

i.MX8 HSM API Rev 3.3

NXP Copyright

Generated by Doxygen 1.8.13

## Contents

<b>1</b>	<b>HSM API</b>	<b>2</b>
<b>2</b>	<b>Revision History</b>	<b>2</b>
<b>3</b>	<b>General concepts related to the API</b>	<b>4</b>
3.1	Session . . . . .	4
3.2	Service flow . . . . .	4
3.3	Example . . . . .	5
3.4	Key store . . . . .	5
3.4.1	Key management . . . . .	5
3.4.2	NVM writing . . . . .	5
3.5	Implementation specificities . . . . .	6
<b>4</b>	<b>Module Index</b>	<b>6</b>
4.1	Modules . . . . .	6
<b>5</b>	<b>Module Documentation</b>	<b>7</b>
5.1	Error codes . . . . .	8
5.1.1	Detailed Description . . . . .	8
5.1.2	Enumeration Type Documentation . . . . .	8
5.2	Session . . . . .	10
5.2.1	Detailed Description . . . . .	10
5.2.2	Data Structure Documentation . . . . .	10
5.2.3	Function Documentation . . . . .	11
5.3	Key store . . . . .	12
5.3.1	Detailed Description . . . . .	12
5.3.2	Data Structure Documentation . . . . .	12
5.3.3	Function Documentation . . . . .	13
5.4	Key management . . . . .	15
5.4.1	Detailed Description . . . . .	17
5.4.2	Data Structure Documentation . . . . .	17

5.4.3	Function Documentation	19
5.5	Ciphering	23
5.5.1	Detailed Description	23
5.5.2	Data Structure Documentation	23
5.5.3	Function Documentation	25
5.6	Signature generation	28
5.6.1	Detailed Description	29
5.6.2	Data Structure Documentation	29
5.6.3	Function Documentation	30
5.7	Signature verification	32
5.7.1	Detailed Description	32
5.7.2	Data Structure Documentation	32
5.7.3	Function Documentation	33
5.8	Random number generation	36
5.8.1	Detailed Description	36
5.8.2	Data Structure Documentation	36
5.8.3	Function Documentation	36
5.9	Hashing	39
5.9.1	Detailed Description	39
5.9.2	Data Structure Documentation	39
5.9.3	Function Documentation	40
5.10	Public key reconstruction	42
5.10.1	Detailed Description	42
5.10.2	Data Structure Documentation	42
5.10.3	Function Documentation	42
5.11	Public key decompression	44
5.11.1	Detailed Description	44
5.11.2	Data Structure Documentation	44
5.11.3	Function Documentation	44
5.12	ECIES encryption	46

5.12.1 Detailed Description . . . . .	46
5.12.2 Data Structure Documentation . . . . .	46
5.12.3 Function Documentation . . . . .	47
5.13 Public key recovery . . . . .	48
5.13.1 Detailed Description . . . . .	48
5.13.2 Data Structure Documentation . . . . .	48
5.13.3 Function Documentation . . . . .	48
5.14 Data storage . . . . .	50
5.14.1 Detailed Description . . . . .	50
5.14.2 Data Structure Documentation . . . . .	50
5.14.3 Function Documentation . . . . .	51
5.15 Root KEK export . . . . .	53
5.15.1 Detailed Description . . . . .	53
5.15.2 Data Structure Documentation . . . . .	53
5.15.3 Function Documentation . . . . .	53
5.16 Get info . . . . .	56
5.16.1 Detailed Description . . . . .	56
5.16.2 Data Structure Documentation . . . . .	56
5.16.3 Function Documentation . . . . .	56
5.17 Mac . . . . .	58
5.17.1 Detailed Description . . . . .	58
5.17.2 Data Structure Documentation . . . . .	58
5.17.3 Function Documentation . . . . .	59
5.18 SM2 Get Z . . . . .	61
5.18.1 Detailed Description . . . . .	61
5.18.2 Data Structure Documentation . . . . .	61
5.18.3 Function Documentation . . . . .	61
5.19 SM2 ECES decryption . . . . .	63
5.19.1 Detailed Description . . . . .	63
5.19.2 Data Structure Documentation . . . . .	63
5.19.3 Function Documentation . . . . .	64
5.20 SM2 ECES encryption . . . . .	66
5.20.1 Detailed Description . . . . .	66
5.20.2 Data Structure Documentation . . . . .	66
5.20.3 Function Documentation . . . . .	66
5.21 Key exchange . . . . .	68
5.21.1 Detailed Description . . . . .	69
5.21.2 Data Structure Documentation . . . . .	69
5.21.3 Function Documentation . . . . .	70
5.22 i.MX8QXP specificities . . . . .	74
5.23 i.MX8DXL specificities . . . . .	77

## 1 HSM API

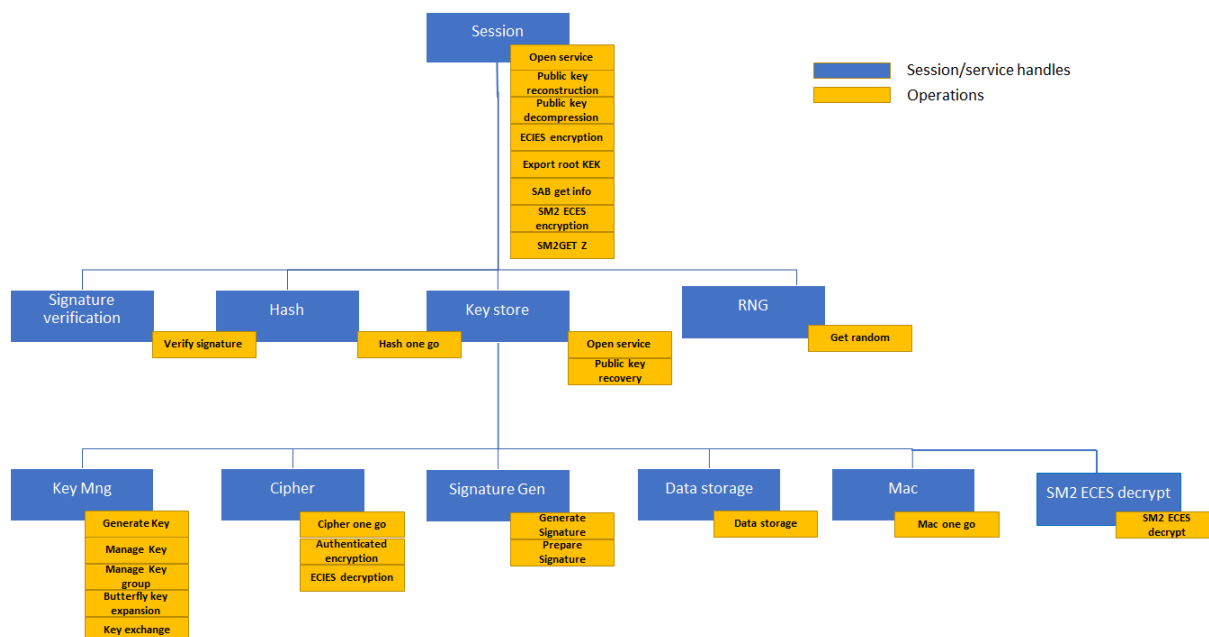
This document is a software reference description of the API provided by the i.MX8 HSM solutions.

