

i.MX8 HSM API

Revision_2.0

Generated by Doxygen 1.8.11

Contents

| | | |
|----------|--|----------|
| 1 | HSM API | 1 |
| 2 | Revision History | 1 |
| 3 | General concepts related to the API | 3 |
| 3.1 | Session | 3 |
| 3.2 | Service flow | 3 |
| 3.3 | Example | 4 |
| 3.4 | Key store | 4 |
| 3.4.1 | Key management | 4 |
| 3.4.2 | NVM writing | 4 |
| 4 | Module Index | 5 |
| 4.1 | Modules | 5 |
| 5 | Module Documentation | 5 |
| 5.1 | Error codes | 5 |
| 5.1.1 | Detailed Description | 6 |
| 5.1.2 | Enumeration Type Documentation | 6 |
| 5.2 | Session | 7 |
| 5.2.1 | Detailed Description | 7 |
| 5.2.2 | Data Structure Documentation | 7 |
| 5.2.3 | Function Documentation | 8 |
| 5.3 | Key store | 9 |
| 5.3.1 | Detailed Description | 9 |
| 5.3.2 | Data Structure Documentation | 9 |
| 5.3.3 | Function Documentation | 10 |
| 5.4 | Key management | 12 |
| 5.4.1 | Detailed Description | 14 |
| 5.4.2 | Data Structure Documentation | 14 |
| 5.4.3 | Function Documentation | 16 |

| | | |
|--------|------------------------------|----|
| 5.5 | Ciphering | 20 |
| 5.5.1 | Detailed Description | 20 |
| 5.5.2 | Data Structure Documentation | 20 |
| 5.5.3 | Function Documentation | 22 |
| 5.6 | Signature generation | 25 |
| 5.6.1 | Detailed Description | 26 |
| 5.6.2 | Data Structure Documentation | 26 |
| 5.6.3 | Function Documentation | 27 |
| 5.7 | Signature verification | 29 |
| 5.7.1 | Detailed Description | 29 |
| 5.7.2 | Data Structure Documentation | 29 |
| 5.7.3 | Function Documentation | 30 |
| 5.8 | Random number generation | 33 |
| 5.8.1 | Detailed Description | 33 |
| 5.8.2 | Data Structure Documentation | 33 |
| 5.8.3 | Function Documentation | 33 |
| 5.9 | Hashing | 35 |
| 5.9.1 | Detailed Description | 35 |
| 5.9.2 | Data Structure Documentation | 35 |
| 5.9.3 | Function Documentation | 36 |
| 5.10 | Public key reconstruction | 38 |
| 5.10.1 | Detailed Description | 38 |
| 5.10.2 | Data Structure Documentation | 38 |
| 5.10.3 | Function Documentation | 38 |
| 5.11 | Public key decompression | 40 |
| 5.11.1 | Detailed Description | 40 |
| 5.11.2 | Data Structure Documentation | 40 |
| 5.11.3 | Function Documentation | 40 |
| 5.12 | ECIES encryption | 42 |
| 5.12.1 | Detailed Description | 42 |

| | |
|---|-----------|
| 5.12.2 Data Structure Documentation | 42 |
| 5.12.3 Function Documentation | 42 |
| 5.13 Public key recovery | 44 |
| 5.13.1 Detailed Description | 44 |
| 5.13.2 Data Structure Documentation | 44 |
| 5.13.3 Function Documentation | 44 |
| 5.14 Data storage | 45 |
| 5.14.1 Detailed Description | 45 |
| 5.14.2 Data Structure Documentation | 45 |
| 5.14.3 Function Documentation | 46 |
| 5.15 Root KEK export | 48 |
| 5.15.1 Detailed Description | 48 |
| 5.15.2 Data Structure Documentation | 48 |
| 5.15.3 Function Documentation | 48 |
| 5.16 Get info | 50 |
| 5.16.1 Detailed Description | 50 |
| 5.16.2 Data Structure Documentation | 50 |
| 5.16.3 Function Documentation | 50 |
| 5.17 Mac | 51 |
| 5.17.1 Detailed Description | 51 |
| 5.17.2 Data Structure Documentation | 51 |
| 5.17.3 Function Documentation | 52 |
| Index | 55 |

