

RM00284

EdgeLock Enclave Hardware Security Module API

Rev. 4.3 — 26 March 2026

Reference manual

Document information

| Information | Content |
|-------------|---|
| Keywords | i.MX, Linux, EdgeLock Enclave, Hardware Security Module, API, RM00284, LF6.18.2_1.0.0 |
| Abstract | This document is a software reference description of the API provided by the i.MX 8ULP, i.MX 91, i.MX 93, i.MX 95, and i.MX 943 Hardware Security Module (HSM) solutions for the EdgeLock Enclave (ELE) Platform. |



1 Overview

This document is a software reference description of the API provided by the i.MX 8ULP, i.MX 91, i.MX 93, i.MX 95, and i.MX 943 Hardware Security Module (HSM) solutions for the EdgeLock Enclave (ELE) Platform.

Note: A reference implementation of this API is available at <https://github.com/nxp-imx/imx-secure-enclave>. All code examples in this document use this implementation.

2 General Concepts Related to the API

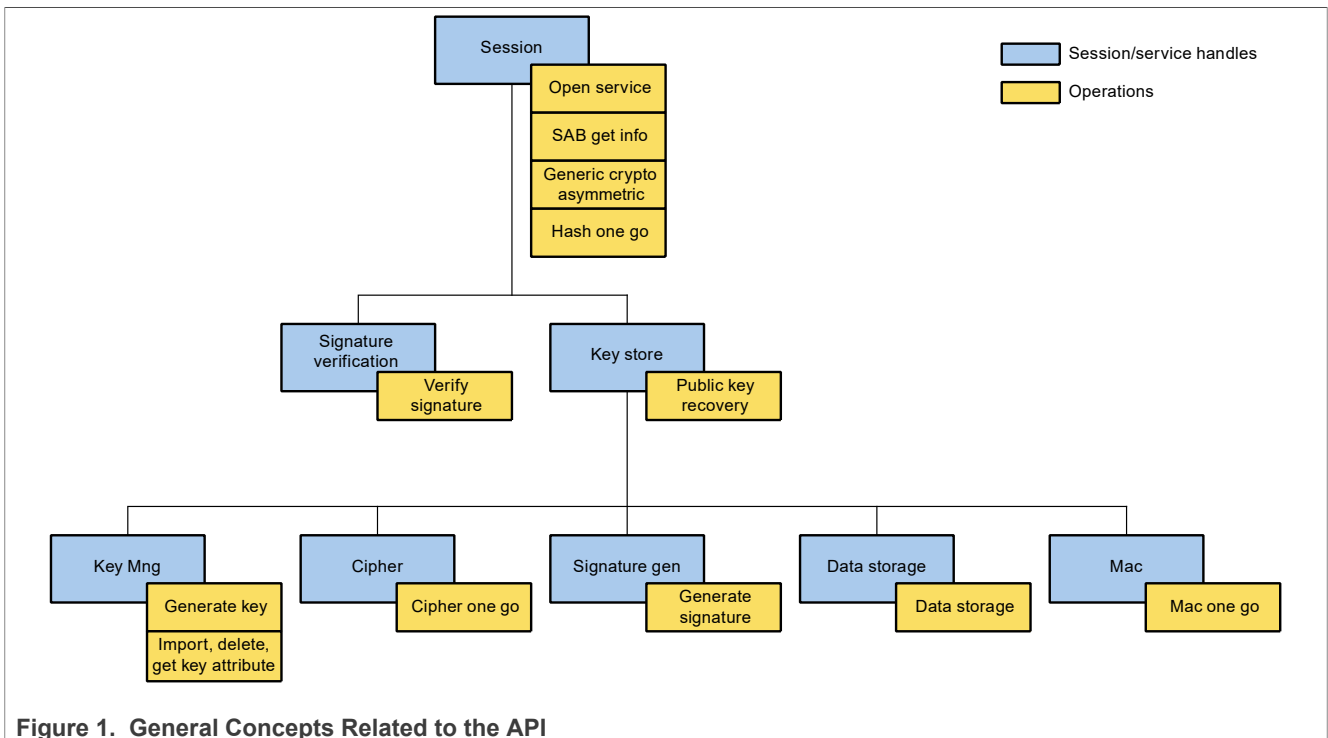


Figure 1. General Concepts Related to the API

2.1 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.

2.2 Service flow

For a given category of services that require service handle, 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.

2.3 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 cipher service it grants access to ciphering operations */
hsm_open_cipher_service(key_store_hdl, &open_svc_cipher_args, &cipher_hdl);

/* Perform ECB, CCB ... */
hsm_cipher_one_go (cipher_hdl, &op_cipher_one_go_args);
/* Perform authenticate and encryption algos: e.g 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);
```

2.4 Key store

A key store can be created by specifying the CREATE flag in the `hsm_open_key_store_service` API. The created key store is not stored in the NVM until a key is generated or 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.

2.4.1 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 2 key groups can be handled in the HSM local memory (those immediately available to perform crypto operations), while up to 100 key groups can be handled in the external NVM and imported in the local memory as needed.

If the local memory is full (2 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 to 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.

2.4.2 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. 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 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, the user should, when possible, perform multiple update before setting the "STRICT OPERATION" flag (i.e., keys to be updated should be kept in the same key group).

When 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.

2.5 Cyber-Resilient Recovery Module (CRRM)

It enables early detection of attacks and adds the capability to recover to a trusted state without human intervention.

2.6 Implementation specificities

HSM API with common features are supported on i.MX 8ULP, i.MX 91, i.MX 93, i.MX 95, and i.MX 943. The details of the supported features per chip are listed in the platform specificities.

3 Module Documentation

3.1 Session

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.

Data structures

- struct [hsm_session_hdl_s](#)
- struct [hsm_service_hdl_s](#)
- struct [open_session_args_t](#)

Macros

- #define HSM_MAX_SESSIONS (8u)
Maximum sessions supported.
- #define HSM_MAX_SERVICES (32u)
Maximum services supported.
- #define HSM_OPEN_SESSION_PRIORITY_LOW (0x00U)
Low priority. Default setting on platforms that does not support session 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 sessions are 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 is attached to this session. May provide better performances on some operations depending on the implementation. Usage of the session is restricted to the operations that do not involve secret keys (e.g., hash, signature verification, and 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.
- #define HSM1 (0x0u)
HSM1 channel on SE, best effort high priority.
- #define HSM2 (0x1u)
HSM2 channel on SE, best effort high priority.
- #define HSM_NVM (0x80u)
HSM NVM channel on SE.

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)
- struct [hsm_session_hdl_s](#) * [session_hdl_to_ptr](#) (uint32_t hdl)
- struct [hsm_service_hdl_s](#) * [service_hdl_to_ptr](#) (uint32_t hdl)
- void [delete_session](#) (struct [hsm_session_hdl_s](#) *s_ptr)
- void [delete_service](#) (struct [hsm_service_hdl_s](#) *s_ptr)
- struct [hsm_session_hdl_s](#) * [add_session](#) (void)
- struct [hsm_service_hdl_s](#) * [add_service](#) (struct [hsm_session_hdl_s](#) *session)

3.1.1 Detailed description

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

When a session is closed, all the associated service flows are closed by the HSM.

3.1.2 Data structure documentation

3.1.2.1 struct hsm_session_hdl_s

Structure describing the session handle members.

Data Fields

| | | |
|--------------------------|-----------------|--------------------------------|
| struct plat_os_abs_hdl * | phdl | Pointer to the OS device node. |
| uint32_t | session_hdl | Session handle. |
| uint32_t | mu_type | Session MU type. |
| se_lib_hdl_t | se_lib_serv_hdl | SE library service handle. |

3.1.2.2 struct hsm_service_hdl_s

Structure describing the service handle members.

Data Fields

| | | |
|--|-------------|----------------------------|
| struct hsm_session_hdl_s * | session | Pointer to session handle. |
| uint32_t | service_hdl | Service handle. |

3.1.2.3 struct open_session_args_t

Structure detailing the open session operation member arguments.

Data Fields

| | | |
|----------|---------------|--|
| uint32_t | session_hdl | Session handle. |
| uint8_t | mu_type | MU with associated functionality on which the session is opened. |
| uint8_t | interrupt_idx | Interrupt number of the MU used to indicate data availability. |

3.1.3 Typedef documentation

3.1.3.1 hsm_hdl_t

```
typedef uint32_t hsm_hdl_t
```

Define the HSM handle type.

3.1.4 Function documentation

3.1.4.1 hsm_open_session()

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

Parameters

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

Returns

Error code.

3.1.4.2 hsm_close_session()

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

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

Parameters

| | |
|-------------|---|
| session_hdl | Pointer to the handle identifying the session to be closed. |
|-------------|---|

Returns

Error code.

3.1.4.3 session_hdl_to_ptr()

```
struct hsm_session_hdl_s* session_hdl_to_ptr (uint32_t hdl)
```

Return pointer to the session handle.

Parameters

| | |
|------------|---------------------------------|
| <i>hdl</i> | Identifying the session handle. |
|------------|---------------------------------|

Returns

Pointer to the session handle.

3.1.4.4 service_hdl_to_ptr()

```
struct hsm\_service\_hdl\_s* service_hdl_to_ptr (uint32_t hdl)
```

Return pointer to the service handle.

Parameters

| | |
|------------|---------------------------------|
| <i>hdl</i> | Identifying the session handle. |
|------------|---------------------------------|

Returns

Pointer to the service handle.

3.1.4.5 delete_session()

```
void delete_session (struct hsm\_session\_hdl\_s * s_ptr)
```

Delete the session.

Parameters

| | |
|--------------|----------------------------------|
| <i>s_ptr</i> | Pointer identifying the session. |
|--------------|----------------------------------|

3.1.4.6 delete_service()

```
void delete_service (struct hsm\_service\_hdl\_s * s_ptr)
```

Delete the service.

Parameters

| | |
|--------------|----------------------------------|
| <i>s_ptr</i> | Pointer identifying the service. |
|--------------|----------------------------------|

3.1.4.7 add_session()

```
struct hsm\_session\_hdl\_s* add_session (void )
```

Add the session.

Returns

Pointer to the session.

3.1.4.8 add_service()

