# i.MX8 HSM API Rev 2.9

NXP Copyright

Generated by Doxygen 1.8.15

# 1 HSM API

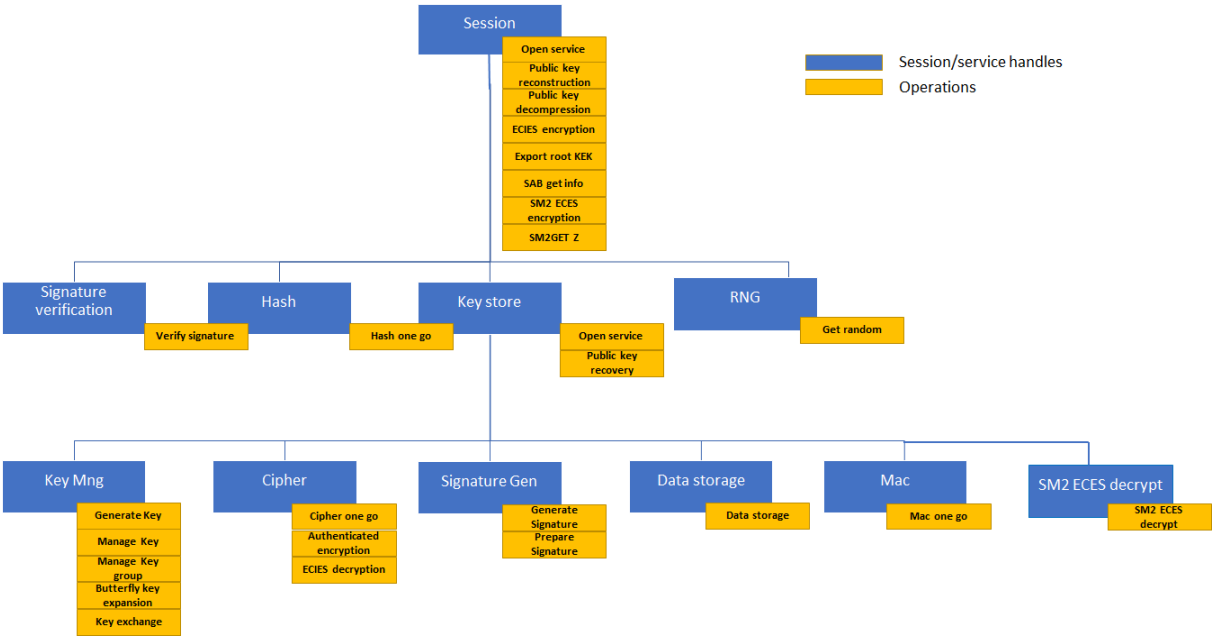
This document is a software referece description of the API provided by the i.MX8 HSM solutions.

# 2 Revision History

| Revision | date | description |
|---|---|---|
| 0.1 | Mar 29 2019 | Preliminary draft |
| 0.8 | May 24 2019 | It adds the following API:<br>-signature generation<br>-signature verification<br>-rng<br>-hash<br>-butterfly key expansion<br>-ECIES enc/dec<br>-public key reconstruction<br>-public key decompression |
| 0.9 | May 28 2019 | Explicit addresses are replaced by pointers. |
| 1.0 | May 29 2019 | - bug/typos fix.<br>- Change HSM_SVC_KEY_STORE_FLAGS definition |
| 1.1 | July 31 2019 | - hsm_butterfly_key_expansion argument definition: dest_key_identifier is now a pointer.<br>- add error code definition.<br>- improve argument comments clarity |
| 1.5 | Sept 13 2019 | - manage key argument: fix padding size<br>- butterfly key expansion: change argument definition<br>- introduce public key recovery API |

| Revision | date | description |
|---|---|---|
| 1.6 | Oct 14 2019 | - add Key store section in chapter 3<br>- change key_info and flags definition, substitute key_type_ext with group_id<br>- hsm_generate_key, hsm_manage_key, hsm_butterfly_key_expansion: change argument definition<br>- hsm_manage_key: change argument definition<br>- add hsm_manage_key_group API |
| 1.7 | Dec 20 2019 | - add generic data storage API<br>- add GCM and CMAC support<br>- add support for AES 192/256 key size for all cipher algorithms<br>- add root KEK export API<br>- add key import functionality<br>- add get info API |
| 2.0 | Feb 21 2020 | - fix HSM_KEY_INFO_TRANSIENT definition: delete erroneous "not supported" comment<br>- add Key Encryption Key (HSM_KEY_INFO_KEK) support<br>- key store open service API: adding signed message support for key store reprovisionning<br>- naming consistency: remove "hsm_" prefix from<br>hsm_op_ecies_dec_args_t<br>hsm_op_pub_key_rec_args_t<br>hsm_op_pub_key_dec_args_t<br>hsm_op_ecies_enc_args_t<br>hsm_op_pub_key_recovery_args_t<br>hsm_op_get_info_args_t |
| 2.1 | Apr 16 2020 | - Preliminary version: Add the support of the chinese algorithms and update for i.MX8DXL |
| 2.2 | Apr 30 2020 | - fix erroneous number of supported key groups (correct number is 1000 while 1024 was indicated)<br>- add missing status code definition<br>- remove hsm_open_key_store_service unused flags: HSM_SVC_KEY_STORE_FLAGS_UPDATE, HSM_SVC_KEY_STORE_FLAGS_DELETE |
| 2.3 | June 30 2020 | - hsm_get_info fips mode definition: now specifying "FIPS mode of operation" and "FIPS certified part" bits.<br>- Update i.MX8QXP specificities section specifying operations disabled when in FIPS approved mode.<br>- Update comments related to cipher_one_go and SM2 ECES APIs for i.MX8DXL |
| 2.4 | July 9 2020 | - clarify support of hsm_import_public key API. |
| 2.5 | July 28 2020 | - add section in "i.MX8QXP specificities" chapter indicating the maximum number of keys per group. |
| 2.6 | Jul 29 2020 | - Key Exchange: add the definition of ECDH_P384 and TLS KDFs<br>- mac_one_go: add definition of HMAC SHA256/384. |
| 2.7 | Sep 25 2020 | - Key Exchange: additional TLS KDFs support, CMAC KDF replaced by SHA-256 KDF<br>- mac_one_go: add support of HMAC SHA224/523. |
| 2.8 | Sep 30 2020 | - Key Exchange: add details related to the SM2 key exchange. |
| 2.9 | Oct 14 2020 | - key_store_open: add STRICT_OPERATION flag. This flag allows to export the key store in the external NVM at the key store creation. |

# 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.

## 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 specyfing 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 monolitic block.
Up to 3 key groups can be handled in the HSM local memory (those immediatly available to perform crypto operations), while up to 1000 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 creating a key store (open key store API) or modyfing its content (key generation, key_management, key derivation functions) provide a "STRICT OPERATION" flag. If the flag is set, the HSM exports the relevant key store blocks into the external NVM and increments (blows one bit) the OTP monotonic counter used as roll back protection. In case of key generation/derivation/update the "STRICT OPERATION" has effect only on the target key

group.
Any update to the key store must be considered as effective only after an operation specifing the flag "STRICT O↩ PERATION" is aknowledged 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 udates 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 key store export to the NVM to inform the user that the new updates are not roll-back protected.

