

# i.MX8 HSM API

Revision\_1.6

Generated by Doxygen 1.8.15

---

<b>1 HSM API</b>	<b>2</b>
<b>2 Revision History</b>	<b>2</b>
<b>3 General concepts related to the API</b>	<b>2</b>
3.1 Session . . . . .	2
3.2 Service flow . . . . .	2
3.3 Key store . . . . .	3
3.3.1 Key management . . . . .	3
3.3.2 NVM writing . . . . .	3
<b>4 Module Index</b>	<b>3</b>
4.1 Modules . . . . .	3
<b>5 Data Structure Index</b>	<b>4</b>
5.1 Data Structures . . . . .	4
<b>6 Module Documentation</b>	<b>5</b>
6.1 Error codes . . . . .	5
6.1.1 Detailed Description . . . . .	5
6.1.2 Enumeration Type Documentation . . . . .	5
6.2 Session . . . . .	7
6.2.1 Detailed Description . . . . .	7
6.2.2 Function Documentation . . . . .	7
6.3 Key store . . . . .	9
6.3.1 Detailed Description . . . . .	9
6.3.2 Function Documentation . . . . .	9
6.4 Key management . . . . .	11
6.4.1 Detailed Description . . . . .	13
6.4.2 Function Documentation . . . . .	13
6.5 Ciphering . . . . .	17
6.5.1 Detailed Description . . . . .	17
6.5.2 Function Documentation . . . . .	17
6.6 Signature generation . . . . .	20
6.6.1 Detailed Description . . . . .	21
6.6.2 Function Documentation . . . . .	21
6.7 Signature verification . . . . .	23
6.7.1 Detailed Description . . . . .	23
6.7.2 Function Documentation . . . . .	23
6.8 Random number generation . . . . .	26
6.8.1 Detailed Description . . . . .	26
6.8.2 Function Documentation . . . . .	26
6.9 Hashing . . . . .	28
6.9.1 Detailed Description . . . . .	28

6.9.2 Function Documentation	28
6.10 Public key reconstruction	30
6.10.1 Detailed Description	30
6.10.2 Function Documentation	30
6.11 Public key decompression	31
6.11.1 Detailed Description	31
6.11.2 Function Documentation	31
6.12 ECIES encryption	32
6.12.1 Detailed Description	32
6.12.2 Function Documentation	32
6.13 Public key recovery	33
6.13.1 Detailed Description	33
6.13.2 Function Documentation	33
<b>7 Data Structure Documentation</b>	<b>34</b>
7.1 hsm_op_ecies_dec_args_t Struct Reference	34
7.2 hsm_op_ecies_enc_args_t Struct Reference	34
7.3 hsm_op_pub_key_dec_args_t Struct Reference	35
7.4 hsm_op_pub_key_rec_args_t Struct Reference	35
7.5 hsm_op_pub_key_recovery_args_t Struct Reference	36
7.6 op_but_key_exp_args_t Struct Reference	36
7.7 op_cipher_one_go_args_t Struct Reference	37
7.8 op_generate_key_args_t Struct Reference	38
7.9 op_generate_sign_args_t Struct Reference	38
7.10 op_get_random_args_t Struct Reference	38
7.11 op_hash_one_go_args_t Struct Reference	39
7.12 op_import_public_key_args_t Struct Reference	39
7.13 op_manage_key_args_t Struct Reference	39
7.14 op_manage_key_group_args_t Struct Reference	40
7.15 op_prepare_sign_args_t Struct Reference	40
7.16 op_verify_sign_args_t Struct Reference	40
7.17 open_session_args_t Struct Reference	41
7.18 open_svc_cipher_args_t Struct Reference	41
7.19 open_svc_hash_args_t Struct Reference	41
7.20 open_svc_key_management_args_t Struct Reference	41
7.21 open_svc_key_store_args_t Struct Reference	41
7.22 open_svc_rng_args_t Struct Reference	42
7.23 open_svc_sign_gen_args_t Struct Reference	42
7.24 open_svc_sign_ver_args_t Struct Reference	42
<b>Index</b>	<b>43</b>

## 1 HSM API

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

## 2 Revision History

Revision	date	description
0.1 - subject to change	Mar 29 2019	Savari preliminary draf
0.8 - subject to change	May 24 2019	It adds the following API: -signature generation -signature verification -rng -hash -butterfly key expansion -ECIES enc/dec -public key reconstruction -public key decompression
0.9 - subject to change	May 28 2019	Explicit addresses are replaced by pointers.
1.0 - subject to change	May 29 2019	- bug/typos fix. - Change HSM_SVC_KEY_STORE_FLAGS definition
1.1 - subject to change	July 31 2019	- hsm_butterfly_key_expansion argument definition: dest_key_↔ identifier is now a pointer. - Add error code definition. - improve argument comments clarity
1.5 - subject to change	Sept 13 2019	- manage key argument: fix padding size - butterfly key expansion: change argument definition - introduce public key recovery API
1.6 - subject to change	Oct 14 2019	- add Key store section in chapter 3 - change key_info and flags definition, substitute key_type_ext with group_id - hsm_generate_key, hsm_manage_key, hsm_butterfly_key_↔ expansion: change argument definition - hsm_manage_key: change argument definition - add hsm_manage_key_group 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 requestor and the HSM. When a session is opened, the HSM returns a handle identifying the session to the requestor.

### 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 and return a handle identifying the service flow.

The context is preserved until the service flow, or the session, are closed by the user and it is used by the HSM to proceed with the sub-subsequent operations requested by the user on the service flow.

### 3.3 Key store

The HSM handles only symmetric and private keys into the key store, the public, optionally exported during the key pair generation operation, can be recalculated through the ref group12 API.

Secret keys cannot be exported under any circumstances, while a way to import keys in encrypted form is provided.

#### 3.3.1 Key management

Keys are divided in groups, keys belonging to the same group are written/read from the NVM as a monolithic block. Up to 3 key groups can be handled in the HSM local memory (those immediately available to perform crypto operation), while up to 1024 key groups can be handled in the external NVM and imported in the local memory as needed. If the local memory is full (3 key groups already reside in the HSM local memory) and a new key group is needed by an incoming user request, the HSM swaps one of the local key group with the one needed by the user request. A control of which key group should be kept in the local memory (cached) is provide through the manage\_key\_group API lock/unlock mechanism.

As general concept, frequently used keys should be kept, when possible, in the same key group and locked in the local memory for performance optimization.

#### 3.3.2 NVM writing

All the APIs modifying the content of the key store (key generation/management) provide a "STRICT UPDATE" flag. If the flag is set, the HSM triggers and export of the encrypted key group into the external NVM and blows one bit of the OTP monotonic counter.

Any update to the key store must be considered as effective only after an operation specifying the flag "STRICT UPDATE" is acknowledged by the HSM. All the operations not specifying the "STRICT UPDATE" flags impact the HSM local memory only and will be lost in case of system reset

Due to the limited monotonic counter size (QXPB0 up to 1620 update available), the user should, when possible, perform multiple updates before setting the "STRICT UPDATE" flag.

Once the monotonic counter is completely blown a warning is returned on each update operation to inform the user that the new updates are not roll-back protected.

## 4 Module Index

### 4.1 Modules

Here is a list of all modules:

<b>Error codes</b>	<b>5</b>
<b>Session</b>	<b>7</b>
<b>Key store</b>	<b>9</b>

Key management	11
Ciphering	17
Signature generation	20
Signature verification	23
Random number generation	26
Hashing	28
Public key reconstruction	30
Public key decompression	31
ECIES encryption	32
Public key recovery	33

## 5 Data Structure Index

### 5.1 Data Structures

Here are the data structures with brief descriptions:

<a href="#">hsm_op_ecies_dec_args_t</a>	34
<a href="#">hsm_op_ecies_enc_args_t</a>	34
<a href="#">hsm_op_pub_key_dec_args_t</a>	35
<a href="#">hsm_op_pub_key_rec_args_t</a>	35
<a href="#">hsm_op_pub_key_recovery_args_t</a>	36
<a href="#">op_butl_key_exp_args_t</a>	36
<a href="#">op_cipher_one_go_args_t</a>	37
<a href="#">op_generate_key_args_t</a>	38
<a href="#">op_generate_sign_args_t</a>	38
<a href="#">op_get_random_args_t</a>	38
<a href="#">op_hash_one_go_args_t</a>	39
<a href="#">op_import_public_key_args_t</a>	39
<a href="#">op_manage_key_args_t</a>	39
<a href="#">op_manage_key_group_args_t</a>	40
<a href="#">op_prepare_sign_args_t</a>	40
<a href="#">op_verify_sign_args_t</a>	40
<a href="#">open_session_args_t</a>	41

<a href="#">open_svc_cipher_args_t</a>	41
<a href="#">open_svc_hash_args_t</a>	41
<a href="#">open_svc_key_management_args_t</a>	41
<a href="#">open_svc_key_store_args_t</a>	41
<a href="#">open_svc_rng_args_t</a>	42
<a href="#">open_svc_sign_gen_args_t</a>	42
<a href="#">open_svc_sign_ver_args_t</a>	42

## 6 Module Documentation

### 6.1 Error codes

#### 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\\_STORE\\_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\\_KEY\\_STORE\\_COUNTER](#) = 0x10,  
[HSM\\_FEATURE\\_NOT\\_SUPPORTED](#) = 0x11,  
[HSM\\_GENERAL\\_ERROR](#) = 0xFF }

#### 6.1.1 Detailed Description

#### 6.1.2 Enumeration Type Documentation

##### 6.1.2.1 [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 store authentication fails.
HSM_KEY_STORE_ERROR	An error occurred in the key store internal processing.
HSM_ID_CONFLICT	An element (key store, 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	A key store with the same attributes already exists.
HSM_KEY_STORE_COUNTER	The current key store reaches the max number of monotonic counter updates, updates are still allowed but monotonic counter will not be blown.
HSM_FEATURE_NOT_SUPPORTED	The requested feature is not supported by the firmware.
HSM_GENERAL_ERROR	Error not covered by other codes occurred.



