

i.MX8 HSM API

Revision_1.0

Generated by Doxygen 1.8.15

1 HSM API	1
2 Revision History	1
3 General concepts related to the API	2
3.1 Session	2
3.2 Service flow	2
4 Module Index	2
4.1 Modules	2
5 Data Structure Index	3
5.1 Data Structures	3
6 Module Documentation	4
6.1 Error codes	4
6.1.1 Detailed Description	4
6.1.2 Enumeration Type Documentation	4
6.2 Session	6
6.2.1 Detailed Description	6
6.2.2 Function Documentation	6
6.3 Key store	8
6.3.1 Detailed Description	8
6.3.2 Function Documentation	8
6.4 Key management	10
6.4.1 Detailed Description	11
6.4.2 Function Documentation	11
6.5 Cipherring	14
6.5.1 Detailed Description	14
6.5.2 Function Documentation	14
6.6 Signature generation	17
6.6.1 Detailed Description	18
6.6.2 Function Documentation	18
6.7 Signature verification	21
6.7.1 Detailed Description	21
6.7.2 Function Documentation	21
6.8 Random number generation	24
6.8.1 Detailed Description	24
6.8.2 Function Documentation	24
6.9 Hashing	26
6.9.1 Detailed Description	26
6.9.2 Function Documentation	26
6.10 Public key reconstruction	28
6.10.1 Detailed Description	28

6.10.2 Function Documentation	28
6.11 Public key decompression	29
6.11.1 Detailed Description	29
6.11.2 Function Documentation	29
6.12 ECIES encryption	30
6.12.1 Detailed Description	30
6.12.2 Function Documentation	30
7 Data Structure Documentation	31
7.1 hsm_op_ecies_dec_args_t Struct Reference	31
7.2 hsm_op_ecies_enc_args_t Struct Reference	31
7.3 hsm_op_pub_key_dec_args_t Struct Reference	32
7.4 hsm_op_pub_key_rec_args_t Struct Reference	32
7.5 op_butl_key_exp_args_t Struct Reference	33
7.6 op_cipher_one_go_args_t Struct Reference	33
7.7 op_finalize_sign_args_t Struct Reference	34
7.8 op_generate_key_args_t Struct Reference	34
7.9 op_generate_sign_args_t Struct Reference	35
7.10 op_get_random_args_t Struct Reference	35
7.11 op_hash_one_go_args_t Struct Reference	36
7.12 op_import_public_key_args_t Struct Reference	36
7.13 op_manage_key_args_t Struct Reference	36
7.14 op_prepare_sign_args_t Struct Reference	37
7.15 op_verify_sign_args_t Struct Reference	37
7.16 open_session_args_t Struct Reference	37
7.17 open_svc_cipher_args_t Struct Reference	37
7.18 open_svc_hash_args_t Struct Reference	38
7.19 open_svc_key_management_args_t Struct Reference	38
7.20 open_svc_key_store_args_t Struct Reference	38
7.21 open_svc_rng_args_t Struct Reference	38
7.22 open_svc_sign_gen_args_t Struct Reference	38
7.23 open_svc_sign_ver_args_t Struct Reference	38
Index	39

1 HSM API

This document is a software referece 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 |

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. 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 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-sequent operations requested by the user on the service flow.

4 Module Index

4.1 Modules

Here is a list of all modules:

Error codes	4
Session	6
Key store	8
Key management	10
Ciphering	14

Signature generation	17
Signature verification	21
Random number generation	24
Hashing	26
Public key reconstruction	28
Public key decompression	29
ECIES encryption	30

5 Data Structure Index

5.1 Data Structures

Here are the data structures with brief descriptions:

hsm_op_ecies_dec_args_t	31
hsm_op_ecies_enc_args_t	31
hsm_op_pub_key_dec_args_t	32
hsm_op_pub_key_rec_args_t	32
op_butk_key_exp_args_t	33
op_cipher_one_go_args_t	33
op_finalize_sign_args_t	34
op_generate_key_args_t	34
op_generate_sign_args_t	35
op_get_random_args_t	35
op_hash_one_go_args_t	36
op_import_public_key_args_t	36
op_manage_key_args_t	36
op_prepare_sign_args_t	37
op_verify_sign_args_t	37
open_session_args_t	37
open_svc_cipher_args_t	37
open_svc_hash_args_t	38
open_svc_key_management_args_t	38
open_svc_key_store_args_t	38

open_svc_rng_args_t	38
open_svc_sign_gen_args_t	38
open_svc_sign_ver_args_t	38

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_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