### 3.5 Implementation specificities

HSM API is supported on different versions of the i.MX8 family. The API description below is the same for all of them but some features may not be available on some chips. The details of the supported features per chip can be found here:

- for i.MX8QXP: i.MX8QXP specificities
- for i.MX8DXL: i.MX8DXL specificities

## 4 Module Index

### 4.1 Modules

Here is a list of all modules:

# 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_SELF_TEST_FAILURE = 0x12,
  HSM_NOT_READY_RATING = 0x13,
  HSM_FEATURE_DISABLED = 0x14,
  HSM_GENERAL_ERROR = 0xFF }

### 5.1.1 Detailed Description

### 5.1.2 Enumeration Type Documentation

#### 5.1.2.1 hsm_err_t

`enum hsm_err_t`

Error codes returned by HSM functions.

**Enumerator**

| | |
|---|---|
| HSM_NO_ERROR | Success. |
| HSM_INVALID_MESSAGE | The received message is invalid or unknown. |
| HSM_INVALID_ADDRESS | The provided address is invalid or doesn't respect the API requirements. |
| HSM_UNKNOWN_ID | The provided identifier is not known. |
| HSM_INVALID_PARAM | One of the parameter provided in the command is invalid. |
| HSM_NVM_ERROR | NVM generic issue. |
| HSM_OUT_OF_MEMORY | There is not enough memory to handle the requested operation. |
| HSM_UNKNOWN_HANDLE | Unknown session/service handle. |
| HSM_UNKNOWN_KEY_STORE | The key store identified by the provided "key store Id" doesn't exist and the "create" flag is not set. |

**Enumerator**

| | |
|---|---|
| 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 firwmare. |
| HSM_SELF_TEST_FAILURE | Self tests report an issue |
| HSM_NOT_READY_RATING | The HSM is not ready to handle the current request |
| HSM_FEATURE_DISABLED | The required service/operation is disabled |
| HSM_GENERAL_ERROR | Error not covered by other codes occured. |

## 5.2 Session

### Modules

- i.MX8QXP specificities
- i.MX8DXL specificities

### 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 (1u << 0)

  *Only FIPS certified operations authorized in this session.*
- #define HSM_OPEN_SESSION_EXCLUSIVE_MASK (1u << 1)

  *No other HSM session will be authorized on the same security enclave.*
- #define HSM_OPEN_SESSION_LOW_LATENCY_MASK (1u << 3)

  *Use a low latency HSM implementation.*
- #define HSM_OPEN_SESSION_NO_KEY_STORE_MASK (1u << 4)

  *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 ((1u << 2) | (1u << 5) | (1u << 6) | (1u << 7))

  *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). ∗/. |
| uint16_t | reserved | |

### 5.2.3 Function Documentation

#### 5.2.3.1 hsm_open_session()

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

**Parameters**

| | |
|---|---|
| *args* | pointer to the structure containing the function arguments. |
| *session_hdl* | pointer to where the session handle must be written. |

**Returns**

error_code error code.

#### 5.2.3.2 hsm_close_session()

```
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**

| | |
|---|---|
| *session_hdl* | 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 if the STRICT OPERATION flag is set.*
- #define HSM_SVC_KEY_STORE_FLAGS_STRICT_OPERATION ((hsm_svc_key_store_flags_t)(1u << 7))

  *The request is completed only when the new key store has been written in the NVM. This applicable for CREATE operations only.*

### 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.<br>This parameter has the goal to limit the occupation of the monotonic counter used as anti-rollback protection.<br>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 | it must be 0. |
| 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_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. Only one key store service can be opened on a given key store.

**Parameters**

| | |
|---|---|
| *session_hdl* | pointer to the handle identifying the current session. |
| *args* | pointer to the structure containing the function arguments. |
| *key_store_hdl* | pointer to where the key store service flow handle must be written. |

**Returns**

error_code error code.

#### 5.3.3.2 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. The key store is deleted from the HSM local memory, any update not written in the NVM is lost

**Parameters**

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

**Returns**

error_code error code.

## 5.4 Key management

### Modules

- i.MX8QXP specificities
- i.MX8DXL specificities

### Data Structures

- struct open_svc_key_management_args_t
- struct op_generate_key_args_t
- struct op_manage_key_args_t
- struct op_manage_key_group_args_t
- struct op_butt_key_exp_args_t

### Macros

- #define **HSM_KEY_TYPE_ECDSA_NIST_P256** ((hsm_key_type_t)0x02u)
- #define **HSM_KEY_TYPE_ECDSA_NIST_P384** ((hsm_key_type_t)0x03u)
- #define **HSM_KEY_TYPE_ECDSA_NIST_P521** ((hsm_key_type_t)0x04u)
- #define **HSM_KEY_TYPE_ECDSA_BRAINPOOL_R1_256** ((hsm_key_type_t)0x13u)
- #define **HSM_KEY_TYPE_ECDSA_BRAINPOOL_R1_320** ((hsm_key_type_t)0x14u)
- #define **HSM_KEY_TYPE_ECDSA_BRAINPOOL_R1_384** ((hsm_key_type_t)0x15u)
- #define **HSM_KEY_TYPE_ECDSA_BRAINPOOL_R1_512** ((hsm_key_type_t)0x16u)
- #define **HSM_KEY_TYPE_ECDSA_BRAINPOOL_T1_256** ((hsm_key_type_t)0x23u)
- #define **HSM_KEY_TYPE_ECDSA_BRAINPOOL_T1_320** ((hsm_key_type_t)0x24u)
- #define **HSM_KEY_TYPE_ECDSA_BRAINPOOL_T1_384** ((hsm_key_type_t)0x25u)
- #define **HSM_KEY_TYPE_ECDSA_BRAINPOOL_T1_512** ((hsm_key_type_t)0x26u)
- #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_KEY_TYPE_DSA_SM2_FP_256** ((hsm_key_type_t)0x42u)
- #define **HSM_KEY_TYPE_SM4_128** ((hsm_key_type_t)0x50u)
- #define **HSM_KEY_TYPE_HMAC_224** ((hsm_key_type_t)0x60u)
- #define **HSM_KEY_TYPE_HMAC_256** ((hsm_key_type_t)0x61u)
- #define **HSM_KEY_TYPE_HMAC_384** ((hsm_key_type_t)0x62u)
- #define **HSM_KEY_TYPE_HMAC_512** ((hsm_key_type_t)0x63u)
- #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_mange↩ _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_KEK ((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_KEK ((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 is only applicable for persistent key.*

- #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))

  *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_flags_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_flags_t | flags | bitmap specifying the operation properties. |
| hsm_key_type_t | key_type | indicates which type of key must be generated. |
| hsm_key_group_t | key_group | Key group of the generated key, 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 throug 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. |
| 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 hsm_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 hsm_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_butt_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.<br>In case of explicit certificate, the hash value address must be set to 0. |

**Data Fields**

| | | |
|---|---|---|
| uint8_t * | pr_reconstruction_value | pointer to the private reconstruction value input.<br>In case of explicit certificate, the pr_reconstruction_value address must be set to 0. |
| uint8_t | expansion_function_value_size | length in bytes of the expansion function input |
| uint8_t | hash_value_size | length in bytes of the hash value input.<br>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.<br>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.<br>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.<br>Keys belonging to the same group can be cached in the HSM local memory throug the hsm_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_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**

| | |
|---|---|
| *key_store_hdl* | handle identifying 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_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**

| | |
|---|---|
| *key_management_hdl* | handle identifying the key management service flow. |
| *args* | pointer to the structure containing the function arguments. |

**Returns**

error code

#### 5.4.3.3 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 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**

| | |
|---|---|
| *key_management_hdl* | handle identifying the key management service flow. |
| *args* | pointer to the structure containing the function arguments. |

**Returns**

error code

#### 5.4.3.4 hsm_manage_key_group()

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

This command is designed to perform the following operations:

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

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

**Parameters**

| | |
|---|---|
| *key_management_hdl* | handle identifying the key management service flow. |
| *args* | pointer to the structure containing the function arguments. |

**Returns**

error code

#### 5.4.3.5 hsm_butterfly_key_expansion()

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

This command is designed to perform the butterfly key expansion operation on an ECC private key in case of implicit and explicit certificates. Optionally the resulting public key is exported.
The result of the key expansion function f_k is calculated outside the HSM and passed as input. The expansion function is defined as f_k = f_k_int mod l , where l 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 in the derivation of the pseudonym ECC key,
- pr_v = private reconstruction value

out_key = (Key + f_k)∗hash + pr_v

**Parameters**

| | |
|---|---|
| *key_management_hdl* | handle identifying the key store management service flow. |
| *args* | pointer to the structure containing the function arguments. |

**Returns**

error code

#### 5.4.3.6 hsm_close_key_management_service()

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

Terminate a previously opened key management service flow

**Parameters**

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

**Returns**

error code

## 5.5 Ciphering

### Modules

- i.MX8QXP specificities
- i.MX8DXL specificities

### 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_ALGO_SM4_ECB** ((hsm_op_cipher_one_go_algo_t)(0x10u))
- #define **HSM_CIPHER_ONE_GO_ALGO_SM4_CBC** ((hsm_op_cipher_one_go_algo_t)(0x11u))
- #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<br>it must be 0 for algorithms not using the initialization vector.<br>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<br>plaintext for encryption<br>ciphertext for decryption (in case of CCM is the purported ciphertext) |
| uint8_t * | output | pointer to the output area<br>ciphertext for encryption (in case of CCM is the output of the generation-encryption process)<br>plaintext for decryption |
| uint32_t | input_size | length in bytes of the input. In case of CBC and ECB, the input size should be multiple of a block cipher size (16 bytes). |
| 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<br>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<br>plaintext for encryption<br>Ciphertext + Tag (16 bytes) for decryption |
| uint8_t * | output | pointer to the output area<br>Ciphertext + Tag (16 bytes) for encryption<br>plaintext for decryption if the Tag is verified |
| uint32_t | input_size | length in bytes of the input |
| 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 |
| hsm_op_ecies_dec_flags_t | flags | bitmap specifying the operation attributes. |

### 5.5.3 Function Documentation

#### 5.5.3.1 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**

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

**Returns**

error code

#### 5.5.3.2 hsm_cipher_one_go()

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

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

**Parameters**

| | |
|---|---|
| *cipher_hdl* | handle identifying the cipher service flow. |
| *args* | pointer to the structure containing the function arguments. |

**Returns**

error code

#### 5.5.3.3 hsm_auth_enc()

