# i.MX8 HSM API

Revision_0.1

Generated by Doxygen 1.8.15

# 1 Main Page

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

# 2 Revision History

Revision 0.1: 29/03/2019 Savari preliminary draft - subject to change Revision 0.8: 20/05/2019 Adding butterfly key expansion operation; adding signature, rng, hash services.

# 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 HSM, and grants the usage of a specified key store through a password authentication.
When a session is opened, the HSM returns a handle identifying the session to the requester.

## 3.2 Service flow

For a given category of services, the requestor is expected to open a service flow by invoking the appropriate HSM API. The session handle, as well as the control data needed for the service flow are provided as parameters of the call. Upon reception of the open request, the HSM allocates a context in which the session handle, as well as the provided control parameters are stored. The context is preserved until the service flow is closed by the user and it is used by the HSM to proceed with the sub-sequent operations requested by the user on the service flow.

# 4 Module Index

## 4.1 Modules

Here is a list of all modules:



# 5 Module Documentation

## 5.1 Hsm_api

i.MX8 HSM API header file

### Macros

- #define HSM_SVC_KEY_STORE_FLAGS_CREATE ((hsm_svc_key_store_flags_t)(1 << 0))
- #define HSM_SVC_KEY_STORE_FLAGS_UPDATE ((hsm_svc_key_store_flags_t)(1 << 1))
- #define HSM_SVC_KEY_STORE_FLAGS_DELETE ((hsm_svc_key_store_flags_t)(1 << 3))
- #define HSM_KEY_TYPE_ECDSA_NIST_P224 ((hsm_key_type_t)0x01)
- #define HSM_KEY_TYPE_ECDSA_NIST_P256 ((hsm_key_type_t)0x02)
- #define HSM_KEY_TYPE_ECDSA_NIST_P384 ((hsm_key_type_t)0x03)
- #define HSM_KEY_TYPE_ECDSA_BRAINPOOL_R1_224 ((hsm_key_type_t)0x12)
- #define HSM_KEY_TYPE_ECDSA_BRAINPOOL_R1_256 ((hsm_key_type_t)0x13)
- #define HSM_KEY_TYPE_ECDSA_BRAINPOOL_R1_384 ((hsm_key_type_t)0x15)
- #define HSM_KEY_TYPE_ECDSA_BRAINPOOL_T1_224 ((hsm_key_type_t)0x22)
- #define HSM_KEY_TYPE_ECDSA_BRAINPOOL_T1_256 ((hsm_key_type_t)0x23)
- #define HSM_KEY_TYPE_ECDSA_BRAINPOOL_T1_384 ((hsm_key_type_t)0x25)
- #define HSM_KEY_TYPE_AES_128 ((hsm_key_type_t)0x30)
- #define HSM_KEY_TYPE_AES_192 ((hsm_key_type_t)0x31)
- #define HSM_KEY_TYPE_AES_256 ((hsm_key_type_t)0x32)
- #define HSM_KEY_INFO_PERMANENT ((hsm_key_info_t)(1 << 0))
- #define HSM_OP_KEY_GENERATION_FLAGS_UPDATE ((hsm_op_key_gen_flags_t)(1 << 0))
- #define HSM_OP_KEY_GENERATION_FLAGS_CREATE_PERSISTENT ((hsm_op_key_gen_flags_t)(1 << 1))
- #define HSM_OP_KEY_GENERATION_FLAGS_CREATE_TRANSIENT ((hsm_op_key_gen_flags_t)(1 << 2))
- #define HSM_OP_KEY_GENERATION_FLAGS_STRICT_OPERATION ((hsm_op_key_gen_flags_t)(1 << 7))
- #define HSM_OP_MANGE_KEY_FLAGS_UPDATE ((hsm_op_manage_key_flags_t)(1 << 0))
- #define HSM_OP_MANGE_KEY_FLAGS_DELETE ((hsm_op_manage_key_flags_t)(1 << 1))
- #define HSM_OP_MANGE_KEY_FLAGS_STRICT_OPERATION ((hsm_op_manage_key_flags_t)(1 << 7))
- #define HSM_CIPHER_ONE_GO_ALGO_AES_ECB ((hsm_op_cipher_one_go_algo_t)(0x00))
- #define HSM_CIPHER_ONE_GO_ALGO_AES_CBC ((hsm_op_cipher_one_go_algo_t)(0x01))
- #define HSM_CIPHER_ONE_GO_ALGO_AES_CCM ((hsm_op_cipher_one_go_algo_t)(0x02))
- #define HSM_CIPHER_ONE_GO_FLAGS_ENCRYPT ((hsm_op_cipher_one_go_flags_t)(1 << 0))
- #define HSM_CIPHER_ONE_GO_FLAGS_DECRYPT ((hsm_op_cipher_one_go_flags_t)(1 << 1))
- #define HSM_OP_SIGNATURE_GENERATION_INPUT_DIGEST ((hsm_op_signature_gen_flags_t)(0 << 0))

- #define HSM_OP_SIGNATURE_GENERATION_INPUT_MESSAGE ((hsm_op_signature_gen_flags_t)(1 << 1))
- #define HSM_OP_SIGNATURE_GENERATION_COMPRESSED_POINT ((hsm_op_signature_gen_flags_t)(2 << 1))
- #define HSM_SIGNATURE_SCHEME_ECDSA_NIST_P224_SHA_256 ((hsm_signature_scheme_id_t)0x01)
- #define HSM_SIGNATURE_SCHEME_ECDSA_NIST_P256_SHA_256 ((hsm_signature_scheme_id_t)0x02)
- #define HSM_SIGNATURE_SCHEME_ECDSA_NIST_P384_SHA_384 ((hsm_signature_scheme_id_t)0x03)
- #define HSM_SIGNATURE_SCHEME_ECDSA_BRAINPOOL_R1_224_SHA_256 ((hsm_signature_scheme_id_t)0x12)
- #define HSM_SIGNATURE_SCHEME_ECDSA_BRAINPOOL_R1_256_SHA_256 ((hsm_signature_scheme_id_t)0x13)
- #define HSM_SIGNATURE_SCHEME_ECDSA_BRAINPOOL_R1_384_SHA_384 ((hsm_signature_scheme_id_t)0x15)
- #define HSM_SIGNATURE_SCHEME_ECDSA_BRAINPOOL_T1_224_SHA_256 ((hsm_signature_scheme_id_t)0x22)
- #define HSM_SIGNATURE_SCHEME_ECDSA_BRAINPOOL_T1_256_SHA_256 ((hsm_signature_scheme_id_t)0x23)
- #define HSM_SIGNATURE_SCHEME_ECDSA_BRAINPOOL_T1_384_SHA_384 ((hsm_signature_scheme_id_t)0x25)
- #define HSM_OP_SIGNATURE_VERIFICATION_INPUT_DIGEST ((hsm_op_signature_ver_flags_t)(0 << 0))
- #define HSM_OP_SIGNATURE_VERIFICATION_INPUT_MESSAGE ((hsm_op_signature_ver_flags_t)(1 << 1))
- #define HSM_VERIFICATION_STATUS_SUCCESS ((hsm_verification_status_t)(0x5A3CC3A5))
- #define HSM_VERIFICATION_STATUS_FAILURE ((hsm_verification_status_t)(0xA5C33C5A))
- #define HSM_OP_FAST_SIGNATURE_GENERATION_INPUT_DIGEST ((hsm_op_fast_signature_gen_flags_t)(0 << 0))
- #define HSM_OP_FAST_SIGNATURE_GENERATION_INPUT_MESSAGE ((hsm_op_fast_signature_gen_flags_t)(1 << 1))
- #define HSM_OP_FAST_SIGNATURE_GENERATION_COMPRESSED_POINT ((hsm_op_fast_signature_gen_flags_t)(2 << 1))
- #define HSM_OP_FAST_SIGNATURE_VERIFICATION_INPUT_DIGEST ((hsm_op_fast_signature_ver_flags_t)(0 << 0))
- #define HSM_OP_FAST_SIGNATURE_VERIFICATION_INPUT_MESSAGE ((hsm_op_fast_signature_ver_flags_t)(1 << 1))
- #define HSM_HASH_ALGO_SHA2_224 ((hsm_hash_algo_t)(0x0))
- #define HSM_HASH_ALGO_SHA2_256 ((hsm_hash_algo_t)(0x1))
- #define HSM_HASH_ALGO_SHA2_384 ((hsm_hash_algo_t)(0x2))