## 2 Revision History

Revision	date	description
0.1	Mar 29 2019	Preliminary draft
0.8	May 24 2019	It adds the following API: -signature generation -signature verification -rng -hash -butterfly key expansion -ECIES enc/dec -public key reconstruction -public key decompression
0.9	May 28 2019	Explicit addresses are replaced by pointers.
1.0	May 29 2019	- bug/typos fix. - 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. - add error code definition. - improve argument comments clarity
1.5	Sept 13 2019	- manage key argument: fix padding size - butterfly key expansion: change argument definition - introduce public key recovery API
1.6	Oct 14 2019	- add Key store section in chapter 3 - change key_info and flags definition, substitute key_type_ext with group_id - hsm_generate_key, hsm_manage_key, hsm_butterfly_key_expansion: change argument definition - hsm_manage_key: change argument definition - add hsm_manage_key_group API
1.7	Dec 20 2019	- add generic data storage API - add GCM and CMAC support - add support for AES 192/256 key size for all cipher algorithms - add root KEK export API - add key import functionality - add get info API
2.0	Feb 21 2020	- fix HSM_KEY_INFO_TRANSIENT definition: delete erroneous "not supported" comment - add Key Encryption Key (HSM_KEY_INFO_KEK) support - key store open service API: adding signed message support for key store reprovisionning - naming consistency: remove "hsm_" prefix from hsm_op_ecies_dec_args_t hsm_op_pub_key_rec_args_t hsm_op_pub_key_dec_args_t hsm_op_ecies_enc_args_t hsm_op_pub_key_recovery_args_t hsm_op_get_info_args_t

Revision	date	description
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) - add missing status code definition - remove hsm_open_key_store_service unused flags: HSM_SVC_KEY_STORE_UPDATE_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. - Update i.MX8QXP specificities section specifying operations disabled when in FIPS approved mode. - 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 - 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 - 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.
3.0	Nov 16 2020	hsm_open_key_store_service: add min_mac_length argument. hsm_mac_one_go - verification: add HSM_OP_MAC_ONE_GO_FLAGS_MAC_LENGTH_IN_BITS to represent mac_length in bit. hsm_key_exchange: - enforce new constraints on KEK and TLS key generations - add signed message arguments for KEK generation. - rename HSM_KDF_ALG_SHA_256 in HSM_KDF_ONE_STEP_SHA_256. - rename HSM_OP_KEY_EXCHANGE_FLAGS_USE_EPHEMERAL in HSM_OP_KEY_EXCHANGE_FLAGS_GENERATE_EPHEMERAL
3.1	Nov 20 2020	Enable support of key_exchange and HMAC on QXP
3.2	Dec 1 2020	hsm_generate_key, hsm_manage_key: fix key_group argument wrong description. User must specify the key group for CREATE/UPDATE/DELETE operations.
3.3	Dec 04 2020	Add support of hsm_tls_finish on QXP, update hsm_key_exchange to describe creation of master_secret for TLS KDF

### 3 General concepts related to the API



#### 3.1 Session

The API must be initialized by a potential requestor by opening a session.

The session establishes a route (MU, DomainID...) between the requestor and the HSM. When a session is opened, the HSM returns a handle identifying the session to the requestor.

#### 3.2 Service flow

For a given category of services, the requestor is expected to open a service flow by invoking the appropriate HSM API.

The session handle, as well as the control data needed for the service flow, are provided as parameters of the call. Upon reception of the open request, the HSM allocates a context in which the session handle, as well as the provided control parameters are stored and return a handle identifying the service flow.

The context is preserved until the service flow, or the session, are closed by the user and it is used by the HSM to proceed with the sub-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 specifying the "STRICT OPERATION" flag.

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

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

#### 3.4.1 Key management

Keys are divided in groups, keys belonging to the same group are written/read from the NVM as a monolithic block. Up to 3 key groups can be handled in the HSM local memory (those immediately available to perform crypto 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 modifying 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 specifying the flag "STRICT OPERATION" is acknowledged by the HSM. All the operations not specifying the "STRICT OPERATION" flags impact the HSM local memory only and will be lost in case of system reset

Due to the limited monotonic counter size (QXPB0 up to 1620 update available by default), the user should, when possible, perform multiple updates before setting the "STRICT OPERATION" flag (i.e. keys to be updated should be kept in the same key group).

Once the monotonic counter is completely blown a warning is returned on each 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:

<b>Error codes</b>	<b><a href="#">8</a></b>
<b>Session</b>	<b><a href="#">10</a></b>
<b>i.MX8QXP specificities</b>	<b><a href="#">74</a></b>
<b>i.MX8DXL specificities</b>	<b><a href="#">77</a></b>
<b>Key store</b>	<b><a href="#">12</a></b>
<b>Key management</b>	<b><a href="#">15</a></b>
<b>i.MX8QXP specificities</b>	<b><a href="#">74</a></b>
<b>i.MX8DXL specificities</b>	<b><a href="#">77</a></b>
<b>Ciphering</b>	<b><a href="#">23</a></b>
<b>i.MX8QXP specificities</b>	<b><a href="#">74</a></b>
<b>i.MX8DXL specificities</b>	<b><a href="#">77</a></b>
<b>Signature generation</b>	<b><a href="#">28</a></b>
<b>i.MX8QXP specificities</b>	<b><a href="#">74</a></b>
<b>i.MX8DXL specificities</b>	<b><a href="#">77</a></b>
<b>Signature verification</b>	<b><a href="#">32</a></b>

i.MX8QXP specificities	74
i.MX8DXL specificities	77
Random number generation	36
Hashing	39
i.MX8QXP specificities	74
Public key reconstruction	42
i.MX8QXP specificities	74
i.MX8DXL specificities	77
Public key decompression	44
i.MX8QXP specificities	74
ECIES encryption	46
i.MX8QXP specificities	74
i.MX8DXL specificities	77
Public key recovery	48
Data storage	50
Root KEK export	53
Get info	56
Mac	58
i.MX8DXL specificities	77
SM2 Get Z	61
i.MX8QXP specificities	74
SM2 ECES decryption	63
i.MX8QXP specificities	74
i.MX8DXL specificities	77
SM2 ECES encryption	66
i.MX8QXP specificities	74
i.MX8DXL specificities	77
Key exchange	68
i.MX8QXP specificities	74
i.MX8DXL specificities	77

## 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 firmware.
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 occurred.