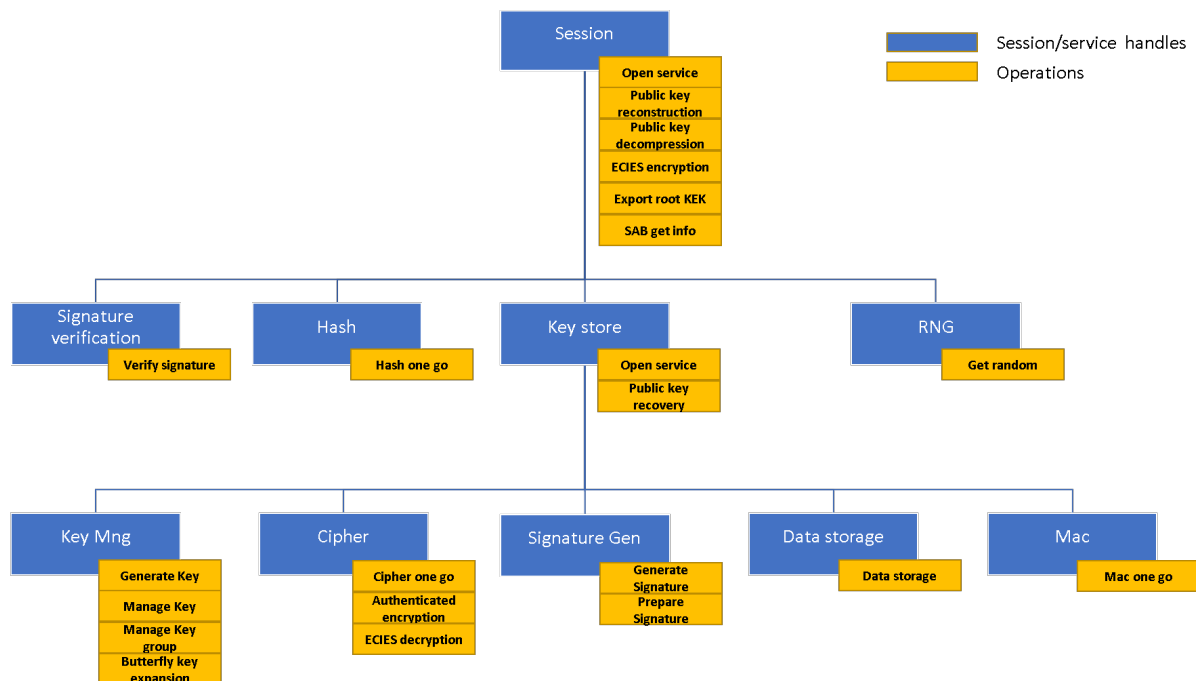
1 HSM API

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 | 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/tipos 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 |
| 1.7 - subject to change | Dec 20 2019 | - add generic data storage API - add GCM and CMAC support - add support for AES 192/256 key size for all cipher algorithms - add root KEK export API - add key import functionality - add get info API |
| 2.0 - subject to change | Feb 21 2020 | - fix HSM_KEY_INFO_TRANSIENT definition: delete erroneous "not supported" comment - add Key Encryption Key (HSM_KEY_INFO_KEK) support - key store open service API: adding signed message support for key store reprovisionning - naming consistency: remove "hsm_" prefix from hsm_op_ecies_dec_args_t hsm_op_pub_key_rec_args_t hsm_op_pub_key_dec_args_t hsm_op_ecies_enc_args_t hsm_op_pub_key_recovery_args_t hsm_op_get_info_args_t |

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 Example

```

/* Open a session: create a route between the user and the HSM */
hsm_open_session(&open_session_args, &session_hdl);

/* Open a key store - user is authenticated */
hsm_open_key_store_service(session_hdl, &open_svc_key_store_args, &key_store_hdl);
/* Open hash service - it grants access to hashing operations */
hsm_open_hash_service (session_hdl, &open_svc_hash_args, &hash_hdl);
/* Open cipher service - it grants access to ciphering operations */
hsm_open_cipher_service(key_store_hdl, &open_svc_cipher_args, &cipher_hdl);

/* Perform AES ECB, CCB ... */
hsm_cipher_one_go (cipher_hdl, &op_cipher_one_go_args);
/* Perform authenticate and encryption algos: e.g AES GCM */
hsm_auth_enc (cipher_hdl, &op_auth_enc_args);
/* Perform hashing operations: e.g SHA */
hsm_hash_one_go (hash_hdl, &op_hash_one_go_args);

/* Close the session and all the related services */
hsm_close_session(session_hdl);

```

3.4 Key store

A key store can be created by specifying the CREATE flag in the hsm_open_key_store_service API. Please note that the created key store will be not stored in the NVM till a key is generated/imported specifying the "STRICT OPERATION" flag.

Only symmetric and private keys are stored into the key store. Public keys can be exported during the key pair generation operation or recalculated through the hsm_pub_key_recovery API.

Secret keys cannot be exported under any circumstances, while they can be imported in encrypted form.

3.4.1 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. The user can control which key group must be kept in the local memory (cached) 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.

3.4.2 NVM writing

All the APIs modifying the content of the key store (key generation, key_management, key derivation functions) provide a "STRICT OPERATION" flag. If the flag is set, the HSM triggers and export of the encrypted key group into the external NVM and increments (blows one bit) the OTP monotonic counter used as roll back protection. Please note that the "STRICT OPERATION" has effect only on the current key group.

Any update to the key store must be considered as effective only after an operation specifying the flag "STRICT OPERATION" is acknowledged by the HSM. All the operations not specifying the "STRICT OPERATION" 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 by default), the user should, when possible, perform multiple updates before setting the "STRICT OPERATION" flag (i.e. keys to be updated should be kept in the same key group).

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:

| | |
|----------------------------------|-----------|
| Error codes | 5 |
| Session | 7 |
| Key store | 9 |
| Key management | 12 |
| Ciphering | 20 |
| Signature generation | 25 |
| Signature verification | 29 |
| Random number generation | 33 |
| Hashing | 35 |
| Public key reconstruction | 38 |
| Public key decompression | 40 |
| ECIES encryption | 42 |
| Public key recovery | 44 |
| Data storage | 45 |
| Root KEK export | 48 |
| Get info | 50 |
| Mac | 51 |