### Typedefs

- typedef uint8_t hsm_svc_key_store_flags_t
- typedef uint8_t hsm_svc_key_management_flags_t
- typedef uint8_t hsm_svc_cipher_flags_t
- typedef uint8_t hsm_svc_signature_flags_t
- typedef uint8_t hsm_svc_fast_signature_verification_flags_t
- typedef uint8_t hsm_svc_fast_signature_generation_flags_t
- typedef uint8_t hsm_svc_rng_flags_t
- typedef uint8_t hsm_svc_hash_flags_t
- typedef uint8_t hsm_op_key_gen_flags_t
- typedef uint8_t hsm_op_manage_key_flags_t
- typedef uint8_t hsm_op_but_key_exp_flags_t
- typedef uint8_t hsm_op_cipher_one_go_algo_t
- typedef uint8_t hsm_op_cipher_one_go_flags_t
- typedef uint8_t hsm_op_signature_gen_flags_t
- typedef uint8_t hsm_op_signature_ver_flags_t
- typedef uint8_t hsm_op_fast_signature_gen_flags_t
- typedef uint8_t hsm_op_fast_signature_ver_flags_t
- typedef uint16_t hsm_key_type_t
- typedef uint16_t hsm_key_info_t

- typedef uint8_t hsm_signature_scheme_id_t
- typedef uint8_t hsm_hash_algo_t
- typedef uint32_t hsm_verification_status_t
- typedef uint32_t hsm_addr_msb_t
- typedef uint32_t hsm_addr_lsb_t

### Enumerations

- enum hsm_err_t {
  HSM_NO_ERROR = 0x0,
  HSM_INVALID_MESSAGE = 0x1,
  HSM_INVALID_ADDRESS = 0x2,
  HSM_UNKNOWN_ID = 0x3,
  HSM_INVALID_PARAM = 0x4,
  HSM_NVM_ERROR = 0x5,
  HSM_OUT_OF_MEMORY = 0x6,
  HSM_UNKNOWN_HANDLE = 0x7,
  HSM_UNKNOWN_KEY_STORE = 0x8,
  HSM_KEY_STORE_AUTH = 0x9,
  HSM_KEY_STORAGE_ERROR = 0xA,
  HSM_ID_CONFLICT = 0xB,
  HSM_RNG_NOT_STARTED = 0xC,
  HSM_CMD_NOT_SUPPORTED = 0xD,
  HSM_INVALID_LIFECYCLE = 0xE,
  HSM_KEY_STORE_CONFLICT = 0xF,
  HSM_GENERAL_ERROR = 0xFF }

  *Error codes returned by HSM functions.*

### Functions

- struct hsm_hdl_s * hsm_open_session (uint8_t session_priority, uint8_t operating_mode, hsm_err_t *error_code)
- hsm_err_t hsm_close_session (struct hsm_hdl_s *session_hdl)
- struct hsm_hdl_s * hsm_open_key_store_service (struct hsm_hdl_s *session_hdl, struct hsm_hdl_s **key_store_hdl, uint32_t key_store_identifier, uint32_t authentication_nonce, uint16_t max_updates_number, hsm_svc_key_store_flags_t flags, hsm_err_t *error_code)
- hsm_err_t hsm_close_key_store_service (struct hsm_hdl_s *key_store_hdl)
- struct hsm_hdl_s * hsm_open_key_management_service (struct hsm_hdl_s *key_store_hdl, hsm_addr_msb_t input_address_ext, hsm_addr_msb_t output_address_ext, hsm_err_t *error_code)
- hsm_err_t hsm_generate_key (struct hsm_hdl_s *key_management_hdl, uint32_t key_identifier, hsm_addr_lsb_t output, uint16_t output_size, hsm_key_type_t key_type, hsm_key_info_t key_info, hsm_op_key_gen_flags_t flags)
- hsm_err_t hsm_manage_key (struct hsm_hdl_s *key_management_hdl, uint32_t key_identifier, hsm_addr_lsb_t key, uint16_t key_size, hsm_key_type_t key_type, hsm_key_info_t key_info, hsm_op_manage_key_flags_t flags)
- hsm_err_t hsm_butterfly_key_expansion (struct hsm_hdl_s *key_management_hdl, uint32_t key_identifier, hsm_addr_lsb_t add_data_1, hsm_addr_lsb_t add_data_2, hsm_addr_lsb_t multiply_data, uint16_t data_1_size, uint16_t data_2_size, uint16_t multiply_data_size, uint32_t dest_key_identifier, hsm_addr_lsb_t output, uint32_t output_size, hsm_op_but_key_exp_flags_t flags)
- hsm_err_t hsm_close_key_management_service (struct hsm_hdl_s *key_management_hdl)
- struct hsm_hdl_s * hsm_open_cipher_service (struct hsm_hdl_s *key_store_hdl, hsm_addr_msb_t input_address_ext, hsm_addr_msb_t output_address_ext, hsm_svc_cipher_flags_t flags, hsm_err_t *error_code)
- hsm_err_t hsm_cipher_one_go (struct hsm_hdl_s *chiper_hdl, uint32_t key_identifier, hsm_addr_lsb_t input, hsm_addr_lsb_t output, hsm_addr_lsb_t iv, uint32_t input_size, uint32_t output_size, uint32_t iv_size, hsm_op_cipher_one_go_algo_t cipher_algo, hsm_op_cipher_one_go_flags_t flags)

- hsm_err_t hsm_close_cipher_service (struct hsm_hdl_s *chiper_hdl)
- struct hsm_hdl_s * hsm_open_signature_service (struct hsm_hdl_s *key_store_hdl, hsm_addr_msb_t input_address_ext, hsm_addr_msb_t output_address_ext, hsm_svc_signature_flags_t flags, hsm_err_t *error_code)
- hsm_err_t hsm_signature_generation (struct hsm_hdl_s *signature_hdl, uint32_t key_identifier, hsm_signature_scheme_id_t scheme_id, hsm_addr_lsb_t message, hsm_addr_lsb_t signature, uint32_t message_size, uint32_t signature_size, hsm_op_signature_gen_flags_t flags)
- hsm_err_t hsm_signature_verification (struct hsm_hdl_s *signature_hdl, hsm_addr_lsb_t key_address, hsm_signature_scheme_id_t scheme_id, hsm_addr_lsb_t message, hsm_addr_lsb_t signature, uint32_t message_size, uint32_t signature_size, hsm_verification_status_t *status, hsm_op_signature_ver_flags_t flags)
- hsm_err_t hsm_close_signature_service (struct hsm_hdl_s *signature_hdl)
- struct hsm_hdl_s * hsm_open_fast_signature_generation_service (struct hsm_hdl_s *key_store_hdl, hsm_addr_msb_t input_address_ext, hsm_addr_msb_t output_address_ext, uint32_t key_identifier, hsm_signature_scheme_id_t scheme_id, hsm_svc_fast_signature_generation_flags_t flags, hsm_err_t *error_code)
- hsm_err_t hsm_fast_signature_generation (struct hsm_hdl_s *fast_signature_gen_hdl, hsm_addr_lsb_t message, hsm_addr_lsb_t signature, uint32_t message_size, uint32_t signature_size, hsm_op_fast_signature_gen_flags_t flags)
- hsm_err_t hsm_close_fast_signature_generation_service (struct hsm_hdl_s *fast_signature_gen_hdl)
- struct hsm_hdl_s * hsm_open_fast_signature_verification_service (struct hsm_hdl_s *key_store_hdl, hsm_addr_msb_t input_address_ext, hsm_addr_msb_t output_address_ext, hsm_addr_msb_t key_address_ext, hsm_addr_lsb_t key_address, hsm_svc_fast_signature_verification_flags_t flags, hsm_signature_scheme_id_t scheme_id, hsm_err_t *error_code)
- hsm_err_t hsm_fast_signature_verification (struct hsm_hdl_s *fast_signature_ver_hdl, hsm_addr_lsb_t message, hsm_addr_lsb_t signature, uint32_t message_size, uint32_t signature_size, hsm_verification_status_t *status, hsm_op_fast_signature_ver_flags_t flags)
- hsm_err_t hsm_close_fast_signature_verification_service (struct hsm_hdl_s *fast_signature_ver_hdl)
- struct hsm_hdl_s * hsm_open_rng_service (struct hsm_hdl_s *session_hdl, hsm_addr_msb_t input_address_ext, hsm_addr_msb_t output_address_ext, hsm_svc_rng_flags_t flags, hsm_err_t *error_code)
- hsm_err_t hsm_rng_get_random (uint32_t rng_hdl, hsm_addr_lsb_t output, uint32_t output_size)
- hsm_err_t hsm_close_rng_service (struct hsm_hdl_s *rng_hdl)
- struct hsm_hdl_s * hsm_open_hash_service (struct hsm_hdl_s *session_hdl, hsm_addr_msb_t input_address_ext, hsm_addr_msb_t output_address_ext, hsm_svc_hash_flags_t flags, hsm_err_t *error_code)
- hsm_err_t hsm_hash_one_go (struct hsm_hdl_s *hash_hdl, hsm_addr_lsb_t input, hsm_addr_lsb_t output, uint32_t input_size, uint32_t output_size, hsm_hash_algo_t algo)
- hsm_err_t hsm_close_hash_service (struct hsm_hdl_s *hash_hdl)