```
struct hsm\_service\_hdl\_s* add_service (
```

```
struct hsm\_session\_hdl\_s * session)
```

Add the service.

Returns

Pointer to the service.

3.2 Key management

Data Structures

- struct [op_delete_key_args_t](#)
- struct [op_get_key_attr_args_t](#)
- struct [op_import_key_args_t](#)
- struct [kek_enc_key_hdr_t](#)
- struct [op_generate_key_args_t](#)
- struct [open_svc_key_management_args_t](#)
- struct [op_manage_key_group_args_t](#)

Macros

- #define [HSM_OP_DEL_KEY_FLAGS_STRICT_OPERATION](#) (([hsm_op_delete_key_flags_t](#))(1u << 7))
- #define [HSM_OP_IMPORT_KEY_INPUT_E2GO_TLV](#) (([hsm_op_import_key_flags_t](#))(1u << 0))
Bit 0: set 1 means input is E2GO_TLV.
- #define [HSM_OP_IMPORT_KEY_INPUT_SIGNED_MSG](#) (([hsm_op_import_key_flags_t](#))(0u << 0))
Bit 0: set 0 means input is signed message.
- #define [HSM_OP_IMPORT_KEY_FLAGS_AUTOMATIC_GROUP](#) (([hsm_op_import_key_flags_t](#))(0u << 2))
Bit 2: set 0 means ELE automatically chooses key group.
- #define [HSM_OP_IMPORT_KEY_FLAGS_GROUP_FIELD](#) (([hsm_op_import_key_flags_t](#))(1u << 2))
Bit 2: set 1 means ELE stores key in the key group set by the key group field.
- #define [HSM_OP_IMPORT_KEY_FLAGS_STRICT_OPERATION](#) (([hsm_op_import_key_flags_t](#)) (1u << 7))
Bit 7: Strict: Request completed - New key written to NVM with updated MC.
- #define [HSM_KEY_USAGE_ENCRYPT](#) (([hsm_key_usage_t](#)) (1u << 8))
- #define [HSM_KEY_USAGE_DECRYPT](#) (([hsm_key_usage_t](#)) (1u << 9))
- #define [HSM_KEY_USAGE_SIGN_MSG](#) (([hsm_key_usage_t](#)) (1u << 10))
- #define [HSM_KEY_USAGE_VERIFY_MSG](#) (([hsm_key_usage_t](#)) (1u << 11))
- #define [HSM_KEY_USAGE_SIGN_HASH](#) (([hsm_key_usage_t](#)) (1u << 12))
- #define [HSM_KEY_USAGE_VERIFY_HASH](#) (([hsm_key_usage_t](#)) (1u << 13))
- #define [HSM_KEY_USAGE_DERIVE](#) (([hsm_key_usage_t](#)) (1u << 14))
- #define [HSM_OP_KEY_GENERATION_FLAGS_STRICT_OPERATION](#) (([hsm_op_key_gen_flags_t](#))(1u << 7))
- #define [HSM_OP_KEY_GENERATION_FLAGS_PLAINTEXT_KEY](#) (([hsm_op_key_gen_flags_t](#))(1u << 3))
- #define [HSM_OP_KEY_GENERATION_FLAGS_MONOTONIC_COUNTER](#) (([hsm_op_key_gen_flags_t](#))(1u << 5))
- #define [HSM_OP_MANAGE_KEY_GROUP_FLAGS_CACHE_LOCKDOWN](#) (([hsm_op_manage_key_group_flags_t](#)) (1u << 0))
- #define [HSM_OP_MANAGE_KEY_GROUP_FLAGS_CACHE_UNLOCK](#) (([hsm_op_manage_key_group_flags_t](#)) (1u << 1))
Import the key group.
- #define [HSM_OP_MANAGE_KEY_GROUP_FLAGS_IMPORT](#) (([hsm_op_manage_key_group_flags_t](#)) (1u << 2))

Export the key group.

- #define [HSM_OP_MANAGE_KEY_GROUP_FLAGS_EXPORT](#) ((hsm_op_manage_key_group_flags_t) (1u << 3))
- #define [HSM_OP_MANAGE_KEY_GROUP_FLAGS_MONOTONIC](#)
- ((hsm_op_manage_key_group_flags_t) (1u << 5))
- #define [HSM_OP_MANAGE_KEY_GROUP_FLAGS_SYNC_KEYSTORE](#) ((hsm_op_manage_key_group_flags_t) (1u << 6))
- #define [HSM_OP_MANAGE_KEY_GROUP_FLAGS_STRICT_OPERATION](#) ((hsm_op_manage_key_group_flags_t) (1u << 7))

Typedefs

- typedef uint8_t [hsm_op_delete_key_flags_t](#)
- typedef uint8_t [hsm_op_import_key_flags_t](#)
- typedef uint32_t [hsm_key_usage_t](#)
- typedef uint16_t [hsm_key_group_t](#)
- typedef uint16_t [hsm_key_info_t](#)
- typedef uint8_t [hsm_op_key_gen_flags_t](#)
Reserved Bits 0 - 6.
- typedef uint8_t [hsm_svc_key_management_flags_t](#)
- typedef uint8_t [hsm_op_manage_key_group_flags_t](#)

Enumerations

- enum [hsm_storage_loc_t](#) {
HSM_SE_KEY_STORAGE = 0x00000000 }
- enum [hsm_storage_persist_lvl_t](#) {
HSM_VOLATILE_STORAGE = 0x0,
HSM_PERSISTENT_STORAGE = 0x1,
HSM_PERMANENT_STORAGE = 0xFF }
- enum [hsm_key_lifetime_t](#) {
HSM_SE_KEY_STORAGE_VOLATILE = HSM_SE_KEY_STORAGE | HSM_VOLATILE_STORAGE,
HSM_SE_KEY_STORAGE_PERSISTENT = HSM_SE_KEY_STORAGE | HSM_PERSISTENT_STORAGE,
HSM_SE_KEY_STORAGE_PERS_PERM = HSM_SE_KEY_STORAGE | HSM_PERMANENT_STORAGE }
- enum [hsm_pubkey_type_t](#) {
HSM_PUBKEY_TYPE_RSA = 0x4001,
HSM_PUBKEY_TYPE_ECC_BP_R1 = 0x4130,
HSM_PUBKEY_TYPE_ECC_NIST = 0x4112,
HSM_PUBKEY_TYPE_ECC_MONTGOMERY = 0x4141,
HSM_PUBKEY_TYPE_ECC_TWISTED_EDWARDS = 0x4142,
HSM_PUBKEY_TYPE_ECC_BP_T1 = 0xC180 }
- enum [hsm_key_type_t](#) {
HSM_KEY_TYPE_DERIVE = 0x1200,
HSM_KEY_TYPE_HMAC = 0x1100,
HSM_KEY_TYPE_CHACHA20_POLY1305 = 0x2004,
HSM_KEY_TYPE_AES = 0x2400,
HSM_KEY_TYPE_SM4 = 0x2405,
HSM_KEY_TYPE_RSA = 0x7001,
HSM_KEY_TYPE_ECC_BP_R1 = 0x7130,
HSM_KEY_TYPE_ECC_NIST = 0x7112,
HSM_KEY_TYPE_ECC_MONTGOMERY = 0x7141,
HSM_KEY_TYPE_ECC_TWISTED_EDWARDS = 0x7142,
HSM_KEY_TYPE_OEM_IMPORT_MK_SK = 0x9200 }

- enum [hsm_bit_key_sz_t](#) {
HSM_KEY_SIZE_DERIVE_256 = 256,
HSM_KEY_SIZE_DERIVE_384 = 384,
HSM_KEY_SIZE_HMAC_224 = 224,
HSM_KEY_SIZE_HMAC_256 = 256,
HSM_KEY_SIZE_HMAC_384 = 384,
HSM_KEY_SIZE_HMAC_512 = 512,
HSM_KEY_SIZE_AES_128 = 128,
HSM_KEY_SIZE_AES_192 = 192,
HSM_KEY_SIZE_AES_256 = 256,
HSM_KEY_SIZE_CHACHA20_POLY1305_256 = 256,
HSM_KEY_SIZE_SM4_128 = 128,
HSM_KEY_SIZE_RSA_2048 = 2048,
HSM_KEY_SIZE_RSA_3072 = 3072,
HSM_KEY_SIZE_RSA_4096 = 4096,
HSM_KEY_SIZE_ECC_BP_R1_224 = 224,
HSM_KEY_SIZE_ECC_BP_R1_256 = 256,
HSM_KEY_SIZE_ECC_BP_R1_320 = 320,
HSM_KEY_SIZE_ECC_BP_R1_384 = 384,
HSM_KEY_SIZE_ECC_BP_R1_512 = 512,
HSM_KEY_SIZE_ECC_NIST_224 = 224,
HSM_KEY_SIZE_ECC_NIST_256 = 256,
HSM_KEY_SIZE_ECC_NIST_384 = 384,
HSM_KEY_SIZE_ECC_NIST_521 = 521,
HSM_KEY_SIZE_ECC_MONTGOMERY_255 = 255,
HSM_KEY_SIZE_ECC_MONTGOMERY_448 = 448,
HSM_KEY_SIZE_ECC_TWISTED_EDWARDS_255 = 255,
HSM_KEY_SIZE_ECC_TWISTED_EDWARDS_448 = 448,
HSM_KEY_SIZE_ECC_BP_T1_224 = 224,
HSM_KEY_SIZE_ECC_BP_T1_256 = 256,
HSM_KEY_SIZE_ECC_BP_T1_320 = 320,
HSM_KEY_SIZE_ECC_BP_T1_384 = 384,
HSM_KEY_SIZE_OEM_IMPORT_MK_SK_128 = 128,
HSM_KEY_SIZE_OEM_IMPORT_MK_SK_192 = 192,
HSM_KEY_SIZE_OEM_IMPORT_MK_SK_256 = 256 }
• enum [hsm_permitted_algo_t](#) {
PERMITTED_ALGO_SHA224 = ALGO_HASH_SHA224,
PERMITTED_ALGO_SHA256 = ALGO_HASH_SHA256,
PERMITTED_ALGO_SHA384 = ALGO_HASH_SHA384,
PERMITTED_ALGO_SHA512 = ALGO_HASH_SHA512,
PERMITTED_ALGO_SM3 = ALGO_HASH_SM3,
PERMITTED_ALGO_HMAC_SHA256 = ALGO_HMAC_SHA256,
PERMITTED_ALGO_HMAC_SHA384 = ALGO_HMAC_SHA384,
PERMITTED_ALGO_CMAC = ALGO_CMAC,
PERMITTED_ALGO_CBCMAC = ALGO_CBCMAC,
PERMITTED_ALGO_CTR = ALGO_CIPHER_CTR,
PERMITTED_ALGO_CFB = ALGO_CIPHER_CFB,
PERMITTED_ALGO_OFB = ALGO_CIPHER_OFB,
PERMITTED_ALGO_ECB_NO_PADDING = ALGO_CIPHER_ECB_NO_PAD,
PERMITTED_ALGO_CBC_NO_PADDING = ALGO_CIPHER_CBC_NO_PAD,
PERMITTED_ALGO_CCM = ALGO_CCM,
PERMITTED_ALGO_GCM = ALGO_GCM,

PERMITTED_ALGO_CHACHA20_POLY1305 = ALGO_CHACHA20_POLY1305,
 PERMITTED_ALGO_RSA_PKCS1_V15_SHA_ANY = ALGO_RSA_PKCS1_V15_SHA_ANY,
 PERMITTED_ALGO_RSA_PKCS1_PSS_MGF1_SHA_ANY = ALGO_RSA_PKCS1_PSS_MGF1_SHA_ANY,
 PERMITTED_ALGO_RSA_PKCS1_V15_SHA224 = ALGO_RSA_PKCS1_V15_SHA224,
 PERMITTED_ALGO_RSA_PKCS1_V15_SHA256 = ALGO_RSA_PKCS1_V15_SHA256,
 PERMITTED_ALGO_RSA_PKCS1_V15_SHA384 = ALGO_RSA_PKCS1_V15_SHA384,
 PERMITTED_ALGO_RSA_PKCS1_V15_SHA512 = ALGO_RSA_PKCS1_V15_SHA512,
 PERMITTED_ALGO_RSA_PKCS1_PSS_MGF1_SHA224 = ALGO_RSA_PKCS1_PSS_MGF1_SHA224,
 PERMITTED_ALGO_RSA_PKCS1_PSS_MGF1_SHA256 = ALGO_RSA_PKCS1_PSS_MGF1_SHA256,
 PERMITTED_ALGO_RSA_PKCS1_PSS_MGF1_SHA384 = ALGO_RSA_PKCS1_PSS_MGF1_SHA384,
 PERMITTED_ALGO_RSA_PKCS1_PSS_MGF1_SHA512 = ALGO_RSA_PKCS1_PSS_MGF1_SHA512,
 PERMITTED_ALGO_RSA_SIGNATURE_ALL = ALGO_RSA_SIGNATURE_ALL,
 PERMITTED_ALGO_ECDSA_SHA224 = ALGO_ECDSA_SHA224,
 PERMITTED_ALGO_ECDSA_SHA256 = ALGO_ECDSA_SHA256,
 PERMITTED_ALGO_ECDSA_SHA384 = ALGO_ECDSA_SHA384,
 PERMITTED_ALGO_ECDSA_SHA512 = ALGO_ECDSA_SHA512,
 PERMITTED_ALGO_ED25519PH = ALGO_ED25519PH,
 PERMITTED_ALGO_ED448PH = ALGO_ED448PH,
 PERMITTED_ALGO_PURE_EDDSA = ALGO_PURE_EDDSA,
 PERMITTED_ALGO_EDDSA_ALL = ALGO_EDDSA_ALL,
 PERMITTED_ALGO_HMAC_KDF_SHA256 = ALGO_HMAC_KDF_SHA256,
 PERMITTED_ALGO_ALL_CIPHER = ALGO_CIPHER_ALL,
 PERMITTED_ALGO_ALL_AEAD = ALGO_ALL_AEAD,
 PERMITTED_ALGO_OTH_KEK_CBC = ALGO_CIPHER_KEK_CBC,
 PERMITTED_ALGO_KEY_EX_ECDH_HKDF_SHA256_KEY_IMPORT =
 ALGO_ECDH_HKDF_SHA256_KEY_IMPORT,
 PERMITTED_ALGO_KEY_EX_ECDH_HKDF_SHA384_KEY_IMPORT =
 ALGO_ECDH_HKDF_SHA384_KEY_IMPORT,
 PERMITTED_ALGO_KEY_EX_ECDH_HKDF_SHA_ANY_KEY_IMPORT =
 ALGO_ECDH_HKDF_SHA_ANY_KEY_IMPORT,
 PERMITTED_ALGO_KEY_EX_ECDH_HKDF_SHA256 = ALGO_ECDH_HKDF_SHA256,
 PERMITTED_ALGO_KEY_EX_ECDH_HKDF_SHA384 = ALGO_ECDH_HKDF_SHA384,
 PERMITTED_ALGO_KEY_EX_ECDH_HKDF_SHA_ANY = ALGO_ECDH_HKDF_SHA_ANY,
 PERMITTED_ALGO_ECDH_HKDF_SHA256 = ALGO_ECDH_HKDF_SHA256_KEY_IMPORT,
 PERMITTED_ALGO_ECDH_HKDF_SHA384 = ALGO_ECDH_HKDF_SHA384_KEY_IMPORT,
 PERMITTED_ALGO_HKDF_EXTRACT_SHA256 = ALGO_HKDF_EXTRACT_SHA256,
 PERMITTED_ALGO_HKDF_EXTRACT_SHA384 = ALGO_HKDF_EXTRACT_SHA384,
 PERMITTED_ALGO_HKDF_EXTRACT_SHA_ANY = ALGO_HKDF_EXTRACT_SHA_ANY,
 PERMITTED_ALGO_HKDF_EXPAND_SHA256 = ALGO_HKDF_EXPAND_SHA256,
 PERMITTED_ALGO_HKDF_EXPAND_SHA384 = ALGO_HKDF_EXPAND_SHA384,
 PERMITTED_ALGO_TLS1_2_MASTER_SECRET_SHA256 = ALGO_TLS1_2_MASTER_SECRET_SHA256,
 PERMITTED_ALGO_TLS1_2_MASTER_SECRET_SHA384 = ALGO_TLS1_2_MASTER_SECRET_SHA384,
 PERMITTED_ALGO_TLS1_2_MASTER_SECRET_SHA_ANY =
 ALGO_TLS1_2_MASTER_SECRET_SHA_ANY,
 PERMITTED_ALGO_TLS1_3_MASTER_SECRET_SHA256 = ALGO_TLS1_3_MS_SHA256,
 PERMITTED_ALGO_TLS1_3_MASTER_SECRET_SHA384 = ALGO_TLS1_3_MS_SHA384,
 PERMITTED_ALGO_TLS1_3_MASTER_SECRET_SHA_ANY = ALGO_TLS1_3_MS_SHA_ANY,
 PERMITTED_ALGO_ATTEST_CMACE = ALGO_ATTEST_CMACE,
 PERMITTED_ALGO_ATTEST_ECDSA_SHA224 = ALGO_ATTEST_ECDSA_SHA224,
 PERMITTED_ALGO_ATTEST_ECDSA_SHA256 = ALGO_ATTEST_ECDSA_SHA256,
 PERMITTED_ALGO_ATTEST_ECDSA_SHA384 = ALGO_ATTEST_ECDSA_SHA384,
 PERMITTED_ALGO_ATTEST_ECDSA_SHA512 = ALGO_ATTEST_ECDSA_SHA512,

```

PERMITTED_ALGO_RSA_PKCS1_V15_CRYPT = ALGO_RSA_PKCS1_V15_CRYPT,
PERMITTED_ALGO_RSA_PKCS1_OAEP_SHA1 = ALGO_RSA_PKCS1_OAEP_SHA1,
PERMITTED_ALGO_RSA_PKCS1_OAEP_SHA224 = ALGO_RSA_PKCS1_OAEP_SHA224,
PERMITTED_ALGO_RSA_PKCS1_OAEP_SHA256 = ALGO_RSA_PKCS1_OAEP_SHA256,
PERMITTED_ALGO_RSA_PKCS1_OAEP_SHA384 = ALGO_RSA_PKCS1_OAEP_SHA384,
PERMITTED_ALGO_RSA_PKCS1_OAEP_SHA512 = ALGO_RSA_PKCS1_OAEP_SHA512,
PERMITTED_ALGO_RSA_PKCS1_OAEP_SHA_ANY = ALGO_RSA_PKCS1_OAEP_SHA_ANY,
PERMITTED_ALGO_RSA_PKCS1_CRYPT_ALL = ALGO_RSA_PKCS1_CRYPT_ALL }
    
```

- enum [hsm_key_lifecycle_t](#) {
 - HSM_KEY_LIFECYCLE_OPEN = 0x1,
 - HSM_KEY_LIFECYCLE_CLOSED = 0x2,
 - HSM_KEY_LIFECYCLE_CLOSED_LOCKED = 0x4,
 - HSM_KEY_LIFECYCLE_INVALID = 0xFF }

Functions

- [hsm_err_t hsm_delete_key](#) ([hsm_hdl_t](#) key_management_hdl, [op_delete_key_args_t](#) *args)
- [hsm_err_t hsm_get_key_attr](#) ([hsm_hdl_t](#) key_management_hdl, [op_get_key_attr_args_t](#) *args)
- [hsm_err_t hsm_import_key](#) ([hsm_hdl_t](#) key_management_hdl, [op_import_key_args_t](#) *args)
- [hsm_err_t hsm_generate_key](#) ([hsm_hdl_t](#) key_management_hdl, [op_generate_key_args_t](#) *args)
- [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_close_key_management_service](#) ([hsm_hdl_t](#) key_management_hdl)
- [hsm_err_t hsm_manage_key_group](#) ([hsm_hdl_t](#) key_management_hdl, [op_manage_key_group_args_t](#) *args)

The entire key group is cached in the HSM local memory.

3.2.1 Detailed description

3.2.2 Data structure documentation

3.2.2.1 struct op_delete_key_args_t

Structure detailing the delete key operation member arguments.

Data Fields

| | | |
|---|----------------|---|
| uint32_t | key_identifier | Identifier of the key to be used for the operation. |
| hsm_op_delete_key_flags_t | flags | Bitmap specifying the operation properties. |

3.2.2.2 struct op_get_key_attr_args_t

Structure describing the get key attribute operation arguments.

Data Fields

| | | |
|------------------------------------|----------------|--|
| uint32_t | key_identifier | Identifier of the key to be used for the operation. |
| hsm_key_type_t | key_type | Indicates which type of key must be generated. |
| hsm_bit_key_sz_t | bit_key_sz | Indicates key security size in bits. |
| hsm_key_lifetime_t | key_lifetime | This attribute comprises of two indicators-key persistence level and location where the key is stored. |
| hsm_key_usage_t | key_usage | Indicates the cryptographic operations that key can execute. |

Data Fields...continued

| | | |
|--------------------------------------|----------------|--|
| hsm_permitted_algo_t | permitted_algo | Indicates the key permitted algorithm. |
| hsm_key_lifecycle_t | lifecycle | Indicates the device lifecycle in which key is usable. |

3.2.2.3 struct op_import_key_args_t

Structure detailing the import key operation member arguments.

Data Fields

| | | |
|---|----------------|---|
| uint32_t | key_identifier | Identifier of the KEK used to encrypt the key to be imported (Ignored if KEK is not used as set as part of the "flags" field). |
| uint8_t * | input_lsb_addr | Address in the requester space where: <ul style="list-style-type: none"> EdgeLock 2GO TLV can be found. Ignore this field if not E2GO_TLV. |
| uint32_t | input_size | Size in bytes of: <ul style="list-style-type: none"> EdgeLock 2GO TLV can be found. Ignore this field if not E2GO_TLV. |
| hsm_op_import_key_flags_t | flags | Bitmap specifying the operation properties. |
| uint16_t | key_group | In case of import key ELE option: <ul style="list-style-type: none"> The imported key group. Ignore this field if it is not the ELE option. |

3.2.2.4 struct kek_enc_key_hdr_t

Structure describing the encryption key header.

Data Fields

| | | |
|-----------|---------------|---|
| uint8_t | iv[IV_LENGTH] | - |
| uint8_t * | key | - |
| uint32_t | tag | - |

3.2.2.5 struct op_generate_key_args_t

Structure describing the generate key operation member arguments.

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 is stored in this location. To use the key identifier (for example, opaque key), ensure that the flag for the PLAINTEXT key is not set. |
| uint16_t | out_size | Length of the generated public key in bytes. 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 of 0-99. The keys belonging to the same group can be cached in the HSM local memory through the hsm_manage_key_group API. |

Data Fields...continued

| | | |
|----------------------|---------------------|--|
| | | Reserved in case of PLAINTEXT flag usage. |
| uint8_t * | out_key | Pointer to the output area where the generated public key must be written. |
| uint16_t | exp_out_size | Expected public key output buffer size, valid in case of HSM_OUT_TOO_SMALL (0x1D) error code. |
| hsm_bit_key_sz_t | bit_key_sz | Indicates the key security size in bits. |
| hsm_key_lifecycle_t | key_lifecycle | Defines the key lifecycle in which the key is usable. If it is set to 0 , the current key lifecycle is used. Reserved in case of PLAINTEXT flag usage. |
| hsm_key_lifetime_t | key_lifetime | This attribute includes two indicators: key persistence level and location where the key is stored. Reserved in case of PLAINTEXT flag usage. |
| hsm_key_usage_t | key_usage | Indicates the cryptographic operations that the key can execute. Reserved in case of PLAINTEXT flag usage. |
| hsm_permitted_algo_t | permitted_algo | Indicates the key permitted algorithm. Reserved in case of PLAINTEXT flag usage. |
| uint16_t | priv_out_key_sz | Length of the generated private key in bytes. Only valid with the PLAINTEXT flag. |
| uint8_t * | priv_out_key | Pointer to the output area where the generated private key must be written. Only valid with the PLAINTEXT flag. |
| uint16_t | exp_priv_out_key_sz | Expected private key output buffer size, valid in case of HSM_OUT_TOO_SMALL (0x1D) error code. Only valid with the PLAINTEXT flag. |

3.2.2.6 struct open_svc_key_management_args_t

Structure detailing the key management open service member arguments.

Data Fields

| | | |
|--|--------------------|---|
| hsm_hdl_t | key_management_hdl | Handle identifying the key management service flow. |
| hsm_svc_key_management_flags_t | flags | Bitmap specifying the services properties. |

3.2.2.7 struct op_manage_key_group_args_t

Data Fields

| | | |
|---------------------------------|-----------|--|
| hsm_key_group_t | key_group | It must be a value in the range of 0-99. Keys belonging to the same group can be cached in the HSM local memory through the <code>hsm_manage_key_group</code> API. |
| hsm_op_manage_key_group_flags_t | flags | Bitmap specifying the operation properties. |
| uint8_t | reserved | - |

3.2.3 Macro definition documentation

3.2.3.1 HSM_OP_DEL_KEY_FLAGS_STRICT_OPERATION

```
#define HSM_OP_DEL_KEY_FLAGS_STRICT_OPERATION ((hsm_op_delete_key_flags_t) (1u << 7))
```

Bitmap detailing the delete key operation properties.

- Bits 0-4: Reserved.
- Bit 5: Monotonic counter increment.
- Bit 6: Reserved.
- Bit 7: Strict: Request completed - New key written to NVM with updated MC.

3.2.3.2 HSM_KEY_USAGE_ENCRYPT

```
#define HSM_KEY_USAGE_ENCRYPT ((hsm_key_usage_t) (1u << 8))
```

Bit indicating the permission to encrypt a message with the key.

3.2.3.3 HSM_KEY_USAGE_DECRYPT

```
#define HSM_KEY_USAGE_DECRYPT ((hsm_key_usage_t) (1u << 9))
```

Bit indicating the permission to decrypt a message with the key.

3.2.3.4 HSM_KEY_USAGE_SIGN_MSG

```
#define HSM_KEY_USAGE_SIGN_MSG ((hsm_key_usage_t) (1u << 10))
```

Bit indicating the permission to sign a message with the key.

3.2.3.5 HSM_KEY_USAGE_VERIFY_MSG

```
#define HSM_KEY_USAGE_VERIFY_MSG ((hsm_key_usage_t) (1u << 11))
```

Bit indicating the permission to verify a message signature with the key.

3.2.3.6 HSM_KEY_USAGE_SIGN_HASH

```
#define HSM_KEY_USAGE_SIGN_HASH ((hsm_key_usage_t) (1u << 12))
```

Bit indicating the permission to sign a hashed message with the key.

3.2.3.7 HSM_KEY_USAGE_VERIFY_HASH

```
#define HSM_KEY_USAGE_VERIFY_HASH ((hsm_key_usage_t) (1u << 13))
```

Bit indicating the permission to verify a hashed message signature with the key.

3.2.3.8 HSM_KEY_USAGE_DERIVE

```
#define HSM_KEY_USAGE_DERIVE ((hsm_key_usage_t) (1u << 14))
```

Bit indicating the permission to derive other keys from this key.

3.2.3.9 HSM_OP_KEY_GENERATION_FLAGS_PLAINTEXT_KEY

```
#define HSM_OP_KEY_GENERATION_FLAGS_PLAINTEXT_KEY ((hsm_op_key_gen_flags_t) (1u << 3))
```

Plaintext key. The key is generated and exported to user's memory.

3.2.3.10 HSM_OP_KEY_GENERATION_FLAGS_MONOTONIC_COUNTER

```
#define HSM_OP_KEY_GENERATION_FLAGS_MONOTONIC_COUNTER ((hsm_op_key_gen_flags_t) (1u << 5))
```

Monotonic counter increment used in conjunction with SYNC. The request is completed only when the monotonic counter has been updated.

3.2.3.11 HSM_OP_KEY_GENERATION_FLAGS_STRICT_OPERATION

```
#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 is written in the NVM. This is applicable for the persistent and permanent keys.