5 Module Documentation

5.1 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 }

```

5.1.1 Detailed Description

5.1.2 Enumeration Type Documentation

5.1.2.1 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.

5.2 Session

Data Structures

- struct [open_session_args_t](#)

Macros

- #define [HSM_OPEN_SESSION_PRIORITY_LOW](#) (0x00U)
Low priority. Should be the default setting on platforms that doesn't support sessions priorities.
- #define [HSM_OPEN_SESSION_PRIORITY_HIGH](#) (0x01U)
High Priority session.
- #define [HSM_OPEN_SESSION_FIPS_MODE_MASK](#) (0x01U)
Only FIPS certified operations authorized in this session.
- #define [HSM_OPEN_SESSION_EXCLUSIVE_MASK](#) (0x02U)
No other HSM session will be authorized on the same security enclave.
- #define [HSM_OPEN_SESSION_LOW_LATENCY_MASK](#) (0x08U)
Use a low latency HSM implementation.
- #define [HSM_OPEN_SESSION_NO_KEY_STORE_MASK](#) (0x10U)
No key store will be attached to this session. May provide better performances on some operation depending on the implementation. Usage of the session will be restricted to operations that doesn't involve secret keys (e.g. hash, signature verification, random generation).
- #define [HSM_OPEN_SESSION_RESERVED_MASK](#) (0x74U)
Bits reserved for future use. Should be set to 0.

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)

5.2.1 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.

5.2.2 Data Structure Documentation

5.2.2.1 struct open_session_args_t

Data Fields

| | | |
|---------|------------------|---|
| uint8_t | session_priority | Priority of the operations performed in this session. */. |
| uint8_t | operating_mode | Options for the session to be opened (bitfield). */. |
| uint8_t | Reserved | |

5.2.3 Function Documentation

5.2.3.1 `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.

5.2.3.2 `hsm_err_t hsm_close_session (hsm_hdl_t session_hdl)`

Terminate a previously opened session. All the services opened under this session are closed as well

Parameters

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

Returns

`error_code` error code.

5.3 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)(1u << 0))
It must be specified to create a new key store. The key store will be stored in the NVM only once a key is generated/imported specifying the STRICT OPERATION flag.
- #define [HSM_SVC_KEY_STORE_FLAGS_UPDATE](#) ((hsm_svc_key_store_flags_t)(1u << 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)(1u << 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)

5.3.1 Detailed Description

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

- create a new key store
- perform operations involving keys stored in the key store (ciphering, signature generation...)
- perform a key store reprovisioning using a signed message. A key store re-provisioning results in erasing all the key stores handled by the HSM.

To grant access to the key store, the caller is authenticated against the domain ID (DID) and Messaging Unit used at the keystore creation, additionally an authentication nonce can be provided.

5.3.2 Data Structure Documentation

5.3.2.1 struct open_svc_key_store_args_t

Data Fields

| | | |
|--------------------------------|----------------------|--|
| uint32_t | key_store_identifier | user defined id identifying the key store. Only one key store service can be opened on a given key_store_identifier. |
| 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. This parameter has the goal to limit the occupation of the monotonic counter used as anti-rollback protection. If the maximum number of updates is reached, HSM still allows key store updates but without updating the monotonic counter giving the opportunity for rollback attacks. |
| hsm_svc_key_store_↔ flags_t | flags | bitmap specifying the services properties. |
| uint8_t | reserved | |
| uint8_t * | signed_message | pointer to signed_message to be sent only in case of key store re-provisioning |
| uint16_t | signed_msg_size | size of the signed_message to be sent only in case of key store re-provisioning |
| uint8_t | reserved_1[2] | |

5.3.3 Function Documentation

5.3.3.1 **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. Only one key store service can be opened on a given 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.

5.3.3.2 **hsm_err_t hsm_close_key_store_service (hsm_hdl_t key_store_hdl)**

Close a previously opened key store service flow. The key store is deleted from the HSM local memory, any update not written in the NVM is lost

Parameters

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

Returns

`error_code` error code.

5.4 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_but_key_exp_args_t](#)

Macros

- #define [HSM_KEY_TYPE_ECDSA_NIST_P224](#) ((hsm_key_type_t)0x01u)
not supported
- #define [HSM_KEY_TYPE_ECDSA_NIST_P256](#) ((hsm_key_type_t)0x02u)
- #define [HSM_KEY_TYPE_ECDSA_NIST_P384](#) ((hsm_key_type_t)0x03u)
- #define [HSM_KEY_TYPE_ECDSA_BRAINPOOL_R1_224](#) ((hsm_key_type_t)0x12u)
not supported
- #define [HSM_KEY_TYPE_ECDSA_BRAINPOOL_R1_256](#) ((hsm_key_type_t)0x13u)
- #define [HSM_KEY_TYPE_ECDSA_BRAINPOOL_R1_384](#) ((hsm_key_type_t)0x15u)
- #define [HSM_KEY_TYPE_ECDSA_BRAINPOOL_T1_224](#) ((hsm_key_type_t)0x22u)
not supported
- #define [HSM_KEY_TYPE_ECDSA_BRAINPOOL_T1_256](#) ((hsm_key_type_t)0x23u)
not supported
- #define [HSM_KEY_TYPE_ECDSA_BRAINPOOL_T1_384](#) ((hsm_key_type_t)0x25u)
not supported
- #define [HSM_KEY_TYPE_AES_128](#) ((hsm_key_type_t)0x30u)
- #define [HSM_KEY_TYPE_AES_192](#) ((hsm_key_type_t)0x31u)
- #define [HSM_KEY_TYPE_AES_256](#) ((hsm_key_type_t)0x32u)
- #define [HSM_OP_KEY_GENERATION_FLAGS_UPDATE](#) ((hsm_op_key_gen_flags_t)(1u << 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)(1u << 1))
Create a new key.
- #define [HSM_OP_KEY_GENERATION_FLAGS_STRICT_OPERATION](#) ((hsm_op_key_gen_flags_t)(1u << 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_PERSISTENT](#) ((hsm_key_info_t)(0u << 1))
Persistent keys are stored in the external NVM. The entire key group is written in the NVM at the next STRICT operation.
- #define [HSM_KEY_INFO_PERMANENT](#) ((hsm_key_info_t)(1u << 0))
When set, the key is permanent (write locked). Once created, it will not be possible to update or delete the key anymore. Transient keys will be anyway deleted after a 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)(1u << 1))
Transient keys are deleted when the corresponding key store service flow is closed or after a PoR. Transient keys cannot be in the same key group than persistent keys.
- #define [HSM_KEY_INFO_MASTER](#) ((hsm_key_info_t)(1u << 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_KEY_INFO_KEK](#) ((hsm_key_info_t)(1u << 3))