## 6.2 Session

### Data Structures

- struct [open\\_session\\_args\\_t](#)

### Typedefs

- typedef uint32\_t **hsm\_hdl\_t**

### Functions

- [hsm\\_err\\_t hsm\\_open\\_session](#) ([open\\_session\\_args\\_t](#) \*args, hsm\_hdl\_t \*session\_hdl)
- [hsm\\_err\\_t hsm\\_close\\_session](#) (hsm\_hdl\_t session\_hdl)

#### 6.2.1 Detailed Description

The API must be initialized by a potential requestor by opening a session. Once a session is closed all the associated service flows are closed by the HSM.

#### 6.2.2 Function Documentation

##### 6.2.2.1 hsm\_open\_session()

```
hsm_err_t hsm_open_session (
    open_session_args_t * args,
    hsm_hdl_t * session_hdl )
```

##### Parameters

<i>args</i>	pointer to the structure containing the function arguments.
<i>session_hdl</i>	pointer to where the session handle must be written.

##### Returns

`error_code` error code.

##### 6.2.2.2 hsm\_close\_session()

```
hsm_err_t hsm_close_session (
    hsm_hdl_t session_hdl )
```

Terminate a previously opened session.

**Parameters**

<i>session_hdl</i>	pointer to the handle identifying the session to be closed.
--------------------	---

**Returns**

error\_code error code.

## 6.3 Key store

### Data Structures

- struct [open\\_svc\\_key\\_store\\_args\\_t](#)

### Macros

- #define [HSM\\_SVC\\_KEY\\_STORE\\_FLAGS\\_CREATE](#) ((hsm\_svc\_key\_store\_flags\_t)(1 << 0))  
*It must be specified to create a new key store.*
- #define [HSM\\_SVC\\_KEY\\_STORE\\_FLAGS\\_UPDATE](#) ((hsm\_svc\_key\_store\_flags\_t)(1 << 2))  
*Not supported - It must be specified in order to open a key management service flow.*
- #define [HSM\\_SVC\\_KEY\\_STORE\\_FLAGS\\_DELETE](#) ((hsm\_svc\_key\_store\_flags\_t)(1 << 3))  
*Not supported - It must be specified to delete an existing key store.*

### Typedefs

- typedef uint8\_t [hsm\\_svc\\_key\\_store\\_flags\\_t](#)

### Functions

- [hsm\\_err\\_t hsm\\_open\\_key\\_store\\_service](#) (hsm\_hdl\_t session\_hdl, [open\\_svc\\_key\\_store\\_args\\_t](#) \*args, hsm\_hdl\_t \*key\_store\_hdl)
- [hsm\\_err\\_t hsm\\_close\\_key\\_store\\_service](#) (hsm\_hdl\_t key\_store\_hdl)

#### 6.3.1 Detailed Description

User must open a key store service flow in order to perform the following operations:

- create a new key store
- update an existing key store
- delete an existing key store
- perform operations involving keys stored in the key store (ciphering, signature generation...)

The authentication is based on the user domain ID and messaging unit, additionally an authentication nonce is provided.

#### 6.3.2 Function Documentation

##### 6.3.2.1 hsm\_open\_key\_store\_service()

```
hsm_err_t hsm_open_key_store_service (
    hsm_hdl_t session_hdl,
    open_svc_key_store_args_t * args,
    hsm_hdl_t * key_store_hdl )
```

Open a service flow on the specified key store.

**Parameters**

<i>session_hdl</i>	pointer to the handle indentifying the current session.
<i>args</i>	pointer to the structure containing the function arugments.
<i>key_store_hdl</i>	pointer to where the key store service flow handle must be written.

**Returns**

error\_code error code.

**6.3.2.2 hsm\_close\_key\_store\_service()**

```
hsm_err_t hsm_close_key_store_service (
    hsm_hdl_t key_store_hdl )
```

Close a previously opened key store service flow.

**Parameters**

<i>handle</i>	indentifying the key store service flow to be closed.
---------------	---

**Returns**

error\_code error code.

## 6.4 Key management

### Data Structures

- struct [open\\_svc\\_key\\_management\\_args\\_t](#)
- struct [op\\_generate\\_key\\_args\\_t](#)
- struct [op\\_manage\\_key\\_args\\_t](#)
- struct [op\\_manage\\_key\\_group\\_args\\_t](#)
- struct [op\\_butt\\_key\\_exp\\_args\\_t](#)

### Macros

- #define [HSM\\_KEY\\_TYPE\\_ECDSA\\_NIST\\_P224](#) ((hsm\_key\_type\_t)0x01)  
*not supported*
- #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)  
*not supported*
- #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)  
*not supported*
- #define [HSM\\_KEY\\_TYPE\\_ECDSA\\_BRAINPOOL\\_T1\\_256](#) ((hsm\_key\_type\_t)0x23)  
*not supported*
- #define [HSM\\_KEY\\_TYPE\\_ECDSA\\_BRAINPOOL\\_T1\\_384](#) ((hsm\_key\_type\_t)0x25)  
*not supported*
- #define [HSM\\_KEY\\_TYPE\\_AES\\_128](#) ((hsm\_key\_type\_t)0x30)
- #define [HSM\\_KEY\\_TYPE\\_AES\\_192](#) ((hsm\_key\_type\_t)0x31)  
*not supported*
- #define [HSM\\_KEY\\_TYPE\\_AES\\_256](#) ((hsm\_key\_type\_t)0x32)  
*not supported*
- #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.*
- #define [HSM\\_OP\\_KEY\\_GENERATION\\_FLAGS\\_CREATE](#) ((hsm\_op\_key\_gen\_flags\_t)(1 << 1))  
*Create a new key.*
- #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.*
- #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. Transient keys will be anyway delated each PoR or when the corresponding key store service flow is closed. This bit can never be reset.*
- #define [HSM\\_KEY\\_INFO\\_TRANSIENT](#) ((hsm\_key\_info\_t)(1 << 1))  
*Transient keys are deleted when the corresponding key store service flow is closed. Transient key cannot be in the same key group than persistent keys.*
- #define [HSM\\_KEY\\_INFO\\_MASTER](#) ((hsm\_key\_info\_t)(1 << 2))  
*When set, the key is considered as a master key. Only master keys can be used as input of key derivation functions (i.e butterfly key expansion)*
- #define [HSM\\_OP\\_MANAGE\\_KEY\\_FLAGS\\_UPDATE](#) ((hsm\_op\_manage\_key\_flags\_t)(1 << 0))  
*not supported - User can replace an existing key only by importing a key with the same type of the original one.*
- #define [HSM\\_OP\\_MANAGE\\_KEY\\_FLAGS\\_CREATE](#) ((hsm\_op\_manage\_key\_flags\_t)(1 << 1))  
*not supported - Create a new key id.*

- `#define HSM_OP_MANAGE_KEY_FLAGS_DELETE ((hsm_op_manage_key_flags_t)(1 << 2))`  
*delete an existing key*
- `#define HSM_OP_MANAGE_KEY_FLAGS_STRICT_OPERATION ((hsm_op_manage_key_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.*
- `#define HSM_OP_MANAGE_KEY_GROUP_FLAGS_CACHE_LOCKDOWN ((hsm_op_manage_key_group_flags_t)(1 << 0))`  
*The entire key group will be cached in the HSM local memory.*
- `#define HSM_OP_MANAGE_KEY_GROUP_FLAGS_CACHE_UNLOCK ((hsm_op_manage_key_group_flags_t)(1 << 1))`  
*HSM may export the key group in the external NVM to free up the local memory. HSM will copy the key group in the local memory again in case of key group usage/update.*
- `#define HSM_OP_MANAGE_KEY_GROUP_FLAGS_DELETE ((hsm_op_manage_key_group_flags_t)(1 << 2))`  
*not supported - delete an existing key group*
- `#define HSM_OP_MANAGE_KEY_GROUP_FLAGS_STRICT_OPERATION ((hsm_op_manage_key_group_flags_t)(1 << 7))`  
*The request is completed only when the update has been written in the NVM. Not applicable for cache lock-down/unlock.*
- `#define HSM_OP_BUTTERFLY_KEY_FLAGS_UPDATE ((hsm_op_but_key_exp_flags_t)(1 << 0))`  
*User can replace an existing key only by generating a key with the same type of the original one.*
- `#define HSM_OP_BUTTERFLY_KEY_FLAGS_CREATE ((hsm_op_but_key_exp_flags_t)(1 << 1))`  
*Create a new key.*
- `#define HSM_OP_BUTTERFLY_KEY_FLAGS_IMPLICIT_CERTIF ((hsm_op_but_key_exp_flags_t)(0 << 2))`  
*butterfly key expansion using implicit certificate.*
- `#define HSM_OP_BUTTERFLY_KEY_FLAGS_EXPLICIT_CERTIF ((hsm_op_but_key_exp_flags_t)(1 << 2))`  
*butterfly key expansion using explicit certificate.*
- `#define HSM_OP_BUTTERFLY_KEY_FLAGS_STRICT_OPERATION ((hsm_op_but_key_exp_flags_t)(1 << 7))`  
*The request is completed only when the new key has been written in the NVM.*