3.2.3.12 HSM_OP_MANAGE_KEY_GROUP_FLAGS_CACHE_LOCKDOWN

```
#define HSM_OP_MANAGE_KEY_GROUP_FLAGS_CACHE_LOCKDOWN ((hsm_op_manage_key_group_flags_t) (1u << 0))
```

The entire key group is cached in the HSM local memory.

3.2.3.13 HSM_OP_MANAGE_KEY_GROUP_FLAGS_EXPORT

```
#define HSM_OP_MANAGE_KEY_GROUP_FLAGS_EXPORT ((hsm_op_manage_key_group_flags_t) (1u << 3))
```

Export the key group.

3.2.3.14 HSM_OP_MANAGE_KEY_GROUP_FLAGS_MONOTONIC

```
#define HSM_OP_MANAGE_KEY_GROUP_FLAGS_MONOTONIC ((hsm_op_manage_key_group_flags_t) (1u << 5))
```

When used in conjunction with the SYNC key group or SYNC key store and storage only, the request is completed only when the monotonic counter is updated.

3.2.3.15 HSM_OP_MANAGE_KEY_GROUP_FLAGS_SYNC_KEYSTORE

```
#define
HSM_OP_MANAGE_KEY_GROUP_FLAGS_SYNC_KEYSTORE ((hsm_op_manage_key_group_flags_t)
(1u << 6))
```

The request is completed only when the update is written in the NVM. Not applicable for cache lockdown/unlock.

3.2.4 Typedef documentation

3.2.4.1 hsm_op_delete_key_flags_t

```
typedef uint8_t hsm\_op\_delete\_key\_flags\_t
```

Bitmap describing the delete key operation properties.

3.2.4.2 hsm_op_import_key_flags_t

```
typedef uint8_t hsm\_op\_import\_key\_flags\_t
```

Bitmap specifying the import key operation supported properties.

- Bit 0: Defines input configuration.
- Bits 1-4: Reserved.
- Bit 5: Monotonic counter increment.
- Bit 6: Reserved.
- Bit 7: Strict.

3.2.4.3 hsm_key_usage_t

```
typedef uint32_t hsm\_key\_usage\_t
```

Bitmap indicating the cryptographic operations that the key can execute.

3.2.4.4 hsm_key_group_t

```
typedef uint16_t hsm\_key\_group\_t
```

Bit field indicating the key group.

3.2.4.5 hsm_key_info_t

```
typedef uint16_t hsm\_key\_info\_t
```

Bit field indicating the key information.

3.2.4.6 hsm_op_key_gen_flags_t

```
typedef uint8_t hsm\_op\_key\_gen\_flags\_t
```

Reserved Bits 0 - 6.

Bitmap specifying the key generate operation supported properties.

3.2.4.7 hsm_svc_key_management_flags_t

```
typedef uint8_t hsm\_svc\_key\_management\_flags\_t
```

Bitmap specifying the key management service supported properties.

3.2.5 Enumeration type documentation

3.2.5.1 hsm_storage_loc_t

```
enum hsm\_storage\_loc\_t
```

Enum indicating the key location indicator.

3.2.5.2 hsm_storage_persist_lvl_t

```
enum hsm\_storage\_persist\_lvl\_t
```

Enum indicating the key persistent level indicator.

3.2.5.3 hsm_key_lifetime_t

```
enum hsm\_key\_lifetime\_t
```

Enum indicating the key lifetime.

3.2.5.4 hsm_pubkey_type_t

```
enum hsm\_pubkey\_type\_t
```

Enum indicating the public key type.

3.2.5.5 hsm_key_type_t

```
enum hsm\_key\_type\_t
```

Enum indicating the key type.

3.2.5.6 hsm_bit_key_sz_t

```
enum hsm\_bit\_key\_sz\_t
```

Enum indicating the key security size in bits.

3.2.5.7 hsm_permitted_algo_t

```
enum hsm\_permitted\_algo\_t
```

Enum describing the permitted algorithm.

Permitted algorithm attribute.

- MAC default algorithms supported
- HMAC truncated permitted algorithm encoding, CMAC truncated permitted algorithm encoding, and Cipher algorithms supported
- AEAD algorithms supported
- Signature algorithms supported

Note: Only one of the previous algorithms could be set.

3.2.5.8 hsm_key_lifecycle_t

```
enum hsm\_key\_lifecycle\_t
```

Enum detailing the permitted key lifecycle.

3.2.6 Function documentation

3.2.6.1 hsm_delete_key()

```
hsm\_err\_t hsm_delete_key (
hsm\_hdl\_t key_management_hdl,
op\_delete\_key\_args\_t * args)
```

This command is designed to perform the following operation:

- Delete an existing key.

Parameters

| | |
|------------------------------------|---|
| key_management_hdl | Handle identifying the key management service flow. |
| args | Pointer to the structure containing the function arguments. |

Returns

Error code

3.2.6.2 hsm_get_key_attr()

```
hsm\_err\_t hsm_get_key_attr(
hsm\_hdl\_t key_management_hdl,
op\_get\_key\_attr\_args\_t * args)
```

This command is designed to perform the following operation:

- Get attributes of an existing key.

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

3.2.6.3 hsm_import_key()

```
hsm_err_t hsm_import_key (
hsm_hdl_t key_management_hdl,
op_import_key_args_t * args)
```

This API is used to import the key.

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

3.2.6.4 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.

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

3.2.6.5 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.

Users must open this service flow 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.

3.2.6.6 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

3.2.6.7 hsm_manage_key_group()

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

The entire key group is cached in the HSM local memory.

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 the local memory as needed.
- Delete an existing key group.

Users 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

3.3 Cipherring**Modules**

- [i.MX 8ULP](#)

Data Structures

- struct [op_auth_enc_args_t](#)
- struct [open_svc_cipher_args_t](#)
- struct [op_auth_enc_new_args_t](#)
- struct [op_cipher_args_t](#)

Macros

- #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))
- #define [HSM_AUTH_ENC_FLAGS_GENERATE_FULL_IV](#) ((hsm_op_auth_enc_flags_t)(1u << 1))
- #define [HSM_AUTH_ENC_FLAGS_GENERATE_COUNTER_IV](#) ((hsm_op_auth_enc_flags_t)(1u << 2))
- #define [HSM_AUTH_ENC_FLAGS_PLAINTEXT_KEY](#) ((hsm_op_auth_enc_new_flags_t)(1u << 3))
- #define [HSM_AUTH_ENC_FLAGS_ONE_SHOT](#) ((hsm_op_auth_enc_new_flags_t)(1u << 4))
- #define [HSM_AUTH_ENC_FLAGS_INIT](#) ((hsm_op_auth_enc_new_flags_t)(1u << 8))
- #define [HSM_AUTH_ENC_FLAGS_UPDATE_AAD](#) ((hsm_op_auth_enc_new_flags_t)(1u << 9))
- #define [HSM_AUTH_ENC_FLAGS_UPDATE_DATA](#) ((hsm_op_auth_enc_new_flags_t)(1u << 10))
- #define [HSM_AUTH_ENC_FLAGS_FINALIZE](#) ((hsm_op_auth_enc_new_flags_t)(1u << 11))
- #define [HSM_AUTH_ENC_FLAGS_FINALIZE_VERIFY](#) ((hsm_op_auth_enc_new_flags_t)(1u << 12))
- #define [HSM_AUTH_ENC_FLAGS_ABORT](#) ((hsm_op_auth_enc_new_flags_t)(1u << 13))
- #define [HSM_AUTH_ENC_FLAGS_GET_CTX_SIZE](#) ((hsm_op_auth_enc_new_flags_t)(1u << 15))
- #define [HSM_AEAD_VERIFICATION_STATUS_SUCCESS](#) ((hsm_aead_verification_status_t)(0x5A3CC3A5u))
- #define [HSM_AEAD_VERIFICATION_STATUS_FAILURE](#) ((hsm_aead_verification_status_t)(0x2B4DD4B2u))
- #define [HSM_AUTH_ENC_IV_OUT_SIZE](#) (12)
- #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_CIPHER_FLAGS_DECRYPT](#) ((hsm_op_cipher_flags_t)(0u << 0))
- #define [HSM_CIPHER_FLAGS_ENCRYPT](#) ((hsm_op_cipher_flags_t)(1u << 0))
- #define [HSM_CIPHER_FLAGS_PLAINTEXT_KEY](#) ((hsm_op_cipher_flags_t)(1u << 3))
- #define [HSM_CIPHER_FLAGS_INIT](#) ((hsm_op_cipher_flags_t)(1u << 8))
- #define [HSM_CIPHER_FLAGS_UPDATE_DATA](#) ((hsm_op_cipher_flags_t)(1u << 10))
- #define [HSM_CIPHER_FLAGS_FINALIZE](#) ((hsm_op_cipher_flags_t)(1u << 11))
- #define [HSM_CIPHER_FLAGS_ABORT](#) ((hsm_op_cipher_flags_t)(1u << 13))
- #define [HSM_CIPHER_FLAGS_GET_CTX_SIZE](#) ((hsm_op_cipher_flags_t)(1u << 15))

Typedefs

- typedef uint8_t [hsm_op_auth_enc_flags_t](#)
- typedef uint8_t [hsm_svc_cipher_flags_t](#)
- typedef uint16_t [hsm_op_auth_enc_new_flags_t](#)
- typedef uint16_t [hsm_op_cipher_one_go_flags_t](#)
- typedef uint32_t [hsm_aead_verification_status_t](#)
- typedef [hsm_op_cipher_one_go_flags_t](#) [hsm_op_cipher_flags_t](#)
- typedef [op_cipher_args_t](#) [op_cipher_one_go_args_t](#)

Enumerations

- enum [hsm_op_auth_enc_algo_t](#) {
HSM_AEAD_ALGO_CCM = ALGO_CCM,
HSM_AEAD_ALGO_GCM = ALGO_GCM,
HSM_AEAD_ALGO_CHACHA20_POLY1305 = ALGO_CHACHA20_POLY1305,
HSM_AEAD_ALGO_ALL_AEAD = ALGO_ALL_AEAD }

- enum [hsm_op_cipher_one_go_algo_t](#) {
HSM_CIPHER_ONE_GO_ALGO_CTR = ALGO_CIPHER_CTR,
HSM_CIPHER_ONE_GO_ALGO_CFB = ALGO_CIPHER_CFB,
HSM_CIPHER_ONE_GO_ALGO_OFB = ALGO_CIPHER_OFB,
HSM_CIPHER_ONE_GO_ALGO_ECB = ALGO_CIPHER_ECB_NO_PAD,
HSM_CIPHER_ONE_GO_ALGO_CBC = ALGO_CIPHER_CBC_NO_PAD }
- enum [hsm_op_cipher_algo_t](#) {
HSM_CIPHER_ALGO_CTR = ALGO_CIPHER_CTR,
HSM_CIPHER_ALGO_CFB = ALGO_CIPHER_CFB,
HSM_CIPHER_ALGO_OFB = ALGO_CIPHER_OFB,
HSM_CIPHER_ALGO_ECB = ALGO_CIPHER_ECB_NO_PAD,
HSM_CIPHER_ALGO_CBC = ALGO_CIPHER_CBC_NO_PAD,
}

Functions

- [hsm_err_t hsm_do_cipher](#) ([hsm_hdl_t](#) cipher_hdl, [op_cipher_args_t](#) *args)
- [hsm_err_t hsm_auth_enc](#) ([hsm_hdl_t](#) cipher_hdl, [op_auth_enc_args_t](#) *args)
- [hsm_err_t hsm_auth_enc_new](#) ([hsm_hdl_t](#) cipher_hdl, [op_auth_enc_new_args_t](#) *args);
- [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_cipher](#) ([hsm_hdl_t](#) cipher_hdl, [op_cipher_args_t](#) *args)
- [hsm_err_t hsm_close_cipher_service](#) ([hsm_hdl_t](#) cipher_hdl)

3.3.1 Detailed description

3.3.2 Data structure documentation

3.3.2.1 struct op_auth_enc_args_t

Structure describing the authenticated encryption operation arguments.

Data Fields

| | | |
|---|----------------|---|
| uint32_t | key_identifier | Identifier of the key to be used for the operation. |
| uint8_t * | iv | Pointer to the user supplied part of initialization vector or nonce, when applicable, otherwise 0 . |
| uint16_t | iv_size | Size of the fixed part of the initialization vector. For GCM algorithm, it can be: <ul style="list-style-type: none"> • 0 when the full IV is internally generated (RBG-based construction, Bit 1 of “Flags” field). • 4 when the counter IV generation option is used (deterministic construction, Bit 2 of “Flags” field). • 12 when the IV is entirely generated by the user (Bit 1 and Bit 2 of “Flags” field must be set to 0). For CCM algorithm IV size, 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. |

Data Fields...continued

| | | |
|-----------|-----------------|--|
| uint8_t * | input | Pointer to the input area. <ul style="list-style-type: none"> Plaintext for encryption Ciphertext + Tag (16 bytes) for decryption |
| uint8_t * | output | Pointer to the output area. Ciphertext + Tag (16 bytes) <ul style="list-style-type: none"> IV 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. |
| uint32_t | exp_output_size | Expected output buffer size in bytes, valid in case of HSM_OUT_TOO_SMALL (0x1D) error code. |

3.3.2.2 struct op_auth_enc_new_args_t

Structure describing the authenticated encryption (new) operation arguments.

Data Fields

| | | |
|-----------------------------|----------|---|
| hsm_op_auth_enc_algo_t | ae_algo | Algorithm to be used for the operation. |
| hsm_op_auth_enc_new_flags_t | flags | Bitmap specifying the operation attributes. |
| uint32_t | key_id | Identifier of the key. Only valid with INIT or ONE_SHOT_PLAINTEXT key flag must not be set. |
| uint8_t * | key | PLAINTEXT key to be used for operation. Only valid with INIT or ONE_SHOT and PLAINTEXT key flag. |
| uint16_t | key_size | Size in Bytes of Key. Only needed for ONE_SHOT or INIT with PLAINTEXT. |
| uint16_t | key_type | Key Type attribute. Only needed for ONE_SHOT or INIT with PLAINTEXT. |
| uint16_t | iv_size | IV size in bytes. CCM: Must be 12 bytes. GCM: The following are the possible input sizes: <ul style="list-style-type: none"> 0 when the Full IV is internally generated (Bit 1 flag) 4 when the Counter IV generation option is used (Bit 2 flag) 12 when the IV is entirely generated by the user |
| uint8_t * | iv_in | Pointer to the user input IV (or nonce) when applicable, otherwise 0. Only valid with INIT or ONE_SHOT. |
| uint8_t * | iv_out | IV (or nonce) output if generated randomly by ELE, depends on flags. Only valid with INIT or ONE_SHOT and IV generated by ELE (Bit 1 or Bit 2 flag). 12 bytes in size. User responsible to free the buffer after usage. |
| uint8_t * | tag | Depending on the operation, usage is different: <ul style="list-style-type: none"> Encrypt: Output buffer for the tag. Decrypt: ONE_SHOT or FINALIZE_VERIFY: Input tag buffer to be verified. |

Data Fields...continued

| | | |
|--------------------------------|-----------------|--|
| | | <ul style="list-style-type: none"> • Decrypt: FINALIZE: output buffer where the tag is outputted. Only valid with FINALIZE (output), FINALIZE VERIFY (input), or ONE SHOT. |
| uint16_t | tag_size | Size in Bytes of Tag buffer. Must be at least 16 bytes, if applicable. Only valid with FINALIZE, FINALIZE VERIFY, or ONE SHOT. |
| uint8_t * | aad | Pointer to the additional authentication data, if applicable. Only valid with ONE SHOT or UPDATE AAD. |
| uint32_t | aad_size | Size in Bytes of AAD. Must be set to valid value during init phase for AES CCM algo. Must be set to 0 for other algorithm init phase. Only valid with ONE SHOT or UPDATE AAD. |
| uint8_t * | input | Pointer to the input area. Depending on the operation, signification is different: <ul style="list-style-type: none"> • Encrypt: Plaintext to encrypt • Decrypt: Ciphertext to decrypt This field is valid with ONE SHOT or UPDATE DATA. |
| uint32_t | input_size | Size in Bytes of Input data. Must be set to a valid value during the initialization phase for AES CCM algo. Must be set to 0 for other algorithm init phase. Only valid with ONE SHOT or UPDATE DATA. |
| uint8_t * | output | Pointer to the output area. Depending on the operation, signification is different: <ul style="list-style-type: none"> • Encrypt: Ciphertext • Decrypt: Plaintext Only valid with ONE SHOT, UPDATE DATA, FINALIZE, or FINALIZE VERIFY. |
| uint32_t | output_size | Length in bytes of the output. Only valid with ONE SHOT, UPDATE DATA, FINALIZE, or FINALIZE VERIFY. |
| uint8_t * | context | Input context buffer. Only valid with PLAINTEXT key. |
| uint16_t | context_size | Size in Bytes of context buffer. Only valid with PLAINTEXT key. |
| uint32_t | exp_output_size | Expected output buffer size in bytes, valid in case of HSM_OUT_TOO_SMALL (0x1D) error code. In case of streaming operation, expected context size in bytes if operation failed with 0x1E. |
| hsm_aead_verification_status_t | verify_status | Verification status. |

3.3.2.3 struct op_cipher_args_t

Structure describing the cipher operation arguments.