```
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**

| | |
|---|---|
| *cipher_hdl* | handle identifying the cipher service flow. |
| *args* | pointer to the structure containing the function arguments. |

**Returns**

error code

#### 5.5.3.4 hsm_ecies_decryption()

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

Decrypt data usign ECIES
User can call this function only after having opened a cipher store service flow.
ECIES is supported with the constraints specified in 1609.2-2016.

**Parameters**

| | |
|---|---|
| *session_hdl* | handle identifying the current session. |
| *args* | pointer to the structure containing the function arguments. |

**Returns**

error code

#### 5.5.3.5 hsm_close_cipher_service()

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

Terminate a previously opened cipher service flow

**Parameters**

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

**Returns**

error code

## 5.6 Signature generation

### Modules

- i.MX8QXP specificities
- i.MX8DXL specificities

### 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_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_NIST_P521_SHA_512** ((hsm_signature_scheme_id_t)0x04u)
- #define **HSM_SIGNATURE_SCHEME_ECDSA_BRAINPOOL_R1_256_SHA_256** ((hsm_signature_scheme_id_t)0x13u)
- #define **HSM_SIGNATURE_SCHEME_ECDSA_BRAINPOOL_R1_320_SHA_384** ((hsm_signature_scheme_id_t)0x14u)
- #define **HSM_SIGNATURE_SCHEME_ECDSA_BRAINPOOL_R1_384_SHA_384** ((hsm_signature_scheme_id_t)0x15u)
- #define **HSM_SIGNATURE_SCHEME_ECDSA_BRAINPOOL_R1_512_SHA_512** ((hsm_signature_scheme_id_t)0x16u)
- #define **HSM_SIGNATURE_SCHEME_ECDSA_BRAINPOOL_T1_256_SHA_256** ((hsm_signature_scheme_id_t)0x23u)
- #define **HSM_SIGNATURE_SCHEME_ECDSA_BRAINPOOL_T1_320_SHA_384** ((hsm_signature_scheme_id_t)0x24u)
- #define **HSM_SIGNATURE_SCHEME_ECDSA_BRAINPOOL_T1_384_SHA_384** ((hsm_signature_scheme_id_t)0x25u)
- #define **HSM_SIGNATURE_SCHEME_ECDSA_BRAINPOOL_T1_512_SHA_512** ((hsm_signature_scheme_id_t)0x26u)
- #define **HSM_SIGNATURE_SCHEME_DSA_SM2_FP_256_SM3** ((hsm_signature_scheme_id_t)0x43u)
- #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**

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

#### 5.6.2.2 struct op_generate_sign_args_t

**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\|\|Ry where Ry is an additional byte containing the lsb of y. Ry has to be considered valid only if the HSM_OP_GENER↩ ATE_SIGN_FLAGS_COMPRESSED_POINT is set. |
| uint32_t | message_size | length in bytes of the input |
| uint16_t | signature_size | length in bytes of the output |
| hsm_signature_scheme_id_t | scheme_id | identifier of the digital signature scheme to be used for the operation |
| hsm_op_generate_sign_flags_t | flags | bitmap specifying the operation attributes |

#### 5.6.2.3 struct op_prepare_sign_args_t

**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 | |

### 5.6.3 Function Documentation

#### 5.6.3.1 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**

| | |
|---|---|
| *key_store_hdl* | handle identifying the key store service flow. |
| *args* | pointer to the structure containing the function arguments. |
| *signature_gen_hdl* | pointer to where the signature generation service flow handle must be written. |

**Returns**

error code

#### 5.6.3.2 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**

| | |
|---|---|
| *signature_gen_hdl* | handle identifying the signature generation service flow to be closed. |

**Returns**

error code

#### 5.6.3.3 hsm_generate_signature()

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

Generate a digital signature according to the signature scheme
User can call this function only after having opened a signature generation service flow
The signature S=(r,s) is stored in the format r||s||Ry where Ry is an additional byte containing the lsb of y. Ry has to be considered valid only if the HSM_OP_GENERATE_SIGN_FLAGS_COMPRESSED_POINT is set.

In case of HSM_SIGNATURE_SCHEME_DSA_SM2_FP_256_SM3, message of op_generate_sign_args_t should be (as specified in GB/T 32918):

- equal to Z||M in case of HSM_OP_GENERATE_SIGN_FLAGS_INPUT_MESSAGE
- equal to SM3(Z||M) in case of HSM_OP_GENERATE_SIGN_FLAGS_INPUT_DIGEST

**Parameters**

| | |
|---|---|
| *signature_gen_hdl* | handle identifying the signature generation service flow. |
| *args* | pointer to the structure containing the function arguments. |

**Returns**

error code

#### 5.6.3.4 hsm_prepare_signature()

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

Prepare the creation of a signature by pre-calculating the operations having not dependencies on the input message. The pre-calculated value will be stored internally and used once call hsm_generate_signature
User can call this function only after having opened a signature generation service flow
The signature S=(r,s) is stored in the format r||s||Ry where Ry is an additional byte containing the lsb of y, Ry has to be considered valid only if the HSM_OP_PREPARE_SIGN_COMPRESSED_POINT is set.