### 5.1.1 Detailed Description

i.MX8 HSM API header file

### 5.1.2 Macro Definition Documentation

#### 5.1.2.1 HSM_SVC_KEY_STORE_FLAGS_CREATE

```
#define HSM_SVC_KEY_STORE_FLAGS_CREATE ((hsm_svc_key_store_flags_t)(1 << 0))
```

It must be specified to create a new key storage

#### 5.1.2.2 HSM_SVC_KEY_STORE_FLAGS_UPDATE

```
#define HSM_SVC_KEY_STORE_FLAGS_UPDATE ((hsm_svc_key_store_flags_t)(1 << 1))
```

#### 5.1.2.3 HSM_SVC_KEY_STORE_FLAGS_DELETE

```
#define HSM_SVC_KEY_STORE_FLAGS_DELETE ((hsm_svc_key_store_flags_t)(1 << 3))
```

#### 5.1.2.4 HSM_KEY_TYPE_ECDSA_NIST_P224

```
#define HSM_KEY_TYPE_ECDSA_NIST_P224 ((hsm_key_type_t)0x01)
```

#### 5.1.2.5 HSM_KEY_TYPE_ECDSA_NIST_P256

```
#define HSM_KEY_TYPE_ECDSA_NIST_P256 ((hsm_key_type_t)0x02)
```

#### 5.1.2.6 HSM_KEY_TYPE_ECDSA_NIST_P384

```
#define HSM_KEY_TYPE_ECDSA_NIST_P384 ((hsm_key_type_t)0x03)
```

#### 5.1.2.7 HSM_KEY_TYPE_ECDSA_BRAINPOOL_R1_224

```
#define HSM_KEY_TYPE_ECDSA_BRAINPOOL_R1_224 ((hsm_key_type_t)0x12)
```

#### 5.1.2.8 HSM_KEY_TYPE_ECDSA_BRAINPOOL_R1_256

```
#define HSM_KEY_TYPE_ECDSA_BRAINPOOL_R1_256 ((hsm_key_type_t)0x13)
```

#### 5.1.2.9 HSM_KEY_TYPE_ECDSA_BRAINPOOL_R1_384

```
#define HSM_KEY_TYPE_ECDSA_BRAINPOOL_R1_384 ((hsm_key_type_t)0x15)
```

#### 5.1.2.10 HSM_KEY_TYPE_ECDSA_BRAINPOOL_T1_224

```
#define HSM_KEY_TYPE_ECDSA_BRAINPOOL_T1_224 ((hsm_key_type_t)0x22)
```

#### 5.1.2.11 HSM_KEY_TYPE_ECDSA_BRAINPOOL_T1_256

```
#define HSM_KEY_TYPE_ECDSA_BRAINPOOL_T1_256 ((hsm_key_type_t)0x23)
```

#### 5.1.2.12 HSM_KEY_TYPE_ECDSA_BRAINPOOL_T1_384

```
#define HSM_KEY_TYPE_ECDSA_BRAINPOOL_T1_384 ((hsm_key_type_t)0x25)
```

#### 5.1.2.13 HSM_KEY_TYPE_AES_128

```
#define HSM_KEY_TYPE_AES_128 ((hsm_key_type_t)0x30)
```

#### 5.1.2.14 HSM_KEY_TYPE_AES_192

```
#define HSM_KEY_TYPE_AES_192 ((hsm_key_type_t)0x31)
```

#### 5.1.2.15 HSM_KEY_TYPE_AES_256

```
#define HSM_KEY_TYPE_AES_256 ((hsm_key_type_t)0x32)
```

#### 5.1.2.16 HSM_KEY_INFO_PERMANENT

```
#define HSM_KEY_INFO_PERMANENT ((hsm_key_info_t)(1 << 0))
```

When set, the key is permanent. Once created, it will not be possible to update or delete the key anymore. This bit can never be reset.

#### 5.1.2.17 HSM_OP_KEY_GENERATION_FLAGS_UPDATE

```
#define HSM_OP_KEY_GENERATION_FLAGS_UPDATE ((hsm_op_key_gen_flags_t)(1 << 0))
```

User can replace an existing key only by generating a key with the same type of the original one.

#### 5.1.2.18 HSM_OP_KEY_GENERATION_FLAGS_CREATE_PERSISTENT

```
#define HSM_OP_KEY_GENERATION_FLAGS_CREATE_PERSISTENT ((hsm_op_key_gen_flags_t)(1 << 1))
```

Persistent keys are saved in the non volatile memory.

#### 5.1.2.19 HSM_OP_KEY_GENERATION_FLAGS_CREATE_TRANSIENT

```
#define HSM_OP_KEY_GENERATION_FLAGS_CREATE_TRANSIENT ((hsm_op_key_gen_flags_t)(1 << 2))
```

Transient keys are deleted when the corresponding key store service flow is closed.

#### 5.1.2.20 HSM_OP_KEY_GENERATION_FLAGS_STRICT_OPERATION

```
#define HSM_OP_KEY_GENERATION_FLAGS_STRICT_OPERATION ((hsm_op_key_gen_flags_t)(1 << 7))
```

The request is completed only when the new key has been written in the NVM. This applicable for persistent key only.

#### 5.1.2.21 HSM_OP_MANGE_KEY_FLAGS_UPDATE

```
#define HSM_OP_MANGE_KEY_FLAGS_UPDATE ((hsm_op_manage_key_flags_t)(1 << 0))
```

#### 5.1.2.22 HSM_OP_MANGE_KEY_FLAGS_DELETE

```
#define HSM_OP_MANGE_KEY_FLAGS_DELETE ((hsm_op_manage_key_flags_t)(1 << 1))
```

#### 5.1.2.23 HSM_OP_MANGE_KEY_FLAGS_STRICT_OPERATION

```
#define HSM_OP_MANGE_KEY_FLAGS_STRICT_OPERATION ((hsm_op_manage_key_flags_t)(1 << 7))
```

The request is completed only when the modification has been written in the NVM. This applicable for persistent key only.