Data Fields

| | | |
|-----------|----------------|---|
| uint32_t | key_identifier | Identifier of the key to be used for the operation. The flag for PLAINTEXT key must not be set. |
| uint8_t * | key | Pointer to the plaintext key buffer. |

Data Fields...continued

| | | |
|------------------------|------------------------------|---|
| | | The flag for <code>PLAINTEXT</code> key must be set. Only valid for <code>INIT</code> and <code>ONE_SHOT</code> operations. |
| <code>uint8_t *</code> | <code>iv</code> | Pointer to the initialization vector (nonce in case of AES CCM). Only valid for <code>INIT</code> and <code>ONE_SHOT</code> operations. |
| <code>uint16_t</code> | <code>iv_size</code> | Length in bytes of the initialization vector. It must be <code>0</code> for algorithms not using the initialization vector. CBC, CTR, OFB, CFB size is fixed to 16 bytes. Only valid for <code>INIT</code> and <code>ONE_SHOT</code> operations. |
| <code>uint8_t</code> | <code>svc_flags</code> | Bitmap specifying the services properties. |
| <code>uint16_t</code> | <code>flags</code> | Bitmap specifying the operation attributes. One-Shot operation configured, if none of bits 8, 10, 11, 15 are set. |
| <code>uint32_t</code> | <code>cipher_algo</code> | Algorithm to be used for the operation |
| <code>uint8_t *</code> | <code>input</code> | Pointer to the input area: <ul style="list-style-type: none"> Plaintext for encryption Ciphertext for decryption – Only valid with <code>ONE_SHOT</code> or <code>UPDATE_DATA</code> . |
| <code>uint8_t *</code> | <code>output</code> | Pointer to the output area: <ul style="list-style-type: none"> Ciphertext for encryption Plaintext for decryption – Only valid with <code>ONE_SHOT</code> , <code>UPDATE_DATA</code> , or <code>FINALIZE</code> . |
| <code>uint32_t</code> | <code>input_size</code> | 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). <ul style="list-style-type: none"> Only valid with valid with <code>ONE_SHOT</code> or <code>UPDATE_DATA</code>. |
| <code>uint32_t</code> | <code>output_size</code> | Length in bytes of the output. Only valid with <code>ONE_SHOT</code> or <code>UPDATE_DATA</code> . |
| <code>uint16_t</code> | <code>key_size</code> | Key size in bytes. Only valid with the <code>PLAINTEXT</code> flag. Only valid with <code>ONE_SHOT</code> or <code>INIT</code> operations. |
| <code>uint16_t</code> | <code>key_type</code> | Key type. Only valid with <code>PLAINTEXT</code> flag. Only valid with <code>ONE_SHOT</code> or <code>INIT</code> operations. |
| <code>uint8_t *</code> | <code>context</code> | Context buffer. Only valid with <code>PLAINTEXT</code> flag. |
| <code>uint16_t</code> | <code>context_size</code> | Size in Bytes of context buffer. Only valid with <code>PLAINTEXT</code> flag. |
| <code>uint32_t</code> | <code>exp_output_size</code> | Expected output buffer size in bytes, valid in case of (0x1D) error code. Expected context buffer size in bytes, valid in case of (0x1E) error code. |

3.3.2.4 struct open_svc_cipher_args_t

Structure describing the open cipher service members.

Data Fields

| | | |
|-----------------------|--------------------------|---|
| <code>uint32_t</code> | <code>cipher_hdl</code> | Handle identifying the cipher service flow. |
| <code>uint8_t</code> | <code>flags</code> | Bitmap specifying the services properties. |
| <code>uint8_t</code> | <code>reserved[3]</code> | Reserved bits. |

3.3.3 Macro definition documentation

3.3.3.1 HSM_AUTH_ENC_FLAGS_DECRYPT

```
#define HSM_AUTH_ENC_FLAGS_DECRYPT ((hsm_op_auth_enc_flags_t) (0u << 0))
```

Bit indicating the decryption operation.

3.3.3.2 HSM_AUTH_ENC_FLAGS_ENCRYPT

```
#define HSM_AUTH_ENC_FLAGS_ENCRYPT ((hsm_op_auth_enc_flags_t) (1u << 0))
```

Bit indicating the encryption operation.

3.3.3.3 HSM_AUTH_ENC_FLAGS_GENERATE_FULL_IV

```
#define HSM_AUTH_ENC_FLAGS_GENERATE_FULL_IV ((hsm_op_auth_enc_flags_t) (1u << 1))
```

Bit indicating the Full IV is internally generated (only relevant for encryption).

3.3.3.4 HSM_AUTH_ENC_FLAGS_GENERATE_COUNTER_IV

```
#define HSM_AUTH_ENC_FLAGS_GENERATE_COUNTER_IV ((hsm_op_auth_enc_flags_t) (1u << 2))
```

Bit indicating 4 bytes supplied other bytes internally generated (only relevant for encryption).

3.3.3.5 HSM_AUTH_ENC_FLAGS_PLAINTEXT_KEY

```
#define HSM_AUTH_ENC_FLAGS_PLAINTEXT_KEY ((hsm_op_auth_enc_new_flags_t) (1u << 3))
```

Bit indicating the plaintext key user input.

3.3.3.6 HSM_AUTH_ENC_FLAGS_ONE_SHOT

```
#define HSM_AUTH_ENC_FLAGS_ONE_SHOT ((hsm_op_auth_enc_new_flags_t) (1u << 4))
```

Bit indicating the ONE SHOT operation.

3.3.3.7 HSM_AUTH_ENC_FLAGS_INIT

```
#define HSM_AUTH_ENC_FLAGS_INIT ((hsm_op_auth_enc_new_flags_t) (1u << 8))
```

Bit indicating the initialization operation. To initialize a multipart context.

3.3.3.8 HSM_AUTH_ENC_FLAGS_UPDATE_AAD

```
#define HSM_AUTH_ENC_FLAGS_UPDATE_AAD ((hsm_op_auth_enc_new_flags_t) (1u << 9))
```

Bit indicating the UPDATE AAD operation. To update the additional authenticated data.

3.3.3.9 HSM_AUTH_ENC_FLAGS_UPDATE_DATA

```
#define HSM_AUTH_ENC_FLAGS_UPDATE_DATA ((hsm_op_auth_enc_new_flags_t) (1u << 10))
```

Bit indicating the UPDATE DATA operation. To update the data to be encrypted/decrypted.

3.3.3.10 HSM_AUTH_ENC_FLAGS_FINALIZE

```
#define HSM_AUTH_ENC_FLAGS_FINALIZE ((hsm_op_auth_enc_new_flags_t) (1u << 11))
```

Bit indicating the FINALIZE operation. The tag is output in the buffer.

3.3.3.11 HSM_AUTH_ENC_FLAGS_FINALIZE_VERIFY

```
#define HSM_AUTH_ENC_FLAGS_FINALIZE_VERIFY ((hsm_op_auth_enc_new_flags_t) (1u << 12))
```

Bit indicating the FINALIZE VERIFY operation. The tag is compared internally with the one provided in the Tag buffer. Valid only for decryption.

3.3.3.12 HSM_AUTH_ENC_FLAGS_ABORT

```
#define HSM_AUTH_ENC_FLAGS_ABORT ((hsm_op_auth_enc_new_flags_t) (1u << 13))
```

Bit indicating the ABORT operation. To abort a multipart operation and clean the context.

3.3.3.13 HSM_AUTH_ENC_FLAGS_GET_CTX_SIZE

```
#define HSM_AUTH_ENC_FLAGS_GET_CTX_SIZE ((hsm_op_auth_enc_new_flags_t) (1u << 15))
```

Bit indicating the GET CTX SIZE operation. Size of the needed allocated context, returned in the output size response field.

3.3.3.14 HSM_AEAD_VERIFICATION_STATUS_SUCCESS

```
#define HSM_AEAD_VERIFICATION_STATUS_SUCCESS ((hsm_aead_verification_status_t) (0x5A3CC3A5u))
```

Authenticated Encryption response verification status success.

3.3.3.15 HSM_AEAD_VERIFICATION_STATUS_FAILURE

```
#define HSM_AEAD_VERIFICATION_STATUS_FAILURE ((hsm_aead_verification_status_t) (0x2B4DD4B2u))
```

Authenticated Encryption response verification status failure.

3.3.3.16 HSM_AUTH_ENC_IV_OUT_SIZE

```
#define HSM_AUTH_ENC_IV_OUT_SIZE (12)
```

It indicates the size (in bytes) of the IV output buffer, received with the response of the Authenticated Encryption operation. Valid only if the flag is set for:

- HSM_AUTH_ENC_FLAGS_GENERATE_FULL_IV
- HSM_AUTH_ENC_FLAGS_GENERATE_COUNTER_IV

3.3.3.17 HSM_CIPHER_ONE_GO_FLAGS_DECRYPT

```
#define HSM_CIPHER_ONE_GO_FLAGS_DECRYPT ((hsm_op_cipher_one_go_flags_t) (0u << 0))
```

Bit indicating the decrypt operation.

3.3.3.18 HSM_CIPHER_ONE_GO_FLAGS_ENCRYPT

```
#define HSM_CIPHER_ONE_GO_FLAGS_ENCRYPT ((hsm_op_cipher_one_go_flags_t) (1u << 0))
```

Bit indicating the encrypt operation.

3.3.3.19 HSM_CIPHER_FLAGS_DECRYPT

```
#define HSM_CIPHER_FLAGS_DECRYPT ((hsm_op_cipher_flags_t) (0u << 0))
```

Bit indicating the decrypt operation.

3.3.3.20 HSM_CIPHER_FLAGS_ENCRYPT

```
#define HSM_CIPHER_FLAGS_ENCRYPT ((hsm_op_cipher_flags_t) (1u << 0))
```

Bit indicating the encrypt operation.

3.3.3.21 HSM_CIPHER_FLAGS_PLAINTEXT_KEY

```
#define HSM_CIPHER_FLAGS_PLAINTEXT_KEY ((hsm_op_cipher_flags_t) (1u << 3))
```

Bit indicating the plaintext key user input.

3.3.3.22 HSM_CIPHER_FLAGS_INIT

```
#define HSM_CIPHER_FLAGS_INIT ((hsm_op_cipher_flags_t) (1u << 8))
```

Bit indicating the initialization for a multipart context.

3.3.3.23 HSM_CIPHER_FLAGS_UPDATE_DATA

```
#define HSM_CIPHER_FLAGS_UPDATE_DATA ((hsm_op_cipher_flags_t) (1u << 10))
```

Bit indicating UPDATE DATA to be encrypted/decrypted.

3.3.3.24 HSM_CIPHER_FLAGS_FINALIZE

```
#define HSM_CIPHER_FLAGS_FINALIZE ((hsm_op_cipher_flags_t) (1u << 11))
```

Bit indicating to finalize the multipart operation.

3.3.3.25 HSM_CIPHER_FLAGS_ABORT

```
#define HSM_CIPHER_FLAGS_ABORT ((hsm_op_cipher_flags_t) (1u << 13))
```

Bit indicating to abort a multipart operation and to clean the context.

3.3.3.26 HSM_CIPHER_FLAGS_GET_CTX_SIZE

```
#define HSM_CIPHER_FLAGS_GET_CTX_SIZE ((hsm_op_cipher_flags_t) (1u << 15))
```

Bit indicating to get the context size. Size of the needed allocated context is returned in the output size response field.

3.3.4 Typedef documentation

3.3.4.1 hsm_op_auth_enc_flags_t

```
typedef uint8_t hsm\_op\_auth\_enc\_flags\_t
```

Bit field indicating the authenticated encryption operations.

3.3.4.2 hsm_op_auth_enc_new_flags_t

```
typedef uint16_t hsm_op_auth_enc_new_flags_t
```

Bit field indicating the authenticated encryption operations.

3.3.4.3 hsm_aead_verification_status_t

```
typedef uint32_t hsm_aead_verification_status_t
```

Bit indicating the Authenticated Encryption (New) response verification status.

3.3.4.4 hsm_svc_cipher_flags_t

```
typedef uint8_t hsm_svc_cipher_flags_t
```

Bit field describing the open cipher service requested operation.

3.3.4.5 hsm_op_cipher_one_go_flags_t

```
typedef uint16_t hsm_op_cipher_one_go_flags_t
```

Bit field indicating the requested operations.

3.3.4.6 hsm_op_cipher_flags_t

```
typedef hsm_op_cipher_one_go_flags_t hsm_op_cipher_flags_t
```

Bit field indicating the requested operations.

3.3.4.7 op_cipher_one_go_args_t

```
typedef op_cipher_args_t op_cipher_one_go_args_t
```

Structure describing the cipher one go operation arguments.

3.3.5 Enumeration type documentation

3.3.5.1 hsm_op_auth_enc_algo_t

```
enum hsm_op_auth_enc_algo_t
```

Bit field indicating the supported algorithm.

Enumerator

| | |
|---------------------------------|---------------------|
| HSM_AEAD_ALGO_CCM | CCM (AES CCM) |
| HSM_AEAD_ALGO_GCM | GCM (AES GCM) |
| HSM_AEAD_ALGO_CHACHA20_POLY1305 | CHACHA20_POLY1305 |
| HSM_AEAD_ALGO_ALL_AEAD | ALL AEAD (ALL AEAD) |

3.3.5.2 hsm_op_cipher_one_go_algo_t

```
enum hsm_op_cipher_one_go_algo_t
```

Enum describing the cipher one go operation algorithm.

Enumerator

| | |
|----------------------------|----------------------|
| HSM_CIPHER_ONE_GO_ALGO_CTR | CTR (AES supported). |
| HSM_CIPHER_ONE_GO_ALGO_CFB | CFB (AES supported). |
| HSM_CIPHER_ONE_GO_ALGO_OFB | OFB (AES supported). |

Enumerator...continued

| | |
|----------------------------|---------------------------------|
| HSM_CIPHER_ONE_GO_ALGO_ECB | ECB no padding (AES supported). |
| HSM_CIPHER_ONE_GO_ALGO_CBC | CBC no padding (AES supported). |

3.3.5.3 hsm_op_cipher_algo_t

```
enum hsm_op_cipher_algo_t
```

Enum describing the cipher operation algorithm.

Enumerator

| | |
|---------------------|---------------------------------|
| HSM_CIPHER_ALGO_CTR | CTR (AES supported). |
| HSM_CIPHER_ALGO_CFB | CFB (AES supported). |
| HSM_CIPHER_ALGO_OFB | OFB (AES supported). |
| HSM_CIPHER_ALGO_ECB | ECB no padding (AES supported). |
| HSM_CIPHER_ALGO_CBC | CBC no padding (AES supported). |

3.3.6 Function documentation

3.3.6.1 hsm_do_cipher()

```
hsm_err_t hsm_do_cipher (
hsm_hdl_t cipher_hdl,
op_cipher_args_t *args)
```

Secondary API to perform ciphering operation.

This API does the following operations:

- Opens an Cipher service flow.
- Performs the Ciphering operation.
- Terminates a previously opened Cipher service flow.

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

Parameters

| | |
|------------|---|
| cipher_hdl | Handle identifying the cipher service flow. |
| args | Pointer to the structure containing the function arguments. |

Returns

Error code

3.3.6.2 hsm_auth_enc()

```
hsm_err_t hsm_auth_enc (
hsm_hdl_t cipher_hdl,
op_auth_enc_args_t * args)
```

Perform authenticated encryption operation.

Users can call this function only after having opened a Cipher service flow.

For decryption operations, the full IV is supplied by the caller through the `iv` and `iv_size` parameters.

`HSM_AUTH_ENC_FLAGS_GENERATE_FULL_IV` and `HSM_AUTH_ENC_FLAGS_GENERATE_COUNTER_IV` flags are ignored.

For encryption operations, either `HSM_AUTH_ENC_FLAGS_GENERATE_FULL_IV` or `HSM_AUTH_ENC_FLAGS_GENERATE_COUNTER_IV` must be set when calling this function:

- When `HSM_AUTH_ENC_FLAGS_GENERATE_FULL_IV` is set, the full IV is internally generated, `iv` and `iv_size` must be set to 0.
- When `HSM_AUTH_ENC_FLAGS_GENERATE_COUNTER_IV` is set, the user supplies a 4 byte fixed part of the IV. The other IV bytes are internally generated.

Note: The API is deprecated now.

Parameters

| | |
|-------------------------|---|
| <code>cipher_hdl</code> | Handle identifying the cipher service flow. |
| <code>args</code> | Pointer to the structure containing the function arguments. |

Returns

Error code

3.3.6.3 hsm_auth_enc_new()

```
hsm_err_t hsm_auth_enc_new(
hsm_hdl_t cipher_hdl,
op_auth_enc_new_args_t *args)
```

Perform the authenticated encryption operation (NEW).

