector (128bits) to use for the operation

Returns

error code

5.18 CMD_RND

Macros

- #define **SHE_RND_SIZE** 16u

Functions

- she_err_t she_cmd_rnd (struct she_hdl_s ∗hdl, uint8_t ∗rnd)

5.18.1 Detailed Description

5.18.2 Function Documentation

5.18.2.1 she_cmd_rnd()

```
she_err_t she_cmd_rnd (
            struct she_hdl_s * hdl,
            uint8_t * rnd )
```

returns a vector of 128 random bits. The random number generator has to be initialized by CMD_INIT_RNG before random numbers can be supplied.

Parameters

hdl	pointer to the SHE session handler
rnd	pointer to the output address for the generated 128bits random vector

Returns

error code

5.19 CMD_GET_STATUS

Functions

- she_err_t she_cmd_get_status (struct she_hdl_s *hdl, uint8_t *sreg)

5.19.1 Detailed Description

5.19.2 Function Documentation

5.19.2.1 she_cmd_get_status()

```
she_err_t she_cmd_get_status (
            struct she_hdl_s * hdl,
            uint8_t * sreg )
```

returns the content of the status register

Parameters

hdl	pointer to the SHE session handler
sreg	pointer to the output address for status register(8bits)

Returns

error code

5.20 CMD_GET_ID

Macros

- #define **SHE_CHALLENGE_SIZE** 16u /∗ 128 bits ∗/
- #define **SHE_ID_SIZE** 15u /∗ 120 bits ∗/

Functions

- she_err_t she_cmd_get_id (struct she_hdl_s ∗hdl, uint8_t ∗challenge, uint8_t ∗id, uint8_t ∗sreg, uint8_t ∗mac)

5.20.1 Detailed Description

5.20.2 Function Documentation

5.20.2.1 she_cmd_get_id()

```
she_err_t she_cmd_get_id (
            struct she_hdl_s * hdl,
            uint8_t * challenge,
            uint8_t * id,
            uint8_t * sreg,
            uint8_t * mac )
```

returns the identity (UID) and the value of the status register protected by a MAC over a challenge and the data.

Parameters

hdl	pointer to the SHE session handler
challenge	pointer to the challenge vector (128bits)
id	pointer to the output address for the identity (120bits)
sreg	pointer to the output address for status register(8bits)
mac	pointer to the output address for the computed MAC (128bits)

Returns

error code

5.21 CMD_CANCEL

Functions

- she_err_t she_cmd_cancel (struct she_hdl_s *hdl)

5.21.1 Detailed Description

5.21.2 Function Documentation

5.21.2.1 she_cmd_cancel()

```
she_err_t she_cmd_cancel (
            struct she_hdl_s * hdl )
```

interrupt any given function and discard all calculations and results.

Parameters

hdl	pointer to the SHE session handler

Returns

error code

5.22 last rating code

Functions

- uint32_t she_get_last_rating_code (struct she_hdl_s ∗hdl)

5.22.1 Detailed Description

5.22.2 Function Documentation

5.22.2.1 she_get_last_rating_code()

```
uint32_t she_get_last_rating_code (
            struct she_hdl_s * hdl )
```

Report rating code from last command

SHE API defines standard errors that should be returned by API calls. Error code reported by SECO are "translated" to these SHE error codes. This API allow user to get the error code reported by SECO for the last command before its translation to SHE error codes. This shoudl be used for debug purpose only.

Parameters

hdl	pointer to the SHE session handler

Returns

rating code reported by last command

5.23 Get info

Functions

- she_err_t she_get_info (struct she_hdl_s *hdl, uint32_t *user_sab_id, uint8_t *chip_unique_id, uint16_t *chip_monotonic_counter, uint16_t *chip_life_cycle, uint32_t *she_version)

5.23.1 Detailed Description

Get miscellaneous information. This function return, among others, all the information needed to build a valid signed message.

5.23.2 Function Documentation

5.23.2.1 she_get_info()

```
she_err_t she_get_info (
            struct she_hdl_s * hdl,
            uint32_t * user_sab_id,
            uint8_t * chip_unique_id,
            uint16_t * chip_monotonic_counter,
            uint16_t * chip_life_cycle,
            uint32_t * she_version )
```

Parameters

hdl	pointer to the SHE session handler
user_sab_id	pointer to the output address for the user identity (32bits)
chip_unique_id	pointer to the output address for the chip unique identifier (64bits)
chip_monotonic_counter	pointer to the output address for the chip monotonic counter value (16bits)
chip_life_cycle	pointer to the output address for the chip current life cycle (16bits)
she_version	pointer to the output address for the SHE module version (32bits)