## 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

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

## Returns

error\_code error code.

## 5.2.3.2 hsm\_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

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

## Returns

error\_code error code.

## 5.3 Key store

### Data Structures

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

### Macros

- #define [HSM\\_SVC\\_KEY\\_STORE\\_FLAGS\\_CREATE](#) ((hsm\_svc\_key\_store\_flags\_t)(1u << 0))  
*It must be specified to create a new key store. The key store will be stored in the NVM only if the STRICT OPERATION flag is set.*
- #define [HSM\\_SVC\\_KEY\\_STORE\\_FLAGS\\_SET\\_MAC\\_LEN](#) ((hsm\_svc\_key\_store\_flags\_t)(1u << 3))  
*If set, minimum mac length specified in min\_mac\_length field will be stored in the key store when creating the key store. Must only be set at key store creation.*
- #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 accessing the key store.
uint16_t	max_updates_number	maximum number of updates authorized for the key store. Valid only for create operation. This parameter has the goal to limit the occupation of the monotonic counter used as anti-rollback protection. If the maximum number of updates is reached, HSM still allows key store updates but without updating the monotonic counter giving the opportunity for rollback attacks.
hsm_svc_key_store_flags_t	flags	bitmap specifying the services properties.
uint8_t	min_mac_length	it corresponds to the minimum mac length (in bits) accepted by the HSM to perform MAC verification operations. Only used upon key store creation when HSM_SVC_KEY_STORE_FLAGS_SET_MAC_LEN bit is set. It is effective only for MAC verification operations with the mac length expressed in bits. It can be used to replace the default value (32 bits). It impacts all MAC algorithms and all key lengths. It must be different from 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

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



**Returns**

error\_code error code.

**5.3.3.2 hsm\_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**

<i>handle</i>	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\\_butl\\_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\_manage\_key API can get this attribute.
- **#define HSM\_OP\_MANAGE\_KEY\_FLAGS\_IMPORT\_UPDATE** ((hsm\_op\_manage\_key\_flags\_t)(1u << 0))  
User can replace an existing key only by importing a key with the same type of the original one.
- **#define HSM\_OP\_MANAGE\_KEY\_FLAGS\_IMPORT\_CREATE** ((hsm\_op\_manage\_key\_flags\_t)(1u << 1))  
Import a key and create a new identifier.
- **#define HSM\_OP\_MANAGE\_KEY\_FLAGS\_DELETE** ((hsm\_op\_manage\_key\_flags\_t)(1u << 2))  
Delete an existing key.
- **#define HSM\_OP\_MANAGE\_KEY\_FLAGS\_PART\_UNIQUE\_ROOT\_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\\_but\\_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. It must be a value in the range 0-1023. Keys belonging to the same group can be cached in the HSM local memory through the <a href="#">hsm_manage_key_group</a> 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. It must be a value in the range 0-1023. Keys belonging to the same group can be cached in the HSM local memory through the hsm_manage_key_group API.
hsm_key_info_t	key_info	bitmap specifying the properties of the key, 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 through 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\_butl\_key\_exp\_args\_t

## Data Fields

uint32_t	key_identifier	identifier of the key to be expanded.
uint8_t *	expansion_function_value	pointer to the expansion function value input
uint8_t *	hash_value	pointer to the hash value input. In case of explicit certificate, the hash value address must be set to 0.
uint8_t *	pr_reconstruction_value	pointer to the private reconstruction value input. In case of explicit certificate, the pr_reconstruction_value address must be set to 0.

## Data Fields

uint8_t	expansion_function_value_size	length in bytes of the expansion function input
uint8_t	hash_value_size	length in bytes of the hash value input. In case of explicit certificate, the hash_value_size parameter must be set to 0.
uint8_t	pr_reconstruction_value_size	length in bytes of the private reconstruction value input. In case of explicit certificate, the pr_reconstruction_value_size parameter must be set to 0.
hsm_op_but_key_exp_flags_t	flags	bitmap specifying the operation properties
uint32_t *	dest_key_identifier	pointer to identifier of the derived key to be used for the operation. In case of create operation the new destination key identifier will be stored in this location.
uint8_t *	output	pointer to the output area where the public key must be written.
uint16_t	output_size	length in bytes of the generated key, if the size is 0, no key is copied in the output.
hsm_key_type_t	key_type	indicates the type of the key to be derived.
uint8_t	reserved	
hsm_key_group_t	key_group	it must be a value in the range 0-1023. Keys belonging to the same group can be cached in the HSM local memory through the 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

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

<code>key_management_hdl</code>	handle identifying the key management service flow.
<code>args</code>	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

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

## Returns

error code

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

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

This command is designed to perform the following operations:

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

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

## Parameters

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

## Returns

error code

## 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  $f_k = f_{k\_int} \bmod l$ , 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

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

#### Returns

error code

#### 5.4.3.6 `hsm_close_key_management_service()`

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

Terminate a previously opened key management service flow

#### Parameters

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

#### Returns

error code

## 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 it must be 0 for algorithms not using the initialization vector. It must be 12 for AES in CCM mode
hsm_op_cipher_one_go_algo_t	cipher_algo	algorithm to be used for the operation
hsm_op_cipher_one_go_flags_t	flags	bitmap specifying the operation attributes
uint8_t *	input	pointer to the input area plaintext for encryption ciphertext for decryption (in case of CCM is the purported ciphertext)
uint8_t *	output	pointer to the output area ciphertext for encryption (in case of CCM is the output of the generation-encryption process) plaintext for decryption
uint32_t	input_size	length in bytes of the input. 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 It must be 12 bytes.
uint8_t *	aad	pointer to the additional authentication data
uint16_t	aad_size	length in bytes of the additional authentication data
hsm_op_auth_enc_algo_t	ae_algo	algorithm to be used for the operation
hsm_op_auth_enc_flags_t	flags	bitmap specifying the operation attributes
uint8_t *	input	pointer to the input area plaintext for encryption Ciphertext + Tag (16 bytes) for decryption
uint8_t *	output	pointer to the output area Ciphertext + Tag (16 bytes) for encryption plaintext for decryption if the Tag is verified
uint32_t	input_size	length in bytes of the input
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