#### 5.1.2.24 HSM_CIPHER_ONE_GO_ALGO_AES_ECB

```
#define HSM_CIPHER_ONE_GO_ALGO_AES_ECB ((hsm_op_cipher_one_go_algo_t)(0x00))
```

#### 5.1.2.25 HSM_CIPHER_ONE_GO_ALGO_AES_CBC

```
#define HSM_CIPHER_ONE_GO_ALGO_AES_CBC ((hsm_op_cipher_one_go_algo_t)(0x01))
```

#### 5.1.2.26 HSM_CIPHER_ONE_GO_ALGO_AES_CCM

```
#define HSM_CIPHER_ONE_GO_ALGO_AES_CCM ((hsm_op_cipher_one_go_algo_t)(0x02))
```

Perform AES CCM with following prerequisites:

- Adata = 0 - There is no associated data
- Tlen = 16 bytes

#### 5.1.2.27 HSM_CIPHER_ONE_GO_FLAGS_ENCRYPT

```
#define HSM_CIPHER_ONE_GO_FLAGS_ENCRYPT ((hsm_op_cipher_one_go_flags_t)(1 << 0))
```

#### 5.1.2.28 HSM_CIPHER_ONE_GO_FLAGS_DECRYPT

```
#define HSM_CIPHER_ONE_GO_FLAGS_DECRYPT ((hsm_op_cipher_one_go_flags_t)(1 << 1))
```

#### 5.1.2.29 HSM_OP_SIGNATURE_GENERATION_INPUT_DIGEST

```
#define HSM_OP_SIGNATURE_GENERATION_INPUT_DIGEST ((hsm_op_signature_gen_flags_t)(0 << 0))
```

#### 5.1.2.30 HSM_OP_SIGNATURE_GENERATION_INPUT_MESSAGE

```
#define HSM_OP_SIGNATURE_GENERATION_INPUT_MESSAGE ((hsm_op_signature_gen_flags_t)(1 << 1))
```

#### 5.1.2.31 HSM_OP_SIGNATURE_GENERATION_COMPRESSED_POINT

```
#define HSM_OP_SIGNATURE_GENERATION_COMPRESSED_POINT ((hsm_op_signature_gen_flags_t)(2 << 1))
```

#### 5.1.2.32 HSM_SIGNATURE_SCHEME_ECDSA_NIST_P224_SHA_256

```
#define HSM_SIGNATURE_SCHEME_ECDSA_NIST_P224_SHA_256 ((hsm_signature_scheme_id_t)0x01)
```

#### 5.1.2.33 HSM_SIGNATURE_SCHEME_ECDSA_NIST_P256_SHA_256

```
#define HSM_SIGNATURE_SCHEME_ECDSA_NIST_P256_SHA_256 ((hsm_signature_scheme_id_t)0x02)
```

#### 5.1.2.34 HSM_SIGNATURE_SCHEME_ECDSA_NIST_P384_SHA_384

```
#define HSM_SIGNATURE_SCHEME_ECDSA_NIST_P384_SHA_384 ((hsm_signature_scheme_id_t)0x03)
```

#### 5.1.2.35 HSM_SIGNATURE_SCHEME_ECDSA_BRAINPOOL_R1_224_SHA_256

```
#define HSM_SIGNATURE_SCHEME_ECDSA_BRAINPOOL_R1_224_SHA_256 ((hsm_signature_scheme_id_t)0x12)
```

#### 5.1.2.36 HSM_SIGNATURE_SCHEME_ECDSA_BRAINPOOL_R1_256_SHA_256

```
#define HSM_SIGNATURE_SCHEME_ECDSA_BRAINPOOL_R1_256_SHA_256 ((hsm_signature_scheme_id_t)0x13)
```

#### 5.1.2.37 HSM_SIGNATURE_SCHEME_ECDSA_BRAINPOOL_R1_384_SHA_384

```
#define HSM_SIGNATURE_SCHEME_ECDSA_BRAINPOOL_R1_384_SHA_384 ((hsm_signature_scheme_id_t)0x15)
```

#### 5.1.2.38 HSM_SIGNATURE_SCHEME_ECDSA_BRAINPOOL_T1_224_SHA_256

```
#define HSM_SIGNATURE_SCHEME_ECDSA_BRAINPOOL_T1_224_SHA_256 ((hsm_signature_scheme_id_t)0x22)
```

#### 5.1.2.39 HSM_SIGNATURE_SCHEME_ECDSA_BRAINPOOL_T1_256_SHA_256

```
#define HSM_SIGNATURE_SCHEME_ECDSA_BRAINPOOL_T1_256_SHA_256 ((hsm_signature_scheme_id_t)0x23)
```

#### 5.1.2.40 HSM_SIGNATURE_SCHEME_ECDSA_BRAINPOOL_T1_384_SHA_384

```
#define HSM_SIGNATURE_SCHEME_ECDSA_BRAINPOOL_T1_384_SHA_384 ((hsm_signature_scheme_id_t)0x25)
```

#### 5.1.2.41 HSM_OP_SIGNATURE_VERIFICATION_INPUT_DIGEST

```
#define HSM_OP_SIGNATURE_VERIFICATION_INPUT_DIGEST ((hsm_op_signature_ver_flags_t)(0 << 0))
```

#### 5.1.2.42 HSM_OP_SIGNATURE_VERIFICATION_INPUT_MESSAGE

```
#define HSM_OP_SIGNATURE_VERIFICATION_INPUT_MESSAGE ((hsm_op_signature_ver_flags_t)(1 << 1))
```

#### 5.1.2.43 HSM_VERIFICATION_STATUS_SUCCESS

```
#define HSM_VERIFICATION_STATUS_SUCCESS ((hsm_verification_status_t)(0x5A3CC3A5))
```

#### 5.1.2.44 HSM_VERIFICATION_STATUS_FAILURE

```
#define HSM_VERIFICATION_STATUS_FAILURE ((hsm_verification_status_t)(0xA5C33C5A))
```

#### 5.1.2.45 HSM_OP_FAST_SIGNATURE_GENERATION_INPUT_DIGEST

```
#define HSM_OP_FAST_SIGNATURE_GENERATION_INPUT_DIGEST ((hsm_op_fast_signature_gen_flags_t)(0
<< 0))
```

#### 5.1.2.46 HSM_OP_FAST_SIGNATURE_GENERATION_INPUT_MESSAGE

```
#define HSM_OP_FAST_SIGNATURE_GENERATION_INPUT_MESSAGE ((hsm_op_fast_signature_gen_flags_t)(1
<< 1))
```

#### 5.1.2.47 HSM_OP_FAST_SIGNATURE_GENERATION_COMPRESSED_POINT

```
#define HSM_OP_FAST_SIGNATURE_GENERATION_COMPRESSED_POINT ((hsm_op_fast_signature_gen_flags_t)(2
<< 1))
```

#### 5.1.2.48 HSM_OP_FAST_SIGNATURE_VERIFICATION_INPUT_DIGEST

```
#define HSM_OP_FAST_SIGNATURE_VERIFICATION_INPUT_DIGEST ((hsm_op_fast_signature_ver_flags_t)(0
<< 0))
```

#### 5.1.2.49 HSM_OP_FAST_SIGNATURE_VERIFICATION_INPUT_MESSAGE

```
#define HSM_OP_FAST_SIGNATURE_VERIFICATION_INPUT_MESSAGE ((hsm_op_fast_signature_ver_flags_t)(1
<< 1))
```

#### 5.1.2.50 HSM_HASH_ALGO_SHA2_224

```
#define HSM_HASH_ALGO_SHA2_224 ((hsm_hash_algo_t)(0x0))
```

#### 5.1.2.51 HSM_HASH_ALGO_SHA2_256

```
#define HSM_HASH_ALGO_SHA2_256 ((hsm_hash_algo_t)(0x1))
```

#### 5.1.2.52 HSM_HASH_ALGO_SHA2_384

```
#define HSM_HASH_ALGO_SHA2_384 ((hsm_hash_algo_t)(0x2))
```

### 5.1.3 Typedef Documentation

#### 5.1.3.1 hsm_svc_key_store_flags_t

```
typedef uint8_t hsm_svc_key_store_flags_t
```

#### 5.1.3.2 hsm_svc_key_management_flags_t

```
typedef uint8_t hsm_svc_key_management_flags_t
```

#### 5.1.3.3 hsm_svc_cipher_flags_t

```
typedef uint8_t hsm_svc_cipher_flags_t
```

#### 5.1.3.4 hsm_svc_signature_flags_t

```
typedef uint8_t hsm_svc_signature_flags_t
```

#### 5.1.3.5 hsm_svc_fast_signature_verification_flags_t

```
typedef uint8_t hsm_svc_fast_signature_verification_flags_t
```