When set, the key is considered as a key encryption key. KEK keys can only be used to wrap and import other keys into the key store, all other operation are not allowed. Only keys imported in the key store through the hsm_manage_key API can get this attribute.

- `#define HSM_OP_MANAGE_KEY_FLAGS_IMPORT_UPDATE ((hsm_op_manage_key_flags_t)(1u << 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_IMPORT_CREATE ((hsm_op_manage_key_flags_t)(1u << 1))`
Import a key and create a new identifier.
- `#define HSM_OP_MANAGE_KEY_FLAGS_DELETE ((hsm_op_manage_key_flags_t)(1u << 2))`
Delete an existing key.
- `#define HSM_OP_MANAGE_KEY_FLAGS_PART_UNIQUE_ROOT_KEY ((hsm_op_manage_key_flags_t)(1u << 3))`
The key to be imported is encrypted using the part-unique root kek.
- `#define HSM_OP_MANAGE_KEY_FLAGS_COMMON_ROOT_KEY ((hsm_op_manage_key_flags_t)(1u << 4))`
The key to be imported is encrypted using the common root kek.
- `#define HSM_OP_MANAGE_KEY_FLAGS_STRICT_OPERATION ((hsm_op_manage_key_flags_t)(1u << 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)(1u << 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)(1u << 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)(1u << 2))`
not supported - delete an existing key group
- `#define HSM_OP_MANAGE_KEY_GROUP_FLAGS_STRICT_OPERATION ((hsm_op_manage_key_group_flags_t)(1u << 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)(1u << 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)(1u << 1))`
Create a new key.
- `#define HSM_OP_BUTTERFLY_KEY_FLAGS_IMPLICIT_CERTIF ((hsm_op_but_key_exp_flags_t)(0u << 2))`
butterfly key expansion using implicit certificate.
- `#define HSM_OP_BUTTERFLY_KEY_FLAGS_EXPLICIT_CERTIF ((hsm_op_but_key_exp_flags_t)(1u << 2))`
butterfly key expansion using explicit certificate.
- `#define HSM_OP_BUTTERFLY_KEY_FLAGS_STRICT_OPERATION ((hsm_op_but_key_exp_flags_t)(1u << 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)

5.4.1 Detailed Description

5.4.2 Data Structure Documentation

5.4.2.1 struct open_svc_key_management_args_t

Data Fields

| | | |
|---|-------------|--|
| hsm_svc_key_management_↔_args_t | flags | bitmap specifying the services properties. |
| uint8_t | reserved[3] | |

5.4.2.2 struct op_generate_key_args_t

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_↔_args_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, relevant only 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 hsm_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 |

5.4.2.3 struct op_manage_key_args_t

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. |
|------------|----------------|--|

Data Fields

| | | |
|----------------------------|----------------|---|
| uint32_t | kek_identifier | identifier of the key to be used to decrypt the key to be imported (Key Encryption Key), only AES-256 key can be uses as KEK. It must be 0 if the HSM_OP_MANAGE_KEY↔_FLAGS_PART_UNIQUE_ROOT_KEK or HSM_OP_MANAGE_KEY_FLAGS_COMMON_ROOT_KEK flags are set. |
| uint16_t | input_size | length in bytes of the input key area. It must be eqaul to the length of the IV (12 bytes) + ciphertext + Tag (16 bytes). 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 throug 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_data | pointer to the input buffer. The input buffer is the concatenation of the IV, the encrypted key to be imported and the Tag. It must be 0 in case of delete operation. |

5.4.2.4 struct op_manage_key_group_args_t

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 throug the ham_manage_key_group API |
| hsm_op_manage_key_group↔_flags_t | flags | bitmap specifying the operation properties. |
| uint8_t | reserved | |

5.4.2.5 struct op_buttt_key_exp_args_t

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 |

Data Fields

| | | |
|---------------------------------|------------------------------|---|
| 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 derived. |
| 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. |

5.4.3 Function Documentation

5.4.3.1 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

| | |
|--------------------|--|
| key_store_hdl | handle indentifying the key store service flow. |
| args | pointer to the structure containing the function arguments. |
| key_management_hdl | pointer to where the key management service flow handle must be written. |

Returns

error_code error code.

5.4.3.2 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

5.4.3.3 `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 and create)
- import a key using an existing key identifier (import and update)
- delete an existing key

The key encryption key (KEK) can be previously pre-shared or stored in the key store.

The key to be imported must be encrypted by using the KEK as following:

- Algorithm: AES GCM
- Key: root KEK
- AAD = 0
- IV = 12 bytes
- Tag = 16 bytes
- Plaintext: key to be imported

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

5.4.3.4 `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

5.4.3.5 `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

5.4.3.6 `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

5.5 Ciphering

Data Structures

- struct [open_svc_cipher_args_t](#)
- struct [op_cipher_one_go_args_t](#)
- struct [op_auth_enc_args_t](#)
- struct [op_ecies_dec_args_t](#)

Macros

- #define **HSM_CIPHER_ONE_GO_ALGO_AES_ECB** ((hsm_op_cipher_one_go_algo_t)(0x00u))
- #define **HSM_CIPHER_ONE_GO_ALGO_AES_CBC** ((hsm_op_cipher_one_go_algo_t)(0x01u))
- #define **HSM_CIPHER_ONE_GO_ALGO_AES_CCM** ((hsm_op_cipher_one_go_algo_t)(0x04u))
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)(0u << 0))
- #define **HSM_CIPHER_ONE_GO_FLAGS_ENCRYPT** ((hsm_op_cipher_one_go_flags_t)(1u << 0))
- #define **HSM_AUTH_ENC_ALGO_AES_GCM** ((hsm_op_auth_enc_algo_t)(0x00u))
Perform AES GCM with following constraints: AES GCM where AAD supported, Tag len = 16 bytes, IV len = 12 bytes.
- #define **HSM_AUTH_ENC_FLAGS_DECRYPT** ((hsm_op_auth_enc_flags_t)(0u << 0))
- #define **HSM_AUTH_ENC_FLAGS_ENCRYPT** ((hsm_op_auth_enc_flags_t)(1u << 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_auth_enc_algo_t**
- typedef uint8_t **hsm_op_auth_enc_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_auth_enc](#) (hsm_hdl_t cipher_hdl, [op_auth_enc_args_t](#) *args)
- [hsm_err_t hsm_ecies_decryption](#) (hsm_hdl_t cipher_hdl, [op_ecies_dec_args_t](#) *args)
- [hsm_err_t hsm_close_cipher_service](#) (hsm_hdl_t cipher_hdl)

5.5.1 Detailed Description

5.5.2 Data Structure Documentation

5.5.2.1 struct open_svc_cipher_args_t

Data Fields

| | | |
|-----------------------------|-------------|--|