**Parameters**

| | |
|---|---|
| *signature_gen_hdl* | handle identifying the signature generation service flow |
| *args* | pointer to the structure containing the function arguments. |

**Returns**

error code

## 5.7 Signature verification

### Modules

- i.MX8QXP specificities
- i.MX8DXL specificities

### 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 contain 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_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**

| | |
|---|---|
| *session_hdl* | handle identifying 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_verify_signature()

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

Verify a digital signature according to the signature scheme
User can call this function only after having opened a signature verification service flow
The signature S=(r,s) is expected to be in format r||s||Ry where Ry is an additional byte containing the lsb of y. Ry will be considered as valid only if the HSM_OP_VERIFY_SIGN_FLAGS_COMPRESSED_POINT is set.
Only not-compressed keys (x,y) can be used by this command. Compressed keys can be decompressed by using the dedicated API.

In case of HSM_SIGNATURE_SCHEME_DSA_SM2_FP_256_SM3, message of op_verify_sign_args_t should be (as specified in GB/T 32918):

- equal to Z||M in case of HSM_OP_VERIFY_SIGN_FLAGS_INPUT_MESSAGE
- equal to SM3(Z||M) in case of HSM_OP_VERIFY_SIGN_FLAGS_INPUT_DIGEST

**Parameters**

| | |
|---|---|
| *signature_ver_hdl* | handle identifying the signature verification service flow. |
| *args* | pointer to the structure containing the function arguments. |
| *status* | pointer to where the verification status must be stored<br>if the verification suceed the value HSM_VERIFICATION_STATUS_SUCCESS is returned. |

**Returns**

error code

#### 5.7.3.3 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_F↩ LAGS_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**

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

**Returns**

error code

#### 5.7.3.4 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**

| | |
|---|---|
| *signature_ver_hdl* | 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_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**

| | |
|---|---|
| *session_hdl* | handle identifying the current session. |
| *args* | pointer to the structure containing the function arguments. |
| *rng_hdl* | pointer to where the rng service flow handle must be written. |

**Returns**

error code

#### 5.8.3.2 hsm_close_rng_service()

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

Terminate a previously opened rng service flow

**Parameters**

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

**Returns**

error code

#### 5.8.3.3 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**

| | |
|---|---|
| *rng_hdl* | handle identifying the rng service flow. |
| *args* | pointer to the structure containing the function arguments. |

**Returns**

error code

## 5.9 Hashing

### Modules

- i.MX8QXP specificities

### 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))
- #define **HSM_HASH_ALGO_SM3_256** ((hsm_hash_algo_t)(0x11u))

### 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 |
| 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_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 hash operations.

**Parameters**

| | |
|---|---|
| *session_hdl* | handle identifying the current session. |
| *args* | pointer to the structure containing the function arguments. |
| *hash_hdl* | pointer to where the hash service flow handle must be written. |

**Returns**

error code

#### 5.9.3.2 hsm_close_hash_service()

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

Terminate a previously opened hash service flow

**Parameters**

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

**Returns**

error code

#### 5.9.3.3 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**

| | |
|---|---|
| *hash_hdl* | handle identifying the hash service flow. |
| *args* | pointer to the structure containing the function arguments. |

**Returns**

error code

## 5.10 Public key reconstruction

### Modules

- i.MX8QXP specificities
- i.MX8DXL specificities

### 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 managed key. |
| 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_pub_key_reconstruction()

```
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**

| | |
|---|---|
| *session_hdl* | handle identifying the current session. |
| *args* | pointer to the structure containing the function arguments. |

**Returns**

error code

## 5.11 Public key decompression

### Modules

- i.MX8QXP specificities

### 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 manged 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_pub_key_decompression()

```
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**

| | |
|---|---|
| *session_hdl* | handle identifying the current session. |
| *args* | pointer to the structure containing the function arguments. |

**Returns**

error code

## 5.12 ECIES encryption

### Modules

- i.MX8QXP specificities
- i.MX8DXL specificities

### 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 |
| 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_ecies_encryption()

```
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**

| | |
|---|---|
| *session_hdl* | handle identifying the current session. |
| *args* | 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_pub_key_recovery()

```
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**

| | |
|---|---|
| *key_store_hdl* | handle identifying the current key store. |
| *args* | 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 |
| 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_open_data_storage_service()

```
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/retreive generic data in/from the HSM.

**Parameters**

| | |
|---|---|
| *key_store_hdl* | handle identifying the key store service flow. |
| *args* | pointer to the structure containing the function arguments. |
| *data_storage_hdl* | pointer to where the data storage service flow handle must be written. |

**Returns**

error_code error code.

#### 5.14.3.2 hsm_data_storage()

```
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**

| | |
|---|---|
| *data_storage_hdl* | handle identifying the data storage service flow. |
| *args* | pointer to the structure containing the function arguments. |

**Returns**

error code

#### 5.14.3.3 hsm_close_data_storage_service()

```
hsm_err_t hsm_close_data_storage_service (
            hsm_hdl_t data_storage_hdl )
```

Terminate a previously opened data storage service flow

**Parameters**

| | |
|---|---|
| *data_storage_hdl* | 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_export_root_key_encryption_key()

```
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**

| | |
|---|---|
| *session_hdl* | handle identifying the current session. |
| *args* | 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 bitfield (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 bitfield (8bits) must be written. Bitmask definition:<br>bit0 - FIPS mode of operation:<br>- value 0 - part is running in FIPS non-approved mode.<br>- value 1 - part is running in FIPS approved mode.<br>bit1 - FIPS certified part:<br>- value 0 - part is not FIPS certified.<br>- value 1 - part is FIPS certified.<br>bit2-7: reserved - 0 value. |

#### 5.16.3 Function Documentation

##### 5.16.3.1 hsm_get_info()

```
hsm_err_t hsm_get_info (
            hsm_hdl_t session_hdl,
            op_get_info_args_t * args )
```

**Parameters**

| | |
|---|---|
| *session_hdl* | handle identifying the current session. |
| *args* | pointer to the structure containing the function arguments. |

**Returns**

error code

## 5.17 Mac

### Modules

- i.MX8DXL specificities

### 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_OP_MAC_ONE_GO_ALGO_HMAC_SHA_224** ((hsm_op_mac_one_go_algo_t)(0x05u))
- #define **HSM_OP_MAC_ONE_GO_ALGO_HMAC_SHA_256** ((hsm_op_mac_one_go_algo_t)(0x06u))
- #define **HSM_OP_MAC_ONE_GO_ALGO_HMAC_SHA_384** ((hsm_op_mac_one_go_algo_t)(0x07u))
- #define **HSM_OP_MAC_ONE_GO_ALGO_HMAC_SHA_512** ((hsm_op_mac_one_go_algo_t)(0x08u))
- #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<br>For CMAC the value must be between 4 and 16 bytes.<br>For HMAC the value must be between 4 and the hash size. |

### 5.17.3 Function Documentation

#### 5.17.3.1 hsm_open_mac_service()

```
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**

| | |
|---|---|
| *key_store_hdl* | handle identifying the key store service flow. |
| *args* | pointer to the structure containing the function arguments. |
| *mac_hdl* | pointer to where the mac service flow handle must be written. |