## Typedefs

- `typedef uint8_t hsm_svc_key_management_flags_t`
- `typedef uint8_t hsm_op_key_gen_flags_t`
- `typedef uint8_t hsm_key_type_t`
- `typedef uint16_t hsm_key_info_t`
- `typedef uint16_t hsm_key_group_t`
- `typedef uint8_t hsm_op_manage_key_flags_t`
- `typedef uint8_t hsm_op_manage_key_group_flags_t`
- `typedef uint8_t hsm_op_but_key_exp_flags_t`

## Functions

- `hsm_err_t hsm_open_key_management_service (hsm_hdl_t key_store_hdl, open_svc_key_management_args_t *args, hsm_hdl_t *key_management_hdl)`
- `hsm_err_t hsm_generate_key (hsm_hdl_t key_management_hdl, op_generate_key_args_t *args)`
- `hsm_err_t hsm_manage_key (hsm_hdl_t key_management_hdl, op_manage_key_args_t *args)`
- `hsm_err_t hsm_manage_key_group (hsm_hdl_t key_management_hdl, op_manage_key_group_args_t *args)`
- `hsm_err_t hsm_butterfly_key_expansion (hsm_hdl_t key_management_hdl, op_butt_key_exp_args_t *args)`
- `hsm_err_t hsm_close_key_management_service (hsm_hdl_t key_management_hdl)`

### 6.4.1 Detailed Description

### 6.4.2 Function Documentation

#### 6.4.2.1 hsm\_open\_key\_management\_service()

```
hsm_err_t hsm_open_key_management_service (
    hsm_hdl_t key_store_hdl,
    open_svc_key_management_args_t * args,
    hsm_hdl_t * key_management_hdl )
```

Open a key management service flow

User must open this service flow in order to perform operation on the key store keys (generate, update, delete)

#### Parameters

<i>key_store_hdl</i>	handle indentifying the key store service flow.
<i>args</i>	pointer to the structure containing the function arugments.
<i>key_management_hdl</i>	pointer to where the key management service flow handle must be written.

#### Returns

error\_code error code.

#### 6.4.2.2 hsm\_generate\_key()

```
hsm_err_t hsm_generate_key (
    hsm_hdl_t key_management_hdl,
    op_generate_key_args_t * args )
```

Generate a key or a key pair. Only the confidential keys (symmetric and private keys) are stored in the internal key store, while the non-confidential keys (public key) are exported.

The generated key can be stored using a new or existing key identifier 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

<i>key_management_hdl</i>	handle identifying the key management service flow.
<i>args</i>	pointer to the structure containing the function arugments.

#### Returns

error code

#### 6.4.2.3 hsm\_manage\_key()

```
hsm_err_t hsm_manage_key (
    hsm_hdl_t key_management_hdl,
    op_manage_key_args_t * args )
```

This command is designed to perform the following operations:

- import a key creating a new key identifier
- import a key using an existing key identifier
- delete an existing key

User can call this function only after having opened a key management service flow

##### Parameters

<i>key_management_hdl</i>	handle identifying the key management service flow.
<i>args</i>	pointer to the structure containing the function arguments.

##### Returns

error code

#### 6.4.2.4 hsm\_manage\_key\_group()

```
hsm_err_t hsm_manage_key_group (
    hsm_hdl_t key_management_hdl,
    op_manage_key_group_args_t * args )
```

This command is designed to perform the following operations:

- lock/unlock down a key group in the HSM local memory so that the keys are available to the HSM without additional latency
- un-lock a key group. HSM may export the key group into the external NVM to free up local memory as needed
- delete an existing key group

User can call this function only after having opened a key management service flow

##### Parameters

<i>key_management_hdl</i>	handle identifying the key management service flow.
<i>args</i>	pointer to the structure containing the function arguments.



**Returns**

error code

**6.4.2.5 hsm\_butterfly\_key\_expansion()**

```
hsm_err_t hsm_butterfly_key_expansion (
    hsm_hdl_t key_management_hdl,
    op_butt_key_exp_args_t * args )
```

This command is designed to perform the butterfly key expansion operation on an ECC private key in case of implicit and explicit certificates. Optionally the resulting public key is exported.

The result of the key expansion function  $f_k$  is calculated outside the HSM and passed as input. The expansion function is defined as  $f_k = f_{k\_int} \bmod I$ , where  $I$  is the order of the group of points on the curve.

User can call this function only after having opened a key management service flow.

Explicit certificates:

- $f_k$  = expansion function value

$out\_key = Key + f_k$

Implicit certificates:

- $f_k$  = expansion function value,
- hash = hash value used to in the derivation of the pseudonym ECC key,
- $pr\_v$  = private reconstruction value

$out\_key = (Key + f_k) * hash + pr\_v$

**Parameters**

<i>key_management_hdl</i>	handle identifying the key store management service flow.
<i>args</i>	pointer to the structure containing the function arguments.

**Returns**

error code

**6.4.2.6 hsm\_close\_key\_management\_service()**

```
hsm_err_t hsm_close_key_management_service (
    hsm_hdl_t key_management_hdl )
```

Terminate a previously opened key management service flow

**Parameters**

<i>key_management_hdl</i>	handle identifying the key management service flow.
---------------------------	---

**Returns**

error code

## 6.5 Ciphering

### Data Structures

- struct [open\\_svc\\_cipher\\_args\\_t](#)
- struct [op\\_cipher\\_one\\_go\\_args\\_t](#)
- struct [hsm\\_op\\_ecies\\_dec\\_args\\_t](#)

### Macros

- #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)(0x04))  
*Perform AES CCM with following constraints: AES CCM where Adata = 0, Tlen = 16 bytes, nonce size = 12 bytes.*
- #define **HSM\_CIPHER\_ONE\_GO\_FLAGS\_DECRYPT** ((hsm\_op\_cipher\_one\_go\_flags\_t)(0 << 0))
- #define **HSM\_CIPHER\_ONE\_GO\_FLAGS\_ENCRYPT** ((hsm\_op\_cipher\_one\_go\_flags\_t)(1 << 0))

### Typedefs

- typedef uint8\_t **hsm\_svc\_cipher\_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\_ecies\_dec\_flags\_t**

### Functions

- [hsm\\_err\\_t hsm\\_open\\_cipher\\_service](#) (hsm\_hdl\_t key\_store\_hdl, [open\\_svc\\_cipher\\_args\\_t](#) \*args, hsm\_hdl\_t \*cipher\_hdl)
- [hsm\\_err\\_t hsm\\_cipher\\_one\\_go](#) (hsm\_hdl\_t cipher\_hdl, [op\\_cipher\\_one\\_go\\_args\\_t](#) \*args)
- [hsm\\_err\\_t hsm\\_ecies\\_decryption](#) (hsm\_hdl\_t cipher\_hdl, [hsm\\_op\\_ecies\\_dec\\_args\\_t](#) \*args)
- [hsm\\_err\\_t hsm\\_close\\_cipher\\_service](#) (hsm\_hdl\_t cipher\_hdl)

#### 6.5.1 Detailed Description

#### 6.5.2 Function Documentation

##### 6.5.2.1 hsm\_open\_cipher\_service()

```
hsm_err_t hsm_open_cipher_service (
    hsm_hdl_t key_store_hdl,
    open_svc_cipher_args_t * args,
    hsm_hdl_t * cipher_hdl )
```

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 operation

**Parameters**

<i>key_store_hdl</i>	handle indentifying the key store service flow.
<i>args</i>	pointer to the structure containing the function arugments.
<i>cipher_hdl</i>	pointer to where the cipher service flow handle must be written.

**Returns**

error code

**6.5.2.2 hsm\_cipher\_one\_go()**

```
hsm_err_t hsm_cipher_one_go (
    hsm_hdl_t cipher_hdl,
    op_cipher_one_go_args_t * args )
```

Perform ciphering operation

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

**Parameters**

<i>cipher_hdl</i>	handle identifying the cipher service flow.
<i>args</i>	pointer to the structure containing the function arugments.

**Returns**

error code

**6.5.2.3 hsm\_ecies\_decryption()**

```
hsm_err_t hsm_ecies_decryption (
    hsm_hdl_t cipher_hdl,
    hsm_op_ecies_dec_args_t * args )
```

Decrypt data usign ECIES

User can call this function only after having opened a cipher store service flow.

ECIES is supported with the constraints specified in 1609.2-2016.

**Parameters**

<i>session_hdl</i>	handle identifying the current session.
<i>args</i>	pointer to the structure containing the function arugments.