| hsm_svc_cipher_↔ flags_t | flags | bitmap specifying the services properties. |
| uint8_t | reserved[3] | |

5.5.2.2 struct op_cipher_one_go_args_t

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 |

5.5.2.3 struct op_auth_enc_args_t

Data Fields

| | | |
|------------------------------|----------------|---|
| uint32_t | key_identifier | identifier of the key to be used for the operation |
| uint8_t * | iv | pointer to the initialization vector or nonce |
| uint16_t | iv_size | length in bytes of the initialization vector It must be 12 bytes. |
| uint8_t * | aad | pointer to the additional authentication data |
| uint16_t | aad_size | length in bytes of the additional authentication data |
| hsm_op_auth_enc_algo_↔ _t | ae_algo | algorithm to be used for the operation |
| hsm_op_auth_enc_↔ flags_t | flags | bitmap specifying the operation attributes |
| uint8_t * | input | pointer to the input area plaintext for encryption Ciphertext + Tag (16 bytes) for decryption |
| uint8_t * | output | pointer to the output area Ciphertext + Tag (16 bytes) for encryption plaintext for decryption if the Tag is verified |
| uint32_t | input_size | length in bytes of the input |

Data Fields

| | | |
|----------|-------------|-------------------------------|
| uint32_t | output_size | length in bytes of the output |
|----------|-------------|-------------------------------|

5.5.2.4 struct op_ecies_dec_args_t

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_t ↔ flags_t | flags | bitmap specifying the operation attributes. |

5.5.3 Function Documentation

5.5.3.1 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

| | |
|---------------|--|
| key_store_hdl | handle indentifying the key store service flow. |
| args | pointer to the structure containing the function arugments. |
| cipher_hdl | pointer to where the cipher service flow handle must be written. |

Returns

error code

5.5.3.2 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 arguments. |

Returns

error code

5.5.3.3 hsm_err_t hsm_auth_enc (hsm_hdl_t cipher_hdl, op_auth_enc_args_t * args)

Perform authenticated encryption 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 arguments. |

Returns

error code

5.5.3.4 hsm_err_t hsm_ecies_decryption (hsm_hdl_t cipher_hdl, op_ecies_dec_args_t * args)

Decrypt data using 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 arguments. |

Returns

error code

5.5.3.5 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

5.6 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)0x01u)
not supported
- #define [HSM_SIGNATURE_SCHEME_ECDSA_NIST_P256_SHA_256](#) ((hsm_signature_scheme_id_↵
t)0x02u)
- #define [HSM_SIGNATURE_SCHEME_ECDSA_NIST_P384_SHA_384](#) ((hsm_signature_scheme_id_↵
t)0x03u)
- #define [HSM_SIGNATURE_SCHEME_ECDSA_BRAINPOOL_R1_224_SHA_256](#) ((hsm_signature_↵
scheme_id_t)0x12u)
not supported
- #define [HSM_SIGNATURE_SCHEME_ECDSA_BRAINPOOL_R1_256_SHA_256](#) ((hsm_signature_↵
scheme_id_t)0x13u)
- #define [HSM_SIGNATURE_SCHEME_ECDSA_BRAINPOOL_R1_384_SHA_384](#) ((hsm_signature_↵
scheme_id_t)0x15u)
- #define [HSM_SIGNATURE_SCHEME_ECDSA_BRAINPOOL_T1_224_SHA_256](#) ((hsm_signature_↵
scheme_id_t)0x22u)
not supported
- #define [HSM_SIGNATURE_SCHEME_ECDSA_BRAINPOOL_T1_256_SHA_256](#) ((hsm_signature_↵
scheme_id_t)0x23u)
not supported
- #define [HSM_SIGNATURE_SCHEME_ECDSA_BRAINPOOL_T1_384_SHA_384](#) ((hsm_signature_↵
scheme_id_t)0x25u)
not supported
- #define [HSM_OP_GENERATE_SIGN_FLAGS_INPUT_DIGEST](#) ((hsm_op_generate_sign_flags_t)(0u <<
0))
- #define [HSM_OP_GENERATE_SIGN_FLAGS_INPUT_MESSAGE](#) ((hsm_op_generate_sign_flags_t)(1u
<< 0))
- #define [HSM_OP_GENERATE_SIGN_FLAGS_COMPRESSED_POINT](#) ((hsm_op_generate_sign_flags_↵
t)(1u << 1))
- #define [HSM_OP_GENERATE_SIGN_FLAGS_LOW_LATENCY_SIGNATURE](#) ((hsm_op_generate_sign_↵
_flags_t)(1u << 2))
- #define [HSM_OP_PREPARE_SIGN_INPUT_DIGEST](#) ((hsm_op_prepare_signature_flags_t)(0u << 0))
- #define [HSM_OP_PREPARE_SIGN_INPUT_MESSAGE](#) ((hsm_op_prepare_signature_flags_t)(1u << 0))
- #define [HSM_OP_PREPARE_SIGN_COMPRESSED_POINT](#) ((hsm_op_prepare_signature_flags_t)(1u <<
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`)

5.6.1 Detailed Description

5.6.2 Data Structure Documentation

5.6.2.1 struct open_svc_sign_gen_args_t

Data Fields

| | | |
|---|--------------------------|--|
| <code>hsm_svc_signature_generation_flags_t</code> | <code>flags</code> | bitmap specifying the services properties. |
| <code>uint8_t</code> | <code>reserved[3]</code> | |

5.6.2.2 struct op_generate_sign_args_t

Data Fields

| | | |
|---|-----------------------------|--|
| <code>uint32_t</code> | <code>key_identifier</code> | identifier of the key to be used for the operation |
| <code>uint8_t *</code> | <code>message</code> | pointer to the input (message or message digest) to be signed |
| <code>uint8_t *</code> | <code>signature</code> | 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 <code>HSM_OP_GENERATE_SIGN_FLAGS_COMPRESSED_POINT</code> is set. |
| <code>uint32_t</code> | <code>message_size</code> | length in bytes of the input |
| <code>uint16_t</code> | <code>signature_size</code> | length in bytes of the output |
| <code>hsm_signature_scheme_id_t</code> | <code>scheme_id</code> | identifier of the digital signature scheme to be used for the operation |
| <code>hsm_op_generate_sign_flags_t</code> | <code>flags</code> | bitmap specifying the operation attributes |

5.6.2.3 struct op_prepare_sign_args_t

Data Fields

| | | |
|---|------------------------|---|
| <code>hsm_signature_scheme_id_t</code> | <code>scheme_id</code> | identifier of the digital signature scheme to be used for the operation |
| <code>hsm_op_prepare_signature_flags_t</code> | <code>flags</code> | bitmap specifying the operation attributes |
| <code>uint16_t</code> | <code>reserved</code> | |

5.6.3 Function Documentation

5.6.3.1 `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

5.6.3.2 `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

5.6.3.3 `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 arugments. |

Returns

error code

5.6.3.4 `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 $r||s||R_y$ where R_y is an additional byte containing the lsb of y , R_y has to be considered valid only if the `HSM_OP_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

5.7 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)(0u << 0))
- #define **HSM_OP_VERIFY_SIGN_FLAGS_INPUT_MESSAGE** ((hsm_op_verify_sign_flags_t)(1u << 0))
- #define **HSM_OP_VERIFY_SIGN_FLAGS_COMPRESSED_POINT** ((hsm_op_verify_sign_flags_t)(1u << 1))
- #define **HSM_OP_VERIFY_SIGN_FLAGS_KEY_INTERNAL** ((hsm_op_verify_sign_flags_t)(1u << 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)(0x5A3CC3A5u))