- One-Shot AEAD operation using an opaque or plaintext key.
- Streaming AEAD operation using an opaque or plaintext key.

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

Parameters

| | |
|-------------------------|---|
| <code>cipher_hdl</code> | Handle identifying the cipher service flow. |
| <code>args</code> | Pointer to the structure containing the function arguments. |

Returns

Error code

3.3.6.4 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.
- Users can call this function only after having opened a key-store service flow.
- Users must open this service to perform Ciphering 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>cipher_hdl</code> | Pointer to where the cipher service flow handle must be written. |

Returns

Error code

3.3.6.5 hsm_cipher_one_go()

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

Perform Cipherring operation.

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

Note: This API will be deprecated soon.

Parameters

| | |
|-------------------------|---|
| <code>cipher_hdl</code> | Handle identifying the cipher service flow. |
| <code>args</code> | Pointer to the structure containing the function arguments. |

Returns

Error code

3.3.6.6 hsm_cipher()

```
hsm_err_t hsm_cipher(
hsm_hdl_t cipher_hdl,
op_cipher_args_t *args)
```

Perform the cipherring operation:

- One-Shot cipher operation using an opaque or plaintext key.
- Streaming cipher operation using an opaque or plaintext key.

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

Parameters

| | |
|-------------------------|---|
| <code>cipher_hdl</code> | Handle identifying the cipher service flow. |
| <code>args</code> | Pointer to the structure containing the function arguments. |

Returns

Error code

3.3.6.7 hsm_close_cipher_service()

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

Terminate a previously opened Cipher service flow.

Parameters

| | |
|-------------------------|---|
| <code>cipher_hdl</code> | Pointer to handle identifying the Cipher service flow to be closed. |
|-------------------------|---|

Returns

Error code

3.4 Signature generation

Data Structures

- struct `op_pub_key_attest_args_t`
- struct `open_svc_sign_gen_args_t`
- struct [op_generate_sign_args_t](#)
- struct [op_prepare_sign_args_t](#)

Macros

- #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_PLAINTEXT_KEY (([hsm_op_generate_sign_flags_t](#))(1u << 3))
- #define HSM_OP_PREPARE_SIGN_INPUT_DIGEST (([hsm_op_prepare_signature_flags_t](#))(0u << 0))
Bit indicating input digest.
- #define HSM_OP_PREPARE_SIGN_INPUT_MESSAGE (([hsm_op_prepare_signature_flags_t](#)) (1u << 0))
Bit indicating input message.
- #define HSM_OP_PREPARE_SIGN_COMPRESSED_POINT (([hsm_op_prepare_signature_flags_t](#)) (1u << 1))
Bit indicating compressed point.

Typedefs

- typedef uint8_t [hsm_op_generate_sign_flags_t](#)
- typedef uint8_t [hsm_op_prepare_signature_flags_t](#)

Enumerations

- enum `hsm_op_pub_key_attest_algo_t` {
HSM_PKEY_ATTEST_ALGO_CMAC = ALGO_ATTEST_CMAC,
HSM_PKEY_ATTEST_ALGO_ECDSA_SHA224 = ALGO_ATTEST_ECDSA_SHA224,
HSM_PKEY_ATTEST_ALGO_ECDSA_SHA256 = ALGO_ATTEST_ECDSA_SHA256,
HSM_PKEY_ATTEST_ALGO_ECDSA_SHA384 = ALGO_ATTEST_ECDSA_SHA384,
HSM_PKEY_ATTEST_ALGO_ECDSA_SHA512 = ALGO_ATTEST_ECDSA_SHA512 }
- enum [hsm_signature_scheme_id_t](#) {
HSM_SIGNATURE_SCHEME_RSA_PKCS1_V15_SHA224 = 0x06000208,
HSM_SIGNATURE_SCHEME_RSA_PKCS1_V15_SHA256 = 0x06000209,
HSM_SIGNATURE_SCHEME_RSA_PKCS1_V15_SHA384 = 0x0600020A,
HSM_SIGNATURE_SCHEME_RSA_PKCS1_V15_SHA512 = 0x0600020B,
HSM_SIGNATURE_SCHEME_RSA_PKCS1_V15_ANY_HASH = 0x060002FF,
HSM_SIGNATURE_SCHEME_RSA_PKCS1_PSS_MGF1_SHA224 = 0x06000308,
HSM_SIGNATURE_SCHEME_RSA_PKCS1_PSS_MGF1_SHA256 = 0x06000309,
HSM_SIGNATURE_SCHEME_RSA_PKCS1_PSS_MGF1_SHA384 = 0x0600030A,
HSM_SIGNATURE_SCHEME_RSA_PKCS1_PSS_MGF1_SHA512 = 0x0600030B,

```
HSM_SIGNATURE_SCHEME_RSA_PKCS1_PSS_MGF1_ANY_HASH = 0x060003FF,
HSM_SIGNATURE_SCHEME_ECDSA_ANY = 0x06000600,
HSM_SIGNATURE_SCHEME_ECDSA_SHA224 = 0x06000608,
HSM_SIGNATURE_SCHEME_ECDSA_SHA256 = 0x06000609,
HSM_SIGNATURE_SCHEME_ECDSA_SHA384 = 0x0600060A,
HSM_SIGNATURE_SCHEME_ED25519PH = 0x0600090B,
HSM_SIGNATURE_SCHEME_ED448PH = 0x06000915,
HSM_SIGNATURE_SCHEME_PURE_EDDSA = 0x06000800,
HSM_SIGNATURE_SCHEME_EDDSA_ALL = 0x86000800,
HSM_SIGNATURE_SCHEME_ECDSA_SHA512 = 0x0600060B }
```

Functions

- [hsm_err_t hsm_do_sign](#) ([hsm_hdl_t](#) key_store_hdl, [op_generate_sign_args_t](#) *args)
- [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_pub_key_attest](#) ([hsm_hdl_t](#) signature_gen_hdl, [op_pub_key_attest_args_t](#) args)
- [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)

3.4.1 Detailed description

3.4.2 Data structure documentation

3.4.2.1 struct op_pub_key_attest_args_t

Structure to represent the generate sign operation arguments.

Table 1. Data Fields

| | | |
|------------------------------|----------------------|---|
| uint32_t | key_identifier | Identifier of the key to be attested. |
| uint32_t | key_attestation_id | Identifier of the key to be used for the attestation. |
| hsm_op_pub_key_attest_algo_t | attest_algo | Algorithm to be used for the attestation. |
| uint8_t * | auth_challenge | Pointer to the authentication challenge. |
| uint32_t | auth_challenge_size | Authentication challenge size in bytes. |
| uint8_t * | certificate | Pointer to the output certificate encoded as signed TLV buffer. |
| uint32_t | certificate_size | Certificate size in bytes. |
| uint32_t | exp_certificate_size | Expected certificate size for output, returned by FW in case of HSM_OUT_TOO_SMALL (0x1D) error. |

3.4.2.2 struct open_svc_sign_gen_args_t

Structure to represent the generate sign open service arguments.

Data Fields

| | | |
|---------------------------|-------------------|---|
| hsm_hdl_t | signature_gen_hdl | - |
|---------------------------|-------------------|---|

3.4.2.3 struct op_generate_sign_args_t

Structure to represent the generate sign operation arguments.

Data Fields

| | | |
|------------------------------|--------------------|--|
| uint32_t | key_identifier | Identifier of the key to be used for the operation. To the key identifier (for example, opaque key), ensure that the flag for the PLAINTEXT key is not set. |
| uint8_t * | Message | Pointer to the input (message or message digest) to be signed. |
| uint8_t * | Signature | Pointer to the output area where the signature must be stored. The signature $S = (r, s)$ is stored in the format $r s R_y$, where: R_y is an additional byte containing the lsb of y . R_y should be considered valid only if <code>HSM_OP_GENERATE_SIGN_FLAGS_COMPRESSED_POINT</code> is set. |
| uint16_t | signature_size | Length in bytes of the output. After the signature generation operation, this field contains the expected signature buffer size, if the operation fails due to the provided output buffer size being too short. |
| uint32_t | message_size | Length of the input in bytes. |
| hsm_signature_scheme_id_t | scheme_id | Identifier of the digital signature scheme to be used for the operation. |
| uint16_t | salt_len | Salt length in bytes. |
| uint16_t | key_type | Key Type. Only valid with the PLAINTEXT flag. |
| uint16_t | priv_key_size | Size of the private key input buffer in bytes. Only valid with the PLAINTEXT flag. |
| uint8_t * | priv_key | Pointer to the private key input buffer, to be used to generate the signature. Only valid with the PLAINTEXT flag. |
| uint16_t | key_security_size | Keypair security size in bits. Only valid with the PLAINTEXT flag. |
| uint16_t | exp_signature_size | Expected signature buffer size for output, returned by FW if the input signature size provided is less than the required size. |
| hsm_op_generate_sign_flags_t | flags | Bitmap specifying the operation attributes. |

3.4.2.4 struct op_prepare_sign_args_t

Structure detailing the prepare signature operation member arguments.

Data Fields

| | | |
|--|-----------|--|
| hsm_signature_scheme_id_t | scheme_id | Identifier of the digital signature scheme to be used for the operation. |
| hsm_op_prepare_signature_flags_t | flags | Bitmap specifying the operation attributes. |

3.4.3 Macro definition documentation

3.4.3.1 HSM_OP_GENERATE_SIGN_FLAGS_INPUT_DIGEST

```
#define HSM_OP_GENERATE_SIGN_FLAGS_INPUT_DIGEST ((hsm_op_generate_sign_flags_t)
(0u << 0))
```

Bit field indicating the input is the message digest.

3.4.3.2 HSM_OP_GENERATE_SIGN_FLAGS_INPUT_MESSAGE

```
#define HSM_OP_GENERATE_SIGN_FLAGS_INPUT_MESSAGE ((hsm_op_generate_sign_flags_t)
(1u << 0))
```

Bit field indicating the input is the actual message.

3.4.3.3 HSM_OP_GENERATE_SIGN_FLAGS_PLAINTEXT_KEY

```
#define HSM_OP_GENERATE_SIGN_FLAGS_PLAINTEXT_KEY ((hsm_op_generate_sign_flags_t)
(1u << 3))
```

Bit field indicating the private key input from the user.

3.4.4 Typedef documentation

3.4.4.1 hsm_op_generate_sign_flags_t

```
typedef uint8_t hsm_op_generate_sign_flags_t
```

Bit field indicating the requested operation.

3.4.4.2 hsm_op_prepare_signature_flags_t

```
typedef uint8_t hsm_op_prepare_signature_flags_t
```

Bitmap specifying the prepare signature operation supported attributes.

3.4.5 Enumeration type documentation

3.4.5.1 hsm_op_pub_key_attest_algo_t

Enum containing the Signature Algorithms for Public Key Attestation.

Table 2. Enumerator

| | |
|-----------------------------------|---------------------------|
| HSM_PKEY_ATTEST_ALGO_CMAC | CMAC Attestation |
| HSM_PKEY_ATTEST_ALGO_ECDSA_SHA224 | ECDSA SHA224 Attestation. |
| HSM_PKEY_ATTEST_ALGO_ECDSA_SHA256 | ECDSA SHA256 Attestation. |
| HSM_PKEY_ATTEST_ALGO_ECDSA_SHA384 | ECDSA SHA384 Attestation. |
| HSM_PKEY_ATTEST_ALGO_ECDSA_SHA512 | ECDSA SHA512 Attestation. |

3.4.5.2 hsm_signature_scheme_id_t

```
enum hsm_signature_scheme_id_t
```

Bit field indicating the PSA compliant requested operations:

Bits 2-7: Reserved.

3.4.6 Function documentation

3.4.6.1 hsm_do_sign()

```
hsm_err_t hsm_do_sign (
hsm_hdl_t key_store_hdl,
op_generate_sign_args_t * args)
```

Secondary API to generate the signature on the given message.

This API does the following:

- Open a service flow for signature generation.
- Based on the flag to identify the type of message: Digest or actual message, generate the signature using the key corresponding to the key ID.
- Post performing the operation, terminate the previously opened signature-generation service flow.

Users 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

3.4.6.2 hsm_do_pub_key_attest()

```
hsm_err_t hsm_do_pub_key_attest (
hsm_hdl_t key_store_hdl,
op_pub_key_attest_args_t * args)
```

Secondary API to attest the public key of an asymmetric key present in the ELE FW key storage (generated or imported).

This API does the following:

- Open a service flow for signature generation.
- Perform the operation to attest the public key of required asymmetric key present in the ELE FW key storage.
- Post performing the operation, and terminate the previously opened signature-generation service flow.

Users can call this function only after opening a key store service flow.

Parameters

| | |
|----------------------|---|
| <i>key_store_hdl</i> | Handle identifying the key store service flow. |
| <i>args</i> | Pointer to the structure containing the function arguments. |

Returns

Error code

3.4.6.3 hsm_pub_key_attest()

```
hsm_err_t hsm_pub_key_attest (
```

```
hsm_hdl_t signature_gen_hdl,
op_pub_key_attest_args_t * args)
```

Attest the public key of an asymmetric key present in the ELE FW key storage. Users can call this function only after having opened a signature generation service flow.

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

3.4.6.4 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.
- Users can call this function only after having opened a key store service flow.
- Users must open this service 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

3.4.6.5 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

3.4.6.6 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.

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

The signature $S=(r,s)$ is stored in the format $r || s || R_y$ where:

- R_y is an additional byte containing the lsb of y . R_y has to be considered valid only if the `HSM_OP_GENERATE_SIGN_FLAGS_COMPRESSED_POINT` is set.

Parameters

| | |
|--------------------------------|---|
| <code>signature_gen_hdl</code> | Handle identifying the signature generation service flow. |
| <code>args</code> | Pointer to the structure containing the function arguments. |

Returns

Error code

3.4.6.7 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 no dependencies on the input message.

The pre-calculated value is stored internally and used once `hsm_generate_signature` is called. Up to 20 pre-calculated values can be stored, and additional preparation operations have no effects.

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

The signature $S=(r,s)$ is stored in the format $r || s || R_y$ where:

- R_y is an additional byte containing the lsb of y . R_y has to be considered valid only if the `HSM_OP_PREPARE_SIGN_COMPRESSED_POINT` is set.

Parameters

| | |
|--------------------------------|---|
| <code>signature_gen_hdl</code> | Handle identifying the signature generation service flow. |
| <code>args</code> | Pointer to the structure containing the function arguments. |

Returns

Error code

3.5 Signature verification

Data Structures

- struct [open_svc_sign_ver_args_t](#)
- struct [op_verify_sign_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))

- #define HSM_OP_VERIFY_SIGN_FLAGS_PLAINTEXT_KEY ((hsm_op_verify_sign_flags_t)(0u << 3))
- #define HSM_OP_VERIFY_SIGN_FLAGS_OPAQUE_KEY ((hsm_op_verify_sign_flags_t)(1u << 3))
- #define HSM_VERIFICATION_STATUS_SUCCESS ((hsm_verification_status_t) (0x5A3CC3A5u))
- #define HSM_VERIFICATION_STATUS_FAILURE ((hsm_verification_status_t) (0x2B4DD4B2u))

Typedefs

- typedef uint32_t [hsm_verification_status_t](#)
- typedef uint8_t [hsm_op_verify_sign_flags_t](#)

Functions

- [hsm_err_t hsm_verify_sign](#) ([hsm_hdl_t](#) session_hdl, [op_verify_sign_args_t](#) *args, [hsm_verification_status_t](#) *verification_status)
- [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_close_signature_verification_service](#) ([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)

3.5.1 Detailed description

3.5.2 Data structure documentation

3.5.2.1 struct open_svc_sign_ver_args_t

Structure to represent verify sign open service arguments.

Data Fields

| | | |
|---------------------------|-------------|---|
| hsm_hdl_t | sig_ver_hdl | - |
|---------------------------|-------------|---|

3.5.2.2 struct op_verify_sign_args_t

Structure to represent verify signature operation arguments.

Data Fields

| | | |
|-----------|----------------|---|
| uint8_t * | key | Pointer to the public key to be used for the verification. If HSM_OP_VERIFY_SIGN_FLAGS_KEY_INTERNAL is set, it must point to the key reference returned by the hsm_import_public_key API. Only valid with the PLAINTEXT flag. |
| uint32_t | key_identifier | Identifier of the key to be used for the operation. In case of using key identifier (for example, opaque key), it must be ensured that the flag for the PLAINTEXT key is not set. Supported platforms: i.MX 95 and i.MX 943. |
| uint8_t * | message | Pointer to the input (message or message digest). |
| uint8_t * | signature | Pointer to the input signature. The signature S=(r,s) is expected to be in the format r s Ry, where Ry is an additional byte containing the lsb of y. Ry is 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. Only valid with the PLAINTEXT flag. |

Data Fields...continued

| | | |
|--|---------------------|---|
| uint16_t | signature_size | Length in bytes of the output. It must contain one additional byte where to store the Ry. |
| uint32_t | message_size | Length in bytes of the input message. |
| hsm_verification_status_t | verification_status | Verification status. |
| hsm_signature_scheme_id_t | scheme_id | Identifier of the digital signature scheme to be used for the operation. |
| uint16_t | salt_len | Salt length in bytes. |
| hsm_bit_key_sz_t | key_sz | Indicates key security size in bits. Only valid with the PLAINTEXT flag. |
| hsm_pubkey_type_t | pkey_type | Indicates the public key type. Only valid with the PLAINTEXT flag. |
| hsm_op_verify_sign_flags_t | flags | Bitmap specifying the operation attributes. |

3.5.3 Macro Definition Documentation

3.5.3.1 HSM_OP_VERIFY_SIGN_FLAGS_INPUT_DIGEST

```
#define HSM_OP_VERIFY_SIGN_FLAGS_INPUT_DIGEST ((hsm\_op\_verify\_sign\_flags\_t) (0u << 0))
```

Verify signature bit indicating input is message digest.