<code>HSM_NO_ERROR</code>	Success.
<code>HSM_INVALID_MESSAGE</code>	The received message is invalid or unknown.
<code>HSM_INVALID_ADDRESS</code>	The provided address is invalid or doesn't respect the API requirements.
<code>HSM_UNKNOWN_ID</code>	The provided identifier is not known.
<code>HSM_INVALID_PARAM</code>	One of the parameter provided in the command is invalid.
<code>HSM_NVM_ERROR</code>	NVM generic issue.

Enumerator

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_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

<code>session_hdl</code>	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 << 1))`
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))`
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_but_key_exp_args_t](#)

Macros

- `#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_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_PERSISTENT ((hsm_op_key_gen_flags_t)(1 << 1))`
Persistent keys are saved in the non volatile memory.
- `#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.
- `#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. This bit can never be reset.
- `#define HSM_OP_MANAGE_KEY_FLAGS_UPDATE ((hsm_op_manage_key_flags_t)(1 << 0))`
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_PERSISTENT ((hsm_op_manage_key_flags_t)(1 << 1))`
Persistent keys are saved in the non volatile memory.
- `#define HSM_OP_MANAGE_KEY_FLAGS_CREATE_TRANSIENT ((hsm_op_manage_key_flags_t)(1 << 2))`
Transient keys are deleted when the corresponding key store service flow is closed.
- `#define HSM_OP_MANAGE_KEY_FLAGS_DELETE ((hsm_op_manage_key_flags_t)(1 << 3))`
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.

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_ext_t**
- typedef uint8_t **hsm_key_type_t**
- typedef uint16_t **hsm_key_info_t**
- typedef uint8_t **hsm_op_manage_key_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_butterfly_key_expansion](#) (hsm_hdl_t key_management_hdl, [op_but_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 arguments.

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_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 certificate. Optionally the resulting public key is exported.

The result of the key expansion function $f1/f2$ is calculated outside the HSM and passed as input. User can call this function only after having opened a key management service flow.

The following operation is performed:

$$\text{out_key} = (\text{Key} + \text{data1}) * \text{data2} + \text{data3} \pmod{n}$$

Explicit certificates:

- $\text{data1} = 0$,
- $\text{data2} = 1$
- $\text{data3} = f1/f2(k, i, j)$

$$\text{out_key} = \text{Key} + f1/f2(k, i, j) \pmod{n}$$

Implicit certificates:

- $\text{data1} = f1(k, i, j)$,
- data2 = hash value used to in the derivation of the pseudonym ECC key,
- data3 = private reconstruction value p_{ij}

$$\text{out_key} = (\text{Key} + f1(k, i, j)) * \text{Hash} + p_{ij}$$

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

6.4.2.5 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_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))

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 cipherring 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

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](#)
- struct [op_finalize_sign_args_t](#)

Macros

- `#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_GENERATE_SIGN_FLAGS_INPUT_DIGEST ((hsm_op_generate_sign_flags_t)(1 << 0))`
- `#define HSM_OP_GENERATE_SIGN_FLAGS_INPUT_MESSAGE ((hsm_op_generate_sign_flags_t)(1 << 1))`
- `#define HSM_OP_GENERATE_SIGN_FLAGS_COMPRESSED_POINT ((hsm_op_generate_sign_flags_t)(1 << 2))`
- `#define HSM_OP_FINALIZE_SIGN_INPUT_DIGEST ((hsm_op_finalize_sign_flags_t)(1 << 0))`
- `#define HSM_OP_FINALIZE_SIGN_INPUT_MESSAGE ((hsm_op_finalize_sign_flags_t)(1 << 1))`
- `#define HSM_OP_FINALIZE_SIGN_COMPRESSED_POINT ((hsm_op_finalize_sign_flags_t)(1 << 2))`

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`
- `typedef uint8_t hsm_op_finalize_sign_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)
- [hsm_err_t hsm_finalize_signature](#) (hsm_hdl_t signature_gen_hdl, [op_finalize_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 $r||s||R_y$ 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.