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_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)

5.7.1 Detailed Description

5.7.2 Data Structure Documentation

5.7.2.1 struct open_svc_sign_ver_args_t

Data Fields

| | | |
|--|-------------|---|
| hsm_svc_signature_verification_ flags_t | flags | bitmap indicating the service flow properties |
| uint8_t | reserved[3] | |

5.7.2.2 struct op_verify_sign_args_t

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 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. |
| 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 Ry. |
| 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 | |

5.7.2.3 struct op_import_public_key_args_t

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 |

5.7.3 Function Documentation

5.7.3.1 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

| | |
|-------------------|--|
| session_hdl | handle indentifying the current session. |
| args | pointer to the structure containing the function arguments. |
| signature_ver_hdl | pointer to where the signature verification service flow handle must be written. |

Returns

error code

5.7.3.2 `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 arguments. |
| <i>status</i> | pointer to where the verification status must be stored if the verification succeed the value <code>HSM_VERIFICATION_STATUS_SUCCESS</code> is returned. |

Returns

error code

5.7.3.3 `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 imported 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 <code>hsm_verify_signature</code> will be stored |

Returns

error code

5.7.3.4 `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

5.8 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)

5.8.1 Detailed Description

5.8.2 Data Structure Documentation

5.8.2.1 struct open_svc_rng_args_t

Data Fields

| | | |
|-------------------------------------|-------------|---|
| hsm_svc_rng_flags_t | flags | bitmap indicating the service flow properties |
| uint8_t | reserved[3] | |

5.8.2.2 struct op_get_random_args_t

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. |

5.8.3 Function Documentation

5.8.3.1 [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

5.8.3.2 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

5.8.3.3 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 arugments. |

Returns

error code

5.9 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)(0x0u))
- #define **HSM_HASH_ALGO_SHA_256** ((hsm_hash_algo_t)(0x1u))
- #define **HSM_HASH_ALGO_SHA_384** ((hsm_hash_algo_t)(0x2u))
- #define **HSM_HASH_ALGO_SHA_512** ((hsm_hash_algo_t)(0x3u))

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)

5.9.1 Detailed Description

5.9.2 Data Structure Documentation

5.9.2.1 struct open_svc_hash_args_t

Data Fields

| | | |
|----------------------|-------------|---|
| hsm_svc_hash_flags_t | flags | bitmap indicating the service flow properties |
| uint8_t | reserved[3] | |

5.9.2.2 struct op_hash_one_go_args_t

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 |

Data Fields

| | | |
|---------------------------------|----------|---|
| 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 | |

5.9.3 Function Documentation

5.9.3.1 **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

5.9.3.2 **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

5.9.3.3 **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 arugments. |

Returns

error code

5.10 Public key reconstruction

Data Structures

- struct [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, [op_pub_key_rec_args_t](#) *args)

5.10.1 Detailed Description

5.10.2 Data Structure Documentation

5.10.2.1 struct op_pub_key_rec_args_t

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 | |

5.10.3 Function Documentation

5.10.3.1 hsm_err_t hsm_pub_key_reconstruction (hsm_hdl_t session_hdl, 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 arguments. |

Returns

error code

5.11 Public key decompression

Data Structures

- struct [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, [op_pub_key_dec_args_t](#) *args)

5.11.1 Detailed Description

5.11.2 Data Structure Documentation

5.11.2.1 struct op_pub_key_dec_args_t

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 | |

5.11.3 Function Documentation

5.11.3.1 hsm_err_t hsm_pub_key_decompression (hsm_hdl_t session_hdl, 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

5.12 ECIES encryption

Data Structures

- struct [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, [op_ecies_enc_args_t](#) *args)

5.12.1 Detailed Description

5.12.2 Data Structure Documentation

5.12.2.1 struct op_ecies_enc_args_t

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 | |

5.12.3 Function Documentation

5.12.3.1 hsm_err_t hsm_ecies_encryption (hsm_hdl_t session_hdl, 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 arguments. |

Returns

error code

5.13 Public key recovery

Data Structures

- struct [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, [op_pub_key_recovery_args_t](#) *args)

5.13.1 Detailed Description

5.13.2 Data Structure Documentation

5.13.2.1 struct op_pub_key_recovery_args_t

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. |

5.13.3 Function Documentation

5.13.3.1 hsm_err_t hsm_pub_key_recovery (hsm_hdl_t key_store_hdl, 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

5.14 Data storage

Data Structures

- struct [open_svc_data_storage_args_t](#)
- struct [op_data_storage_args_t](#)

Macros

- #define [HSM_OP_DATA_STORAGE_FLAGS_STORE](#) ((hsm_op_data_storage_flags_t)(1u << 0))
Store data.
- #define [HSM_OP_DATA_STORAGE_FLAGS_RETRIEVE](#) ((hsm_op_data_storage_flags_t)(0u << 0))
Retrieve data.

Typedefs

- typedef uint8_t [hsm_svc_data_storage_flags_t](#)
- typedef uint8_t [hsm_op_data_storage_flags_t](#)

Functions

- [hsm_err_t hsm_open_data_storage_service](#) (hsm_hdl_t key_store_hdl, [open_svc_data_storage_args_t](#) *args, hsm_hdl_t *data_storage_hdl)
- [hsm_err_t hsm_data_storage](#) (hsm_hdl_t data_storage_hdl, [op_data_storage_args_t](#) *args)
- [hsm_err_t hsm_close_data_storage_service](#) (hsm_hdl_t data_storage_hdl)

5.14.1 Detailed Description

5.14.2 Data Structure Documentation

5.14.2.1 struct open_svc_data_storage_args_t

Data Fields

| | | |
|--|-------------|--|
| hsm_svc_data_storage_flags_t | flags | bitmap specifying the services properties. |
| uint8_t | reserved[3] | |

5.14.2.2 struct op_data_storage_args_t

Data Fields

| | | |
|-----------|-----------|--|
| uint8_t * | data | pointer to the data. In case of store request, it will be the input data to store. In case of retrieve, it will be the the pointer where to load data. |