#### 5.1.3.6 hsm_svc_fast_signature_generation_flags_t

```
typedef uint8_t hsm_svc_fast_signature_generation_flags_t
```

#### 5.1.3.7 hsm_svc_rng_flags_t

```
typedef uint8_t hsm_svc_rng_flags_t
```

#### 5.1.3.8 hsm_svc_hash_flags_t

```
typedef uint8_t hsm_svc_hash_flags_t
```

#### 5.1.3.9 hsm_op_key_gen_flags_t

```
typedef uint8_t hsm_op_key_gen_flags_t
```

#### 5.1.3.10 hsm_op_manage_key_flags_t

```
typedef uint8_t hsm_op_manage_key_flags_t
```

#### 5.1.3.11 hsm_op_but_key_exp_flags_t

```
typedef uint8_t hsm_op_but_key_exp_flags_t
```

#### 5.1.3.12 hsm_op_cipher_one_go_algo_t

```
typedef uint8_t hsm_op_cipher_one_go_algo_t
```

#### 5.1.3.13 hsm_op_cipher_one_go_flags_t

```
typedef uint8_t hsm_op_cipher_one_go_flags_t
```

#### 5.1.3.14 hsm_op_signature_gen_flags_t

```
typedef uint8_t hsm_op_signature_gen_flags_t
```

#### 5.1.3.15 hsm_op_signature_ver_flags_t

```
typedef uint8_t hsm_op_signature_ver_flags_t
```

#### 5.1.3.16 hsm_op_fast_signature_gen_flags_t

```
typedef uint8_t hsm_op_fast_signature_gen_flags_t
```

#### 5.1.3.17 hsm_op_fast_signature_ver_flags_t

```
typedef uint8_t hsm_op_fast_signature_ver_flags_t
```

#### 5.1.3.18 hsm_key_type_t

```
typedef uint16_t hsm_key_type_t
```

#### 5.1.3.19 hsm_key_info_t

```
typedef uint16_t hsm_key_info_t
```

#### 5.1.3.20 hsm_signature_scheme_id_t

```
typedef uint8_t hsm_signature_scheme_id_t
```

#### 5.1.3.21 hsm_hash_algo_t

```
typedef uint8_t hsm_hash_algo_t
```

#### 5.1.3.22 hsm_verification_status_t

```
typedef uint32_t hsm_verification_status_t
```

#### 5.1.3.23 hsm_addr_msb_t

```
typedef uint32_t hsm_addr_msb_t
```

#### 5.1.3.24 hsm_addr_lsb_t

```
typedef uint32_t hsm_addr_lsb_t
```

### 5.1.4 Enumeration Type Documentation

#### 5.1.4.1 hsm_err_t

```
enum hsm_err_t
```

Error codes returned by HSM functions.

**Enumerator**

| | |
|---|---|
| HSM_NO_ERROR | Success. |
| HSM_INVALID_MESSAGE | The received message is invalid or unknown. |
| HSM_INVALID_ADDRESS | The provided address is invalid or doesn't respect the API requirements. |
| HSM_UNKNOWN_ID | The provided identifier is not known. |
| HSM_INVALID_PARAM | One of the parameter provided in the command is invalid. |
| HSM_NVM_ERROR | NVM generic issue. |
| HSM_OUT_OF_MEMORY | There is not enough memory to handle the requested operation. |
| HSM_UNKNOWN_HANDLE | Unknown session/service handle. |
| HSM_UNKNOWN_KEY_STORE | The key store identified by the provided "key store Id" doesn't exist and the "create" flag is not set. |
| HSM_KEY_STORE_AUTH | Key storage authentication fails. |
| HSM_KEY_STORAGE_ERROR | An error occurred in the key storage internal processing. |
| HSM_ID_CONFLICT | An element (key storage, key. . . ) with the provided ID already exists. |
| HSM_RNG_NOT_STARTED | The internal RNG is not started. |
| HSM_CMD_NOT_SUPPORTED | The functionality is not supported for the current session/service/key store configuration. |
| HSM_INVALID_LIFECYCLE | Invalid lifecycle for requested operation. |
| HSM_KEY_STORE_CONFLICT | An key store with the same attributes already exists. |
| HSM_GENERAL_ERROR | Error not covered by other codes occured. |

### 5.1.5 Function Documentation

#### 5.1.5.1 hsm_open_session()

```
struct hsm_hdl_s* hsm_open_session (
            uint8_t session_priority,
            uint8_t operating_mode,
            hsm_err_t * error_code )
```

Initiate a HSM session.

**Parameters**

| | |
|---|---|
| *session_priority* | not supported in current release, any value accepted. |
| *operating_mode* | not supported in current release, any value accepted. |
| *error_code* | pointer to where the error code should be written. |

**Returns**

Pointer to the handle identifying the session. NULL in case of error.
The returned pointer is typed with the struct "hsm_hdl_s". The user doesn't need to know or to access the fields of this struct, but it needs to store and pass the pointer to the subsequent services/operaton calls.

#### 5.1.5.2 hsm_close_session()

```
hsm_err_t hsm_close_session (
            struct hsm_hdl_s * session_hdl )
```

Terminate a previously opened HSM session

**Parameters**

| | |
|---|---|
| *session_hdl* | pointer to the handle identifying the session to be closed. |

**Returns**

error_code error code.

#### 5.1.5.3 hsm_open_key_store_service()

```
struct hsm_hdl_s* hsm_open_key_store_service (
            struct hsm_hdl_s * session_hdl,
            struct hsm_hdl_s ** key_store_hdl,
            uint32_t key_store_identifier,
            uint32_t authentication_nonce,
            uint16_t max_updates_number,
            hsm_svc_key_store_flags_t flags,
            hsm_err_t * error_code )
```

Open a service flow on the specified key store.

**Parameters**

| | |
|---|---|
| *session_hdl* | pointer to the handle indentifing the current session. |
| *key_store_identifier* | user defined id identifying the key store. |
| *authentication_nonce* | user defined nonce used as authentication proof for accesing the key storage. |
| *max_updates_number* | maximum number of updates authorized for the storage. Valid only for create operation. |
| *access_flags* | bitmap indicating the requested access to the key store. |
| *error_code* | pointer to where the error code should be written. |

**Returns**

Pointer to the handle indentifying the key store service flow. NULL in case of error. The returned pointer is typed with the struct "hsm_hdl_s". The user doesn't need to know or to access the fields of this struct, but it needs to store and pass the pointer to the subsequent services/operaton calls.

#### 5.1.5.4 hsm_close_key_store_service()

```
hsm_err_t hsm_close_key_store_service (
            struct hsm_hdl_s * key_store_hdl )
```

Close a previously opened key store service flow.

**Parameters**

| | |
|---|---|
| *pointer* | to the handle indentifing the key store service flow to be closed. |

**Returns**

error_code error code.