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

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

#### Returns

error code

### 5.5.3.3 hsm\_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

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

#### Returns

error code

### 5.5.3.4 hsm\_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

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

## Returns

error code

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

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

Terminate a previously opened cipher service flow

## Parameters

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

## Returns

error code

## 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 $r  s  R_y$ where $R_y$ is an additional byte containing the lsb of $y$ . $R_y$ has to be considered valid only if the <a href="#">HSM_OP_GENERATE_SIGN_FLAGS_COMPRESSED_POINT</a> 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

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

## 5.6.3 Function Documentation

5.6.3.1 `hsm_open_signature_generation_service()`

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

Open a signature generation service flow

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

User must open this service in order to perform signature generation operations.

## Parameters

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

## Returns

error code

5.6.3.2 `hsm_close_signature_generation_service()`

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

Terminate a previously opened signature generation service flow

## Parameters

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

## Returns

error code

## 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

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

## Returns

error code

## 5.6.3.4 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_FLAGS_COMPRESSED_POINT` is set.

## Parameters

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

## Returns

error code

## 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 $r  s  R_y$ where $R_y$ is an additional byte containing the lsb of $y$ . $R_y$ will be considered as valid only if the HSM_OP_VERIFY_SIGN_FLAGS_COMPRESSED_POINT is set.
uint16_t	key_size	length in bytes of the input key
uint16_t	signature_size	length in bytes of the output - it must contain one additional byte where to store the $R_y$ .
uint32_t	message_size	length in bytes of the input message
hsm_signature_scheme_id_t	scheme_id	identifier of the digital signature scheme to be used for the operation
hsm_op_verify_sign_flags_t	flags	bitmap specifying the operation attributes
uint16_t	reserved	

## 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

<i>session_hdl</i>	handle identifying the current session.
<i>args</i>	pointer to the structure containing the function arguments.
<i>signature_ver_hdl</i>	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  $r||s||R_y$  where  $R_y$  is an additional byte containing the lsb of  $y$ .  $R_y$  will be considered as valid only if the `HSM_OP_VERIFY_SIGN_FLAGS_COMPRESSED_POINT` is set.

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

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

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

## Returns

error code

## 5.7.3.3 hsm\_import\_public\_key()

```
hsm_err_t hsm_import_public_key (
    hsm_hdl_t signature_ver_hdl,
```

```
op_import_public_key_args_t * args,
uint32_t * key_ref )
```

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

User can use the returned reference in the `hsm_verify_signature` API by setting the `HSM_OP_VERIFY_SIGN_FLAGS_KEY_INTERNAL` flag

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

#### Parameters

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

#### Returns

error code

#### 5.7.3.4 `hsm_close_signature_verification_service()`

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

Terminate a previously opened signature verification service flow

#### Parameters

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

#### Returns

error code

## 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

<a href="#">hsm_svc_rng_flags_t</a>	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**

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

<i>rng_hdl</i>	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**

<i>rng_hdl</i>	handle identifying the rng service flow.
<i>args</i>	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 ↵  
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

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

<i>hash_hdl</i>	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**

<i>hash_hdl</i>	handle identifying the hash service flow.
<i>args</i>	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:

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

**Parameters**

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

**Returns**

error code

## 5.11 Public key decompression

### 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

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

#### Returns

error code



## 5.12 ECIES encryption

### 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 using ECIES

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

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

#### Parameters

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

#### Returns

error code

### 5.13 Public key recovery

#### Data Structures

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

#### Typedefs

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

#### Functions

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

#### 5.13.1 Detailed Description

#### 5.13.2 Data Structure Documentation

##### 5.13.2.1 struct op\_pub\_key\_recovery\_args\_t

#### Data Fields

uint32_t	key_identifier	pointer to the identifier of the key to be used for the operation
uint8_t *	out_key	pointer to the output area where the generated public key must be written
uint16_t	out_key_size	length in bytes of the output key
hsm_key_type_t	key_type	indicates the type of the key to be recovered
hsm_op_pub_key_recovery_flags_t	flags	bitmap specifying the operation attributes.

#### 5.13.3 Function Documentation

##### 5.13.3.1 hsm\_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

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

#### Returns

error code

## 5.14 Data storage

### Data Structures

- struct [open\\_svc\\_data\\_storage\\_args\\_t](#)
- struct [op\\_data\\_storage\\_args\\_t](#)

### Macros

- #define [HSM\\_OP\\_DATA\\_STORAGE\\_FLAGS\\_STORE](#) ((hsm\_op\_data\_storage\_flags\_t)(1u << 0))  
*Store data.*
- #define [HSM\\_OP\\_DATA\\_STORAGE\\_FLAGS\\_RETRIEVE](#) ((hsm\_op\_data\_storage\_flags\_t)(0u << 0))  
*Retrieve data.*

### Typedefs

- typedef uint8\_t [hsm\\_svc\\_data\\_storage\\_flags\\_t](#)
- typedef uint8\_t [hsm\\_op\\_data\\_storage\\_flags\\_t](#)

### Functions

- [hsm\\_err\\_t hsm\\_open\\_data\\_storage\\_service](#) (hsm\_hdl\_t key\_store\_hdl, [open\\_svc\\_data\\_storage\\_args\\_t](#) \*args, hsm\_hdl\_t \*data\_storage\_hdl)
- [hsm\\_err\\_t hsm\\_data\\_storage](#) (hsm\_hdl\_t data\_storage\_hdl, [op\\_data\\_storage\\_args\\_t](#) \*args)
- [hsm\\_err\\_t hsm\\_close\\_data\\_storage\\_service](#) (hsm\_hdl\_t data\_storage\_hdl)

#### 5.14.1 Detailed Description

#### 5.14.2 Data Structure Documentation

##### 5.14.2.1 struct open\_svc\_data\_storage\_args\_t

#### Data Fields

<a href="#">hsm_svc_data_storage_flags_t</a>	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
<a href="#">hsm_op_data_storage_flags_t</a>	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/retrieve generic data in/from the HSM.