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 to the next call of `hsm_generate_signature_finalize`

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

The signature $S=(r,s)$ is stored in the format $r||s||R_y$ where R_y is an additional byte containing the lsb of y . R_y has to be considered valid only if the `HSM_OP_FINALIZE_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

6.6.2.5 hsm_finalize_signature()

```
hsm_err_t hsm_finalize_signature (
    hsm_hdl_t signature_gen_hdl,
    op_finalize_sign_args_t * args )
```

Finalize the computation of a digital signature

User can call this function only after having called the `hsm_prepare_signature` API.

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)(1 << 0))`
- `#define HSM_OP_VERIFY_SIGN_FLAGS_INPUT_MESSAGE ((hsm_op_verify_sign_flags_t)(1 << 1))`
- `#define HSM_OP_VERIFY_SIGN_FLAGS_COMPRESSED_POINT ((hsm_op_verify_sign_flags_t)(1 << 2))`
- `#define HSM_OP_VERIFY_SIGN_FLAGS_KEY_INTERNAL ((hsm_op_verify_sign_flags_t)(1 << 3))`
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 $r||s||R_y$ 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.

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 <code>HSM_VERIFICATION_STATUS_SUCCESS</code> 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))
- #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

Parameters

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

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
- 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
- uint32_t [output_size](#)
length in bytes of the output plaintext
- uint16_t [p1_size](#)
length in bytes of the KDF P1 parameter
- uint16_t [p2_size](#)
length in bytes of the MAC P2 parameter
- uint16_t [mac_size](#)
length in bytes of the requested message authentication code
- hsm_key_type_t [key_type](#)
indicates the type of the used key
- 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
- 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
- uint16_t [p1_size](#)
length in bytes of the KDF P1 parameter

- uint16_t [p2_size](#)
length in bytes of the MAC P2 parameter
- uint16_t [pub_key_size](#)
length in bytes of the recipient public key
- uint16_t [mac_size](#)
length in bytes of the requested message authentication code
- uint32_t [out_size](#)
length in bytes of the output VCT
- hsm_key_type_t [key_type](#)
indicates the type of the recipient public key
- 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 managed 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 op_but_key_exp_args_t Struct Reference

Data Fields

- uint32_t [key_identifier](#)
 - identifier of the key to be expanded*
- uint8_t * [data1](#)
 - pointer to the data1 input*
- uint8_t * [data2](#)
 - pointer to the data2 input*
- uint8_t * [data3](#)
 - pointer to the data3 input*
- uint8_t [data1_size](#)
 - length in bytes of the add_data1 input*
- uint8_t [data2_size](#)
 - length in bytes of the add_data2 input*
- uint8_t [data3_size](#)
 - length in bytes of the data3 input*
- hsm_op_but_key_exp_flags_t [flags](#)
 - bitmap specifying the operation properties*
- uint32_t [dest_key_identifier](#)
 - identifier of the derived key*
- uint8_t * [output](#)
 - pointer to the output area where the public key must be written.*
- uint16_t [output_size](#)
 - length in bytes of the output area, 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**

7.6 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

7.7 op_finalize_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 the 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_FINALIZE↔E_SIGN_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_op_finalize_sign_flags_t [flags](#)
bitmap specifying the operation attributes
- uint8_t **reserved**

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 output area where the generated public key will be copied. It must be 0 in case of symmetric keys.
- hsm_op_key_gen_flags_t [flags](#)

bitmap specifying the operation properties.