Returns

error code

5.24 Abstraction layer

Data Structures

- struct seco_mu_params

Macros

- #define **MU_CHANNEL_UNDEF** (0x00u)
- #define **MU_CHANNEL_SHE** (0x01u)
- #define **MU_CHANNEL_SHE_NVM** (0x02u)
- #define **MU_CHANNEL_HSM** (0x03u)
- #define **MU_CHANNEL_HSM_NVM** (0x04u)
- #define **DATA_BUF_IS_INPUT** 0x01u
- #define **DATA_BUF_USE_SEC_MEM** 0x02u
- #define **DATA_BUF_SHORT_ADDR** 0x04u
- #define **SEC_MEM_SHORT_ADDR_MASK** 0xFFFFu

Functions

- struct seco_os_abs_hdl * **seco_os_abs_open_mu_channel** (uint32_t type, struct seco_mu_params *mu↩_params)
- void seco_os_abs_close_session (struct seco_os_abs_hdl *phdl)
- int32_t seco_os_abs_send_mu_message (struct seco_os_abs_hdl *phdl, uint32_t *message, uint32_t size)
- int32_t seco_os_abs_read_mu_message (struct seco_os_abs_hdl *phdl, uint32_t *message, uint32_t size)
- int32_t seco_os_abs_configure_shared_buf (struct seco_os_abs_hdl *phdl, uint32_t shared_buf_off, uint32_t size)
- uint64_t seco_os_abs_data_buf (struct seco_os_abs_hdl *phdl, uint8_t *src, uint32_t size, uint32_t flags)
- uint32_t seco_os_abs_crc (uint8_t *data, uint32_t size)
- void seco_os_abs_memset (uint8_t *dst, uint8_t val, uint32_t len)
- void seco_os_abs_memcpy (uint8_t *dst, uint8_t *src, uint32_t len)
- uint8_t * seco_os_abs_malloc (uint32_t size)
- void seco_os_abs_free (void *ptr)
- int32_t seco_os_abs_storage_write (struct seco_os_abs_hdl *phdl, uint8_t *src, uint32_t size)
- int32_t seco_os_abs_storage_read (struct seco_os_abs_hdl *phdl, uint8_t *dst, uint32_t size)
- int32_t seco_os_abs_storage_write_chunk (struct seco_os_abs_hdl *phdl, uint8_t *src, uint32_t size, uint64_t blob_id)
- int32_t seco_os_abs_storage_read_chunk (struct seco_os_abs_hdl *phdl, uint8_t *dst, uint32_t size, uint64_t blob_id)
- void seco_os_abs_start_system_rng (struct seco_os_abs_hdl *phdl)
- int32_t seco_os_abs_send_signed_message (struct seco_os_abs_hdl *phdl, uint8_t *signed_message, uint32_t msg_len)

5.24.1 Detailed Description

seco_libs code itself is independent from any OS or platform. This abstraction layer defines the functions that should be implemented on a specific OS or platform when porting seco_libs on it.

5.24.2 Data Structure Documentation

5.24.2.1 struct seco_mu_params

Opens a MU channel Purpose of this function is to setup a communication channel using a messaging unit (MU) between the caller of this function and SECO. This session is uniquely identified by a platform handle. The pointer to this handle is returned here and has to be provided by the user on each call to platform dependent functions. The details of the seco_os_abs_hdl structure are not described in this API so it can be custmized for each OS/platform porting.

One physical MU can be shared between several sessions but concurrent access should be protected. Seco can only process one command at a time. Once a command has been sent to SECO no other command should be sent on the physical MU (even by other sessions sharing same physical MU) until the response has been received. (but a response to a command from Seco should not be blocked - e.g. in storage case)

The mapping between physical MUs and sessions is considered as platform dependent. So no assumption are made about this by the caller.

Parameters

type	constant indicating for which purpose will be used this channel (see defines below)
mu_params	pointer where the MU params for this channel should be written. It is up to this abstraction layer to provide a mapping between channel types and associated MU params. Caller need to know these information to fill some SECO messages.

Returns

pointer to the MU channel handle.

Data Fields

uint8_t	mu_id	index of the MU as per SECO point of view.
uint8_t	interrupt_idx	Interrupt number of the MU used to indicate data availability.
uint8_t	tz	indicate if current partition has TZ enabled.
uint8_t	did	DID of the calling partition.
uint8_t	priority	SECO priority associated to this MU.
uint8_t	operating_mode	SECO operating mode associated to this MU.

5.24.3 Function Documentation

5.24.3.1 seco_os_abs_close_session()

```
void seco_os_abs_close_session (
            struct seco_os_abs_hdl * phdl )
```

Close a previously opened session.

Parameters

phdl	pointer to the session handle to be closed.

5.24.3.2 seco_os_abs_send_mu_message()

```
int32_t seco_os_abs_send_mu_message (
            struct seco_os_abs_hdl * phdl,
            uint32_t * message,
            uint32_t size )
```

Send a message to Seco over a messaging unit.