3.5.3.2 HSM_OP_VERIFY_SIGN_FLAGS_INPUT_MESSAGE

```
#define HSM_OP_VERIFY_SIGN_FLAGS_INPUT_MESSAGE ((hsm\_op\_verify\_sign\_flags\_t) (1u << 0))
```

Verify signature bit indicating input is actual message.

3.5.3.3 HSM_OP_VERIFY_SIGN_FLAGS_COMPRESSED_POINT

```
#define HSM_OP_VERIFY_SIGN_FLAGS_COMPRESSED_POINT ((hsm\_op\_verify\_sign\_flags\_t) (1u << 1))
```

Verify signature bit indicating input based on signature format.

3.5.3.4 HSM_OP_VERIFY_SIGN_FLAGS_KEY_INTERNAL

```
#define HSM_OP_VERIFY_SIGN_FLAGS_KEY_INTERNAL ((hsm\_op\_verify\_sign\_flags\_t) (1u << 2))
```

Verify signature bit indicating input is key argument.

3.5.3.5 HSM_OP_VERIFY_SIGN_FLAGS_PLAINTEXT_KEY

```
#define HSM_OP_VERIFY_SIGN_FLAGS_PLAINTEXT_KEY ((hsm\_op\_verify\_sign\_flags\_t) (0u << 3))
```

Verify signature bit indicating input public key is in plaintext. For example, the public key comes from the user memory.

3.5.3.6 HSM_OP_VERIFY_SIGN_FLAGS_OPAQUE_KEY

```
#define HSM_OP_VERIFY_SIGN_FLAGS_OPAQUE_KEY ((hsm_op_verify_sign_flags_t) (1u << 3))
```

Verify signature bit indicating the input public key is opaque. For example, the public key comes from the Secure Enclave keystore.

3.5.3.7 HSM_VERIFICATION_STATUS_SUCCESS

```
#define HSM_VERIFICATION_STATUS_SUCCESS ((hsm_verification_status_t) (0x5A3CC3A5u))
```

Verify signature response success status.

3.5.3.8 HSM_VERIFICATION_STATUS_FAILURE

```
#define HSM_VERIFICATION_STATUS_FAILURE ((hsm_verification_status_t) (0x2B4DD4B2u))
```

Verify signature response failure status.

3.5.4 Typedef documentation

3.5.4.1 hsm_verification_status_t

```
typedef uint32_t hsm_verification_status_t
```

Bit indicating the response verification status.

3.5.4.2 hsm_op_verify_sign_flags_t

```
typedef uint8_t hsm_op_verify_sign_flags_t
```

Bit indicating the requested operations.

3.5.5 Function documentation

3.5.5.1 hsm_verify_sign()

```
hsm_err_t hsm_verify_sign (
    hsm_hdl_t op_hdl,
    op_verify_sign_args_t * args,
    hsm_verification_status_t * verification_status)
```

Secondary API to verify a message signature.

This API does the following:

- Open a flow for verification of the signature.
- Based on the flag to identify the type of message: Digest or actual message, verification of the signature is done using the public key.
- Post performing the operation, terminate the previously opened signature-verification service flow.

Users can call this function only after having opened a session or keystore, depending on the type of the operation (plaintext or opaque key) and platform.

Parameters

| | |
|----------------------------|--|
| <i>op_hdl</i> | Handle identifying the current session or keystore, depending on the type of the operation (plaintext or opaque key) and platform. |
| <i>args</i> | Pointer to the structure containing the function arguments. |
| <i>verification_status</i> | Pointer for storing the verification status. |

Returns

Error code

3.5.5.2 hsm_open_signature_verification_service()

```
hsm_err_t hsm_open_signature_verification_service(
hsm_hdl_t op_hdl,
open_svc_sign_ver_args_t * args,
hsm_hdl_t * signature_ver_hdl)
```

Users must open this service to perform signature verification operations. Users can call this function only after having opened a session or keystore, depending on the type of the operation (plaintext or opaque key) and platform.

Parameters

| | |
|--------------------------|---|
| <i>op_hdl</i> | Handle identifying the current session or the key store, depending on the type of the operation (plaintext or opaque key) and platform. |
| <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

3.5.5.3 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

3.5.5.4 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. Users can call this function only after opening 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 is considered as valid only if the `HSM_OP_VERIFY_SIGN_FLAGS_COMPRESSED_POINT` is set.

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

Parameters

| | |
|--------------------------|---|
| <i>signature_ver_hdl</i> | Handle identifying the signature verification service flow. |
| <i>args</i> | Pointer to the structure containing the function arguments. |
| <i>status</i> | Pointer to where the verification status must be stored. If the verification succeed, the value <code>HSM_VERIFICATION_STATUS_SUCCESS</code> is returned. |

Returns

Error code

3.6 Random number generation

Data Structures

- struct [op_get_random_args_t](#)

Functions

- [hsm_err_t hsm_do_rng](#) ([hsm_hdl_t](#) session_hdl, [op_get_random_args_t](#) *args)
- [hsm_err_t hsm_get_random](#) ([hsm_hdl_t](#) rng_hdl, [op_get_random_args_t](#) *args)

3.6.1 Detailed description

3.6.2 Data structure documentation

3.6.2.1 struct op_get_random_args_t

Structure detailing the get random number operation member arguments.

Data Fields

| | | |
|------------------------|--------------------------|---|
| <code>uint8_t *</code> | <code>output</code> | Pointer to the output area where the random number must be written. |
| <code>uint32_t</code> | <code>random_size</code> | Length in bytes of the random number to be provided. |

3.6.3 Function documentation

3.6.3.1 hsm_do_rng()

```
hsm_err_t hsm_do_rng (
hsm_hdl_t session_hdl,
op_get_random_args_t * args)
```

Secondary API to fetch the Random Number.

This API does the following:

Get a freshly generated random number.

Parameters

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

Returns

Error code

3.6.3.2 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.

Users can call this function only after opening an 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

3.7 Hashing

Data Structures

- struct [op_hash_one_go_args_t](#)

Macros

- #define HSM_HASH_FLAG_ALLOWED

Enumerations

- enum [hsm_hash_algo_t](#) {
 - HSM_HASH_ALGO_MD5 = 0x02000003,
 - HSM_HASH_ALGO_SHA_1 = 0x02000005,
 - HSM_HASH_ALGO_SHA_224 = 0x02000008,
 - HSM_HASH_ALGO_SHA_256 = 0x02000009,

```
HSM_HASH_ALGO_SHA_384 = 0x0200000A,
HSM_HASH_ALGO_SHA_512 = 0x0200000B,
HSM_HASH_ALGO_SHA3_224 = 0x02000010,
HSM_HASH_ALGO_SHA3_256 = 0x02000011,
HSM_HASH_ALGO_SHA3_384 = 0x02000012,
HSM_HASH_ALGO_SHA3_512 = 0x02000013,
HSM_HASH_ALGO_SM3 = 0x02000014,
HSM_HASH_ALGO_SHAKE_256 = 0x02000015 }
```

- enum [hsm_hash_svc_flags_t](#) {
 - HSM_HASH_FLAG_ONE_SHOT = 0x1,
 - HSM_HASH_FLAG_INIT = 0x2,
 - HSM_HASH_FLAG_UPDATE = 0x4,
 - HSM_HASH_FLAG_FINAL = 0x8,
 - HSM_HASH_FLAG_DESTROY = 0x10,
 - HSM_HASH_FLAG_GET_CONTEXT = 0x80 }

Functions

- [hsm_err_t hsm_do_hash \(hsm_hdl_t session_hdl, op_hash_one_go_args_t *args\)](#)
- [hsm_err_t hsm_hash_one_go \(hsm_hdl_t hash_hdl, op_hash_one_go_args_t *args\)](#)

3.7.1 Detailed description

3.7.2 Data structure documentation

3.7.2.1 struct op_hash_one_go_args_t

Structure describing the hash one go operation arguments.

Data Fields

| | | |
|--------------------------------------|-----------------|---|
| uint8_t * | msb | Pointer to the MSB of address in the requester space where buffers can be found, must be 0 until supported. |
| uint8_t * | ctx | Pointer to the context. |
| 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_hash_svc_flags_t | svc_flags | Flags identifying the operation <code>init()</code> , <code>update()</code> , <code>final()</code> , or one shot operation. |
| uint16_t | ctx_size | Size of context buffer in bytes, ignored in case of one shot operation. |
| uint32_t | exp_output_size | Expected output digest buffer size, returned by FW in case the provided output size is incorrect. |
| uint16_t | context_size | Expected context size to allocate in bytes, if flag Get context size is set or provided context size is incorrect. |

3.7.3 Macro definition documentation

3.7.3.1 HSM_HASH_FLAG_ALLOWED

```
#define HSM_HASH_FLAG_ALLOWED

Value:          (HSM_HASH_FLAG_ONE_SHOT | HSM_HASH_FLAG_INIT \
                | HSM_HASH_FLAG_UPDATE | HSM_HASH_FLAG_FINAL \
                | HSM_HASH_FLAG_DESTROY | HSM_HASH_FLAG_GET_CONTEXT)
```

Bitmap indicating the allowed hash service operations.

3.7.4 Enumeration type documentation

3.7.4.1 hsm_hash_algo_t

```
enum hsm_hash_algo_t
```

Bitmap indicating the supported hash algorithm.

3.7.4.2 hsm_hash_svc_flags_t

```
enum hsm_hash_svc_flags_t
```

Bit field indicating the hash service operations (only one can be set):

Table 3. Data Fields

| | |
|---------------------------|---|
| HSM_HASH_FLAG_ONE_SHOT | One-Shot operation (Context is not used). |
| HSM_HASH_FLAG_INIT | Initialization. |
| HSM_HASH_FLAG_UPDATE | Update. For SHAKE algo, all data needs to be processed by update. |
| HSM_HASH_FLAG_FINAL | Final (Final can be used multiple times if algo is SHAKE). |
| HSM_HASH_FLAG_DESTROY | Destroy (valid for SHAKE algo only). |
| HSM_HASH_FLAG_GET_CONTEXT | Get context size. Return context size to allocate (depends on Algorithm). |

3.7.5 Function documentation

3.7.5.1 hsm_do_hash()

```
hsm_err_t hsm_do_hash (
    hsm_hdl_t session_hdl,
    op_hash_one_go_args_t * args)
```

Secondary API to digest a message.

This API performs hash.

Parameters

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

Returns

Error code

3.7.5.2 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.

Users can call this function only after opening 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

3.8 Data storage**Data Structures**

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

Macros

- #define HSM_OP_DATA_STORAGE_FLAGS_EL2GO (([hsm_op_data_storage_flags_t](#)) (1u << 0))
- #define HSM_OP_DATA_STORAGE_FLAGS_DEFAULT (([hsm_op_data_storage_flags_t](#)) (0u << 0))

Store data.

- #define HSM_OP_DATA_STORAGE_FLAGS_STORE (([hsm_op_data_storage_flags_t](#)) (1u << 1))

Retrieve data.

- #define HSM_OP_DATA_STORAGE_FLAGS_RETRIEVE (([hsm_op_data_storage_flags_t](#)) (0u << 1))
- #define [ENC_DATA_TLV_DEV_UUID_TAG](#) 0x41u
- #define [ENC_DATA_TLV_IV_TAG](#) 0x45u
- #define [ENC_DATA_TLV_ENC_DATA_TAG](#) 0x46u
- #define [ENC_DATA_TLV_SIGN_TAG](#) 0x5Eu
- #define [ENC_DATA_TLV_DEV_UUID_TAG_LEN](#) 0x01u
- #define [ENC_DATA_TLV_IV_TAG_LEN](#) 0x01u
- #define [ENC_DATA_TLV_ENC_DATA_TAG_LEN](#) 0x01u
- #define [ENC_DATA_TLV_SIGN_TAG_LEN](#) 0x01u
- #define HSM_OP_ENC_DATA_STORAGE_FLAGS_RANDOM_IV (([hsm_op_enc_data_storage_flags_t](#)) (1u << 0))

Internally generate random IV, if needed for operation.

- `#define HSM_OP_ENC_DATA_STORAGE_FLAGS_READ_ONCE ((hsm_op_enc_data_storage_flags_t) (1u << 1))`
Read once, and delete data from NVM after retrieve.

Typedefs

- typedef uint8_t [hsm_svc_data_storage_flags_t](#)
- typedef uint8_t [hsm_op_data_storage_flags_t](#)
- typedef uint16_t [hsm_op_enc_data_storage_flags_t](#)

Functions

- [hsm_err_t hsm_data_ops](#) ([hsm_hdl_t](#) key_store_hdl, [op_data_storage_args_t](#) *args)
- [hsm_err_t hsm_data_delete_ops](#) ([hsm_hdl_t](#) key_store_hdl, [op_data_storage_delete_args_t](#) *args)
- [hsm_err_t hsm_enc_data_ops](#) ([hsm_hdl_t](#) key_store_hdl, [op_enc_data_storage_args_t](#) *args)
- [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_data_storage_delete](#) ([hsm_hdl_t](#) data_storage_hdl, [op_data_storage_delete_args_t](#) *args)
- [hsm_err_t hsm_enc_data_storage](#) ([hsm_hdl_t](#) data_storage_hdl, [op_enc_data_storage_args_t](#) *args)
- uint8_t [decode_enc_data_tlv](#) ([op_data_storage_args_t](#) *args)
- [hsm_err_t hsm_close_data_storage_service](#) ([hsm_hdl_t](#) data_storage_hdl)

3.8.1 Detailed description

3.8.2 Data structure documentation

3.8.2.1 struct open_svc_data_storage_args_t

Structure specifying the data storage open service member arguments.

Data Fields

| | | |
|--|---------------------|--|
| hsm_hdl_t | data_storage_handle | Data storage handle. |
| hsm_svc_data_storage_flags_t | flags | Bitmap specifying the services properties. |
| uint8_t | reserved[3] | - |

3.8.2.2 struct op_data_storage_args_t

Structure detailing the data storage operation member arguments.

Data Fields

| | | |
|--|-----------|--|
| uint8_t * | data | Pointer to the data. In case of store request, it is the input data to store. In case of retrieve, it is the pointer where to load data. |
| uint32_t | data_size | Length in bytes of the data. |
| uint32_t | data_id | ID of the data. |
| hsm_op_data_storage_flags_t | flags | Flags bitmap specifying the operation attributes. |
| hsm_svc_data_storage_flags_t | svc_flags | Bitmap specifying the services properties. |
| uint16_t | uuid_len | Device UUID length in bytes. In case of retrieve, if the data retrieved is in the TLV format, which was stored by Encrypted Data Storage API, the TLV format data is decoded to fill the following fields. |

Data Fields...continued

| | | |
|-----------|-----------------|--|
| | | Memory for storing uuid/iv/ciphertext/payload/signature is allocated by the HSM library. Caller of the function decode_enc_data_tlv() , needs to ensure freeing up memory. |
| uint8_t * | uuid | Device UUID. |
| uint16_t | iv_len | IV length in bytes if needed, otherwise 0. |
| uint8_t * | iv | IV buffer, if needed. |
| uint32_t | ciphertext_len | Encrypted text length in bytes. |
| uint8_t * | ciphertext | Encrypted text buffer. |
| uint32_t | payload_len | Payload length in bytes. |
| uint8_t * | payload | Payload data buffer to verify signature. |
| uint16_t | signature_len | Signature length in bytes. |
| uint8_t * | signature | Signature buffer. |
| uint32_t | exp_output_size | Expected output buffer size in bytes, valid in case of HSM_OUT_TOO_SMALL (0x1D) error code. |

3.8.2.3 struct op_enc_data_storage_args_t

Data Fields

| | | |
|---|---------------|--|
| uint32_t | data_id | ID of the data. |
| uint8_t * | data | Pointer to the data, to be encrypted and signed. |
| uint32_t | data_size | Length in bytes of the data. |
| uint32_t | enc_algo | Cipher algorithm to be used for encryption of data. |
| uint32_t | enc_key_id | Identifier of the key to be used for encryption. |
| uint32_t | sign_algo | Signature algorithm to be used for signing the data. |
| uint32_t | sign_key_id | Identifier of the key to be used for signing. |
| uint8_t * | iv | Pointer to the IV buffer. |
| uint16_t | iv_size | IV size in bytes. |
| hsm_op_enc_data_storage_flags_t | flags | Bitmap specifying the operation attributes. |
| hsm_svc_data_storage_flags_t | svc_flags | Bitmap specifying the service attributes. |
| uint16_t | lifecycle | Bitmask of device lifecycle, in which the data can be retrieved. |
| uint32_t | out_data_size | Size (bytes) of the signed TLV stored, received with API resp. |

3.8.2.4 struct op_data_storage_delete_args_t

Structure detailing the data storage delete operation member arguments.

Data Fields

| | | |
|--|-----------|--|
| hsm_svc_data_storage_flags_t | svc_flags | bitmap specifying the services properties. |
| uint32_t | data_id | ID of the data. |

3.8.3 Macro definition documentation

3.8.3.1 ENC_DATA_TLV_DEV_UUID_TAG

```
#define ENC_DATA_TLV_DEV_UUID_TAG 0x41u
```

Encrypted Data TLV Tags.

3.8.3.2 ENC_DATA_TLV_DEV_UUID_TAG_LEN

```
#define ENC_DATA_TLV_DEV_UUID_TAG_LEN 0x01u
```

Encrypted Data TLV Tags lengths.

3.8.4 Typedef documentation

3.8.4.1 hsm_svc_data_storage_flags_t

```
typedef uint8_t hsm\_svc\_data\_storage\_flags\_t
```

Bitmap specifying the data storage open service supported properties.

3.8.4.2 hsm_op_data_storage_flags_t

```
typedef uint8_t hsm\_op\_data\_storage\_flags\_t
```

Bitmap specifying the data storage operation supported attributes.

3.8.4.3 hsm_op_enc_data_storage_flags_t

```
typedef uint16_t hsm\_op\_enc\_data\_storage\_flags\_t
```

Bitmap specifying the encrypted data storage operation supported attributes.