- uint8_t **reserved**
- hsm_key_type_t [key_type](#)
indicates which type of key must be generated.
- hsm_key_type_ext_t [key_type_ext](#)
it must be 0
- 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_IGNORE_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.*
- uint16_t [input_size](#)
length in bytes of the input key area. Not checked in case of delete operation.
- hsm_op_manage_key_flags_t [flags](#)
bitmap specifying the operation properties.
- uint16_t **reserved**
- hsm_key_type_t [key_type](#)
indicates the type of the key to be managed.
- hsm_key_type_ext_t **key_type_ext**
- hsm_key_info_t [key_info](#)
bitmap specifying the properties of the key, in case of update operation it will replace the existing value. Not checked in case of delete operation.
- uint8_t * [input_key](#)
pointer to the key to be imported. Not checked in case of delete operation.

7.14 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**

7.15 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**

7.16 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**

7.17 open_svc_cipher_args_t Struct Reference

Data Fields

- hsm_svc_cipher_flags_t [flags](#)
bitmap specifying the services properties.
- uint8_t **reserved** [3]

7.18 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]

7.19 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]

7.20 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**

7.21 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]

7.22 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]

7.23 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, [14](#)

- [hsm_cipher_one_go](#), [15](#)
- [hsm_close_cipher_service](#), [15](#)
- [hsm_ecies_decryption](#), [15](#)
- [hsm_open_cipher_service](#), [14](#)

ECIES encryption, [30](#)

- [hsm_ecies_encryption](#), [30](#)

Error codes, [4](#)

- [HSM_CMD_NOT_SUPPORTED](#), [5](#)
- [hsm_err_t](#), [4](#)
- [HSM_GENERAL_ERROR](#), [5](#)
- [HSM_ID_CONFLICT](#), [5](#)
- [HSM_INVALID_ADDRESS](#), [4](#)
- [HSM_INVALID_LIFECYCLE](#), [5](#)
- [HSM_INVALID_MESSAGE](#), [4](#)
- [HSM_INVALID_PARAM](#), [4](#)
- [HSM_KEY_STORE_AUTH](#), [5](#)
- [HSM_KEY_STORE_CONFLICT](#), [5](#)
- [HSM_KEY_STORE_ERROR](#), [5](#)
- [HSM_NO_ERROR](#), [4](#)
- [HSM_NVM_ERROR](#), [4](#)
- [HSM_OUT_OF_MEMORY](#), [5](#)
- [HSM_RNG_NOT_STARTED](#), [5](#)
- [HSM_UNKNOWN_HANDLE](#), [5](#)
- [HSM_UNKNOWN_ID](#), [4](#)
- [HSM_UNKNOWN_KEY_STORE](#), [5](#)

Hashing, [26](#)

- [hsm_close_hash_service](#), [27](#)
- [hsm_hash_one_go](#), [27](#)
- [hsm_open_hash_service](#), [26](#)

[hsm_butterfly_key_expansion](#)

- Key management, [12](#)

[hsm_cipher_one_go](#)

- Ciphering, [15](#)

[hsm_close_cipher_service](#)

- Ciphering, [15](#)

[hsm_close_hash_service](#)

- Hashing, [27](#)

[hsm_close_key_management_service](#)

- Key management, [13](#)

[hsm_close_key_store_service](#)

- Key store, [9](#)

[hsm_close_rng_service](#)

- Random number generation, [24](#)

[hsm_close_session](#)

- Session, [6](#)

[hsm_close_signature_generation_service](#)

- Signature generation, [18](#)

[hsm_close_signature_verification_service](#)

- Signature verification, [23](#)

[HSM_CMD_NOT_SUPPORTED](#)

- Error codes, [5](#)

[hsm_ecies_decryption](#)

- Ciphering, [15](#)

[hsm_ecies_encryption](#)

- ECIES encryption, [30](#)

[hsm_err_t](#)

- Error codes, [4](#)

[hsm_finalize_signature](#)

- Signature generation, [19](#)

[HSM_GENERAL_ERROR](#)

- Error codes, [5](#)

[hsm_generate_key](#)

- Key management, [11](#)

[hsm_generate_signature](#)

- Signature generation, [18](#)

[hsm_get_random](#)

- Random number generation, [25](#)

[hsm_hash_one_go](#)

- Hashing, [27](#)

[HSM_ID_CONFLICT](#)

- Error codes, [5](#)

[hsm_import_public_key](#)

- Signature verification, [22](#)