##### Parameters

<i>key_store_hdl</i>	handle identifying the key store service flow.
<i>args</i>	pointer to the structure containing the function arguments.
<i>data_storage_hdl</i>	pointer to where the data storage service flow handle must be written.

##### Returns

error\_code error code.

#### 5.14.3.2 hsm\_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

<i>data_storage_hdl</i>	handle identifying the data storage service flow.
<i>args</i>	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**

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

**Returns**

error code

## 5.15 Root KEK export

### Data Structures

- struct [op\\_export\\_root\\_kek\\_args\\_t](#)

### Macros

- #define **HSM\_OP\_EXPORT\_ROOT\_KEK\_FLAGS\_COMMON\_KEK** ((hsm\_op\_export\_root\_kek\_flags\_t)(1u << 0))
- #define **HSM\_OP\_EXPORT\_ROOT\_KEK\_FLAGS\_UNIQUE\_KEK** ((hsm\_op\_export\_root\_kek\_flags\_t)(0u << 0))

### Typedefs

- typedef uint8\_t **hsm\_op\_export\_root\_kek\_flags\_t**

### Functions

- [hsm\\_err\\_t hsm\\_export\\_root\\_key\\_encryption\\_key](#) (hsm\_hdl\_t session\_hdl, [op\\_export\\_root\\_kek\\_args\\_t](#) \*args)

#### 5.15.1 Detailed Description

#### 5.15.2 Data Structure Documentation

##### 5.15.2.1 struct op\_export\_root\_kek\_args\_t

#### Data Fields

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

#### 5.15.3 Function Documentation

##### 5.15.3.1 hsm\_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**

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

**Returns**

error code

## 5.16 Get info

### Data Structures

- struct [op\\_get\\_info\\_args\\_t](#)

### Functions

- [hsm\\_err\\_t hsm\\_get\\_info](#) ([hsm\\_hdl\\_t session\\_hdl](#), [op\\_get\\_info\\_args\\_t](#) \*args)

#### 5.16.1 Detailed Description

#### 5.16.2 Data Structure Documentation

##### 5.16.2.1 struct op\_get\_info\_args\_t

#### Data Fields

uint32_t *	user_sab_id	pointer to the output area where the user identifier (32bits) must be written
uint8_t *	chip_unique_id	pointer to the output area where the chip unique identifier (64bits) must be written
uint16_t *	chip_monotonic_counter	pointer to the output are where the chip monotonic counter value (16bits) must be written
uint16_t *	chip_life_cycle	pointer to the output area where the chip current life cycle 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: bit0 - FIPS mode of operation: - value 0 - part is running in FIPS non-approved mode. - value 1 - part is running in FIPS approved mode. bit1 - FIPS certified part: - value 0 - part is not FIPS certified. - value 1 - part is FIPS certified. 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**

<i>session_hdl</i>	handle identifying the current session.
<i>args</i>	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_FLAGS_MAC_LENGTH_IN_BITS ((hsm_op_mac_one_go_flags_t)(1u << 1))`
- `#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

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

5.17.2.2 struct `op_mac_one_go_args_t`

## Data Fields

<code>uint32_t</code>	key_identifier	identifier of the key to be used for the operation
<code>hsm_op_mac_one_go_algo_t</code>	algorithm	algorithm to be used for the operation
<code>hsm_op_mac_one_go_flags_t</code>	flags	bitmap specifying the operation attributes
<code>uint8_t *</code>	payload	pointer to the payload area
<code>uint8_t *</code>	mac	pointer to the tag area
<code>uint16_t</code>	payload_size	length in bytes of the payload
<code>uint16_t</code>	mac_size	length of the tag. Specified in bytes if <code>HSM_OP_MAC_ON↵E_GO_FLAGS_MAC_LENGTH_IN_BITS</code> is clear, specified in bits when <code>HSM_OP_MAC_ONE_GO_FLAGS_MAC_LE↵NGTH_IN_BITS</code> is set. When specified in bytes the mac size cannot be less than 4 bytes. When specified in bits the mac size cannot be less than: - the <code>min_mac_length</code> value if specified at the key store provisioning or - the default value (32 bit) if <code>min_mac_length</code> has not been specified.

## 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

<code>key_store_hdl</code>	handle identifying the key store service flow.
<code>args</code>	pointer to the structure containing the function arguments.
<code>mac_hdl</code>	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**

<i>mac_hdl</i>	handle identifying the mac service flow.
<i>args</i>	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**

<i>mac_hdl</i>	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  $Z = \text{SM3}(\text{Entl} \parallel \text{ID} \parallel a \parallel b \parallel xG \parallel yG \parallel \text{xpubk} \parallel \text{ypubk})$

- 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

<i>session_hdl</i>	handle identifying the current session.
<i>args</i>	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

<a href="#">hsm_svc_sm2_eces_flags_t</a>	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

<code>hsm_op_sm2_eces_dec_flags_t</code>	flags	bitmap specifying the operation attributes.
<code>uint16_t</code>	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

<i>session_hdl</i>	handle identifying the current session.
<i>args</i>	pointer to the structure containing the function arguments.
<i>sm2_eces_hdl</i>	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

<i>sm2_eces_hdl</i>	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**

<i>sm2_eces_hdl</i>	handle identifying the SM2 ECES
<i>args</i>	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. It should be at least input_size + 97 bytes (overhead related to C1 and C3 - as specified 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 using 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 an elliptic curve point

C2 = M xor t where t=KDF(x2||y2, input\_size) and (x2,y2) are the coordinates of an elliptic curve point

C3 = SM3 (x2||M||y2)

**Parameters**

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

**Returns**

error code

## 5.21 Key exchange

### Modules

- [i.MX8QXP specificities](#)
- [i.MX8DXL specificities](#)