| uint32_t | data_size | length in bytes of the data |
| uint16_t | data_id | id of the data |

Data Fields

| | | |
|----------------------------------|----------|---|
| hsm_op_data_storage_↔ flags_t | flags | flags bitmap specifying the operation attributes. |
| uint8_t | reserved | |

5.14.3 Function Documentation

5.14.3.1 **hsm_err_t** hsm_open_data_storage_service (**hsm_hdl_t** *key_store_hdl*, **open_svc_data_storage_args_t** * *args*, **hsm_hdl_t** * *data_storage_hdl*)

Open a data storage service flow

User must open this service flow in order to store/retrieve generic data in/from the HSM.

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>data_storage_hdl</i> | pointer to where the data storage service flow handle must be written. |

Returns

error_code error code.

5.14.3.2 **hsm_err_t** hsm_data_storage (**hsm_hdl_t** *data_storage_hdl*, **op_data_storage_args_t** * *args*)

Store or retrieve generic data identified by a data_id.

Parameters

| | |
|-------------------------|---|
| <i>data_storage_hdl</i> | handle identifying the data storage service flow. |
| <i>args</i> | pointer to the structure containing the function arugments. |

Returns

error code

5.14.3.3 **hsm_err_t** hsm_close_data_storage_service (**hsm_hdl_t** *data_storage_hdl*)

Terminate a previously opened data storage service flow

Parameters

| | |
|-------------------------|---|
| <i>data_storage_hdl</i> | handle identifying the data storage service flow. |
|-------------------------|---|

Returns

error code

5.15 Root KEK export

Data Structures

- struct [op_export_root_kek_args_t](#)

Macros

- `#define HSM_OP_EXPORT_ROOT_KEK_FLAGS_COMMON_KEK ((hsm_op_export_root_kek_flags_t)(1u << 0))`
- `#define HSM_OP_EXPORT_ROOT_KEK_FLAGS_UNIQUE_KEK ((hsm_op_export_root_kek_flags_t)(0u << 0))`

Typedefs

- typedef uint8_t [hsm_op_export_root_kek_flags_t](#)

Functions

- [hsm_err_t hsm_export_root_key_encryption_key](#) ([hsm_hdl_t session_hdl](#), [op_export_root_kek_args_t](#) *args)

5.15.1 Detailed Description

5.15.2 Data Structure Documentation

5.15.2.1 struct op_export_root_kek_args_t

Data Fields

| | | |
|--|-----------------|--|
| uint8_t * | signed_message | pointer to signed_message authorizing the operation |
| uint8_t * | out_root_kek | pointer to the output area where the derived root kek (key encryption key) must be written |
| uint16_t | signed_msg_size | size of the signed_message authorizing the operation |
| uint8_t | root_kek_size | length in bytes of the root kek. Must be 32 bytes. |
| hsm_op_export_root_kek_flags_t | flags | flags bitmap specifying the operation attributes. |
| uint8_t | reserved[2] | |

5.15.3 Function Documentation

5.15.3.1 `hsm_err_t hsm_export_root_key_encryption_key (hsm_hdl_t session_hdl, op_export_root_kek_args_t * args)`

Export the root key encryption key. This key is derived on chip. It can be common or chip unique. This key will be used to import key in the key store through the manage key API.

Parameters

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

Returns

error code

5.16 Get info

Data Structures

- struct [op_get_info_args_t](#)

Functions

- [hsm_err_t hsm_get_info](#) ([hsm_hdl_t session_hdl](#), [op_get_info_args_t](#) *args)

5.16.1 Detailed Description

5.16.2 Data Structure Documentation

5.16.2.1 struct op_get_info_args_t

Data Fields

| | | |
|------------|------------------------|---|
| uint32_t * | user_sab_id | pointer to the output area where the user identifier (32bits) must be written |
| uint8_t * | chip_unique_id | pointer to the output area where the chip unique identifier (64bits) must be written |
| uint16_t * | chip_monotonic_counter | pointer to the output are where the chip monotonic counter value (16bits) must be written |
| uint16_t * | chip_life_cycle | pointer to the output area where the chip current life cycle (16bits) must be written |
| uint32_t * | version | pointer to the output area where the module version (32bits) must be written |
| uint32_t * | version_ext | pointer to the output area where module extended version (32bits) must be written |
| uint8_t * | fips_mode | pointer to the output area where the FIPS mode of operation (8bits) must be written |

5.16.3 Function Documentation

5.16.3.1 [hsm_err_t hsm_get_info](#) ([hsm_hdl_t session_hdl](#), [op_get_info_args_t](#) * args)

Parameters

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

Returns

error code

5.17 Mac

Data Structures

- struct [open_svc_mac_args_t](#)
- struct [op_mac_one_go_args_t](#)

Macros

- `#define HSM_OP_MAC_ONE_GO_FLAGS_MAC_VERIFICATION ((hsm_op_mac_one_go_flags_t)(0u << 0))`
- `#define HSM_OP_MAC_ONE_GO_FLAGS_MAC_GENERATION ((hsm_op_mac_one_go_flags_t)(1u << 0))`
- `#define HSM_OP_MAC_ONE_GO_ALGO_AES_CMAC ((hsm_op_mac_one_go_algo_t)(0x01u))`
- `#define HSM_MAC_VERIFICATION_STATUS_SUCCESS ((hsm_mac_verification_status_t)(0x6C1AA1↵C6u))`

Typedefs

- `typedef uint8_t hsm_svc_mac_flags_t`
- `typedef uint8_t hsm_op_mac_one_go_algo_t`
- `typedef uint8_t hsm_op_mac_one_go_flags_t`
- `typedef uint32_t hsm_mac_verification_status_t`

Functions

- `hsm_err_t hsm_open_mac_service (hsm_hdl_t key_store_hdl, open_svc_mac_args_t *args, hsm_hdl_t ↵*mac_hdl)`
- `hsm_err_t hsm_mac_one_go (hsm_hdl_t mac_hdl, op_mac_one_go_args_t *args, hsm_mac_verification↵_status_t *status)`
- `hsm_err_t hsm_close_mac_service (hsm_hdl_t mac_hdl)`

5.17.1 Detailed Description

5.17.2 Data Structure Documentation

5.17.2.1 struct [open_svc_mac_args_t](#)

Data Fields

| | | |
|--------------------------------------|-------------|--|
| hsm_svc_mac_↵flags_t | flags | bitmap specifying the services properties. |
| uint8_t | reserved[3] | |

5.17.2.2 struct [op_mac_one_go_args_t](#)

Data Fields

| | | |
|---------------------------|----------------|---|
| uint32_t | key_identifier | identifier of the key to be used for the operation |
| hsm_op_mac_one_go_algo_t | algorithm | algorithm to be used for the operation |
| hsm_op_mac_one_go_flags_t | flags | bitmap specifying the operation attributes |
| uint8_t * | payload | pointer to the payload area |
| uint8_t * | mac | pointer to the tag area |
| uint16_t | payload_size | length in bytes of the payload |
| uint16_t | mac_size | length in bytes of the tag the value is in range from 4 to 16 bytes. |

5.17.3 Function Documentation

5.17.3.1 **hsm_err_t hsm_open_mac_service (hsm_hdl_t key_store_hdl, open_svc_mac_args_t * args, hsm_hdl_t * mac_hdl)**

Open a mac 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 mac 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>mac_hdl</i> | pointer to where the mac service flow handle must be written. |