[HSM_INVALID_ADDRESS](#)

- Error codes, [4](#)

[HSM_INVALID_LIFECYCLE](#)

- Error codes, [5](#)

[HSM_INVALID_MESSAGE](#)

- Error codes, [4](#)

[HSM_INVALID_PARAM](#)

- Error codes, [4](#)

[HSM_KEY_STORE_AUTH](#)

- Error codes, [5](#)

[HSM_KEY_STORE_CONFLICT](#)

- Error codes, [5](#)

[HSM_KEY_STORE_ERROR](#)

- Error codes, [5](#)

[hsm_manage_key](#)

- Key management, [12](#)

[HSM_NO_ERROR](#)

- Error codes, [4](#)

[HSM_NVM_ERROR](#)

- Error codes, [4](#)

[hsm_op_ecies_dec_args_t](#), [31](#)

[hsm_op_ecies_enc_args_t](#), [31](#)

[hsm_op_pub_key_dec_args_t](#), [32](#)

[hsm_op_pub_key_rec_args_t](#), [32](#)

[hsm_open_cipher_service](#)

- Ciphering, [14](#)

[hsm_open_hash_service](#)

- Hashing, [26](#)

[hsm_open_key_management_service](#)

- Key management, [11](#)

[hsm_open_key_store_service](#)

- Key store, [8](#)

[hsm_open_rng_service](#)

- Random number generation, [24](#)

[hsm_open_session](#)

- Session, 6
- hsm_open_signature_generation_service
 - Signature generation, 18
- hsm_open_signature_verification_service
 - Signature verification, 21
- HSM_OUT_OF_MEMORY
 - Error codes, 5
- hsm_prepare_signature
 - Signature generation, 19
- hsm_pub_key_decompression
 - Public key decompression, 29
- hsm_pub_key_reconstruction
 - Public key reconstruction, 28
- HSM_RNG_NOT_STARTED
 - Error codes, 5
- HSM_UNKNOWN_HANDLE
 - Error codes, 5
- HSM_UNKNOWN_ID
 - Error codes, 4
- HSM_UNKNOWN_KEY_STORE
 - Error codes, 5
- hsm_verify_signature
 - Signature verification, 22
- Key management, 10
 - hsm_butterfly_key_expansion, 12
 - hsm_close_key_management_service, 13
 - hsm_generate_key, 11
 - hsm_manage_key, 12
 - hsm_open_key_management_service, 11
- Key store, 8
 - hsm_close_key_store_service, 9
 - hsm_open_key_store_service, 8
- op_butt_key_exp_args_t, 33
- op_cipher_one_go_args_t, 33
- op_finalize_sign_args_t, 34
- op_generate_key_args_t, 34
- op_generate_sign_args_t, 35
- op_get_random_args_t, 35
- op_hash_one_go_args_t, 36
- op_import_public_key_args_t, 36
- op_manage_key_args_t, 36
- op_prepare_sign_args_t, 37
- op_verify_sign_args_t, 37
- open_session_args_t, 37
- open_svc_cipher_args_t, 37
- open_svc_hash_args_t, 38
- open_svc_key_management_args_t, 38
- open_svc_key_store_args_t, 38
- open_svc_rng_args_t, 38
- open_svc_sign_gen_args_t, 38
- open_svc_sign_ver_args_t, 38
- Public key decompression, 29
 - hsm_pub_key_decompression, 29
- Public key reconstruction, 28
 - hsm_pub_key_reconstruction, 28
- Random number generation, 24
 - hsm_close_rng_service, 24
 - hsm_get_random, 25
 - hsm_open_rng_service, 24
- Session, 6
 - hsm_close_session, 6
 - hsm_open_session, 6
- Signature generation, 17
 - hsm_close_signature_generation_service, 18
 - hsm_finalize_signature, 19
 - hsm_generate_signature, 18
 - hsm_open_signature_generation_service, 18
 - hsm_prepare_signature, 19
- Signature verification, 21
 - hsm_close_signature_verification_service, 23
 - hsm_import_public_key, 22
 - hsm_open_signature_verification_service, 21
 - hsm_verify_signature, 22