### Data Structures

- struct [op\\_key\\_exchange\\_args\\_t](#)
- struct [op\\_tls\\_finish\\_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_ONE_STEP_SHA_256 ((hsm_kdf_algo_id_t)0x31u)`  
*One-Step Key Derivation using SHA256 as per NIST SP80056C. It can only be used, together with a signed message, to generate KEKs (key encryption keys) for key injection (hsm\_manage\_key API).*
- `#define HSM_KEY_SCHEME_ECDH_NIST_P256 ((hsm_key_exchange_scheme_id_t)0x02u)`
- `#define HSM_KEY_SCHEME_ECDH_NIST_P384 ((hsm_key_exchange_scheme_id_t)0x03u)`
- `#define HSM_KEY_SCHEME_ECDH_BRAINPOOL_R1_256 ((hsm_key_exchange_scheme_id_t)0x13u)`
- `#define HSM_KEY_SCHEME_ECDH_BRAINPOOL_R1_384 ((hsm_key_exchange_scheme_id_t)0x15u)`
- `#define HSM_KEY_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_GENERATE_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\_KEY\_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.*
- `#define HSM_OP_TLS_FINISH_HASH_ALGO_SHA256 (0x06)`
- `#define HSM_OP_TLS_FINISH_HASH_ALGO_SHA384 (0x07)`
- `#define HSM_OP_TLS_FINISH_FLAGS_CLIENT (1 << 0)`  
*Use "client finished" label for PRF.*
- `#define HSM_OP_TLS_FINISH_FLAGS_SERVER (1 << 1)`  
*Use "server finished" label for PRF.*

## 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**
- typedef uint8\_t **hsm\_op\_tls\_finish\_algo\_id\_t**
- typedef uint8\_t **hsm\_op\_tls\_finish\_flags\_t**

## Functions

- [hsm\\_err\\_t hsm\\_key\\_exchange](#) (hsm\_hdl\_t key\_management\_hdl, [op\\_key\\_exchange\\_args\\_t](#) \*args)
- [hsm\\_err\\_t hsm\\_tls\\_finish](#) (hsm\_hdl\_t key\_management\_hdl, [op\\_tls\\_finish\\_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_GENERATE_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.
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.
uint8_t *	kdf_input	pointer to the input data of the KDF.
uint8_t *	kdf_output	pointer to the output area where the non sensitive output data related to the KDF are written.
hsm_key_group_t	shared_key_group	It specifies the group where the derived keys will be stored. It must be a value in the range 0-1023. Keys belonging to the same group can be cached in the HSM local memory through the hsm_manage_key_group API.



## Data Fields

hsm_key_info_t	shared_key_info	bitmap specifying the properties of the derived keys, it will be applied to all the derived keys.
hsm_key_type_t	shared_key_type	indicates the type of the derived key.
hsm_key_type_t	initiator_public_data_type	indicates the public data type specified by the initiator, e.g. public key type.
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.
uint8_t	kdf_output_size	length in bytes of the non sensitive output data related to the KDF.
hsm_op_key_exchange_flags_t	flags	bitmap specifying the operation properties
uint8_t *	signed_message	pointer to the signed_message authorizing the operation.
uint16_t	signed_msg_size	size of the signed_message authorizing the operation.
uint8_t	reserved[2]	It must be 0.

## 5.21.2.2 struct op\_tls\_finish\_args\_t

## Data Fields

uint32_t	key_identifier	identifier of the master_secret key used for the PRF.
uint8_t *	handshake_hash_input	pointer to the input area containing the hash of the handshake messages.
uint8_t *	verify_data_output	pointer to the output area where the verify_data contents will be written.
uint16_t	handshake_hash_input_size	size of the hash of the handshake messages
uint16_t	verify_data_output_size	size of the required verify_data output
hsm_op_tls_finish_flags_t	flags	bitmap specifying the operation properties
hsm_op_tls_finish_algo_id_t	hash_algorithm	hash algorithm to be used for the PRF
uint8_t	reserved[2]	It must be 0.

## 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 compute secret keys through a key exchange protocol and the use of a key derivation function. The resulting secret keys are stored into the key store as new keys or as an update of existing keys. A freshly generated key or an existing key can be used as input of the shared secret calculation. User can call this function only after having opened a key management service flow.

This API support three use cases:

- Key Encryption Key generation:
  - shared\_key\_identifier\_array: it must corresponds to the KEK key id.
  - The kdf\_input must be 0
  - The kdf\_output must be 0
  - The shared\_key\_info must have the HSM\_KEY\_INFO\_KEY bit set (only Key Encryption Keys can be generated).
  - The shared\_key\_type must be HSM\_KEY\_TYPE\_AES\_256
  - The initiator\_public\_data\_type must be HSM\_KEY\_TYPE\_ECDSA\_NIST\_P256 or HSM\_KEY\_TYPE\_↵\_ECDSA\_BRAINPOOL\_R1\_256.
  - The key\_exchange\_scheme must be HSM\_KEY\_SCHEME\_ECDH\_NIST\_P256 or HSM\_KEY\_SCHEM\_↵E\_ECDH\_BRAINPOOL\_R1\_256
  - The kdf\_algorithm must be HSM\_KDF\_ONE\_STEP\_SHA\_256. As per as per SP800-56C rev2, the KEK is generated using the formula  $SHA\_256(counter || Z || FixedInput)$ , where:
    - \* counter is the value 1 expressed in 32 bit and 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).
  - The kdf\_input\_size must be 0.
  - The kdf\_output\_size must be 0.
  - Flags: the use of the HSM\_OP\_KEY\_EXCHANGE\_FLAGS\_GENERATE\_EPHEMERAL flag is mandatory (only freshly generated keys can be used as input of the Z derivation)
  - signed\_message: mandatory in OEM CLOSED life cycle.
- TLS Key generation:
  - Only an ephemeral key pair is supported as input of the TLS key\_exchange negotiation. This can be:
    - \* either a TRANSIENT private key already stored into the key store indicated by its key identifier. To prevent any misuse non-transient key will be rejected, additionally the private key will be deleted from the key store as part of this command handling.
    - \* either a key pair freshly generated by the use of HSM\_OP\_KEY\_EXCHANGE\_FLAGS\_GENER\_↵ATE\_EPHEMERAL flag.
  - shared\_key\_identifier\_array: it must correspond to the concatenation of client\_write\_MAC\_key id (4 bytes, if any), server\_write\_MAC\_key id (4 bytes, if any), client\_write\_key id (4 bytes), the server\_↵write\_key id (4 bytes), and the master\_secret key id (4 bytes).
  - The kdf\_input must correspond to the concatenation of clientHello\_random (32 bytes), serverHello\_↵random (32 bytes), server\_random (32 bytes) and client\_random (32 bytes).
  - kdf\_output: the concatenation of client\_write\_iv (4 bytes) and server\_write\_iv (4 bytes) will be stored at this address.