3.8.5 Function Documentation

3.8.5.1 hsm_data_ops()

```
hsm\_err\_t hsm_data_ops (  
hsm\_hdl\_t key_store_hdl,  
op\_data\_storage\_args\_t * args)
```

Secondary API to store and retrieve data from the Linux filesystem managed by EdgeLock Enclave Firmware.

This API does the following:

- Open a data storage service flow.
- Based on the flag for operation attribute: Store or Retrieve
 - Store the data.
 - Retrieve the data, from the non-volatile storage.
- Post performing the operation, and terminate the previously opened data-storage service flow.

Users can call this function only after opening 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

3.8.5.2 hsm_data_delete_ops()

```
hsm_err_t hsm_data_delete_ops (
hsm_hdl_t key_store_hdl,
op_data_storage_delete_args_t *args)
```

Secondary API to delete data from the Linux filesystem managed by EdgeLock Enclave Firmware.

This API does the following:

- Open a data storage service flow.
- Delete the data identified by input Data ID, and from the non-volatile storage.
- Post performing the operation, and terminate the previously opened data-storage service flow.

Users can call this function only after opening 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

3.8.5.3 hsm_enc_data_ops()

```
hsm_err_t hsm_enc_data_ops (
hsm_hdl_t key_store_hdl,
op_enc_data_storage_args_t * args)
```

Secondary API to store the encrypted and signed data in NVM.

This API does the following:

- Open a data storage service Flow.
- Store the encrypted and signed data in NVM. The stored data can be retrieved through Data Storage API.
- Post performing the operation, terminate the previously opened data-storage service flow.

Users 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

3.8.5.4 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.

Users must open this service flow 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.

3.8.5.5 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

3.8.5.6 hsm_data_storage_delete()

```
hsm_err_t hsm_data_storage_delete(
hsm_hdl_t data_storage_hdl,
op_data_storage_delete_args_t *args);
```

Delete the data object identified by *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

3.8.5.7 hsm_enc_data_storage()

```
hsm_err_t hsm_enc_data_storage (
hsm_hdl_t data_storage_hdl,
op_enc_data_storage_args_t * args)
```

Store encrypted and signed data in the NVM.

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

3.8.5.8 decode_enc_data_tlv()

```
uint8_t decode_enc_data_tlv (op_data_storage_args_t * args)
```

Decode and populate the data storage op args for Encrypted Data TLV fields.

Parameters

| | |
|-------------|---|
| <i>args</i> | Pointer to the structure containing Retrieved Encrypted Data TLV buffer and to be populated with decoded data from TLV. |
|-------------|---|

Returns

Error code **0** for success

3.8.5.9 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

3.9 Authenticated encryption

Functions

- [hsm_err_t hsm_do_auth_enc](#) ([hsm_hdl_t](#) key_store_hdl, [op_auth_enc_args_t](#) *auth_enc_args)
- [hsm_err_t hsm_do_auth_enc_new](#) ([hsm_hdl_t](#) key_store_hdl, [op_auth_enc_new_args_t](#) *args)

3.9.1 Detailed description

3.9.2 Function documentation

3.9.2.1 hsm_do_auth_enc()

```
hsm_err_t hsm_do_auth_enc (
hsm_hdl_t key_store_hdl,
op_auth_enc_args_t * auth_enc_args)
```

Secondary API to perform Authenticated Encryption.

This API does the following:

- Opens Cipher service flow.
- Perform authenticated encryption operation.
- Terminates the previously opened Cipher service flow.

Users can call this function only after opening a key store service flow.

Note: The API is deprecated now.

Parameters

| | |
|---------------|---|
| key_store_hdl | Handle identifying the key store service flow. |
| auth_enc_args | Pointer to the structure containing the function arguments. |

Returns

Error code

3.9.2.2 hsm_do_auth_enc_new()

```
hsm_err_t hsm_do_auth_enc_new(
hsm_hdl_t key_store_hdl,
op_auth_enc_new_args_t *args)
```

Secondary API to perform Authenticated Encryption (New).

This API does the following:

- Opens Cipher service flow.
- Perform authenticated encryption operation.
 - One-Shot AEAD operation using an opaque or plaintext key.
 - Streaming AEAD operation using an opaque or plaintext key.
- Terminates the previously opened Cipher service flow.

Users can call this function only after opening a key store service flow.

Parameters

| | |
|---------------|---|
| key_store_hdl | Handle identifying the key store service flow. |
| args | Pointer to the structure containing the function arguments. |

Returns

Error code

3.10 MAC

Data Structures

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

Macros

- #define HSM_OP_MAC_FLAGS_MAC_VERIFICATION ((hsm_op_mac_flags_t)(0u << 0))
- #define HSM_OP_MAC_FLAGS_MAC_GENERATION ((hsm_op_mac_flags_t)(1u << 0))
- #define HSM_OP_MAC_FLAGS_PLAINTEXT_KEY ((hsm_op_mac_flags_t)(1u << 3))
- #define HSM_OP_MAC_FLAGS_INIT ((hsm_op_mac_flags_t)(1u << 8))
- #define HSM_OP_MAC_FLAGS_UPDATE_DATA ((hsm_op_mac_flags_t)(1u << 10))
- #define HSM_OP_MAC_FLAGS_FINALIZE ((hsm_op_mac_flags_t)(1u << 11))
- #define HSM_OP_MAC_FLAGS_FINALIZE_VERIFY ((hsm_op_mac_flags_t)(1u << 12))
- #define HSM_OP_MAC_FLAGS_ABORT ((hsm_op_mac_flags_t)(1u << 13))
- #define HSM_OP_MAC_FLAGS_GET_CTX_SIZE ((hsm_op_mac_flags_t)(1u << 15))
- #define HSM_MAC_VERIFICATION_STATUS_FAILURE ((hsm_mac_verification_status_t)(0x0u))
- #define [HSM_OP_MAC_ONE_GO_FLAGS_MAC_VERIFICATION](#) (HSM_OP_MAC_FLAGS_MAC_VERIFICATION)
- #define [HSM_OP_MAC_ONE_GO_FLAGS_MAC_GENERATION](#) (HSM_OP_MAC_FLAGS_MAC_GENERATION)
- #define [HSM_MAC_VERIFICATION_STATUS_SUCCESS](#) ((hsm_mac_verification_status_t) (0x6C1AA1C6u))

Typedefs

- typedef uint16_t [hsm_op_mac_flags_t](#)
- typedef [hsm_op_mac_flags_t](#) [hsm_op_mac_one_go_flags_t](#)
- typedef uint32_t [hsm_mac_verification_status_t](#)
- typedef [hsm_permitted_algo_t](#) [hsm_op_mac_algo_t](#)
- typedef [hsm_op_mac_algo_t](#) [hsm_op_mac_one_go_algo_t](#)
- typedef [op_mac_args_t](#) [op_mac_one_go_args_t](#)

Functions

- [hsm_err_t hsm_do_mac](#) ([hsm_hdl_t](#) key_store_hdl, [op_mac_args_t](#) *args)
- [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_mac](#) ([hsm_hdl_t](#) mac_hdl, [op_mac_args_t](#) *args, [hsm_mac_verification_status_t](#) *status)
- [hsm_err_t hsm_close_mac_service](#) ([hsm_hdl_t](#) mac_hdl)

3.10.1 Detailed description

3.10.2 Data structure documentation

3.10.2.1 struct open_svc_mac_args_t

Structure describing the MAC open service member arguments.

Data Fields

| | | |
|---------------------------|--------------|---------------------------|
| hsm_hdl_t | mac_serv_hdl | Indicates the MAC handle. |
|---------------------------|--------------|---------------------------|

3.10.2.2 struct op_mac_args_t

Structure describing the MAC operation member arguments.

Data Fields

| | | |
|-------------------------------|---------------------|---|
| uint32_t | key_identifier | Identifier of the key to be used for the operation. The flag for plaintext key must not be set. |
| uint8_t * | key | Pointer to the plaintext key buffer. The flag for plaintext key must be set. Only valid for INIT and ONE SHOT operations. |
| hsm_op_mac_algo_t | algorithm | Algorithm to be used for the operation. |
| hsm_op_mac_flags_t | flags | Bitmap specifying the operation attributes. |
| uint8_t * | payload | Pointer to the payload area. Only valid for UPDATE DATA and ONE SHOT operations. |
| uint8_t * | mac | Pointer to the MAC area. <ul style="list-style-type: none"> MAC generation: Output MAC value MAC verification: Input MAC value to verify <ul style="list-style-type: none"> Only valid for FINALIZE, FINALIZE VERIFY, and ONE SHOT operations. |
| uint32_t | payload_size | Length in bytes of the payload. Only valid for UPDATE DATA and ONE SHOT operations. |
| uint16_t | mac_size | Length of the MAC. Only valid for FINALIZE, FINALIZE VERIFY, and ONE SHOT operations. |
| uint16_t | key_type | Key type. Only valid with PLAINTEXT flag. Only valid with ONE SHOT or INIT operations. |
| uint16_t | key_size | Key size in bytes. Only valid with PLAINTEXT flag. Only valid with ONE SHOT or INIT operations. |
| uint8_t * | context | Context buffer. Only valid with PLAINTEXT flag. |
| uint16_t | context_size | Size in Bytes of context buffer. Only valid with PLAINTEXT flag. |
| hsm_mac_verification_status_t | verification_status | MAC verification status. |
| uint16_t | exp_mac_size | Expected MAC size for output, returned by FW if the MAC size provided is less than the expected MAC size calculated from the MAC algorithm. Note: This field will be deprecated soon. exp_output_size to be used instead. |
| uint16_t | exp_output_size | Expected output buffer size in bytes, valid in case of HSM_OUT_TOO_SMALL (0x1D) error code. In case of streaming operation, expected context size in bytes if operation failed with 0x1E. |

3.10.3 Macro definition documentation

3.10.3.1 HSM_OP_MAC_FLAGS_MAC_VERIFICATION

```
#define HSM_OP_MAC_FLAGS_MAC_VERIFICATION ((hsm_op_mac_flags_t) (0u << 0))
```

Bit indicating the MAC Verify operation.

3.10.3.2 HSM_OP_MAC_FLAGS_MAC_GENERATION

```
#define HSM_OP_MAC_FLAGS_MAC_GENERATION ((hsm_op_mac_flags_t) (1u << 0))
```

Bit indicating the MAC Generate operation.

3.10.3.3 HSM_OP_MAC_FLAGS_PLAINTEXT_KEY

```
#define HSM_OP_MAC_FLAGS_PLAINTEXT_KEY ((hsm_op_mac_flags_t) (1u << 3))
```

Bit indicating the PLAINTEXT key user input.

3.10.3.4 HSM_OP_MAC_FLAGS_INIT

```
#define HSM_OP_MAC_FLAGS_INIT ((hsm_op_mac_flags_t) (1u << 8))
```

Bit indicating the initialization of a multipart context.

3.10.3.5 HSM_OP_MAC_FLAGS_UPDATE_DATA

```
#define HSM_OP_MAC_FLAGS_UPDATE_DATA ((hsm_op_mac_flags_t) (1u << 10))
```

Bit indicating the UPDATE DATA to be processed for the multipart operation.

3.10.3.6 HSM_OP_MAC_FLAGS_FINALIZE

```
#define HSM_OP_MAC_FLAGS_FINALIZE ((hsm_op_mac_flags_t) (1u << 11))
```

Bit indicating the FINALIZE multipart operation, for example, MAC Generation.

3.10.3.7 HSM_OP_MAC_FLAGS_FINALIZE_VERIFY

```
#define HSM_OP_MAC_FLAGS_FINALIZE_VERIFY ((hsm_op_mac_flags_t) (1u << 12))
```

Bit indicating the FINALIZE VERIFY multipart operation, for example, finalize the operation and internal MAC verification.

3.10.3.8 HSM_OP_MAC_FLAGS_ABORT

```
#define HSM_OP_MAC_FLAGS_ABORT ((hsm_op_mac_flags_t) (1u << 13))
```

Bit indicating to ABORT the multipart operation and clean the context.

3.10.3.9 HSM_OP_MAC_FLAGS_GET_CTX_SIZE

```
#define HSM_OP_MAC_FLAGS_GET_CTX_SIZE ((hsm_op_mac_flags_t) (1u << 15))
```

Bit indicating to get the context size for multipart operation. Size of the needed allocated context, returned in the output size response field.

3.10.3.10 HSM_MAC_VERIFICATION_STATUS_FAILURE

```
#define HSM_MAC_VERIFICATION_STATUS_FAILURE ((hsm_mac_verification_status_t) (0x0u))
```

Bit indicating the MAC verification failure status.

3.10.3.11 HSM_OP_MAC_ONE_GO_FLAGS_MAC_VERIFICATION

```
#define HSM_OP_MAC_ONE_GO_FLAGS_MAC_VERIFICATION (HSM_OP_MAC_FLAGS_MAC_VERIFICATION)
```

Bit indicating MAC one go verification operation.

3.10.3.12 HSM_OP_MAC_ONE_GO_FLAGS_MAC_GENERATION

```
#define HSM_OP_MAC_ONE_GO_FLAGS_MAC_GENERATION (HSM_OP_MAC_FLAGS_MAC_GENERATION)
```

Bit indicating MAC one go generate operation.

3.10.3.13 HSM_MAC_VERIFICATION_STATUS_SUCCESS

```
#define HSM_MAC_VERIFICATION_STATUS_SUCCESS ((hsm\_mac\_verification\_status\_t) (0x6C1AA1C6u))
```

Bit indicating MAC verification success status.

3.10.4 Typedef documentation

3.10.4.1 hsm_op_mac_flags_t

```
typedef uint16_t hsm_op_mac_flags_t
```

Bitmap describing the MAC operation.

3.10.4.2 hsm_op_mac_one_go_flags_t

```
typedef hsm_op_mac_flags_t hsm\_op\_mac\_one\_go\_flags\_t
```

Bitmap describing the MAC one go operation.

3.10.4.3 hsm_mac_verification_status_t

```
typedef uint32_t hsm_mac_verification_status_t
```

Bitmap describing the MAC verification status.

3.10.4.4 hsm_op_mac_algo_t

```
typedef hsm_permitted_algo_t hsm_op_mac_algo_t
```

Bitmap describing the MAC operation permitted algorithm.

The following three permitted algos are allowed:

- PERMITTED_ALGO_HMAC_SHA256 = 0x03800009
- PERMITTED_ALGO_HMAC_SHA384 = 0x0380000A
- PERMITTED_ALGO_CMAC = 0x03C00200
- PERMITTED_ALGO_CBCMAC (SM4_ONLY) = 0x03C00100

3.10.4.5 hsm_op_mac_one_go_algo_t

```
typedef hsm_op_mac_algo_t hsm_op_mac_one_go_algo_t
```

Bitmap describing the MAC one go operation permitted algorithm.

3.10.4.6 op_mac_one_go_args_t

```
typedef op_mac_args_t op_mac_one_go_args_t
```

Structure describing the MAC one go operation member arguments.

3.10.5 Function documentation

3.10.5.1 hsm_do_mac()

```
hsm_err_t hsm_do_mac (
hsm_hdl_t key_store_hdl,
op_mac_args_t *args)
```

Secondary API to perform MAC operation.

This API does the following:

- Open an MAC Service Flow.
- Perform MAC operation.
- Terminate a previously opened MAC service flow.

Users can call this function only after opening a key store service flow.

Parameters

| | |
|----------------------|---|
| <i>key_store_hdl</i> | Handle identifying the key store service flow. |
| <i>args</i> | Pointer to the structure containing the function arguments. |

Returns

Error code

3.10.5.2 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.

Users can call this function only after opening a key store service flow.

Users must open this service to perform MAC 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>mac_hdl</i> | Pointer to where the MAC service flow handle must be written. |

Returns

Error code

3.10.5.3 hsm_mac_one_go()

```
hsm_err_t hsm_mac_one_go (
hsm_hdl_t mac_hdl,
op_mac_args_t * args,
hsm_mac_verification_status_t * status)
```

Perform MAC operation.

Users can call this function only after opening a MAC service flow.

Note: This API will be deprecated soon.

Parameters

| | |
|----------------|---|
| <i>mac_hdl</i> | Handle identifying the MAC service flow. |
| <i>args</i> | Pointer to the structure containing the function arguments. |
| <i>status</i> | Pointer for storing the verification status. |

Returns

Error code

3.10.5.4 hsm_mac()

```
hsm_err_t hsm_mac (
hsm_hdl_t mac_hdl,
op_mac_args_t *args,
hsm_mac_verification_status_t *status)
```

Perform MAC operation.

- One-Shot MAC generation or verification operation using opaque or plaintext key.
- Streaming MAC generation or verification operation using opaque or plaintext key.

Users can call this function only after opening a MAC service flow.

Parameters

| | |
|----------------|---|
| <i>mac_hdl</i> | Handle identifying the MAC service flow. |
| <i>args</i> | Pointer to the structure containing the function arguments. |
| <i>status</i> | Pointer for storing the verification status. |

Returns

Error code

3.10.5.5 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

3.11 Dump firmware log

Data Structures

- struct [op_debug_dump_args_t](#)

Functions

- [hsm_err_t dump_firmware_log](#) ([hsm_hdl_t](#) session_hdl)

3.11.1 Detailed description

3.11.2 Data structure documentation

3.11.2.1 struct op_debug_dump_args_t

Structure detailing the debug dump operation member arguments.

Data Fields

| | | |
|----------|------------------------|---|
| bool | is_dump_pending | - |
| uint32_t | dump_buf_len | - |
| uint32_t | dump_buf[MAC_BUFF_LEN] | - |

3.11.3 Function documentation

3.11.3.1 dump_firmware_log()

```
hsm_err_t dump_firmware_log (hsm_hdl_t session_hdl)
```

This command is designed to dump the firmware logs.