#### 5.1.5.5 hsm_open_key_management_service()

```
struct hsm_hdl_s* hsm_open_key_management_service (
            struct hsm_hdl_s * key_store_hdl,
            hsm_addr_msb_t input_address_ext,
            hsm_addr_msb_t output_address_ext,
            hsm_err_t * error_code )
```

Open a key management service flow
User must open this service in order to perform operation on the key store content: key generate, delete, update

**Parameters**

| | |
|---|---|
| *key_store_hdl* | pointer to the handle indentifing the key management service flow. |
| *input_address_ext* | most significant 32 bits address to be used by HSM for input memory transactions in the requester address space for the commands handled by the service flow. |
| *output_address_ext* | most significant 32 bits address to be used by HSM for output memory transactions in the requester address space for the commands handled by the service flow. |
| *error_code* | pointer to where the error code should be written. |
| *Pointer* | to the handle indentifing the key management service flow. NULL in case of error. The returned pointer is typed with the struct "hsm_hdl_s". The user doesn't need to know or to access the fields of this struct, but it needs to store and pass the pointer to the subsequent services/operaton calls. |

#### 5.1.5.6 hsm_generate_key()

```
hsm_err_t hsm_generate_key (
            struct hsm_hdl_s * key_management_hdl,
            uint32_t key_identifier,
            hsm_addr_lsb_t output,
            uint16_t output_size,
            hsm_key_type_t key_type,
            hsm_key_info_t key_info,
            hsm_op_key_gen_flags_t flags )
```

Generate a key or a key pair in the key store. In case of asymetic keys, the public key can optionally be exported. The generated key can be stored in a new or in an existing key slot with the restriction that an existing key can be replaced only by a key of the same type.
User can call this function only after having opened a key management service flow

**Parameters**

| | |
|---|---|
| *key_management_hdl* | pointer to handle identifying the key management service flow. |
| *key_identifier* | pointer to the identifier of the key to be used for the operation. In case of create operation the new key identifier will be stored in this location. |
| *output* | LSB of the address in the requester space where to store the public key. This address is combined with the 32 bits UOA extension provided for the service flow |
| *output_size* | lenght in bytes of the output area, if the size is 0, no key is copied in the output. |
| *key_type* | indicates which type of key must be generated |
| *key_info* | bitmap specifying the properties of the key |
| *flags* | bitmap specifying the operation properties |

**Returns**

error code

#### 5.1.5.7 hsm_manage_key()

```
hsm_err_t hsm_manage_key (
            struct hsm_hdl_s * key_management_hdl,
            uint32_t key_identifier,
            hsm_addr_lsb_t key,
            uint16_t key_size,
            hsm_key_type_t key_type,
            hsm_key_info_t key_info,
            hsm_op_manage_key_flags_t flags )
```

This command is designed to perform operation on an existing key.
User can call this function only after having opened a key management service flow

**Parameters**

| | |
|---|---|
| *key_management_hdl* | pointer to handle identifying the key management service flow. |
| *key_identifier* | identifier of the key to be used for the operation. |
| *key_address* | LSB of the address in the requester space where the new key value can be found. This address is combined with the 32 bits UIA extension provided for the service flow. Not checked in case of delete operation. |
| *key_size* | lenght in bytes of the input key area. Not checked in case of delete operation. |
| *key_type* | indicates the type of the key to be managed. |
| *key_info* | bitmap specifying the properties of the key, it will replace the existing value. Not checked in case of delete operation.. |
| *flags* | bitmap specifying the operation properties |

**Returns**

error code

#### 5.1.5.8 hsm_butterfly_key_expansion()

```
hsm_err_t hsm_butterfly_key_expansion (
            struct hsm_hdl_s * key_management_hdl,
            uint32_t key_identifier,
            hsm_addr_lsb_t add_data_1,
            hsm_addr_lsb_t add_data_2,
            hsm_addr_lsb_t multiply_data,
            uint16_t data_1_size,
            uint16_t data_2_size,
            uint16_t multiply_data_size,
            uint32_t dest_key_identifier,
            hsm_addr_lsb_t output,
            uint32_t output_size,
            hsm_op_but_key_exp_flags_t flags )
```

This command is designed to perform the butterfly key expansion operation on an ECC private key in case of implicit certificate. Optionally the resulting public key is exported.
User can call this function only after having opened a key management service flow

The following operation is performed: ButKey = (Key + AddData1) ∗ MultiplyData + AddData2 (mod n)

**Parameters**

| | |
|---|---|
| *key_management_hdl* | pointer to handle identifying the key store management service flow. |
| *key_identifier* | identifier of the key to be used for the operation. |
| *add_data_1* | LSB of the address in the requester space where the add_data_1 input can be found value 0 in case of explicit certificate<br>expansion function f1(k, i, j) result value in case of implicit certificate. |
| *add_data_2* | LSB of the address in the requester space where the add_data_2 input can be found expansion function f1/f2(k, i, j) result value in case of explicit certificate<br>the private reconstruction value used in the derivation of the pseudonym ECC key in case of implicit certificate |
| *multiply_data* | LSB of the address in the requester space where the multiply_data input can be found value 1 in case of explicit certificate<br>the hash value used to in the derivation of the pseudonym ECC key |
| *data_1_size* | lenght in bytes of the add_data_1 input |
| *data_2_size* | lenght in bytes of the add_data_2 input |
| *multiply_date_size* | lenght in bytes of the multiply_data input |
| *output* | LSB of the address in the requester space where to store the public key. This address is combined with the 32 bits UOA extension provided for the service flow |
| *output_size* | lenght in bytes of the output area, if the size is 0, no key is copied in the output. |
| *flags* | bitmap specifying the operation properties |

**Returns**

error code

#### 5.1.5.9 hsm_close_key_management_service()

```
hsm_err_t hsm_close_key_management_service (
            struct hsm_hdl_s * key_management_hdl )
```

Terminate a previously opened key management service flow

**Parameters**

| | |
|---|---|
| *key_management_hdl* | pointer to handle identifying the key management service flow. |

**Returns**

error code

#### 5.1.5.10 hsm_open_cipher_service()

```
struct hsm_hdl_s* hsm_open_cipher_service (
            struct hsm_hdl_s * key_store_hdl,
            hsm_addr_msb_t input_address_ext,
            hsm_addr_msb_t output_address_ext,
            hsm_svc_cipher_flags_t flags,
            hsm_err_t * error_code )
```

Open a cipher service flow
User can call this function only after having opened a key store service flow. User must open this service in order to perform cipher operations.

**Parameters**

| | |
|---|---|
| *key_store_hdl* | pointer to the handle indentifing the key management service flow. |
| *input_address_ext* | most significant 32 bits address to be used by HSM for input memory transactions in the requester address space for the operations handled by the service flow. |
| *output_address_ext* | most significant 32 bits address to be used by HSM for output memory transactions in the requester address space for the opeartion handled by the service flow. |
| *flags* | bitmap indicating the service flow properties - not supported in current release, any value accepted. |
| *error_code* | pointer to where the error code should be written. |
| *pointer* | to the handle indentifing the cipher service flow. NULL in case of error. The returned pointer is typed with the struct "hsm_hdl_s". The user doesn't need to know or to access the fields of this struct, but it needs to store and pass the pointer to the subsequent services/operaton calls. |

#### 5.1.5.11 hsm_cipher_one_go()

```
hsm_err_t hsm_cipher_one_go (
            struct hsm_hdl_s * chiper_hdl,
            uint32_t key_identifier,
            hsm_addr_lsb_t input,
            hsm_addr_lsb_t output,
            hsm_addr_lsb_t iv,
            uint32_t input_size,
            uint32_t output_size,
            uint32_t iv_size,
```

```
            hsm_op_cipher_one_go_algo_t cipher_algo,
            hsm_op_cipher_one_go_flags_t flags )
```

Perform ciphering operation
User can call this function only after having opened a cipher service flow

**Parameters**

| | |
|---|---|
| *chiper_hdl* | pointer to handle identifying the cipher service flow. |
| *key_identifier* | identifier of the key to be used for the operation |
| *input* | LSB of the address in the requester space where the input to be processed can be found<br>plaintext for encryption<br>ciphertext for decryption (tag is concatenated for CCM) |
| *output* | LSB of the address in the requester space where the output must be stored<br>ciphertext for encryption (tag is concatenated for CCM)<br>plaintext for decryption |
| *iv* | LSB of the address in the requester space where the initialization vector can be found |
| *input_size* | lenght in bytes of the input |
| *iv_size* | lenght in bytes of the initialization vector<br>it must be 0 for algorithms not using the initialization vector.<br>It must be 12 for AES in CCM mode |
| *cipher_algo* | algorithm to be used for the operation |
| *flags* | bitmap specifying the operation attributes |

**Returns**

error code

#### 5.1.5.12 hsm_close_cipher_service()

```
hsm_err_t hsm_close_cipher_service (
            struct hsm_hdl_s * chiper_hdl )
```

Terminate a previously opened cipher service flow

**Parameters**

| | |
|---|---|
| *chiper_hdl* | pointer to handle identifying the cipher service flow to be closed. |

**Returns**

error code

#### 5.1.5.13 hsm_open_signature_service()

```
struct hsm_hdl_s* hsm_open_signature_service (
            struct hsm_hdl_s * key_store_hdl,
```

```
            hsm_addr_msb_t input_address_ext,
            hsm_addr_msb_t output_address_ext,
            hsm_svc_signature_flags_t flags,
            hsm_err_t * error_code )
```

Open a signature service flow
User can call this function only after having opened a key store service flow. User must open this service in order to perform signature generation/verification operations.