**Returns**

error code

#### 5.17.3.2 hsm_mac_one_go()

```
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
For CMAC algorithm, a key of type HSM_KEY_TYPE_AES_XXX must be used
For HMAC algorithm, a key of type HSM_KEY_TYPE_HMAC_XXX must be used

**Parameters**

| | |
|---|---|
| *mac_hdl* | handle identifying the mac service flow. |
| *args* | pointer to the structure containing the function arguments. |

**Returns**

error code

#### 5.17.3.3 hsm_close_mac_service()

```
hsm_err_t hsm_close_mac_service (
            hsm_hdl_t mac_hdl )
```

Terminate a previously opened mac service flow

**Parameters**

| | |
|---|---|
| *mac_hdl* | pointer to handle identifying the mac service flow to be closed. |

**Returns**

error code

## 5.18 SM2 Get Z

### Modules

- i.MX8QXP specificities

### Data Structures

- struct op_sm2_get_z_args_t

### Typedefs

- typedef uint8_t **hsm_op_sm2_get_z_flags_t**

### Functions

- hsm_err_t hsm_sm2_get_z (hsm_hdl_t session_hdl, op_sm2_get_z_args_t *args)

#### 5.18.1 Detailed Description

#### 5.18.2 Data Structure Documentation

##### 5.18.2.1 struct op_sm2_get_z_args_t

**Data Fields**

| | | |
|---|---|---|
| uint8_t * | public_key | pointer to the sender public key |
| uint8_t * | identifier | pointer to the sender identifier |
| uint8_t * | z_value | pointer to the output area where the Z value must be written |
| uint16_t | public_key_size | length in bytes of the sender public key should be equal to 64 bytes |
| uint8_t | id_size | length in bytes of the identifier |
| uint8_t | z_size | length in bytes of Z should be at least 32 bytes |
| hsm_key_type_t | key_type | indicates the type of the sender public key. Only HSM_KEY_TYPE_DSA_SM2_FP_256 is supported. |
| hsm_op_sm2_get_z_flags_t | flags | bitmap specifying the operation attributes. |
| uint8_t | reserved[2] | |

#### 5.18.3 Function Documentation

##### 5.18.3.1 hsm_sm2_get_z()

```
hsm_err_t hsm_sm2_get_z (
            hsm_hdl_t session_hdl,
            op_sm2_get_z_args_t * args )
```

This command is designed to compute Z = SM3(Entl || ID || a || b || xG || yG || xpubk || ypubk)

- ID, Entl: user distinguishing identifier and length,
- a, b, xG and yG : curve parameters,
- xpubk , ypubk : public key

  This value is used for SM2 public key cryptography algorithms, as specified in GB/T 32918. User can call this function only after having opened a session.

**Parameters**

| | |
|---|---|
| *session_hdl* | handle identifying the current session. |
| *args* | pointer to the structure containing the function arguments. |

**Returns**

error code

## 5.19 SM2 ECES decryption

### Modules

- i.MX8QXP specificities
- i.MX8DXL specificities

### Data Structures

- struct open_svc_sm2_eces_args_t
- struct op_sm2_eces_dec_args_t

### Typedefs

- typedef uint8_t **hsm_svc_sm2_eces_flags_t**
- typedef uint8_t **hsm_op_sm2_eces_dec_flags_t**

### Functions

- hsm_err_t hsm_open_sm2_eces_service (hsm_hdl_t key_store_hdl, open_svc_sm2_eces_args_t *args, hsm_hdl_t *sm2_eces_hdl)
- hsm_err_t hsm_close_sm2_eces_service (hsm_hdl_t sm2_eces_hdl)
- hsm_err_t hsm_sm2_eces_decryption (hsm_hdl_t sm2_eces_hdl, op_sm2_eces_dec_args_t *args)

### 5.19.1 Detailed Description

### 5.19.2 Data Structure Documentation

#### 5.19.2.1 struct open_svc_sm2_eces_args_t

**Data Fields**

| | | |
|---|---|---|
| hsm_svc_sm2_eces_flags_t | flags | bitmap indicating the service flow properties |
| uint8_t | reserved[3] | |

#### 5.19.2.2 struct op_sm2_eces_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 input ciphertext |
| uint8_t * | output | pointer to the output area where the plaintext must be written |
| uint32_t | input_size | length in bytes of the input ciphertext. |
| uint32_t | output_size | length in bytes of the output plaintext |
| hsm_key_type_t | key_type | indicates the type of the used key. Only HSM_KEY_TYPE_DSA_SM2_FP_256 is supported. |

**Data Fields**

| | | |
|---|---|---|
| hsm_op_sm2_eces_dec_flags_t | flags | bitmap specifying the operation attributes. |
| uint16_t | reserved | |

### 5.19.3 Function Documentation

#### 5.19.3.1 hsm_open_sm2_eces_service()

```
hsm_err_t hsm_open_sm2_eces_service (
            hsm_hdl_t key_store_hdl,
            open_svc_sm2_eces_args_t * args,
            hsm_hdl_t * sm2_eces_hdl )
```

Open a SM2 ECES decryption service flow
User can call this function only after having opened a key store.
User must open this service in order to perform SM2 decryption.

**Parameters**

| | |
|---|---|
| *session_hdl* | handle identifying the current session. |
| *args* | pointer to the structure containing the function arguments. |
| *sm2_eces_hdl* | pointer to where the sm2 eces service flow handle must be written. |

**Returns**

error code

#### 5.19.3.2 hsm_close_sm2_eces_service()

```
hsm_err_t hsm_close_sm2_eces_service (
            hsm_hdl_t sm2_eces_hdl )
```

Terminate a previously opened SM2 ECES service flow

**Parameters**

| | |
|---|---|
| *sm2_eces_hdl* | handle identifying the SM2 ECES service flow to be closed. |

**Returns**

error code

#### 5.19.3.3 hsm_sm2_eces_decryption()

```
hsm_err_t hsm_sm2_eces_decryption (
            hsm_hdl_t sm2_eces_hdl,
            op_sm2_eces_dec_args_t * args )
```

Decrypt data usign SM2 ECES
User can call this function only after having opened a SM2 ECES service flow.
SM2 ECES is supported with the requirements specified in the GB/T 32918.4.

**Parameters**

| | |
|---|---|
| *sm2_eces_hdl* | handle identifying the SM2 ECES |
| *args* | pointer to the structure containing the function arguments. |

**Returns**

error code

## 5.20 SM2 ECES encryption

### Modules

- i.MX8QXP specificities
- i.MX8DXL specificities

### Data Structures

- struct op_sm2_eces_enc_args_t

### Typedefs

- typedef uint8_t **hsm_op_sm2_eces_enc_flags_t**

### Functions

- hsm_err_t hsm_sm2_eces_encryption (hsm_hdl_t session_hdl, op_sm2_eces_enc_args_t *args)

#### 5.20.1 Detailed Description

#### 5.20.2 Data Structure Documentation

##### 5.20.2.1 struct op_sm2_eces_enc_args_t

**Data Fields**

| | | |
|---|---|---|
| uint8_t * | input | pointer to the input plaintext |
| uint8_t * | output | pointer to the output area where the ciphertext must be written |
| uint8_t * | pub_key | pointer to the input recipient public key |
| uint32_t | input_size | length in bytes of the input plaintext |
| uint32_t | output_size | length in bytes of the output ciphertext.<br>It should be at least input_size + 97 bytes (overhead related to C1 and C3 - as specifed below) + size alignment constraints specific to a given implementation (see related chapter). |
| uint16_t | pub_key_size | length in bytes of the recipient public key should be equal to 64 bytes |
| hsm_key_type_t | key_type | indicates the type of the recipient public key. Only HSM_KEY_TYPE_DSA_SM2_FP_256 is supported. |
| hsm_op_sm2_eces_enc_flags_t | flags | bitmap specifying the operation attributes. |

#### 5.20.3 Function Documentation

#### 5.20.3.1 hsm_sm2_eces_encryption()