Returns

error code

5.17.3.2 **hsm_err_t hsm_mac_one_go (hsm_hdl_t mac_hdl, op_mac_one_go_args_t * args, hsm_mac_verification_status_t * status)**

Perform mac operation

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

Parameters

| | |
|----------------|---|
| <i>mac_hdl</i> | handle identifying the mac service flow. |
| <i>args</i> | pointer to the structure containing the function arugments. |

Returns

error code

5.17.3.3 `hsm_err_t hsm_close_mac_service (hsm_hdl_t mac_hdl)`

Terminate a previously opened mac service flow

Parameters

| | |
|----------------|--|
| <i>mac_hdl</i> | pointer to handle identifying the mac service flow to be closed. |
|----------------|--|

Returns

error code

Index

Ciphering, [20](#)

- [hsm_auth_enc](#), [23](#)
- [hsm_cipher_one_go](#), [22](#)
- [hsm_close_cipher_service](#), [23](#)
- [hsm_ecies_decryption](#), [23](#)
- [hsm_open_cipher_service](#), [22](#)

Data storage, [45](#)

- [hsm_close_data_storage_service](#), [46](#)
- [hsm_data_storage](#), [46](#)
- [hsm_open_data_storage_service](#), [46](#)

ECIES encryption, [42](#)

- [hsm_ecies_encryption](#), [42](#)

Error codes, [5](#)

- [HSM_CMD_NOT_SUPPORTED](#), [6](#)
- [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](#)
- [hsm_err_t](#), [6](#)

Get info, [50](#)

- [hsm_get_info](#), [50](#)

[HSM_CMD_NOT_SUPPORTED](#)

- Error codes, [6](#)

[HSM_FEATURE_NOT_SUPPORTED](#)

- Error codes, [6](#)

[HSM_GENERAL_ERROR](#)

- Error codes, [6](#)

[HSM_ID_CONFLICT](#)

- Error codes, [6](#)

[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_NO_ERROR](#)

- Error codes, [6](#)

[HSM_NVM_ERROR](#)

- Error codes, [6](#)

[HSM_OUT_OF_MEMORY](#)

- Error codes, [6](#)

[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](#)

Hashing, [35](#)

- [hsm_close_hash_service](#), [36](#)
- [hsm_hash_one_go](#), [36](#)
- [hsm_open_hash_service](#), [36](#)

[hsm_auth_enc](#)

- Ciphering, [23](#)

[hsm_butterfly_key_expansion](#)

- Key management, [18](#)

[hsm_cipher_one_go](#)

- Ciphering, [22](#)

[hsm_close_cipher_service](#)

- Ciphering, [23](#)

[hsm_close_data_storage_service](#)

- Data storage, [46](#)

[hsm_close_hash_service](#)

- Hashing, [36](#)

[hsm_close_key_management_service](#)

- Key management, [18](#)

[hsm_close_key_store_service](#)

- Key store, [10](#)

[hsm_close_mac_service](#)

- Mac, [52](#)

[hsm_close_rng_service](#)

- Random number generation, [34](#)

[hsm_close_session](#)

- Session, [8](#)

[hsm_close_signature_generation_service](#)

- Signature generation, [27](#)

[hsm_close_signature_verification_service](#)

- Signature verification, [31](#)

[hsm_data_storage](#)

- Data storage, [46](#)

[hsm_ecies_decryption](#)

- Ciphering, [23](#)

- hsm_ecies_encryption
 - ECIES encryption, [42](#)
- hsm_err_t
 - Error codes, [6](#)
- hsm_export_root_key_encryption_key
 - Root KEK export, [48](#)
- hsm_generate_key
 - Key management, [16](#)
- hsm_generate_signature
 - Signature generation, [27](#)
- hsm_get_info
 - Get info, [50](#)
- hsm_get_random
 - Random number generation, [34](#)
- hsm_hash_one_go
 - Hashing, [36](#)
- hsm_import_public_key
 - Signature verification, [31](#)
- hsm_mac_one_go
 - Mac, [52](#)
- hsm_manage_key
 - Key management, [17](#)
- hsm_manage_key_group
 - Key management, [17](#)
- hsm_open_cipher_service
 - Ciphering, [22](#)
- hsm_open_data_storage_service
 - Data storage, [46](#)
- hsm_open_hash_service
 - Hashing, [36](#)
- hsm_open_key_management_service
 - Key management, [16](#)
- hsm_open_key_store_service
 - Key store, [10](#)
- hsm_open_mac_service
 - Mac, [52](#)
- hsm_open_rng_service
 - Random number generation, [33](#)
- hsm_open_session
 - Session, [8](#)
- hsm_open_signature_generation_service
 - Signature generation, [27](#)
- hsm_open_signature_verification_service
 - Signature verification, [30](#)
- hsm_prepare_signature
 - Signature generation, [27](#)
- hsm_pub_key_decompression
 - Public key decompression, [40](#)
- hsm_pub_key_reconstruction
 - Public key reconstruction, [38](#)
- hsm_pub_key_recovery
 - Public key recovery, [44](#)
- hsm_verify_signature
 - Signature verification, [31](#)
- Key management, [12](#)
 - hsm_butterfly_key_expansion, [18](#)
 - hsm_close_key_management_service, [18](#)
 - hsm_generate_key, [16](#)
 - hsm_manage_key, [17](#)
 - hsm_manage_key_group, [17](#)
 - hsm_open_key_management_service, [16](#)
 - Key store, [9](#)
 - hsm_close_key_store_service, [10](#)
 - hsm_open_key_store_service, [10](#)
 - Mac, [51](#)
 - hsm_close_mac_service, [52](#)
 - hsm_mac_one_go, [52](#)
 - hsm_open_mac_service, [52](#)
 - op_auth_enc_args_t, [21](#)
 - op_butl_key_exp_args_t, [15](#)
 - op_cipher_one_go_args_t, [21](#)
 - op_data_storage_args_t, [45](#)
 - op_ecies_dec_args_t, [22](#)
 - op_ecies_enc_args_t, [42](#)
 - op_export_root_kek_args_t, [48](#)
 - op_generate_key_args_t, [14](#)
 - op_generate_sign_args_t, [26](#)
 - op_get_info_args_t, [50](#)
 - op_get_random_args_t, [33](#)
 - op_hash_one_go_args_t, [35](#)
 - op_import_public_key_args_t, [30](#)
 - op_mac_one_go_args_t, [51](#)
 - op_manage_key_args_t, [14](#)
 - op_manage_key_group_args_t, [15](#)
 - op_prepare_sign_args_t, [26](#)
 - op_pub_key_dec_args_t, [40](#)
 - op_pub_key_rec_args_t, [38](#)
 - op_pub_key_recovery_args_t, [44](#)
 - op_verify_sign_args_t, [29](#)
 - open_session_args_t, [7](#)
 - open_svc_cipher_args_t, [20](#)
 - open_svc_data_storage_args_t, [45](#)
 - open_svc_hash_args_t, [35](#)
 - open_svc_key_management_args_t, [14](#)
 - open_svc_key_store_args_t, [9](#)
 - open_svc_mac_args_t, [51](#)
 - open_svc_rng_args_t, [33](#)
 - open_svc_sign_gen_args_t, [26](#)
 - open_svc_sign_ver_args_t, [29](#)
 - Public key decompression, [40](#)
 - hsm_pub_key_decompression, [40](#)
 - Public key reconstruction, [38](#)
 - hsm_pub_key_reconstruction, [38](#)
 - Public key recovery, [44](#)
 - hsm_pub_key_recovery, [44](#)
 - Random number generation, [33](#)
 - hsm_close_rng_service, [34](#)
 - hsm_get_random, [34](#)
 - hsm_open_rng_service, [33](#)
 - Root KEK export, [48](#)
 - hsm_export_root_key_encryption_key, [48](#)
 - Session, [7](#)

- hsm_close_session, [8](#)
- hsm_open_session, [8](#)
- Signature generation, [25](#)
 - hsm_close_signature_generation_service, [27](#)
 - hsm_generate_signature, [27](#)
 - hsm_open_signature_generation_service, [27](#)
 - hsm_prepare_signature, [27](#)
- Signature verification, [29](#)
 - hsm_close_signature_verification_service, [31](#)
 - hsm_import_public_key, [31](#)
 - hsm_open_signature_verification_service, [30](#)
 - hsm_verify_signature, [31](#)