**Returns**

error code

#### 6.5.2.4 hsm\_close\_cipher\_service()

```
hsm_err_t hsm_close_cipher_service (  
    hsm_hdl_t cipher_hdl )
```

Terminate a previously opened cipher service flow

##### Parameters

<i>cipher_hdl</i>	pointer to handle identifying the cipher service flow to be closed.
-------------------	---

##### Returns

error code

## 6.6 Signature generation

### Data Structures

- struct [open\\_svc\\_sign\\_gen\\_args\\_t](#)
- struct [op\\_generate\\_sign\\_args\\_t](#)
- struct [op\\_prepare\\_sign\\_args\\_t](#)

### Macros

- `#define HSM_SIGNATURE_SCHEME_ECDSA_NIST_P224_SHA_256 ((hsm_signature_scheme_id_t)0x01)`  
*not supported*
- `#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)`  
*not supported*
- `#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)`  
*not supported*
- `#define HSM_SIGNATURE_SCHEME_ECDSA_BRAINPOOL_T1_256_SHA_256 ((hsm_signature_scheme_id_t)0x23)`  
*not supported*
- `#define HSM_SIGNATURE_SCHEME_ECDSA_BRAINPOOL_T1_384_SHA_384 ((hsm_signature_scheme_id_t)0x25)`  
*not supported*
- `#define HSM_OP_GENERATE_SIGN_FLAGS_INPUT_DIGEST ((hsm_op_generate_sign_flags_t)(0 << 0))`
- `#define HSM_OP_GENERATE_SIGN_FLAGS_INPUT_MESSAGE ((hsm_op_generate_sign_flags_t)(1 << 0))`
- `#define HSM_OP_GENERATE_SIGN_FLAGS_COMPRESSED_POINT ((hsm_op_generate_sign_flags_t)(1 << 1))`
- `#define HSM_OP_GENERATE_SIGN_FLAGS_LOW_LATENCY_SIGNATURE ((hsm_op_generate_sign_flags_t)(1 << 2))`
- `#define HSM_OP_PREPARE_SIGN_INPUT_DIGEST ((hsm_op_prepare_signature_flags_t)(0 << 0))`
- `#define HSM_OP_PREPARE_SIGN_INPUT_MESSAGE ((hsm_op_prepare_signature_flags_t)(1 << 0))`
- `#define HSM_OP_PREPARE_SIGN_COMPRESSED_POINT ((hsm_op_prepare_signature_flags_t)(1 << 1))`

### Typedefs

- typedef uint8\_t [hsm\\_svc\\_signature\\_generation\\_flags\\_t](#)
- typedef uint8\_t [hsm\\_signature\\_scheme\\_id\\_t](#)
- typedef uint8\_t [hsm\\_op\\_generate\\_sign\\_flags\\_t](#)
- typedef uint8\_t [hsm\\_op\\_prepare\\_signature\\_flags\\_t](#)

## Functions

- [hsm\\_err\\_t hsm\\_open\\_signature\\_generation\\_service](#) (hsm\_hdl\_t key\_store\_hdl, [open\\_svc\\_sign\\_gen\\_args\\_t](#) \*args, hsm\_hdl\_t \*signature\_gen\_hdl)
- [hsm\\_err\\_t hsm\\_close\\_signature\\_generation\\_service](#) (hsm\_hdl\_t signature\_gen\_hdl)
- [hsm\\_err\\_t hsm\\_generate\\_signature](#) (hsm\_hdl\_t signature\_gen\_hdl, [op\\_generate\\_sign\\_args\\_t](#) \*args)
- [hsm\\_err\\_t hsm\\_prepare\\_signature](#) (hsm\_hdl\_t signature\_gen\_hdl, [op\\_prepare\\_sign\\_args\\_t](#) \*args)

## 6.6.1 Detailed Description

## 6.6.2 Function Documentation

## 6.6.2.1 hsm\_open\_signature\_generation\_service()

```
hsm_err_t hsm_open_signature_generation_service (
    hsm_hdl_t key_store_hdl,
    open_svc_sign_gen_args_t * args,
    hsm_hdl_t * signature_gen_hdl )
```

Open a 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 signature generation operations.

## Parameters

<i>key_store_hdl</i>	handle indentifying the key store service flow.
<i>args</i>	pointer to the structure containing the function arugments.
<i>signature_gen_hdl</i>	pointer to where the signature generation service flow handle must be written.

## Returns

error code

## 6.6.2.2 hsm\_close\_signature\_generation\_service()

```
hsm_err_t hsm_close_signature_generation_service (
    hsm_hdl_t signature_gen_hdl )
```

Terminate a previously opened signature generation service flow

## Parameters

<i>signature_gen_hdl</i>	handle identifying the signature generation service flow to be closed.
--------------------------	--

**Returns**

error code

**6.6.2.3 hsm\_generate\_signature()**

```
hsm_err_t hsm_generate_signature (
    hsm_hdl_t signature_gen_hdl,
    op_generate_sign_args_t * args )
```

Generate a digital signature according to the signature scheme

User can call this function only after having opened a signature generation service flow

The signature S=(r,s) is stored in the format r||s||Ry where Ry is an additional byte containing the lsb of y. Ry has to be considered valid only if the HSM\_OP\_GENERATE\_SIGN\_FLAGS\_COMPRESSED\_POINT is set.

**Parameters**

<i>signature_gen_hdl</i>	handle identifying the signature generation service flow
<i>args</i>	pointer to the structure containing the function arguments.

**Returns**

error code

**6.6.2.4 hsm\_prepare\_signature()**

```
hsm_err_t hsm_prepare_signature (
    hsm_hdl_t signature_gen_hdl,
    op_prepare_sign_args_t * args )
```

Prepare the creation of a signature by pre-calculating the operations having not dependencies on the input message.

The pre-calculated value will be stored internally and used once call hsm\_generate\_signature

User can call this function only after having opened a signature generation service flow

The signature S=(r,s) is stored in the format r||s||Ry where Ry is an additional byte containing the lsb of y, Ry has to be considered valid only if the HSM\_OP\_PREPARE\_SIGN\_COMPRESSED\_POINT is set.

**Parameters**

<i>signature_gen_hdl</i>	handle identifying the signature generation service flow
<i>args</i>	pointer to the structure containing the function arguments.

**Returns**

error code



## 6.7 Signature verification

### Data Structures

- struct [open\\_svc\\_sign\\_ver\\_args\\_t](#)
- struct [op\\_verify\\_sign\\_args\\_t](#)
- struct [op\\_import\\_public\\_key\\_args\\_t](#)

### Macros

- `#define HSM_OP_VERIFY_SIGN_FLAGS_INPUT_DIGEST ((hsm_op_verify_sign_flags_t)(0 << 0))`
- `#define HSM_OP_VERIFY_SIGN_FLAGS_INPUT_MESSAGE ((hsm_op_verify_sign_flags_t)(1 << 0))`
- `#define HSM_OP_VERIFY_SIGN_FLAGS_COMPRESSED_POINT ((hsm_op_verify_sign_flags_t)(1 << 1))`
- `#define HSM_OP_VERIFY_SIGN_FLAGS_KEY_INTERNAL ((hsm_op_verify_sign_flags_t)(1 << 2))`  
*when set the value passed by the key argument is considered as the internal reference of a key imported through the hsm\_import\_pub\_key API.*
- `#define HSM_VERIFICATION_STATUS_SUCCESS ((hsm_verification_status_t)(0x5A3CC3A5))`

### Typedefs

- `typedef uint8_t hsm_svc_signature_verification_flags_t`
- `typedef uint8_t hsm_op_verify_sign_flags_t`
- `typedef uint32_t hsm_verification_status_t`
- `typedef uint8_t hsm_op_import_public_key_flags_t`

### Functions

- `hsm_err_t hsm_open_signature_verification_service (hsm_hdl_t session_hdl, open\_svc\_sign\_ver\_args\_t *args, hsm_hdl_t *signature_ver_hdl)`
- `hsm_err_t hsm_verify_signature (hsm_hdl_t signature_ver_hdl, op\_verify\_sign\_args\_t *args, hsm_hdl_t *signature_ver_hdl, hsm_verification_status_t *status)`
- `hsm_err_t hsm_import_public_key (hsm_hdl_t signature_ver_hdl, op\_import\_public\_key\_args\_t *args, uint32_t *key_ref)`
- `hsm_err_t hsm_close_signature_verification_service (hsm_hdl_t signature_ver_hdl)`

#### 6.7.1 Detailed Description

#### 6.7.2 Function Documentation

##### 6.7.2.1 hsm\_open\_signature\_verification\_service()

```
hsm_err_t hsm_open_signature_verification_service (
    hsm_hdl_t session_hdl,
    open\_svc\_sign\_ver\_args\_t * args,
    hsm_hdl_t * signature_ver_hdl )
```

User must open this service in order to perform signature verification operations.  
 User can call this function only after having opened a session.

**Parameters**

<i>session_hdl</i>	handle indentifying the current session.
<i>args</i>	pointer to the structure containing the function arugments.
<i>signature_ver_hdl</i>	pointer to where the signature verification service flow handle must be written.

**Returns**

error code

**6.7.2.2 hsm\_verify\_signature()**

```
hsm_err_t hsm_verify_signature (
    hsm_hdl_t signature_ver_hdl,
    op_verify_sign_args_t * args,
    hsm_verification_status_t * status )
```

Verify a digital signature according to the signature scheme

User can call this function only after having opened a signature verification service flow

The signature S=(r,s) is expected to be in format r||s||Ry where Ry is an additional byte containing the lsb of y. Ry will be considered as valid only if the HSM\_OP\_VERIFY\_SIGN\_FLAGS\_COMPRESSED\_POINT is set.