```
hsm_err_t hsm_sm2_eces_encryption (
            hsm_hdl_t session_hdl,
            op_sm2_eces_enc_args_t * args )
```

Encrypt data usign SM2 ECES
User can call this function only after having opened a session.
SM2 ECES is supported with the requirements specified in the GB/T 32918.4.
The output (i.e. ciphertext) is stored in the format C= C1||C2||C3 :
C1 = PC||x1||y1 where PC=04 and (x1,y1) are the coordinates of a an elliptic curve point
C2 = M xor t where t=KDF(x2||y2, input_size) and (x2,y2) are the coordinates of a an elliptic curve point
C3 = SM3 (x2||M||y2)

**Parameters**

| | |
|---|---|
| *session_hdl* | handle identifying the current session. |
| *args* | pointer to the structure containing the function arguments. |

**Returns**

error code

## 5.21 Key exchange

### Modules

- i.MX8QXP specificities
- i.MX8DXL specificities

### Data Structures

- struct op_key_exchange_args_t

### Macros

- #define **HSM_KDF_ALG_FOR_SM2** ((hsm_kdf_algo_id_t)0x10u)
- #define HSM_KDF_HMAC_SHA_256_TLS_0_16_4 ((hsm_kdf_algo_id_t)0x20u)

  *TLS PRF based on HMAC with SHA-256, the resulting mac_key_length is 0 bytes, enc_key_length is 16 bytes and fixed_iv_length is 4 bytes.*
- #define HSM_KDF_HMAC_SHA_384_TLS_0_32_4 ((hsm_kdf_algo_id_t)0x21u)

  *TLS PRF based on HMAC with SHA-384, the resulting mac_key_length is 0 bytes, enc_key_length is 32 bytes and fixed_iv_length is 4 bytes.*
- #define HSM_KDF_HMAC_SHA_256_TLS_0_32_4 ((hsm_kdf_algo_id_t)0x22u)

  *TLS PRF based on HMAC with SHA-256, the resulting mac_key_length is 0 bytes, enc_key_length is 32 bytes and fixed_iv_length is 4 bytes.*
- #define HSM_KDF_HMAC_SHA_256_TLS_32_16_4 ((hsm_kdf_algo_id_t)0x23u)

  *TLS PRF based on HMAC with SHA-256, the resulting mac_key_length is 32 bytes, enc_key_length is 16 bytes and fixed_iv_length is 4 bytes.*
- #define HSM_KDF_HMAC_SHA_384_TLS_48_32_4 ((hsm_kdf_algo_id_t)0x24u)

  *TLS PRF based on HMAC with SHA-384, the resulting mac_key_length is 48 bytes, enc_key_length is 32 bytes and fixed_iv_length is 4 bytes.*
- #define HSM_KDF_ALG_SHA_256 ((hsm_kdf_algo_id_t)0x31u)

  *SHA KDF can only be used to generate KEKs (key encryption keys) for key injection (hsm_manage_key API)*
- #define **HSM_KE_SCHEME_ECDH_NIST_P256** ((hsm_key_exchange_scheme_id_t)0x02u)
- #define **HSM_KE_SCHEME_ECDH_NIST_P384** ((hsm_key_exchange_scheme_id_t)0x03u)
- #define **HSM_KE_SCHEME_SM2_FP_256** ((hsm_key_exchange_scheme_id_t)0x42u)
- #define HSM_OP_KEY_EXCHANGE_FLAGS_UPDATE ((hsm_op_key_exchange_flags_t)(1u << 0))

  *User can replace an existing key only by the derived key which should have the same type of the original one.*
- #define HSM_OP_KEY_EXCHANGE_FLAGS_CREATE ((hsm_op_key_exchange_flags_t)(1u << 1))

  *Create a new key.*
- #define HSM_OP_KEY_EXCHANGE_FLAGS_USE_EPHEMERAL ((hsm_op_key_exchange_flags_t)(1u << 2))

  *Use an ephemeral key (freshly generated key)*
- #define HSM_OP_KEY_EXCHANGE_FLAGS_KEY_CONF_EN ((hsm_op_key_exchange_flags_t)(1u << 3))

  *Enable key confirmation (valid only in case of HSM_KE_SCHEME_SM2_FP_256)*
- #define HSM_OP_KEY_EXCHANGE_FLAGS_STRICT_OPERATION ((hsm_op_key_exchange_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.*

### Typedefs

- typedef uint8_t **hsm_kdf_algo_id_t**
- typedef uint8_t **hsm_key_exchange_scheme_id_t**
- typedef uint8_t **hsm_op_key_exchange_flags_t**

#### Functions

- hsm_err_t hsm_key_exchange (hsm_hdl_t key_management_hdl, op_key_exchange_args_t *args)

### 5.21.1 Detailed Description

### 5.21.2 Data Structure Documentation

#### 5.21.2.1 struct op_key_exchange_args_t

#### Data Fields

| | | |
|---|---|---|
| uint32_t | key_identifier | identifier of the key used for derivation. It must be zero, if HSM_OP_KEY_EXCHANGE_FLAGS_USE_EPHEMERAL is set. |
| uint8_t * | shared_key_identifier_array | pointer to the identifiers of the derived keys. In case of create operation the new destination key identifiers will be stored in this location. In case of update operation the destination key identifiers to update are provided by the caller in this location. In case of HSM_KDF_ALG_SHA_256 it contains the KEK key id. In case of HSM_KDF_HMAC_SHA_256_TLS_0_16_4, HSM_KDF_HMAC_SHA_384_TLS_0_32_4 or HSM_KDF_HMAC_SHA_256_TLS_0_32_4 KDF it contains the concatenation of client_write_key id (4 bytes) and the server_write_key id (4 bytes). In case of HSM_KDF_HMAC_SHA_256_TLS_32_16_4 or HSM_KDF_HMAC_SHA_384_TLS_48_32_4 KDF it contains the concatenation of client_write_MAC_key id (4 bytes), server_write_MAC_key id (4 bytes), client_write_key id (4 bytes) and the server_write_key id (4 bytes). |
| uint8_t * | ke_input | pointer to the initiator input data related to the key exchange function. |
| uint8_t * | ke_output | pointer to the output area where the data related to the key exchange function must be written. It corresponds to the receiver public data. |

**Data Fields**

| | | |
|---|---|---|
| uint8_t * | kdf_input | pointer to the input data of the KDF.<br>In case of HSM_KDF_HMAC_S↩<br>HA_256_TLS_0_16_4,<br>HSM_KDF_HMAC_SHA_384_T↩<br>LS_0_32_4 KDF,<br>HSM_KDF_HMAC_SHA_256_T↩<br>LS_0_32_4,<br>HSM_KDF_HMAC_SHA_256_T↩<br>LS_32_16_4 or<br>HSM_KDF_HMAC_SHA_384_T↩<br>LS_48_32_4 it must contain to the concatenarion of clientHello_random (32 bytes), serverHello_random (32 bytes), server_random (32 bytes) and client_random (32 bytes), it must be 0 otherwise |
| uint8_t * | kdf_output | pointer to the output area where the non sensitive output data related to the KDF are written. In case of HSM_KDF_HMAC_SHA↩<br>_256_TLS_0_16_4,<br>HSM_KDF_HMAC_SHA_384_T↩<br>LS_0_32_4 KDF,<br>HSM_KDF_HMAC_SHA_256_T↩<br>LS_0_32_4,<br>HSM_KDF_HMAC_SHA_256_T↩<br>LS_32_16_4 or<br>HSM_KDF_HMAC_SHA_384_T↩<br>LS_48_32_4 KDF the concatenation of client_write_iv (4 bytes) and server_write_iv (4 bytes) will be stored at this address, it must be 0 otherwise |
| hsm_key_group_t | shared_key_group | It specifies the group where the derived keys will be stored.<br>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 hsm_manage_key_group API. |
| hsm_key_info_t | shared_key_info | bitmap specifying the properties of the derived keys, it will be applied to all the derived keys. |

**Data Fields**

| | | |
|---|---|---|
| hsm_key_type_t | shared_key_type | indicates the type of the derived key.<br>In case of HSM_KDF_ALG_SHA_256 it must be HSM_KEY_TYPE_AES_256.<br>Not relevant in case of HSM_KDF_HMAC_SHA_256_TLS_0_16_4, HSM_KDF_HMAC_SHA_384_TLS_0_32_4 KDF, HSM_KDF_HMAC_SHA_256_TLS_0_32_4, HSM_KDF_HMAC_SHA_256_TLS_32_16_4 or HSM_KDF_HMAC_SHA_384_TLS_48_32_4 KDF. |
| hsm_key_type_t | initiator_public_data_type | indicates the public data type specified by the initiator, e.g. public key type.<br>For SHA KDF, this must be HSM_KEY_TYPE_ECDSA_NIST_P256.<br>For HMAC KDF, this can be HSM_KEY_TYPE_ECDSA_NIST_P256 or HSM_KEY_TYPE_ECDSA_NIST_P384. |
| hsm_key_exchange_scheme_id_t | key_exchange_scheme | indicates the key exchange scheme |
| hsm_kdf_algo_id_t | kdf_algorithm | indicates the KDF algorithm |
| uint16_t | ke_input_size | length in bytes of the input data of the key exchange function |
| uint16_t | ke_output_size | length in bytes of the output data of the key exchange function |
| uint8_t | shared_key_identifier_array_size | length in byte of the area containing the shared key identifiers |
| uint8_t | kdf_input_size | length in bytes of the input data of the KDF. It must be 128 bytes in case of HSM_KDF_HMAC_SHA_256_TLS_0_16_4, HSM_KDF_HMAC_SHA_384_TLS_0_32_4 KDF, HSM_KDF_HMAC_SHA_256_TLS_0_32_4 KDF, HSM_KDF_HMAC_SHA_256_TLS_32_16_4 or HSM_KDF_HMAC_SHA_384_TLS_48_32_4, 0 otherwise. |

**Data Fields**

| | | |
|---|---|---|
| uint8_t | kdf_output_size | length in bytes of the non sensitive output data related to the KDF. It must be 8 bytes in case of HSM_KDF_HMAC_SHA_256_TLS_0_16_4, HSM_KDF_HMAC_SHA_384_TLS_0_32_4, HSM_KDF_HMAC_SHA_256_TLS_0_32_4 KDF, HSM_KDF_HMAC_SHA_256_TLS_32_16_4 or HSM_KDF_HMAC_SHA_384_TLS_48_32_4 KDF, 0 otherwise. |
| hsm_op_key_exchange_flags_t | flags | bitmap specifying the operation properties |

### 5.21.3 Function Documentation

#### 5.21.3.1 hsm_key_exchange()