**Parameters**

| | |
|---|---|
| *key_store_hdl* | pointer to the handle indentifing the key management service flow. |
| *input_address_ext* | most significant 32 bits address to be used by HSM for input memory transactions in the requester address space for the operations handled by the service flow. |
| *output_address_ext* | most significant 32 bits address to be used by HSM for output memory transactions in the requester address space for the opeartion handled by the service flow. |
| *flags* | bitmap indicating the service flow properties - not supported in current release, any value accepted. |
| *error_code* | pointer to where the error code should be written. |
| *pointer* | to the handle indentifing the signature service flow. NULL in case of error. The returned pointer is typed with the struct "hsm_hdl_s". The user doesn't need to know or to access the fields of this struct, but it needs to store and pass the pointer to the subsequent services/operaton calls. |

#### 5.1.5.14 hsm_signature_generation()

```
hsm_err_t hsm_signature_generation (
            struct hsm_hdl_s * signature_hdl,
            uint32_t key_identifier,
            hsm_signature_scheme_id_t scheme_id,
            hsm_addr_lsb_t message,
            hsm_addr_lsb_t signature,
            uint32_t message_size,
            uint32_t signature_size,
            hsm_op_signature_gen_flags_t flags )
```

Generate a digital signature according to the signature scheme
User can call this function only after having opened a signature service flow

**Parameters**

| | |
|---|---|
| *signature_hdl* | pointer to handle identifying the signature service flow |
| *key_identifier* | identifier of the key to be used for the operation |
| *scheme_id* | identifier of the digital signature scheme to be used for the operation |
| *message* | LSB of the address in the requester space where the input (message or message digest) to be processed can be found |
| *signature* | LSB of the address in the requester space where the signature must be stored<br>the signature S=(c,d) is stored as c\|\|d\|\|lsb_y in case of compressed point signature, c\|\|d otherwhise. |
| *message_size* | lenght in bytes of the input |
| *signature_size* | lenght in bytes of the output - it must contains additional 32bits where to store the Ry last significant bit |
| *flags* | bitmap specifying the operation attributes |

**Returns**

error code

#### 5.1.5.15 hsm_signature_verification()

```
hsm_err_t hsm_signature_verification (
            struct hsm_hdl_s * signature_hdl,
            hsm_addr_lsb_t key_address,
            hsm_signature_scheme_id_t scheme_id,
            hsm_addr_lsb_t message,
            hsm_addr_lsb_t signature,
            uint32_t message_size,
            uint32_t signature_size,
            hsm_verification_status_t * status,
            hsm_op_signature_ver_flags_t flags )
```

Verify a digital signature according to the signature scheme
User can call this function only after having opened a signature service flow

**Parameters**

| | |
|---|---|
| *signature_hdl* | pointer to handle identifying the signature service flow. |
| *key_address* | pointer to the key to be used for the operation |
| *key_identifier* | identifier of the key to be used for the operation |
| *ecc_domain↩ _id* | identifier of the supported ECC domains to be used for the operation |
| *message* | LSB of the address in the requester space where the input (message or message digest) to be processed can be found |
| *signature* | LSB of the address in the requester space where the signature can be found<br>the signature S=(c,d) must be in the format c\|\|d. |
| *message_size* | lenght in bytes of the input |
| *signature_size* | lenght in bytes of the output - it must contains additional 32bits where to store the Ry last significant bit |
| *status* | pointer to where the verification status must be stored<br>if the verification suceed the value<br>HSM_OP_SIGNATURE_VERIFICATION_STATUS_SUCCESS is returned. |
| *flags* | bitmap specifying the operation attributes |

**Returns**

error code

#### 5.1.5.16 hsm_close_signature_service()

```
hsm_err_t hsm_close_signature_service (
            struct hsm_hdl_s * signature_hdl )
```

Terminate a previously opened signature service flow

**Parameters**

| | |
|---|---|
| *signature_hdl* | pointer to handle identifying the signature service flow to be closed. |

**Returns**

error code

#### 5.1.5.17 hsm_open_fast_signature_generation_service()

```
struct hsm_hdl_s* hsm_open_fast_signature_generation_service (
            struct hsm_hdl_s * key_store_hdl,
            hsm_addr_msb_t input_address_ext,
            hsm_addr_msb_t output_address_ext,
            uint32_t key_identifier,
            hsm_signature_scheme_id_t scheme_id,
            hsm_svc_fast_signature_generation_flags_t flags,
            hsm_err_t * error_code )
```

Open a fast signature generation service flow
User can call this function only after having opened a key store service flow. User must open this service in order to perform several signature generation by using the same private key.

**Parameters**

| | |
|---|---|
| *key_store_hdl* | pointer to the handle indentifing the key management service flow. |
| *input_address_ext* | most significant 32 bits address to be used by HSM for input memory transactions in the requester address space for the operations handled by the service flow. |
| *output_address_ext* | most significant 32 bits address to be used by HSM for output memory transactions in the requester address space for the opeartion handled by the service flow. |
| *key_identifier* | identifier of the private key to be used for the subsequent operations |
| *flags* | bitmap indicating the service flow properties - not supported in current release, any value accepted. |
| *error_code* | pointer to where the error code should be written. |
| *pointer* | to the handle indentifing the fast signature generation service flow. NULL in case of error. The returned pointer is typed with the struct "hsm_hdl_s". The user doesn't need to know or to access the fields of this struct, but it needs to store and pass the pointer to the subsequent services/operaton calls. |

#### 5.1.5.18 hsm_fast_signature_generation()

```
hsm_err_t hsm_fast_signature_generation (
            struct hsm_hdl_s * fast_signature_gen_hdl,
            hsm_addr_lsb_t message,
            hsm_addr_lsb_t signature,
            uint32_t message_size,
            uint32_t signature_size,
            hsm_op_fast_signature_gen_flags_t flags )
```

Generate a digital signature according to the signature scheme
User can call this function only after having opened a fast signature generation service flow (key_identifier is omitted in the command)

**Parameters**

| | |
|---|---|
| *fast_signature_gen_hdl* | pointer to handle identifying the fast signature generation service flow |
| *scheme_id* | identifier of the digital signature scheme to be used for the operation |
| *message* | LSB of the address in the requester space where the input to be processed (message or message digest) can be found. |
| *signature* | LSB of the address in the requester space where the signature must be stored the signature S=(c,d) is stored as c\|\|d\|lsb_y in case of compressed point signature, c\|\|d otherwhise. |
| *message_size* | lenght in bytes of the input |
| *signature_size* | lenght in bytes of the output - In case of compressed point signature additional 32bit must be provided. |
| *flags* | bitmap specifying the operation attributes |

**Returns**

error code

#### 5.1.5.19 hsm_close_fast_signature_generation_service()

```
hsm_err_t hsm_close_fast_signature_generation_service (
            struct hsm_hdl_s * fast_signature_gen_hdl )
```

Terminate a previously opened fast signature generation service flow

**Parameters**

| | |
|---|---|
| *fast_signature_gen_hdl* | pointer to handle identifying the signature service flow to be closed. |

**Returns**

error code

#### 5.1.5.20 hsm_open_fast_signature_verification_service()

```
struct hsm_hdl_s* hsm_open_fast_signature_verification_service (
            struct hsm_hdl_s * key_store_hdl,
            hsm_addr_msb_t input_address_ext,
            hsm_addr_msb_t output_address_ext,
            hsm_addr_msb_t key_address_ext,
            hsm_addr_lsb_t key_address,
            hsm_svc_fast_signature_verification_flags_t flags,
```

```
            hsm_signature_scheme_id_t scheme_id,
            hsm_err_t * error_code )
```

Open a fast signature verification service flow
User can call this function only after having opened a key store service flow. User must open this service in order to perform several signature generation by using the same private key.