Only not-compressed keys (x,y) can be used by this command. Compressed keys can be decompressed by using the dedicated API.

**Parameters**

<i>signature_ver_hdl</i>	handle identifying the signature verification service flow.
<i>args</i>	pointer to the structure containing the function arugments.
<i>status</i>	pointer to where the verification status must be stored if the verification suceed the value HSM_VERIFICATION_STATUS_SUCCESS is returned.

**Returns**

error code

**6.7.2.3 hsm\_import\_public\_key()**

```
hsm_err_t hsm_import_public_key (
    hsm_hdl_t signature_ver_hdl,
    op_import_public_key_args_t * args,
    uint32_t * key_ref )
```

Import a public key to be used for several verification operations, a reference to the imported key is returned.

User can use the returned reference in the hsm\_verify\_signature API by setting the HSM\_OP\_VERIFY\_SIGN\_FLAGS\_KEY\_INTERNAL flag

Only not-compressed keys (x,y) can be imprted by this command. Compressed keys can be decompressed by using the dedicated API. User can call this function only after having opened a signature verification service flow.

## Parameters

<i>signature_ver_hdl</i>	handle identifying the signature verification service flow.
<i>args</i>	pointer to the structure containing the function arguments.
<i>key_ref</i>	pointer to where the 4 bytes key reference to be used as key in the hsm_verify_signature will be stored

## Returns

error code

## 6.7.2.4 hsm\_close\_signature\_verification\_service()

```
hsm_err_t hsm_close_signature_verification_service (  
    hsm_hdl_t signature_ver_hdl )
```

Terminate a previously opened signature verification service flow

## Parameters

<i>signature_ver_hdl</i>	handle identifying the signature verification service flow to be closed.
--------------------------	--

## Returns

error code

## 6.8 Random number generation

### Data Structures

- struct [open\\_svc\\_rng\\_args\\_t](#)
- struct [op\\_get\\_random\\_args\\_t](#)

### Typedefs

- typedef uint8\_t **hsm\_svc\_rng\_flags\_t**

### Functions

- [hsm\\_err\\_t hsm\\_open\\_rng\\_service](#) (hsm\_hdl\_t session\_hdl, [open\\_svc\\_rng\\_args\\_t](#) \*args, hsm\_hdl\_t \*rng\_hdl)
- [hsm\\_err\\_t hsm\\_close\\_rng\\_service](#) (hsm\_hdl\_t rng\_hdl)
- [hsm\\_err\\_t hsm\\_get\\_random](#) (hsm\_hdl\_t rng\_hdl, [op\\_get\\_random\\_args\\_t](#) \*args)

#### 6.8.1 Detailed Description

#### 6.8.2 Function Documentation

##### 6.8.2.1 hsm\_open\_rng\_service()

```
hsm_err_t hsm_open_rng_service (
    hsm_hdl_t session_hdl,
    open_svc_rng_args_t * args,
    hsm_hdl_t * rng_hdl )
```

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

<i>session_hdl</i>	handle indentifying the current session.
<i>args</i>	pointer to the structure containing the function arugments.
<i>rng_hdl</i>	pointer to where the rng service flow handle must be written.

#### Returns

error code

### 6.8.2.2 hsm\_close\_rng\_service()

```
hsm_err_t hsm_close_rng_service (
    hsm_hdl_t rng_hdl )
```

Terminate a previously opened rng service flow

#### Parameters

<i>rng_hdl</i>	handle identifying the rng service flow to be closed.
----------------	---

#### Returns

error code

### 6.8.2.3 hsm\_get\_random()

```
hsm_err_t hsm_get_random (
    hsm_hdl_t rng_hdl,
    op_get_random_args_t * args )
```

Get a freshly generated random number

User can call this function only after having opened a rng service flow

#### Parameters

<i>rng_hdl</i>	handle identifying the rng service flow.
<i>args</i>	pointer to the structure containing the function arguments.

#### Returns

error code

## 6.9 Hashing

### Data Structures

- struct [open\\_svc\\_hash\\_args\\_t](#)
- struct [op\\_hash\\_one\\_go\\_args\\_t](#)

### Macros

- #define [HSM\\_HASH\\_ALGO\\_SHA\\_224](#) ((hsm\_hash\_algo\_t)(0x0))  
*not supported*
- #define [HSM\\_HASH\\_ALGO\\_SHA\\_256](#) ((hsm\_hash\_algo\_t)(0x1))
- #define [HSM\\_HASH\\_ALGO\\_SHA\\_384](#) ((hsm\_hash\_algo\_t)(0x2))

### Typedefs

- typedef uint8\_t [hsm\\_svc\\_hash\\_flags\\_t](#)
- typedef uint8\_t [hsm\\_hash\\_algo\\_t](#)
- typedef uint8\_t [hsm\\_op\\_hash\\_one\\_go\\_flags\\_t](#)

### Functions

- [hsm\\_err\\_t hsm\\_open\\_hash\\_service](#) (hsm\_hdl\_t session\_hdl, [open\\_svc\\_hash\\_args\\_t](#) \*args, hsm\_hdl\_t \*hash\_hdl)
- [hsm\\_err\\_t hsm\\_close\\_hash\\_service](#) (hsm\_hdl\_t hash\_hdl)
- [hsm\\_err\\_t hsm\\_hash\\_one\\_go](#) (hsm\_hdl\_t hash\_hdl, [op\\_hash\\_one\\_go\\_args\\_t](#) \*args)

#### 6.9.1 Detailed Description

#### 6.9.2 Function Documentation

##### 6.9.2.1 hsm\_open\_hash\_service()

```
hsm_err_t hsm_open_hash_service (
    hsm_hdl_t session_hdl,
    open_svc_hash_args_t * args,
    hsm_hdl_t * hash_hdl )
```

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

<i>session_hdl</i>	handle indentifying the current session.
<i>args</i>	pointer to the structure containing the function arugments.
<i>hash_hdl</i>	pointer to where the hash service flow handle must be written.

**Returns**

error code

**6.9.2.2 hsm\_close\_hash\_service()**

```
hsm_err_t hsm_close_hash_service (
    hsm_hdl_t hash_hdl )
```

Terminate a previously opened hash service flow

**Parameters**

<i>hash_hdl</i>	handle identifying the hash service flow to be closed.
-----------------	--

**Returns**

error code

**6.9.2.3 hsm\_hash\_one\_go()**

```
hsm_err_t hsm_hash_one_go (
    hsm_hdl_t hash_hdl,
    op_hash_one_go_args_t * args )
```

Perform the hash operation on a given input

User can call this function only after having opened a hash service flow

**Parameters**

<i>hash_hdl</i>	handle identifying the hash service flow.
<i>args</i>	pointer to the structure containing the function arguments.

**Returns**

error code

## 6.10 Public key reconstruction

### Data Structures

- struct [hsm\\_op\\_pub\\_key\\_rec\\_args\\_t](#)

### Typedefs

- typedef uint8\_t [hsm\\_op\\_pub\\_key\\_rec\\_flags\\_t](#)

### Functions

- [hsm\\_err\\_t hsm\\_pub\\_key\\_reconstruction](#) (hsm\_hdl\_t session\_hdl, [hsm\\_op\\_pub\\_key\\_rec\\_args\\_t](#) \*args)

#### 6.10.1 Detailed Description

#### 6.10.2 Function Documentation

##### 6.10.2.1 hsm\_pub\_key\_reconstruction()

```
hsm\_err\_t hsm_pub_key_reconstruction (
    hsm_hdl_t session_hdl,
    hsm\_op\_pub\_key\_rec\_args\_t * args )
```

Reconstruct an ECC public key provided by an implicit certificate

User can call this function only after having opened a session

This API implements the followign formula:

$out\_key = (pub\_rec * hash) + ca\_key$

#### Parameters

<i>session_hdl</i>	handle identifying the current session.
<i>args</i>	pointer to the structure containing the function arugments.

#### Returns

error code



## 6.11 Public key decompression

### Data Structures

- struct [hsm\\_op\\_pub\\_key\\_dec\\_args\\_t](#)

### Typedefs

- typedef uint8\_t [hsm\\_op\\_pub\\_key\\_dec\\_flags\\_t](#)

### Functions

- [hsm\\_err\\_t hsm\\_pub\\_key\\_decompression](#) (hsm\_hdl\_t session\_hdl, [hsm\\_op\\_pub\\_key\\_dec\\_args\\_t](#) \*args)

#### 6.11.1 Detailed Description

#### 6.11.2 Function Documentation

##### 6.11.2.1 hsm\_pub\_key\_decompression()

```
hsm_err_t hsm_pub_key_decompression (
    hsm_hdl_t session_hdl,
    hsm_op_pub_key_dec_args_t * args )
```

Decompress an ECC public key

The expected key format is x||lsb\_y where lsb\_y is 1 byte having value 1 if the least-significant bit of the original (uncompressed) y coordinate is set, and 0 otherwise.

User can call this function only after having opened a session

#### Parameters

<i>session_hdl</i>	handle identifying the current session.
<i>args</i>	pointer to the structure containing the function arguments.