A message is made of 1 or more 32bits words. The 1st word is an header containing the length of the message and its type (command or response) The following protocol has to be respected when sending a message to SECO:

- write the header in TR0 of the MU
- send and interrupt to SECO thanks to GI0 bit in CR register of the MU
- Write other words of the message. word at index n in the message has to be written in TR[n%4] register of the MU. Write to a TRx register has to be performed only when the corresponding TEx bit in SR register is equal to 1 (indicating that the TRx register is empty).

Note that a physical MU can be shared between several sessions. Concurent access to the physical MU should be prevented (SECO process commands one by one). So this API should block until the physical MU is available for this session.

Parameters

phdl	pointer to handle identifying the session to be used to carry the message.
message	pointer to the message itself. It has to be aligned on 32bits.
size	size in bytes of the message. It has to be multiple of 4 bytes.

Returns

length in bytes written to the MU or negative value in case of error

5.24.3.3 seco_os_abs_read_mu_message()

```
int32_t seco_os_abs_read_mu_message (
            struct seco_os_abs_hdl * phdl,
            uint32_t * message,
            uint32_t size )
```

Read a message from Seco over a messaging unit.

This API is blocking until a message is sent by SECO

A message is made of 1 or more 32bits words. The 1st word is an header containing the length of the message and its type (command or response)

When receiving MU's GI0 interrupt from SECO this API should:

- read the header in RR0 of the MU and extract its length
- send and interrupt to SECO thanks to GI0 bit in CR register of the MU
- Write other words of the message. word at index n in the message has to be written in TR[n%4] register of the MU. Write to a TRx register has to be performed only when the corresponding TEx bit in SR register is equal to 1 (indicating that the TRx register is empty).

Typically for SHE this API will be called right after having sent a command to SECO in order to get the response. And for Storage an infinite loop will wait on this blocking API for a command from SECO.

Parameters

phdl	pointer to handle identifying the session to be used to carry the message.
message	pointer to the message itself. It has to be aligned on 32bits.
size	size in bytes of the message. It has to be multiple of 4 bytes.

Returns

length in bytes read from the MU or negative value in case of error

5.24.3.4 seco_os_abs_configure_shared_buf()

```
int32_t seco_os_abs_configure_shared_buf (
             struct seco_os_abs_hdl * phdl,
             uint32_t shared_buf_off,
             uint32_t size )
```

Configure the use of shared buffer in secure memory

Seco allocates a shared buffer in secure memory to exchange data. Offset in secure memory and size are provided by Seco in a message decoded by caller. These information are provided to the platform dependent layer through this API to avoid parsing incoming messages here.

Parameters

phdl	pointer to the session handle associated to this shared buffer.
shared_buf_offset	offset of the shared buffer in secure memory.
size	size in bytes of the allocated shared buffer.

Returns

0 in case of success. Any other value means error.

5.24.3.5 seco_os_abs_data_buf()

```
uint64_t seco_os_abs_data_buf (
             struct seco_os_abs_hdl * phdl,
```

```
            uint8_t * src,
            uint32_t size,
            uint32_t flags )
```

Setup data buffer for command processing

The command messages sent to Seco most of the time do not carry the data to be processed. It uses pointers instead. This API is used to make sure the data are available to Seco when it receive a command and also that the result can be accessed by the caller after seco sent the response by either:

- copy to/from dedicated shared buffer in secure memory
- or perform appropriate cache management when using buffers in DDR It also provides the address to be inserted into the message to be sent to Seco: either the physical address or the offset in the shared buffer (see options below) yo be used by Seco.

Several options are available to describe the buffers. DATA_BUF_IS_INPUT: the buffer described here is input to the next command. Otherwise this is an output. DATA_BUF_USE_SEC_MEM: the data should be copied to the shared buffer in secure memory or the output will be stored by Seco in it. It is expected that the buffers will be allocated in a contiguous manner in this shared memory since some optimizations depend on this (fast MAC). If not set any other memory can be used (e.g. DDR). In this case this API should take care of cache coherency and access rights before Seco tries access the physical memory. when using secure mem. DATA_BUF_SHORT_A↩ DDR: (only possible when using secure memory) returns the offset in secure memory (16bits) instead of full 64bits address (used to reduce the size of some Seco messages).

Once this API has been called the buffers should no more be accessed by the caller until the command has been sent to Seco and its response has been received.

Parameters

phdl	pointer to the session handle for which this data buffer is used.
src	pointer to the data if input or to the area where the output should be written.
size	size in bytes of the input data or max size of the output.
flags	data buffer options as described above. Interpreted as a bit-field.

Returns

the address to be inserted in the message to Seco to indicate him this buffer.