**Parameters**

| | |
|---|---|
| *key_store_hdl* | pointer to the handle indentifing the key management service flow. |
| *input_address_ext* | most significant 32 bits address to be used by HSM for input memory transactions in the requester address space for the operations handled by the service flow. |
| *output_address_ext* | most significant 32 bits address to be used by HSM for output memory transactions in the requester address space for the opeartion handled by the service flow. |
| *key_identifier* | identifier of the private key to be used for the subsequent operations |
| *flags* | bitmap indicating the service flow properties - not supported in current release, any value accepted. |
| *error_code* | pointer to where the error code should be written. |
| *pointer* | to the handle indentifing the fast signature generation service flow. NULL in case of error. The returned pointer is typed with the struct "hsm_hdl_s". The user doesn't need to know or to access the fields of this struct, but it needs to store and pass the pointer to the subsequent services/operaton calls. |

#### 5.1.5.21 hsm_fast_signature_verification()

```
hsm_err_t hsm_fast_signature_verification (
            struct hsm_hdl_s * fast_signature_ver_hdl,
            hsm_addr_lsb_t message,
            hsm_addr_lsb_t signature,
            uint32_t message_size,
            uint32_t signature_size,
            hsm_verification_status_t * status,
            hsm_op_fast_signature_ver_flags_t flags )
```

Verify a digital signature according to the signature scheme
User can call this function only after having opened a signature service flow

**Parameters**

| | |
|---|---|
| *signature_hdl* | pointer to handle identifying the signature service flow. |
| *key_address* | pointer to the key to be used for the operation |
| *key_identifier* | identifier of the key to be used for the operation |
| *ecc_domain↩ _id* | identifier of the supported ECC domains to be used for the operation |
| *message* | LSB of the address in the requester space where the input to be processed (message or message digest) can be found. |
| *signature* | message LSB of the address in the requester space where the signature can be foundmust be stored<br>the signature S=(c,d) must be in the c\|\|d format. |
| *message_size* | lenght in bytes of the input |
| *signature_size* | lenght in bytes of the signature. |

**Parameters**

| | |
|---|---|
| *status* | pointer to where the verification status must be stored<br>if the verification suceed the value<br>HSM_OP_SIGNATURE_VERIFICATION_STATUS_SUCCESS is returned. |
| *flags* | bitmap specifying the operation attributes. |

**Returns**

error code

#### 5.1.5.22 hsm_close_fast_signature_verification_service()

```
hsm_err_t hsm_close_fast_signature_verification_service (
            struct hsm_hdl_s * fast_signature_ver_hdl )
```

Terminate a previously opened fast signature generation service flow

**Parameters**

| | |
|---|---|
| *fast_signature_ver_hdl* | pointer to handle identifying the fast signature verification service flow to be closed. |

**Returns**

error code

#### 5.1.5.23 hsm_open_rng_service()

```
struct hsm_hdl_s* hsm_open_rng_service (
            struct hsm_hdl_s * session_hdl,
            hsm_addr_msb_t input_address_ext,
            hsm_addr_msb_t output_address_ext,
            hsm_svc_rng_flags_t flags,
            hsm_err_t * error_code )
```

Open a random number generation service flow
User can call this function only after having opened a session. User must open this service in order to perform rng operations.

**Parameters**

| | |
|---|---|
| *session_hdl* | pointer to the handle indentifing the current session. |
| *input_address_ext* | most significant 32 bits address to be used by HSM for input memory transactions in the requester address space for the operations handled by the service flow. |
| *output_address_ext* | most significant 32 bits address to be used by HSM for output memory transactions in the requester address space for the opeartion handled by the service flow. |
| *flags* | bitmap indicating the service flow properties |
| *error_code* | pointer to where the error code should be written. |
| *pointer* | to the handle indentifing the rng service flow. NULL in case of error. The returned pointer is typed with the struct "hsm_hdl_s". The user doesn't need to know or to access the fields of this struct, but it needs to store and pass the pointer to the subsequent services/operaton calls. |

#### 5.1.5.24 hsm_rng_get_random()

```
hsm_err_t hsm_rng_get_random (
            uint32_t rng_hdl,
            hsm_addr_lsb_t output,
            uint32_t output_size )
```

Get a freshly generated random number
User can call this function only after having opened a rng service flow

**Parameters**

| | |
|---|---|
| *rng_hdl* | pointer to handle identifying the rng service flow. |
| *output* | LSB of the address in the requester space where random number must be stored. |
| *output_size* | length of the random number in bytes |

**Returns**

error code

#### 5.1.5.25 hsm_close_rng_service()

```
hsm_err_t hsm_close_rng_service (
            struct hsm_hdl_s * rng_hdl )
```

Terminate a previously opened rng service flow

**Parameters**

| | |
|---|---|
| *rng_hdl* | pointer to handle identifying the rng service flow to be closed. |

**Returns**

error code

#### 5.1.5.26 hsm_open_hash_service()

```
struct hsm_hdl_s* hsm_open_hash_service (
            struct hsm_hdl_s * session_hdl,
            hsm_addr_msb_t input_address_ext,
            hsm_addr_msb_t output_address_ext,
            hsm_svc_hash_flags_t flags,
            hsm_err_t * error_code )
```

Open an hash service flow
User can call this function only after having opened a session. User must open this service in order to perform an hash operations.

**Parameters**

| | |
|---|---|
| *session_hdl* | pointer to the handle indentifing the current session. |
| *input_address_ext* | most significant 32 bits address to be used by HSM for input memory transactions in the requester address space for the operations handled by the service flow. |
| *output_address_ext* | most significant 32 bits address to be used by HSM for output memory transactions in the requester address space for the opeartion handled by the service flow. |
| *flags* | bitmap indicating the service flow properties |
| *error_code* | pointer to where the error code should be written. |
| *pointer* | to the handle indentifing the hash service flow. NULL in case of error. The returned pointer is typed with the struct "hsm_hdl_s". The user doesn't need to know or to access the fields of this struct, but it needs to store and pass the pointer to the subsequent services/operaton calls. |

#### 5.1.5.27 hsm_hash_one_go()

```
hsm_err_t hsm_hash_one_go (
            struct hsm_hdl_s * hash_hdl,
            hsm_addr_lsb_t input,
            hsm_addr_lsb_t output,
            uint32_t input_size,
            uint32_t output_size,
            hsm_hash_algo_t algo )
```

Perform the hash operation on a given input
User can call this function only after having opened a hash service flow

**Parameters**

| | |
|---|---|
| *hash_hdl* | pointer to handle identifying the hash service flow. |
| *input* | LSB of the address in the requester space where message to be hashed can be found. |
| *output* | LSB of the address in the requester space where the resulting hash must be stored. |
| *input_size* | lenght in bytes of the input |
| *output_size* | lenght in bytes of the output. |
| *algo* | algorithm to be used for the operation |

**Returns**

error code

#### 5.1.5.28 hsm_close_hash_service()

```
hsm_err_t hsm_close_hash_service (
            struct hsm_hdl_s * hash_hdl )
```

Terminate a previously opened hash service flow

**Parameters**

| | |
|---|---|
| *hash_hdl* | pointer to handle identifying the hash service flow to be closed. |

**Returns**

error code

# Index