- The `shared_key_info` must have the `HSM_KEY_INFO_TRANSIENT` bit set (only transient keys can be generated), the `HSM_KEY_INFO_KEK` bit is not allowed.
  - The `shared_key_type` is not applicable and must be left to 0.
  - The `initiator_public_data_type` must be `HSM_KEY_TYPE_ECDSA_NIST_P256/384` or `HSM_KEY_TYPE_ECDSA_BRAINPOOL_R1_256/384`.
  - The `key_exchange_scheme` must be `HSM_KEY_SCHEME_ECDH_NIST_P256/384` or `HSM_KEY_SCHEME_ECDH_BRAINPOOL_R1_256/384`.
  - The `kdf_algorithm` must be `HSM_KDF_HMAC_SHA_xxx_TLS_xxx`. The generated MAC keys will have type `ALG_HMAC_XXX`, where XXX corresponds to the key length in bit of generated MAC key. The generated encryption keys will have type `HSM_KEY_TYPE_AES_XXX`, where XXX corresponds to the key length in bit of the generated AES key. The master\_secret key can only be used for the `hsm_tls_finish` function or be deleted using the `hsm_manage_key` function.
  - `kdf_input_size`: It must be 128 bytes.
  - `kdf_output_size`: It must be 8 bytes
  - `signed_message`: it must be NULL
- SM2 key generation (as specified in GB/T 32918):
    - Only the receiver role is supported.
    - `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.
    - `shared_key_info`: the `HSM_KEY_INFO_KEK` bit is not allowed.
    - The `shared_key_type` must be `HSM_KEY_TYPE_SM4_128` or `HSM_KEY_TYPE_DSA_SM2_FP_256`
    - The `initiator_public_data_type` must be `HSM_KEY_TYPE_DSA_SM2_FP_256`
    - The `key_exchange_scheme` must be `HSM_KEY_SCHEME_SM2_FP_256`.
    - The `kdf_algorithm` must be `HSM_KDF_ALG_FOR_SM2`.
    - Flags: the `HSM_OP_KEY_EXCHANGE_FLAGS_GENERATE_EPHEMERAL` flag is not supported
    - `signed_message`: it must be NULL

#### Parameters

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

#### Returns

error code

#### 5.21.3.2 hsm\_tls\_finish()

```
hsm_err_t hsm_tls_finish (
    hsm_hdl_t key_management_hdl,
    op_tls_finish_args_t * args )
```

This command is designed to compute the `verify_data` block required for the Finished message in the TLS handshake.

The input key must be a `master_secret` key generated by a previous `hsm_key_exchange` call using a TLS KDF. User can call this function only after having opened a key management service flow.

#### Parameters

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

#### Returns

error code

## 5.22 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\_KEY\_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\_TYPE\_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\_TYPE\_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_key_exp_args_t`: Only `HSM_KEY_TYPE_ECDSA_NIST_P256`, `HSM_KEY_TYPE_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_TYPE_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_SIGNATURE_SCHEME_DSA_SM2_FP_256_SM3`.

### Signature verification

- `HSM_OP_VERIFY_SIGN_FLAGS_COMPRESSED_POINT` is not supported, in case of `HSM_SIGNATURE_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_TYPE_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_TYPE_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.
- hsm\_tls\_finish 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

### Ciphering, [23](#)

- [hsm\\_auth\\_enc](#), [26](#)
- [hsm\\_cipher\\_one\\_go](#), [25](#)
- [hsm\\_close\\_cipher\\_service](#), [27](#)
- [hsm\\_ecies\\_decryption](#), [26](#)
- [hsm\\_open\\_cipher\\_service](#), [25](#)

### Data storage, [50](#)

- [hsm\\_close\\_data\\_storage\\_service](#), [51](#)
- [hsm\\_data\\_storage](#), [51](#)
- [hsm\\_open\\_data\\_storage\\_service](#), [51](#)

### ECIES encryption, [46](#)

- [hsm\\_ecies\\_encryption](#), [47](#)

### Error codes, [8](#)

- [hsm\\_err\\_t](#), [8](#)

### Get info, [56](#)

- [hsm\\_get\\_info](#), [56](#)

### Hashing, [39](#)

- [hsm\\_close\\_hash\\_service](#), [40](#)
- [hsm\\_hash\\_one\\_go](#), [41](#)
- [hsm\\_open\\_hash\\_service](#), [40](#)

### [hsm\\_auth\\_enc](#)

- Ciphering, [26](#)

### [hsm\\_butterfly\\_key\\_expansion](#)

- Key management, [21](#)

### [hsm\\_cipher\\_one\\_go](#)

- Ciphering, [25](#)

### [hsm\\_close\\_cipher\\_service](#)

- Ciphering, [27](#)

### [hsm\\_close\\_data\\_storage\\_service](#)

- Data storage, [51](#)

### [hsm\\_close\\_hash\\_service](#)

- Hashing, [40](#)

### [hsm\\_close\\_key\\_management\\_service](#)

- Key management, [22](#)

### [hsm\\_close\\_key\\_store\\_service](#)

- Key store, [14](#)

### [hsm\\_close\\_mac\\_service](#)

- Mac, [60](#)

### [hsm\\_close\\_rng\\_service](#)

- Random number generation, [38](#)

### [hsm\\_close\\_session](#)

- Session, [11](#)

### [hsm\\_close\\_signature\\_generation\\_service](#)

- Signature generation, [30](#)

### [hsm\\_close\\_signature\\_verification\\_service](#)

- Signature verification, [35](#)

### [hsm\\_close\\_sm2\\_eces\\_service](#)

- SM2 ECES decryption, [64](#)

### [hsm\\_data\\_storage](#)

- Data storage, [51](#)

### [hsm\\_ecies\\_decryption](#)

- Ciphering, [26](#)

### [hsm\\_ecies\\_encryption](#)

- ECIES encryption, [47](#)

### [hsm\\_err\\_t](#)

- Error codes, [8](#)

### [hsm\\_export\\_root\\_key\\_encryption\\_key](#)

- Root KEK export, [53](#)

### [hsm\\_generate\\_key](#)

- Key management, [20](#)

### [hsm\\_generate\\_signature](#)

- Signature generation, [30](#)

### [hsm\\_get\\_info](#)

- Get info, [56](#)

### [hsm\\_get\\_random](#)

- Random number generation, [38](#)