```
hsm_err_t hsm_key_exchange (
            hsm_hdl_t key_management_hdl,
            op_key_exchange_args_t * args )
```

This command is designed to to derive a secret key that will be stored in the key store as a new key or as an update of an existing key.
A freshly generated key or an existing key can be used as input for the shared secret calculation.
User can call this function only after having opened a key management service flow.
When using the SHA KDF, only Key Encryption Keys (KEKs) can be generated.
As per as per SP800-56C rev2, the KEK is generated using SHA_256(counter || Z || FixedInput), where:

- counter is a 32 bit value of 1 in big endian format
- Z is the shared secret generated by the DH key-establishment scheme
- FixedInput is the literal 'NXP HSM USER KEY DERIVATION' (27 bytes, no null termination).

In the case of HSM_KE_SCHEME_SM2_FP_256 :

- v2x role is receiver
- only HSM_KDF_ALG_FOR_SM2 is supported
- HSM_OP_KEY_EXCHANGE_FLAGS_USE_EPHEMERAL is not supported
- shared_key_type could only be HSM_KEY_TYPE_SM4_128 or HSM_KEY_TYPE_DSA_SM2_FP_256
  - shared_key info could not be a KEK
  - initiator_public_data_type could only be HSM_KEY_TYPE_DSA_SM2_FP_256
  - ke_input = (x||y) || (xephemeral||yephemeral) of the 2 public keys of initiator

- ke_out = (x||y)|| (xephemeral||yephemeral) of the 2 public keys the receiver
- kdf_input = (Zinitiator||Zinitiator||V1) if HSM_OP_KEY_EXCHANGE_FLAGS_KEY_CONF_EN enabled, where V1 is the verification value calculated on the initiator side
- kdf_output = (VA||VB) if HSM_OP_KEY_EXCHANGE_FLAGS_KEY_CONF_EN enabled, 0 otherwise.
- This SM2 key exchange algorithm is specified in GB/T 32918.

**Parameters**

| | |
|---|---|
| *key_management_hdl* | handle identifying the key store management service flow. |
| *args* | pointer to the structure containing the function arguments. |

**Returns**

error code

## 5.22 i.MX8QXP specificities

### Session

i.MX8QXP HSM is implemented only on SECO core which doesn't offer priority management neither low latencies.

- HSM_OPEN_SESSION_FIPS_MODE_MASK not supported and ignored
- HSM_OPEN_SESSION_EXCLUSIVE_MASK not supported and ignored
- session_priority field of open_session_args_t is ignored.
- HSM_OPEN_SESSION_LOW_LATENCY_MASK not supported and ignored.

### Key management

- HSM_OP_MANAGE_KEY_GROUP_FLAGS_DELETE is not supported.
- HSM_KEY_TYPE_ECDSA_NIST_P521 is not supported.
- HSM_KEY_TYPE_ECDSA_BRAINPOOL_R1_320 is not supported.
- HSM_KEY_TYPE_ECDSA_BRAINPOOL_R1_512 is not supported.
- HSM_KEY_TYPE_ECDSA_BRAINPOOL_T1_256 is not supported.
- HSM_KEY_TYPE_ECDSA_BRAINPOOL_T1_320 is not supported.
- HSM_KEY_TYPE_ECDSA_BRAINPOOL_T1_384 is not supported.
- HSM_KEY_TYPE_ECDSA_BRAINPOOL_T1_512 is not supported.
- HSM_KEY_TYPE_DSA_SM2_FP_256 is not supported.
- HSM_KEY_TYPE_SM4_128 is not supported.
- hsm_butterfly_key_expansion: This feature is disabled when part is running in FIPS approved mode. Any call to this API will results in a HSM_FEATURE_DISABLED error.
- hsm_key_type_t of op_butt_key_exp_args_t: Only HSM_KEY_TYPE_ECDSA_NIST_P256 and HSM_KE↩ Y_TYPE_ECDSA_BRAINPOOL_R1_256 are supported.

### Ciphering

- HSM_CIPHER_ONE_GO_ALGO_SM4_ECB is not supported.
- HSM_CIPHER_ONE_GO_ALGO_SM4_CBC is not supported.
- hsm_ecies_decryption: This feature is disabled when part is running in FIPS approved mode. Any call to this API will results in a HSM_FEATURE_DISABLED error.
- hsm_key_type_t of op_ecies_dec_args_t: Only HSM_KEY_TYPE_ECDSA_NIST_P256 and HSM_KEY_T↩ YPE_ECDSA_BRAINPOOL_R1_256 are supported.

### Signature generation

- HSM_SIGNATURE_SCHEME_ECDSA_NIST_P521_SHA_512 is not supported.
- HSM_SIGNATURE_SCHEME_ECDSA_BRAINPOOL_R1_320_SHA_384 is not supported.

- HSM_SIGNATURE_SCHEME_ECDSA_BRAINPOOL_R1_512_SHA_512 is not supported.
- HSM_SIGNATURE_SCHEME_ECDSA_BRAINPOOL_T1_256_SHA_256 is not supported.
- HSM_SIGNATURE_SCHEME_ECDSA_BRAINPOOL_T1_320_SHA_384 is not supported.
- HSM_SIGNATURE_SCHEME_ECDSA_BRAINPOOL_T1_384_SHA_384 is not supported.
- HSM_SIGNATURE_SCHEME_ECDSA_BRAINPOOL_T1_512_SHA_512 is not supported.
- HSM_SIGNATURE_SCHEME_DSA_SM2_FP_256_SM3 is not supported.

Signature verification

- HSM_OP_VERIFY_SIGN_FLAGS_KEY_INTERNAL is not supported
- hsm_import_public_key: This API is not supported