#### Returns

error code

## 6.12 ECIES encryption

### Data Structures

- struct [hsm\\_op\\_ecies\\_enc\\_args\\_t](#)

### Typedefs

- typedef uint8\_t [hsm\\_op\\_ecies\\_enc\\_flags\\_t](#)

### Functions

- [hsm\\_err\\_t hsm\\_ecies\\_encryption](#) (hsm\_hdl\_t session\_hdl, [hsm\\_op\\_ecies\\_enc\\_args\\_t](#) \*args)

#### 6.12.1 Detailed Description

#### 6.12.2 Function Documentation

##### 6.12.2.1 hsm\_ecies\_encryption()

```
hsm_err_t hsm_ecies_encryption (
    hsm_hdl_t session_hdl,
    hsm_op_ecies_enc_args_t * args )
```

#### Encrypt data usign ECIES

User can call this function only after having opened a session.

ECIES is supported with the constraints specified in 1609.2-2016.

#### Parameters

<i>session_hdl</i>	handle identifying the current session.
<i>args</i>	pointer to the structure containing the function arugments.

#### Returns

error code

## 6.13 Public key recovery

### Data Structures

- struct [hsm\\_op\\_pub\\_key\\_recovery\\_args\\_t](#)

### Typedefs

- typedef uint8\_t [hsm\\_op\\_pub\\_key\\_recovery\\_flags\\_t](#)

### Functions

- [hsm\\_err\\_t hsm\\_pub\\_key\\_recovery](#) (hsm\_hdl\_t key\_store\_hdl, [hsm\\_op\\_pub\\_key\\_recovery\\_args\\_t](#) \*args)

#### 6.13.1 Detailed Description

#### 6.13.2 Function Documentation

##### 6.13.2.1 hsm\_pub\_key\_recovery()

```
hsm_err_t hsm_pub_key_recovery (
    hsm_hdl_t key_store_hdl,
    hsm_op_pub_key_recovery_args_t * args )
```

Recover Public key from private key present in key store  
User can call this function only after having opened a key store.

#### Parameters

<i>key_store_hdl</i>	handle identifying the current key store.
<i>args</i>	pointer to the structure containing the function arguments.

#### Returns

error code

## 7 Data Structure Documentation

### 7.1 hsm\_op\_ecies\_dec\_args\_t Struct Reference

#### Data Fields

- `uint32_t key_identifier`  
*identifier of the private key to be used for the operation*
- `uint8_t * input`  
*pointer to the VCT input*
- `uint8_t * p1`  
*pointer to the KDF P1 input parameter*
- `uint8_t * p2`  
*pointer to the MAC P2 input parameter should be NULL*
- `uint8_t * output`  
*pointer to the output area where the plaintext must be written*
- `uint32_t input_size`  
*length in bytes of the input VCT should be equal to 96 bytes*
- `uint32_t output_size`  
*length in bytes of the output plaintext should be equal to 16 bytes*
- `uint16_t p1_size`  
*length in bytes of the KDF P1 parameter should be equal to 32 bytes*
- `uint16_t p2_size`  
*length in bytes of the MAC P2 parameter should be zero reserved for generic use cases*
- `uint16_t mac_size`  
*length in bytes of the requested message authentication code should be equal to 16 bytes*
- `hsm_key_type_t key_type`  
*indicates the type of the used key (only NIST P256 and Br256r1 are supported)*
- `hsm_op_ecies_dec_flags_t flags`  
*bitmap specifying the operation attributes.*

### 7.2 hsm\_op\_ecies\_enc\_args\_t Struct Reference

#### Data Fields

- `uint8_t * input`  
*pointer to the input plaintext*
- `uint8_t * pub_key`  
*pointer to the input recipient public key*
- `uint8_t * p1`  
*pointer to the KDF P1 input parameter*
- `uint8_t * p2`  
*pointer to the MAC P2 input parameter should be NULL*
- `uint8_t * output`  
*pointer to the output area where the VCT must be written*
- `uint32_t input_size`  
*length in bytes of the input plaintext should be equal to 16 bytes*
- `uint16_t p1_size`  
*length in bytes of the KDF P1 parameter should be equal to 32 bytes*

- uint16\_t [p2\\_size](#)  
*length in bytes of the MAC P2 parameter should be zero reserved for generic use cases*
- uint16\_t [pub\\_key\\_size](#)  
*length in bytes of the recipient public key should be equal to 64 bytes*
- uint16\_t [mac\\_size](#)  
*length in bytes of the requested message authentication code should be equal to 16 bytes*
- uint32\_t [out\\_size](#)  
*length in bytes of the output VCT should be equal to 96 bytes*
- hsm\_key\_type\_t [key\\_type](#)  
*indicates the type of the recipient public key (only NIST P256 and Br256r1 are supported)*
- hsm\_op\_ecies\_enc\_flags\_t [flags](#)  
*bitmap specifying the operation attributes.*
- uint16\_t **reserved**

### 7.3 hsm\_op\_pub\_key\_dec\_args\_t Struct Reference

#### Data Fields

- uint8\_t \* [key](#)  
*pointer to the compressed ECC public key. The expected key format is x||lsb\_y where lsb\_y is 1 byte having value 1 if the least-significant bit of the original (uncompressed) y coordinate is set, and 0 otherwise.*
- uint8\_t \* [out\\_key](#)  
*pointer to the output area where the decompressed public key must be written.*
- uint16\_t [key\\_size](#)  
*length in bytes of the input compressed public key*
- uint16\_t [out\\_key\\_size](#)  
*length in bytes of the resulting public key*
- hsm\_key\_type\_t [key\\_type](#)  
*indicates the type of the manged keys.*
- hsm\_op\_pub\_key\_dec\_flags\_t [flags](#)  
*bitmap specifying the operation attributes.*
- uint16\_t **reserved**

### 7.4 hsm\_op\_pub\_key\_rec\_args\_t Struct Reference

#### Data Fields

- uint8\_t \* [pub\\_rec](#)  
*pointer to the public reconstruction value extracted from the implicit certificate.*
- uint8\_t \* [hash](#)  
*pointer to the input hash value. In the butterfly scheme it corresponds to the hash value calculated over PCA certificate and, concatenated, the implicit certificat.*
- uint8\_t \* [ca\\_key](#)  
*pointer to the CA public key*
- uint8\_t \* [out\\_key](#)  
*pointer to the output area where the reconstructed public key must be written.*
- uint16\_t [pub\\_rec\\_size](#)  
*length in bytes of the public reconstruction value*
- uint16\_t [hash\\_size](#)

- `length` in bytes of the input hash
- `uint16_t` [ca\\_key\\_size](#)  
*length in bytes of the input CA public key*
- `uint16_t` [out\\_key\\_size](#)  
*length in bytes of the output key*
- `hsm_key_type_t` [key\\_type](#)  
*indicates the type of the manged keys.*
- `hsm_op_pub_key_rec_flags_t` [flags](#)  
*flags bitmap specifying the operation attributes.*
- `uint16_t` **reserved**

## 7.5 hsm\_op\_pub\_key\_recovery\_args\_t Struct Reference

### Data Fields

- `uint32_t` [key\\_identifier](#)  
*pointer to the identifier of the key to be used for the operation*
- `uint8_t *` [out\\_key](#)  
*pointer to the output area where the generated public key must be written*
- `uint16_t` [out\\_key\\_size](#)  
*length in bytes of the output key*
- `hsm_key_type_t` [key\\_type](#)  
*indicates the type of the key to be recovered*
- `hsm_op_pub_key_recovery_flags_t` [flags](#)  
*bitmap specifying the operation attributes.*

## 7.6 op\_but\_key\_exp\_args\_t Struct Reference

### Data Fields

- `uint32_t` [key\\_identifier](#)  
*identifier of the key to be expanded*
- `uint8_t *` [expansion\\_function\\_value](#)  
*pointer to the expansion function value input*
- `uint8_t *` [hash\\_value](#)  
*pointer to the hash value input.  
In case of explicit certificate, the hash value address must be set to 0.*
- `uint8_t *` [pr\\_reconstruction\\_value](#)  
*pointer to the private reconstruction value input.  
In case of explicit certificate, the pr\_reconstruction\_value address must be set to 0.*
- `uint8_t` [expansion\\_function\\_value\\_size](#)  
*length in bytes of the expansion function input*
- `uint8_t` [hash\\_value\\_size](#)  
*length in bytes of the hash value input.  
In case of explicit certificate, the hash\_value\_size parameter must be set to 0.*
- `uint8_t` [pr\\_reconstruction\\_value\\_size](#)  
*length in bytes of the private reconstruction value input.  
In case of explicit certificate, the pr\_reconstruction\_value\_size parameter must be set to 0.*
- `hsm_op_but_key_exp_flags_t` [flags](#)  
*bitmap specifying the operation properties*

- uint32\_t \* [dest\\_key\\_identifier](#)  
*pointer to identifier of the derived key to be used for the operation.  
In case of create operation the new destination key identifier will be stored in this location.*
- uint8\_t \* [output](#)  
*pointer to the output area where the public key must be written.*
- uint16\_t [output\\_size](#)  
*length in bytes of the generated key, if the size is 0, no key is copied in the output.*
- hsm\_key\_type\_t [key\\_type](#)  
*indicates the type of the key to be managed.*
- uint8\_t **reserved**
- hsm\_key\_group\_t [key\\_group](#)  
*it must be a value in the range 0-1023. Keys belonging to the same group can be cached in the HSM local memory through the ham\_manage\_key\_group API*
- hsm\_key\_info\_t [key\\_info](#)  
*bitmap specifying the properties of the derived key.*