### [hsm\\_hash\\_one\\_go](#)

- Hashing, [41](#)

### [hsm\\_import\\_public\\_key](#)

- Signature verification, [34](#)

### [hsm\\_key\\_exchange](#)

- Key exchange, [70](#)

### [hsm\\_mac\\_one\\_go](#)

- Mac, [60](#)

### [hsm\\_manage\\_key](#)

- Key management, [20](#)

### [hsm\\_manage\\_key\\_group](#)

- Key management, [21](#)

### [hsm\\_open\\_cipher\\_service](#)

- Ciphering, [25](#)

### [hsm\\_open\\_data\\_storage\\_service](#)

- Data storage, [51](#)

### [hsm\\_open\\_hash\\_service](#)

- Hashing, [40](#)

### [hsm\\_open\\_key\\_management\\_service](#)

- Key management, [19](#)

### [hsm\\_open\\_key\\_store\\_service](#)

- Key store, [13](#)

### [hsm\\_open\\_mac\\_service](#)

- Mac, [59](#)

### [hsm\\_open\\_rng\\_service](#)

- Random number generation, [36](#)

### [hsm\\_open\\_session](#)

- Session, [11](#)

### [hsm\\_open\\_signature\\_generation\\_service](#)

- Signature generation, [30](#)

### [hsm\\_open\\_signature\\_verification\\_service](#)

- Signature verification, [33](#)

### [hsm\\_open\\_sm2\\_eces\\_service](#)

- SM2 ECES decryption, [64](#)

### [hsm\\_prepare\\_signature](#)

- Signature generation, [31](#)

### [hsm\\_pub\\_key\\_decompression](#)

- Public key decompression, [44](#)

### [hsm\\_pub\\_key\\_reconstruction](#)

- Public key reconstruction, [42](#)

### [hsm\\_pub\\_key\\_recovery](#)

- Public key recovery, 48
- hsm\_sm2\_eces\_decryption
  - SM2 ECES decryption, 64
- hsm\_sm2\_eces\_encryption
  - SM2 ECES encryption, 66
- hsm\_sm2\_get\_z
  - SM2 Get Z, 61
- hsm\_tls\_finish
  - Key exchange, 72
- hsm\_verify\_signature
  - Signature verification, 34
- i.MX8DXL specificities, 77
- i.MX8QXP specificities, 74
- Key exchange, 68
  - hsm\_key\_exchange, 70
  - hsm\_tls\_finish, 72
- Key management, 15
  - hsm\_butterfly\_key\_expansion, 21
  - hsm\_close\_key\_management\_service, 22
  - hsm\_generate\_key, 20
  - hsm\_manage\_key, 20
  - hsm\_manage\_key\_group, 21
  - hsm\_open\_key\_management\_service, 19
- Key store, 12
  - hsm\_close\_key\_store\_service, 14
  - hsm\_open\_key\_store\_service, 13
- Mac, 58
  - hsm\_close\_mac\_service, 60
  - hsm\_mac\_one\_go, 60
  - hsm\_open\_mac\_service, 59
- op\_auth\_enc\_args\_t, 24
- op\_butl\_key\_exp\_args\_t, 18
- op\_cipher\_one\_go\_args\_t, 24
- op\_data\_storage\_args\_t, 50
- op\_ecies\_dec\_args\_t, 25
- op\_ecies\_enc\_args\_t, 46
- op\_export\_root\_kek\_args\_t, 53
- op\_generate\_key\_args\_t, 17
- op\_generate\_sign\_args\_t, 29
- op\_get\_info\_args\_t, 56
- op\_get\_random\_args\_t, 36
- op\_hash\_one\_go\_args\_t, 39
- op\_import\_public\_key\_args\_t, 33
- op\_key\_exchange\_args\_t, 69
- op\_mac\_one\_go\_args\_t, 59
- op\_manage\_key\_args\_t, 18
- op\_manage\_key\_group\_args\_t, 18
- op\_prepare\_sign\_args\_t, 29
- op\_pub\_key\_dec\_args\_t, 44
- op\_pub\_key\_rec\_args\_t, 42
- op\_pub\_key\_recovery\_args\_t, 48
- op\_sm2\_eces\_dec\_args\_t, 63
- op\_sm2\_eces\_enc\_args\_t, 66
- op\_sm2\_get\_z\_args\_t, 61
- op\_tls\_finish\_args\_t, 70
- op\_verify\_sign\_args\_t, 33
- open\_session\_args\_t, 10
- open\_svc\_cipher\_args\_t, 23
- open\_svc\_data\_storage\_args\_t, 50
- open\_svc\_hash\_args\_t, 39
- open\_svc\_key\_management\_args\_t, 17
- open\_svc\_key\_store\_args\_t, 12
- open\_svc\_mac\_args\_t, 58
- open\_svc\_rng\_args\_t, 36
- open\_svc\_sign\_gen\_args\_t, 29
- open\_svc\_sign\_ver\_args\_t, 32
- open\_svc\_sm2\_eces\_args\_t, 63
- Public key decompression, 44
  - hsm\_pub\_key\_decompression, 44
- Public key reconstruction, 42
  - hsm\_pub\_key\_reconstruction, 42
- Public key recovery, 48
  - hsm\_pub\_key\_recovery, 48
- Random number generation, 36
  - hsm\_close\_rng\_service, 38
  - hsm\_get\_random, 38
  - hsm\_open\_rng\_service, 36
- Root KEK export, 53
  - hsm\_export\_root\_key\_encryption\_key, 53
- SM2 ECES decryption, 63
  - hsm\_close\_sm2\_eces\_service, 64
  - hsm\_open\_sm2\_eces\_service, 64
  - hsm\_sm2\_eces\_decryption, 64
- SM2 ECES encryption, 66
  - hsm\_sm2\_eces\_encryption, 66
- SM2 Get Z, 61
  - hsm\_sm2\_get\_z, 61
- Session, 10
  - hsm\_close\_session, 11
  - hsm\_open\_session, 11
- Signature generation, 28
  - hsm\_close\_signature\_generation\_service, 30
  - hsm\_generate\_signature, 30
  - hsm\_open\_signature\_generation\_service, 30
  - hsm\_prepare\_signature, 31
- Signature verification, 32
  - hsm\_close\_signature\_verification\_service, 35
  - hsm\_import\_public\_key, 34
  - hsm\_open\_signature\_verification\_service, 33
  - hsm\_verify\_signature, 34