Signature generation

- HSM_HASH_ALGO_SM3_256 is not supported.

Public key reconstruction

- This feature is disabled when part is running in FIPS approved mode. Any call to this API will results in a HSM_FEATURE_DISABLED error.
- hsm_key_type_t of op_pub_key_rec_args_t: Only HSM_KEY_TYPE_ECDSA_NIST_P256 and HSM_KEY↩_TYPE_ECDSA_BRAINPOOL_R1_256 are supported.

Public key decompression

- This feature is disabled when part is running in FIPS approved mode. Any call to this API will results in a HSM_FEATURE_DISABLED error.

ECIES encryption

- hsm_ecies_encryption: This feature is disabled when part is running in FIPS approved mode. Any call to this API will results in a HSM_FEATURE_DISABLED error.
- hsm_key_type_t of op_ecies_enc_args_t: Only HSM_KEY_TYPE_ECDSA_NIST_P256 and HSM_KEY_T↩YPE_ECDSA_BRAINPOOL_R1_256 are supported.

SM2 Get Z

- This API is not supported.

SM2 ECES decryption

- All the APIs related the SM2 ECES decryption are not supported.

SM2 ECES encryption

- This API is not supported.

Key exchange

- HSM_KE_SCHEME_SM2_FP_256 is not supported.
- HSM_KDF_ALG_FOR_SM2 is not supported.
- HSM_OP_KEY_EXCHANGE_FLAGS_KEY_CONF_EN is not supported

Key store

The table below summarizes the maximum number of keys per group in the QXP implementation:

| Key size (bits) | Number of keys per group |
|---|---|
| 128 | 169 |
| 192 | 126 |
| 224 | 101 |
| 256 | 101 |
| 384 | 72 |
| 512 | 56 |

## 5.23 i.MX8DXL specificities

### Session

i.MX8DXL has 2 separate implementations of HSM on SECO and on V2X cores.

- HSM_OPEN_SESSION_FIPS_MODE_MASK not supported and ignored
- HSM_OPEN_SESSION_EXCLUSIVE_MASK not supported and ignored
- If HSM_OPEN_SESSION_LOW_LATENCY_MASK is unset then SECO implementation will be used. In this case session_priority field of open_session_args_t is ignored.
- If HSM_OPEN_SESSION_LOW_LATENCY_MASK is set then V2X implementation is used. session_priority field of open_session_args_t and HSM_OPEN_SESSION_NO_KEY_STORE_MASK are considered.

### Key management

- HSM_OP_MANAGE_KEY_GROUP_FLAGS_DELETE is not supported.
- hsm_key_type_t of op_butt_key_exp_args_t: Only HSM_KEY_TYPE_ECDSA_NIST_P256, HSM_KEY_T↩YPE_ECDSA_BRAINPOOL_R1_256 and HSM_KEY_TYPE_ECDSA_BRAINPOOL_T1_256 are supported.

### Ciphering

- hsm_key_type_t of op_ecies_dec_args_t: Only HSM_KEY_TYPE_ECDSA_NIST_P256, HSM_KEY_TYP↩E_ECDSA_BRAINPOOL_R1_256 and HSM_KEY_TYPE_ECDSA_BRAINPOOL_T1_256 are supported.

### Signature generation

- HSM_OP_GENERATE_SIGN_FLAGS_COMPRESSED_POINT is not supported, in case of HSM_SIGNA↩TURE_SCHEME_DSA_SM2_FP_256_SM3.

### Signature verification

- HSM_OP_VERIFY_SIGN_FLAGS_COMPRESSED_POINT is not supported, in case of HSM_SIGNATUR↩E_SCHEME_DSA_SM2_FP_256_SM3.
- HSM_OP_VERIFY_SIGN_FLAGS_KEY_INTERNAL is not supported
- hsm_import_public_key: This API is a preliminary version

### Public key reconstruction

- hsm_key_type_t of op_pub_key_rec_args_t: Only HSM_KEY_TYPE_ECDSA_NIST_P256, HSM_KEY_T↩YPE_ECDSA_BRAINPOOL_R1_256 and HSM_KEY_TYPE_ECDSA_BRAINPOOL_T1_256 are supported.

### ECIES encryption

- hsm_key_type_t of op_ecies_enc_args_t: Only HSM_KEY_TYPE_ECDSA_NIST_P256, HSM_KEY_TYP↩E_ECDSA_BRAINPOOL_R1_256 and HSM_KEY_TYPE_ECDSA_BRAINPOOL_T1_256 are supported.

### Mac

- HSM_OP_MAC_ONE_GO_ALGO_HMAC_SHA_224 is not supported.
- HSM_OP_MAC_ONE_GO_ALGO_HMAC_SHA_256 is not supported.
- HSM_OP_MAC_ONE_GO_ALGO_HMAC_SHA_384 is not supported.
- HSM_OP_MAC_ONE_GO_ALGO_HMAC_SHA_512 is not supported.

### SM2 ECES decryption

- The output_size should be a multiple of 4 bytes.

### SM2 ECES encryption

- The output_size should be a multiple of 4 bytes.

### Key exchange

- HSM_KDF_HMAC_SHA_256_TLS_0_16_4 is not supported.
- HSM_KDF_HMAC_SHA_384_TLS_0_32_4 is not supported.
- HSM_KDF_HMAC_SHA_256_TLS_0_32_4 is not supported.
- HSM_KDF_HMAC_SHA_256_TLS_32_16_4 is not supported.
- HSM_KDF_HMAC_SHA_384_TLS_48_32_4 is not supported.

### Key store

The table below summarizes the maximum number of keys per group in the DXL implementation:

sessions using V2X implementation (HSM_OPEN_SESSION_LOW_LATENCY_MASK) :

| Key size (bits) | Number of keys per group |
|---|---|
| 128 | 166 |
| 192 | 125 |
| 224 | 111 |
| 256 | 100 |
| 384 | 71 |
| 512 | 52 |

session using SECO implementation : same number as QXP applies

# Index