## 7.7 op\_cipher\_one\_go\_args\_t Struct Reference

### Data Fields

- uint32\_t [key\\_identifier](#)  
*identifier of the key to be used for the operation*
- uint8\_t \* [iv](#)  
*pointer to the initialization vector (nonce in case of AES CCM)*
- uint16\_t [iv\\_size](#)  
*length in bytes of the initialization vector  
it must be 0 for algorithms not using the initialization vector.  
It must be 12 for AES in CCM mode*
- hsm\_op\_cipher\_one\_go\_algo\_t [cipher\\_algo](#)  
*algorithm to be used for the operation*
- hsm\_op\_cipher\_one\_go\_flags\_t [flags](#)  
*bitmap specifying the operation attributes*
- uint8\_t \* [input](#)  
*pointer to the input area  
plaintext for encryption  
ciphertext for decryption (in case of CCM is the purported ciphertext)*
- uint8\_t \* [output](#)  
*pointer to the output area  
ciphertext for encryption (in case of CCM is the output of the generation-encryption process)  
plaintext for decryption*
- uint32\_t [input\\_size](#)  
*length in bytes of the input*
- uint32\_t [output\\_size](#)  
*length in bytes of the output*

## 7.8 op\_generate\_key\_args\_t Struct Reference

### Data Fields

- uint32\_t \* [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.*
- uint16\_t [out\\_size](#)  
*length in bytes of the generated key. It must be 0 in case of symmetric keys.*
- hsm\_op\_key\_gen\_flags\_t [flags](#)  
*bitmap specifying the operation properties.*
- hsm\_key\_type\_t [key\\_type](#)  
*indicates which type of key must be generated.*
- hsm\_key\_group\_t [key\\_group](#)  
*Key group of the generated key, only needed in case of create operation. it must be a value in the range 0-1023. Keys belonging to the same group can be cached in the HSM local memory through the ham\_manage\_key\_group API.*
- hsm\_key\_info\_t [key\\_info](#)  
*bitmap specifying the properties of the key.*
- uint8\_t \* [out\\_key](#)  
*pointer to the output area where the generated public key must be written*

## 7.9 op\_generate\_sign\_args\_t Struct Reference

### Data Fields

- uint32\_t [key\\_identifier](#)  
*identifier of the key to be used for the operation*
- uint8\_t \* [message](#)  
*pointer to the input (message or message digest) to be signed*
- uint8\_t \* [signature](#)  
*pointer to the output area where the signature must be stored. The signature  $S=(r,s)$  is stored in format  $r||s||R_y$  where  $R_y$  is an additional byte containing the lsb of  $y$ .  $R_y$  has to be considered valid only if the `HSM_OP_GENERATE_SIGN_FLAGS_COMPRESSED_POINT` is set.*
- uint32\_t [message\\_size](#)  
*length in bytes of the input*
- uint16\_t [signature\\_size](#)  
*length in bytes of the output*
- hsm\_signature\_scheme\_id\_t [scheme\\_id](#)  
*identifier of the digital signature scheme to be used for the operation*
- hsm\_op\_generate\_sign\_flags\_t [flags](#)  
*bitmap specifying the operation attributes*

## 7.10 op\_get\_random\_args\_t Struct Reference

### Data Fields

- uint8\_t \* [output](#)  
*pointer to the output area where the random number must be written*
- uint32\_t [random\\_size](#)  
*length in bytes of the random number to be provided.*



## 7.11 op\_hash\_one\_go\_args\_t Struct Reference

### Data Fields

- uint8\_t \* [input](#)  
*pointer to the input data to be hashed*
- uint8\_t \* [output](#)  
*pointer to the output area where the resulting digest must be written*
- uint32\_t [input\\_size](#)  
*length in bytes of the input*
- uint32\_t [output\\_size](#)  
*length in bytes of the output*
- hsm\_hash\_algo\_t [algo](#)  
*hash algorithm to be used for the operation*
- hsm\_op\_hash\_one\_go\_flags\_t [flags](#)  
*flags bitmap specifying the operation attributes.*
- uint16\_t **reserved**

## 7.12 op\_import\_public\_key\_args\_t Struct Reference

### Data Fields

- uint8\_t \* [key](#)  
*pointer to the public key to be imported*
- uint16\_t [key\\_size](#)  
*length in bytes of the input key*
- hsm\_key\_type\_t [key\\_type](#)  
*indicates the type of the key to be imported.*
- hsm\_op\_import\_public\_key\_flags\_t [flags](#)  
*bitmap specifying the operation attributes*

## 7.13 op\_manage\_key\_args\_t Struct Reference

### Data Fields

- uint32\_t \* [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.*
- uint32\_t [kek\\_identifier](#)  
*identifier of the key to be used to decrypt the imported key (key encryption key)*
- uint16\_t [input\\_size](#)  
*length in bytes of the input key area. It must be 0 in case of delete operation.*
- hsm\_op\_manage\_key\_flags\_t [flags](#)  
*bitmap specifying the operation properties.*
- hsm\_key\_type\_t [key\\_type](#)  
*indicates the type of the key to be managed.*
- hsm\_key\_group\_t [key\\_group](#)  
*key group of the imported key, only relevant in case of create operation (it must be 0 otherwise). It must be a value in the range 0-1023. Keys belonging to the same group can be cached in the HSM local memory through the ham\_↔manage\_key\_group API*
- hsm\_key\_info\_t [key\\_info](#)  
*bitmap specifying the properties of the key, in case of update operation it will replace the existing value. It must be 0 in case of delete operation.*
- uint8\_t \* [input\\_key](#)  
*pointer to the key to be imported. It must be 0 in case of delete operation.*

## 7.14 op\_manage\_key\_group\_args\_t Struct Reference

### Data Fields

- hsm\_key\_group\_t [key\\_group](#)  
*it must be a value in the range 0-1023. Keys belonging to the same group can be cached in the HSM local memory through the ham\_manage\_key\_group API*
- hsm\_op\_manage\_key\_group\_flags\_t [flags](#)  
*bitmap specifying the operation properties.*
- uint8\_t **reserved**

## 7.15 op\_prepare\_sign\_args\_t Struct Reference

### Data Fields

- hsm\_signature\_scheme\_id\_t [scheme\\_id](#)  
*identifier of the digital signature scheme to be used for the operation*
- hsm\_op\_prepare\_signature\_flags\_t [flags](#)  
*bitmap specifying the operation attributes*
- uint16\_t **reserved**

## 7.16 op\_verify\_sign\_args\_t Struct Reference

### Data Fields

- uint8\_t \* [key](#)  
*pointer to the public key to be used for the verification. If the HSM\_OP\_VERIFY\_SIGN\_FLAGS\_KEY\_INTERNAL is set, it must point to the key reference returned by the hsm\_import\_public\_key API.*
- uint8\_t \* [message](#)  
*pointer to the input (message or message digest)*
- uint8\_t \* [signature](#)  
*pointer to the input signature. The signature  $S=(r,s)$  is expected to be in the format  $r||s||R_y$  where  $R_y$  is an additional byte containing the lsb of  $y$ .  $R_y$  will be considered as valid only if the HSM\_OP\_VERIFY\_SIGN\_FLAGS\_COMPRESSED\_POINT is set.*
- uint16\_t [key\\_size](#)  
*length in bytes of the input key*
- uint16\_t [signature\\_size](#)  
*length in bytes of the output - it must contains one additional byte where to store the  $R_y$ .*
- uint32\_t [message\\_size](#)  
*length in bytes of the input message*
- hsm\_signature\_scheme\_id\_t [scheme\\_id](#)  
*identifier of the digital signature scheme to be used for the operation*
- hsm\_op\_verify\_sign\_flags\_t [flags](#)  
*bitmap specifying the operation attributes*
- uint16\_t **reserved**

## 7.17 open\_session\_args\_t Struct Reference

### Data Fields

- uint8\_t [session\\_priority](#)  
*not supported in current release, any value accepted. \*/*
- uint8\_t [operating\\_mode](#)  
*not supported in current release, any value accepted. \*/*
- uint16\_t **reserved**

## 7.18 open\_svc\_cipher\_args\_t Struct Reference

### Data Fields

- hsm\_svc\_cipher\_flags\_t [flags](#)  
*bitmap specifying the services properties.*
- uint8\_t **reserved** [3]

## 7.19 open\_svc\_hash\_args\_t Struct Reference

### Data Fields

- hsm\_svc\_hash\_flags\_t [flags](#)  
*bitmap indicating the service flow properties*
- uint8\_t **reserved** [3]

## 7.20 open\_svc\_key\_management\_args\_t Struct Reference

### Data Fields

- hsm\_svc\_key\_management\_flags\_t [flags](#)  
*bitmap specifying the services properties.*
- uint8\_t **reserved** [3]

## 7.21 open\_svc\_key\_store\_args\_t Struct Reference

### Data Fields

- uint32\_t [key\\_store\\_identifier](#)  
*user defined id identifying the key store.\*/*
- uint32\_t [authentication\\_nonce](#)  
*user defined nonce used as authentication proof for accesing the key store. \*/*
- uint16\_t [max\\_updates\\_number](#)  
*maximum number of updates authorized for the key store. Valid only for create operation. \*/*
- hsm\_svc\_key\_store\_flags\_t [flags](#)  
*bitmap specifying the services properties. \*/*
- uint8\_t **reserved**