5.24.3.6 seco_os_abs_crc()

```
uint32_t seco_os_abs_crc (
            uint8_t * data,
            uint32_t size )
```

Compute the CRC of a buffer.

Used for basic check of integrity on the storage to avoid sending a corrupted blob to Seco. No strong security requirement here since Seco will perform more robust integrity check on the blob.

CRC computation is abstracted here in order to let the possibility to use some platform optimized library instead of re-implementing.

Parameters

data	pointer to the data on which the CRC must be computed. \size size in bytes of the data

Returns

32bits value of the CRC.

5.24.3.7 seco_os_abs_memset()

```
void seco_os_abs_memset (
            uint8_t * dst,
            uint8_t val,
            uint32_t len )
```

Force all bytes of a buffer to a given value.

Parameters

dst	address of the buffer to be overwriten
val	value to be written in every bytes of the buffer
len	number of bytes to be written

5.24.3.8 seco_os_abs_memcpy()

```
void seco_os_abs_memcpy (
            uint8_t * dst,
            uint8_t * src,
            uint32_t len )
```

Copy the content of a buffer to another location.

Parameters

dst	pointer to the destination buffer where data should be copied
src	pointer to the source buffer from where data should be copied
len	number of bytes to be copied

5.24.3.9 seco_os_abs_malloc()

```
uint8_t* seco_os_abs_malloc (
            uint32_t size )
```

Dynamically allocate memory.

Parameters

size	number of bytes to be allocated

Returns

pointer to the allocated buffer or NULL in case of error.

5.24.3.10 seco_os_abs_free()

```
void seco_os_abs_free (
            void * ptr )
```

Free a previously allocated buffer.

Parameters

ptr	pointer to the buffer to free

5.24.3.11 seco_os_abs_storage_write()

```
int32_t seco_os_abs_storage_write (
            struct seco_os_abs_hdl * phdl,
            uint8_t * src,
            uint32_t size )
```

Write data to the non volatile storage.

Parameters

phdl	pointer to the session handle for which this data buffer is used.
src	pointer to the data to be written to storage.
size	number of bytes to be written.

Returns

number of bytes written.

5.24.3.12 seco_os_abs_storage_read()

```
int32_t seco_os_abs_storage_read (
            struct seco_os_abs_hdl * phdl,
            uint8_t * dst,
            uint32_t size )
```

Read data from the non volatile storage.

Parameters

phdl	pointer to the session handle for which this data buffer is used.
dst	pointer to the data where data read from the storage should be copied.
size	number of bytes to be read.

Returns

number of bytes read.

5.24.3.13 seco_os_abs_storage_write_chunk()

```
int32_t seco_os_abs_storage_write_chunk (
            struct seco_os_abs_hdl * phdl,
            uint8_t * src,
            uint32_t size,
            uint64_t blob_id )
```

Write a subset of data to the non volatile storage.

In case of large storage SECO will split it in "chunks" that will be encrypted in blobs. A unique identifier within the system allow the storage manager to store and read it without ambiguity.

Parameters

phdl	pointer to the session handle for which this data buffer is used.
src	pointer to the data to be written to storage.
size	number of bytes to be written. \patam blob_id unique identifier of the blob corresponding to the storage chunk to be written

Returns

number of bytes written.

5.24.3.14 seco_os_abs_storage_read_chunk()

```
int32_t seco_os_abs_storage_read_chunk (
            struct seco_os_abs_hdl * phdl,
            uint8_t * dst,
            uint32_t size,
            uint64_t blob_id )
```

Read a subset of data from the non volatile storage.

In case of large storage SECO will split it in "chunks" that will be encrypted in blobs. A unique identifier within the system allow the storage manager to store and read it without ambiguity.

Parameters

phdl	pointer to the session handle for which this data buffer is used.
dst	pointer to the data where data read from the storage should be copied.
size	number of bytes to be read. \patam blob_id unique identifier of the blob corresponding to the storage chunk to be read

Returns

number of bytes read.

5.24.3.15 seco_os_abs_start_system_rng()

```
void seco_os_abs_start_system_rng (
            struct seco_os_abs_hdl * phdl )
```

Start the RNG from a system point of view.

Start the RNG HW through a SCU RPC: sc_seco_start_rng()

Parameters

phdl	pointer to the session handle for which this data buffer is used.

5.24.3.16 seco_os_abs_send_signed_message()

```
int32_t seco_os_abs_send_signed_message (
            struct seco_os_abs_hdl * phdl,
            uint8_t * signed_message,
            uint32_t msg_len )
```

Send a signed message to SECO through dedicated SCU RPC

Purpose is to unlock the creation of a storage in some specific cases.

Parameters

signed_message	pointer to the signed message.
msg_len	length of the signed message

Index