## 7.22 open\_svc\_rng\_args\_t Struct Reference

### Data Fields

- hsm\_svc\_rng\_flags\_t [flags](#)  
*bitmap indicating the service flow properties*
- uint8\_t **reserved** [3]

## 7.23 open\_svc\_sign\_gen\_args\_t Struct Reference

### Data Fields

- hsm\_svc\_signature\_generation\_flags\_t [flags](#)  
*bitmap specifying the services properties.*
- uint8\_t **reserved** [3]

## 7.24 open\_svc\_sign\_ver\_args\_t Struct Reference

### Data Fields

- hsm\_svc\_signature\_verification\_flags\_t [flags](#)  
*bitmap indicating the service flow properties*
- uint8\_t **reserved** [3]

## Index

- Ciphering, [17](#)
  - `hsm_cipher_one_go`, [18](#)
  - `hsm_close_cipher_service`, [19](#)
  - `hsm_ecies_decryption`, [18](#)
  - `hsm_open_cipher_service`, [17](#)
- ECIES encryption, [32](#)
  - `hsm_ecies_encryption`, [32](#)
- Error codes, [5](#)
  - `HSM_CMD_NOT_SUPPORTED`, [6](#)
  - `hsm_err_t`, [5](#)
  - `HSM_FEATURE_NOT_SUPPORTED`, [6](#)
  - `HSM_GENERAL_ERROR`, [6](#)
  - `HSM_ID_CONFLICT`, [6](#)
  - `HSM_INVALID_ADDRESS`, [6](#)
  - `HSM_INVALID_LIFECYCLE`, [6](#)
  - `HSM_INVALID_MESSAGE`, [6](#)
  - `HSM_INVALID_PARAM`, [6](#)
  - `HSM_KEY_STORE_AUTH`, [6](#)
  - `HSM_KEY_STORE_CONFLICT`, [6](#)
  - `HSM_KEY_STORE_COUNTER`, [6](#)
  - `HSM_KEY_STORE_ERROR`, [6](#)
  - `HSM_NO_ERROR`, [6](#)
  - `HSM_NVM_ERROR`, [6](#)
  - `HSM_OUT_OF_MEMORY`, [6](#)
  - `HSM_RNG_NOT_STARTED`, [6](#)
  - `HSM_UNKNOWN_HANDLE`, [6](#)
  - `HSM_UNKNOWN_ID`, [6](#)
  - `HSM_UNKNOWN_KEY_STORE`, [6](#)
- Hashing, [28](#)
  - `hsm_close_hash_service`, [29](#)
  - `hsm_hash_one_go`, [29](#)
  - `hsm_open_hash_service`, [28](#)
- `hsm_butterfly_key_expansion`
  - Key management, [15](#)
- `hsm_cipher_one_go`
  - Ciphering, [18](#)
- `hsm_close_cipher_service`
  - Ciphering, [19](#)
- `hsm_close_hash_service`
  - Hashing, [29](#)
- `hsm_close_key_management_service`
  - Key management, [15](#)
- `hsm_close_key_store_service`
  - Key store, [10](#)
- `hsm_close_rng_service`
  - Random number generation, [26](#)
- `hsm_close_session`
  - Session, [7](#)
- `hsm_close_signature_generation_service`
  - Signature generation, [21](#)
- `hsm_close_signature_verification_service`
  - Signature verification, [25](#)
- `HSM_CMD_NOT_SUPPORTED`
  - Error codes, [6](#)
- `hsm_ecies_decryption`
  - Ciphering, [18](#)
- `hsm_ecies_encryption`
  - ECIES encryption, [32](#)
- `hsm_err_t`
  - Error codes, [5](#)
- `HSM_FEATURE_NOT_SUPPORTED`
  - Error codes, [6](#)
- `HSM_GENERAL_ERROR`
  - Error codes, [6](#)
- `hsm_generate_key`
  - Key management, [13](#)
- `hsm_generate_signature`
  - Signature generation, [22](#)
- `hsm_get_random`
  - Random number generation, [27](#)
- `hsm_hash_one_go`
  - Hashing, [29](#)
- `HSM_ID_CONFLICT`
  - Error codes, [6](#)
- `hsm_import_public_key`
  - Signature verification, [24](#)
- `HSM_INVALID_ADDRESS`
  - Error codes, [6](#)
- `HSM_INVALID_LIFECYCLE`
  - Error codes, [6](#)
- `HSM_INVALID_MESSAGE`
  - Error codes, [6](#)
- `HSM_INVALID_PARAM`
  - Error codes, [6](#)
- `HSM_KEY_STORE_AUTH`
  - Error codes, [6](#)
- `HSM_KEY_STORE_CONFLICT`
  - Error codes, [6](#)
- `HSM_KEY_STORE_COUNTER`
  - Error codes, [6](#)
- `HSM_KEY_STORE_ERROR`
  - Error codes, [6](#)
- `hsm_manage_key`
  - Key management, [13](#)
- `hsm_manage_key_group`
  - Key management, [14](#)
- `HSM_NO_ERROR`
  - Error codes, [6](#)
- `HSM_NVM_ERROR`
  - Error codes, [6](#)
- `hsm_op_ecies_dec_args_t`, [34](#)
- `hsm_op_ecies_enc_args_t`, [34](#)
- `hsm_op_pub_key_dec_args_t`, [35](#)
- `hsm_op_pub_key_rec_args_t`, [35](#)
- `hsm_op_pub_key_recovery_args_t`, [36](#)
- `hsm_open_cipher_service`
  - Ciphering, [17](#)
- `hsm_open_hash_service`
  - Hashing, [28](#)

- hsm\_open\_key\_management\_service
  - Key management, 13
- hsm\_open\_key\_store\_service
  - Key store, 9
- hsm\_open\_rng\_service
  - Random number generation, 26
- hsm\_open\_session
  - Session, 7
- hsm\_open\_signature\_generation\_service
  - Signature generation, 21
- hsm\_open\_signature\_verification\_service
  - Signature verification, 23
- HSM\_OUT\_OF\_MEMORY
  - Error codes, 6
- hsm\_prepare\_signature
  - Signature generation, 22
- hsm\_pub\_key\_decompression
  - Public key decompression, 31
- hsm\_pub\_key\_reconstruction
  - Public key reconstruction, 30
- hsm\_pub\_key\_recovery
  - Public key recovery, 33
- HSM\_RNG\_NOT\_STARTED
  - Error codes, 6
- HSM\_UNKNOWN\_HANDLE
  - Error codes, 6
- HSM\_UNKNOWN\_ID
  - Error codes, 6
- HSM\_UNKNOWN\_KEY\_STORE
  - Error codes, 6
- hsm\_verify\_signature
  - Signature verification, 24
- Key management, 11
  - hsm\_butterfly\_key\_expansion, 15
  - hsm\_close\_key\_management\_service, 15
  - hsm\_generate\_key, 13
  - hsm\_manage\_key, 13
  - hsm\_manage\_key\_group, 14
  - hsm\_open\_key\_management\_service, 13
- Key store, 9
  - hsm\_close\_key\_store\_service, 10
  - hsm\_open\_key\_store\_service, 9
- op\_but\_key\_exp\_args\_t, 36
- op\_cipher\_one\_go\_args\_t, 37
- op\_generate\_key\_args\_t, 38
- op\_generate\_sign\_args\_t, 38
- op\_get\_random\_args\_t, 38
- op\_hash\_one\_go\_args\_t, 39
- op\_import\_public\_key\_args\_t, 39
- op\_manage\_key\_args\_t, 39
- op\_manage\_key\_group\_args\_t, 40
- op\_prepare\_sign\_args\_t, 40
- op\_verify\_sign\_args\_t, 40
- open\_session\_args\_t, 41
- open\_svc\_cipher\_args\_t, 41
- open\_svc\_hash\_args\_t, 41
- open\_svc\_key\_management\_args\_t, 41
- open\_svc\_key\_store\_args\_t, 41
- open\_svc\_rng\_args\_t, 42
- open\_svc\_sign\_gen\_args\_t, 42
- open\_svc\_sign\_ver\_args\_t, 42
- Public key decompression, 31
  - hsm\_pub\_key\_decompression, 31
- Public key reconstruction, 30
  - hsm\_pub\_key\_reconstruction, 30
- Public key recovery, 33
  - hsm\_pub\_key\_recovery, 33
- Random number generation, 26
  - hsm\_close\_rng\_service, 26
  - hsm\_get\_random, 27
  - hsm\_open\_rng\_service, 26
- Session, 7
  - hsm\_close\_session, 7
  - hsm\_open\_session, 7
- Signature generation, 20
  - hsm\_close\_signature\_generation\_service, 21
  - hsm\_generate\_signature, 22
  - hsm\_open\_signature\_generation\_service, 21
  - hsm\_prepare\_signature, 22
- Signature verification, 23
  - hsm\_close\_signature\_verification\_service, 25
  - hsm\_import\_public\_key, 24
  - hsm\_open\_signature\_verification\_service, 23
  - hsm\_verify\_signature, 24