Parameters

| | |
|-------------|--|
| session_hdl | Handle identifying the session handle. |
|-------------|--|

Returns

Error code

3.12 Dev attest

Data Structures

- struct [op_dev_attest_args_t](#)

Macros

- #define [DEV_ATTEST_NOUNCE_SIZE_V1](#) (4)
- #define [DEV_ATTEST_NOUNCE_SIZE_V2](#) (16)

Functions

- [hsm_err_t hsm_dev_attest](#) (hsm_hdl_t sess_hdl, op_dev_attest_args_t *args)

3.12.1 Detailed description

3.12.2 Data structure documentation

3.12.2.1 struct op_dev_attest_args_t

Structure describing the device attestation operation member arguments. Memory for storing uid/sha_rom_patch/sha_fw/signature is allocated by the HSM library. Caller of the function [hsm_dev_attest\(\)](#) needs to ensure freeing up memory.

Data Fields

| | | |
|-----------|------------------|--|
| uint16_t | soc_id | SoC ID. |
| uint16_t | soc_rev | SoC Revision. |
| uint16_t | lmda_val | Lmda Lifecycle value. |
| uint8_t | ssm_state | Security Subsystem State Machine state. |
| uint8_t | uid_sz | Buffer size in bytes for Chip Unique Identifier. |
| uint8_t * | uid | Pointer to the Chip Unique Identifier buffer. |
| uint16_t | rom_patch_sha_sz | Buffer size in bytes for SHA256 of Sentinel ROM patch fuses. |
| uint16_t | sha_fw_sz | Buffer size in bytes for the first 256 bits of installed FW SHA. |
| uint8_t * | sha_rom_patch | Pointer to the buffer containing SHA256 of Sentinel ROM patch fuses. |
| uint8_t * | sha_fw | Pointer to the buffer containing first 256 bits of installed FW SHA. |

Data Fields...continued

| | | |
|-----------|-----------------|--|
| uint16_t | nounce_sz | Buffer size in bytes for request nounce value. |
| uint8_t * | nounce | Pointer to the input/request nounce value buffer. |
| uint16_t | rsp_nounce_sz | Size in bytes for FW nounce buffer, returned with FW resp. |
| uint8_t * | rsp_nounce | Pointer to the FW nounce buffer, returned with FW resp. |
| uint16_t | oem_srkh_sz | Buffer size in bytes for OEM SRKH (version 2). |
| uint8_t * | oem_srkh | Pointer to the buffer of OEM SRKH (version 2). |
| uint8_t | imem_state | IMEM state (version 2). |
| uint8_t | csal_state | CSAL state (version 2). |
| uint8_t | trng_state | TRNG state (version 2). |
| uint16_t | oem_pqc_srkh_sz | Buffer size in bytes for OEM PQC SRKH (version 3). |
| uint8_t* | oem_pqc_srkh | Pointer to the buffer of OEM PQC SRKH (version 3). |
| uint16_t | info_buf_sz | Size in bytes for info buffer. |
| uint8_t * | info_buf | Pointer to the info buffer, for verification of the signature. |
| uint16_t | sign_sz | Buffer size in bytes for signature. |
| uint8_t * | signature | Pointer to the signature buffer. |

3.12.3 Macro definition documentation

3.12.3.1 DEV_ATTEST_NOUNCE_SIZE_V1

```
#define DEV_ATTEST_NOUNCE_SIZE_V1 (4)
```

Device Attestation Nounce sizes.

3.12.4 Function documentation

3.12.4.1 hsm_dev_attest()

```
hsm_err_t hsm_dev_attest (
hsm_hdl_t sess_hdl,
op_dev_attest_args_t * args)
```

Perform device attestation operation.

Users can call this function only after opening the session.

Parameters

| | |
|----------|---|
| sess_hdl | Handle identifying the active session. |
| args | Pointer to the structure containing the function arguments. |

Returns

Error code

3.13 Dev Info

Data Structures

- struct [op_dev_getinfo_args_t](#)

Enumerations

- enum [hsm_lmda_val_t](#) {
 HSM_LMDA_OEM_OPEN = 0x10,
 HSM_LMDA_OEM_CLOSED = 0x40,
 HSM_LMDA_OEM_LOCKED = 0x200 }

Functions

- [hsm_err_t hsm_dev_getinfo](#) ([hsm_hdl_t](#) sess_hdl, [op_dev_getinfo_args_t](#) *args)
- [hsm_key_lifecycle_t hsm_get_lc_from_lmda](#) ([hsm_lmda_val_t](#) lmda_val)

3.13.1 Detailed description

3.13.2 Data structure documentation

3.13.2.1 struct op_dev_getinfo_args_t

Structure detailing the device getinfo operation member arguments. Memory for storing uid/sha_rom_patch/sha_fw/signature is allocated by the HSM library. Caller of the function [hsm_dev_getinfo\(\)](#) needs to ensure freeing up memory.

Data Fields

| | | |
|-----------|------------------|---|
| uint16_t | soc_id | SoC ID. |
| uint16_t | soc_rev | SoC revision number. |
| uint16_t | lmda_val | Indicates the lmda lifecycle value. |
| uint8_t | ssm_state | Security subsystem state machine. |
| uint8_t | uid_sz | Chip unique identifier size. |
| uint8_t * | uid | Pointer to the chip unique identifier. |
| uint16_t | rom_patch_sha_sz | Indicates the size of SHA256 of sentinel ROM patch fuses. |
| uint16_t | sha_fw_sz | Indicates the size of first 256 bits of installed FW SHA. |
| uint8_t * | sha_rom_patch | Pointer to the SHA256 of sentinel ROM patch fuses digest. |
| uint8_t * | sha_fw | Pointer to the first 256 bits of installed FW SHA digest. |
| uint16_t | oem_srkh_sz | Indicates the size of FW OEM SRKH. |
| uint8_t * | oem_srkh | Pointer to the FW OEM SRKH. |
| uint8_t | imem_state | Indicates the IMEM state. |
| uint8_t | csal_state | Crypto Lib random context initialization state. |
| uint8_t | trng_state | Indicates TRNG state. |
| uint16_t | oem_pqc_srkh_sz | Indicates the size of FW OEM PQC SRKH. |
| uint8_t* | oem_pqc_srkh | Pointer to the FW OEM PQC SRKH. |

3.13.2.2 hsm_lmda_val_t

```
hsm_lmda_val_t enum hsm_lmda_val_t
```

LMDA values.

Enumerator

| | |
|---------------------|--|
| HSM_LMDA_OEM_OPEN | LMDA value for OEM Open state LMDA value for OEM Closed state. |
| HSM_LMDA_OEM_CLOSED | LMDA value for OEM Locked state. |

3.13.3 Function documentation

3.13.3.1 hsm_dev_getinfo()

```
hsm_err_t hsm_dev_getinfo (
hsm_hdl_t sess_hdl,
op_dev_getinfo_args_t * args)
```

Perform the device attestation operation.

Users can call this function only after opening the session.

Parameters

| | |
|----------|---|
| sess_hdl | Handle identifying the active session. |
| args | Pointer to the structure containing the function arguments. |

Returns

Error code

3.13.3.2 hsm_get_lc_from_lmda()

```
hsm_key_lifecycle_t hsm_get_lc_from_lmda (hsm_lmda_val_t lmda_val)
```

Returns the lifecycle value corresponding to the given LMDA value.

Parameters

| | |
|----------|-------------|
| lmda_val | LMDA value. |
|----------|-------------|

Returns

LC Lifecycle value

3.14 Write Fuse

Data Structures

- struct op_dev_write_fuse_args_t

Functions

- hsm_err_t hsm_dev_write_fuse (hsm_hdl_t sess_hdl, op_dev_write_fuse_args_t a *rgs)

3.14.1 Detailed description

3.14.2 Data structure documentation

3.14.2.1 struct op_dev_write_fuse_args_t

Structure detailing the device write fuse operation member arguments.

Data Fields

| | | |
|----------|---------------|---|
| uint16_t | processed_idx | Last processed fuse. Value is valid if != 0xffff. In case of failure, it can indicate which fuse triggered the error. In case of success, it provide the last fused word index. |
| uint16_t | word_pos | Fuse identifier expressed as its position in word in the fuse map. |
| uint16_t | bit_pos | Fuse identifier expressed as its position in bit in the fuse map. |
| uint16_t | bit_len | Contains bit length. |
| uint32_t | payload[] | Payload. |

3.14.3 Function documentation

3.14.3.1 hsm_dev_write_fuse()

```
hsm_err_t hsm_dev_write_fuse (
hsm_hdl_t sess_hdl, op_dev_write_fuse_args_t * args )
```

Perform the write fuse operation.

Users can call this function only after having opened the session.

Parameters

| | |
|----------|---|
| sess_hdl | Handle identifying the active session. |
| args | Pointer to the structure containing the function arguments. |

Returns

Error code

3.15 Generic Crypto: Asymmetric Crypto

Data Structures

- struct [op_gc_acrypto_args_t](#)

Macros

- #define HSM_OP_GC_ACRYPTO_FLAGS_INPUT_DIGEST ((hsm_op_gc_acrypto_flags_t) (0u << 0))
- #define [HSM_OP_GC_ACRYPTO_FLAGS_INPUT_MESSAGE](#) ((hsm_op_gc_acrypto_flags_t) (1u << 0))
- #define [HSM_GC_ACRYPTO_VERIFICATION_SUCCESS](#) ((hsm_gc_acrypto_verification_status_t) (0x5A3CC3A5u))
- #define [HSM_GC_ACRYPTO_VERIFICATION_FAILURE](#) ((hsm_gc_acrypto_verification_status_t) (0x2B4DD4B2u))

Typedefs

- typedef uint8_t [hsm_op_gc_acrypto_flags_t](#)
- typedef uint32_t [hsm_gc_acrypto_verification_status_t](#)

Enumerations

- enum [hsm_op_gc_acrypto_algo_t](#) {
 - HSM_GC_ACRYPTO_ALGO_ECDSA_SHA224 = ALGO_ECDSA_SHA224,
 - HSM_GC_ACRYPTO_ALGO_ECDSA_SHA256 = ALGO_ECDSA_SHA256,
 - HSM_GC_ACRYPTO_ALGO_ECDSA_SHA384 = ALGO_ECDSA_SHA384,
 - HSM_GC_ACRYPTO_ALGO_ECDSA_SHA512 = ALGO_ECDSA_SHA512,
 - HSM_GC_ACRYPTO_ALGO_RSA_PKCS1_V15_SHA224 = ALGO_RSA_PKCS1_V15_SHA224,
 - HSM_GC_ACRYPTO_ALGO_RSA_PKCS1_V15_SHA256 = ALGO_RSA_PKCS1_V15_SHA256,
 - HSM_GC_ACRYPTO_ALGO_RSA_PKCS1_V15_SHA384 = ALGO_RSA_PKCS1_V15_SHA384,
 - HSM_GC_ACRYPTO_ALGO_RSA_PKCS1_V15_SHA512 = ALGO_RSA_PKCS1_V15_SHA512,
 - HSM_GC_ACRYPTO_ALGO_RSA_PKCS1_PSS_MGF1_SHA224 = ALGO_RSA_PKCS1_PSS_MGF1_SHA224,
 - HSM_GC_ACRYPTO_ALGO_RSA_PKCS1_PSS_MGF1_SHA256 = ALGO_RSA_PKCS1_PSS_MGF1_SHA256,
 - HSM_GC_ACRYPTO_ALGO_RSA_PKCS1_PSS_MGF1_SHA384 = ALGO_RSA_PKCS1_PSS_MGF1_SHA384,
 - HSM_GC_ACRYPTO_ALGO_RSA_PKCS1_PSS_MGF1_SHA512 = ALGO_RSA_PKCS1_PSS_MGF1_SHA512,
 - HSM_GC_ACRYPTO_ALGO_RSA_PKCS1_V15_CRYPT = ALGO_RSA_PKCS1_V15_CRYPT,
 - HSM_GC_ACRYPTO_ALGO_RSA_PKCS1_OAEP_SHA1 = ALGO_RSA_PKCS1_OAEP_SHA1,
 - HSM_GC_ACRYPTO_ALGO_RSA_PKCS1_OAEP_SHA224 = ALGO_RSA_PKCS1_OAEP_SHA224,
 - HSM_GC_ACRYPTO_ALGO_RSA_PKCS1_OAEP_SHA256 = ALGO_RSA_PKCS1_OAEP_SHA256,
 - HSM_GC_ACRYPTO_ALGO_RSA_PKCS1_OAEP_SHA384 = ALGO_RSA_PKCS1_OAEP_SHA384,
 - HSM_GC_ACRYPTO_ALGO_RSA_PKCS1_OAEP_SHA512 = ALGO_RSA_PKCS1_OAEP_SHA512 }
- enum [hsm_gc_acrypto_op_mode_t](#) {
 - HSM_GC_ACRYPTO_OP_MODE_ENCRYPT = 0x01,
 - HSM_GC_ACRYPTO_OP_MODE_DECRYPT = 0x02,
 - HSM_GC_ACRYPTO_OP_MODE_SIGN_GEN = 0x03,
 - HSM_GC_ACRYPTO_OP_MODE_SIGN_VER = 0x04 }

Functions

- [hsm_err_t hsm_gc_acrypto](#) ([hsm_hdl_t](#) session_hdl, [op_gc_acrypto_args_t](#)*args)

3.15.1 Detailed description

3.15.2 Data structure documentation

3.15.2.1 struct op_gc_acrypto_args_t

Structure describing the generic asymmetric crypto member arguments.

Data Fields

| | | |
|---|------------|-------------------------------------|
| hsm_op_gc_acrypto_algo_t | algorithm | Algorithm to use for the operation. |
| hsm_gc_acrypto_op_mode_t | op_mode | Indicates the operation mode. |
| hsm_op_gc_acrypto_flags_t | flags | Indicates operation flags. |
| hsm_bit_key_sz_t | bit_key_sz | Key size in bits. |
| uint8_t * | data_buff1 | Pointer to the data buffer 1: |

Data Fields...continued

| | | |
|--|---------------------|--|
| | | <ul style="list-style-type: none"> • Plaintext in case of encryption/decryption operation. • Digest or message in case of signature generation/verification operation. |
| uint8_t * | data_buff2 | Pointer to the data buffer 2: <ul style="list-style-type: none"> • Ciphertext in case of encryption/decryption operation. • Signature in case of signature generation/verification operation. |
| uint32_t | data_buff1_size | Size in bytes of data buffer 1. |
| uint32_t | data_buff2_size | Size in bytes of data buffer 2. |
| uint8_t * | key_buff1 | Pointer to the key modulus buffer. |
| uint8_t * | key_buff2 | Pointer the key exponent, either private or public. <ul style="list-style-type: none"> • Encryption mode, public exponent • Decryption mode, private exponent • Signature Generation mode, private exponent • Signature Verification mode, public exponent |
| uint16_t | key_buff1_size | Size in bytes of the key buffer 1. |
| uint16_t | key_buff2_size | Size in bytes of the key buffer 2. |
| uint8_t * | rsa_label | RSA label address. Only used for OAEP encryption/decryption operation mode and optional. |
| uint16_t | rsa_label_size | RSA label size in bytes. Only used for OAEP encryption/decryption operation mode. |
| uint16_t | rsa_salt_len | RSA salt length in bytes. Only used for PSS signature algorithm scheme. |
| uint32_t | exp_plaintext_len | Expected plaintext length in bytes, returned by FW in case of DECRYPT operation mode. |
| hsm_gc_acrypto_verification_status_t | verification_status | Signature verification status. |

3.15.3 Macro definition documentation

3.15.3.1 HSM_OP_GC_ACRYPTO_FLAGS_INPUT_DIGEST

```
#define HSM_OP_GC_ACRYPTO_FLAGS_INPUT_DIGEST ((hsm_op_gc_acrypto_flags_t) (0u << 0))
```

Bit indicating the generic asymmetric crypto input message digest operation. Only valid for signature modes.

3.15.3.2 HSM_OP_GC_ACRYPTO_FLAGS_INPUT_MESSAGE

```
#define HSM_OP_GC_ACRYPTO_FLAGS_INPUT_MESSAGE ((hsm_op_gc_acrypto_flags_t) (1u << 0))
```

Bit indicating the generic asymmetric crypto input message operation. Only valid for signature modes.

3.15.3.3 HSM_GC_ACRYPTO_VERIFICATION_SUCCESS

```
#define HSM_GC_ACRYPTO_VERIFICATION_SUCCESS  
((hsm\_gc\_acrypto\_verification\_status\_t) (0x5A3CC3A5u))
```

Bit indicating the generic asymmetric crypto success verification status.

3.15.3.4 HSM_GC_ACRYPTO_VERIFICATION_FAILURE

```
#define HSM_GC_ACRYPTO_VERIFICATION_FAILURE  
((hsm\_gc\_acrypto\_verification\_status\_t) (0x2B4DD4B2u))
```

Bit indicating the generic asymmetric crypto failure verification status.

3.15.4 Typedef documentation

3.15.4.1 hsm_op_gc_acrypto_flags_t

```
typedef uint8_t hsm\_op\_gc\_acrypto\_flags\_t
```

Bitmap describing the generic asymmetric crypto supported operation.

3.15.4.2 hsm_gc_acrypto_verification_status_t

```
typedef uint32_t hsm\_gc\_acrypto\_verification\_status\_t
```

Bitmap describing the generic asymmetric crypto verification status.

3.15.5 Enumeration type documentation

3.15.5.1 hsm_op_gc_acrypto_algo_t

```
enum hsm\_op\_gc\_acrypto\_algo\_t
```

Enum detailing the generic asymmetric crypto supported algorithms.

3.15.5.2 hsm_gc_acrypto_op_mode_t

```
enum hsm\_gc\_acrypto\_op\_mode\_t
```

Enum describing the generic asymmetric crypto supported operating modes.

3.15.6 Function documentation

3.15.6.1 hsm_gc_acrypto()

```
hsm\_err\_t hsm_gc_acrypto (  
hsm\_hdl\_t session_hdl,  
op\_gc\_acrypto\_args\_t * args)
```

This command is designed to perform the following operations:

- Asymmetric crypto
- Encryption/decryption
- Signature generation/verification

Parameters

| | |
|--------------------------|---|
| <code>session_hdl</code> | Handle identifying the current session. |
| <code>args</code> | Pointer to the structure containing the function arguments. |

Returns

Error code

3.16 Generic Crypto Asymmetric Key Generate

Data Structures

- struct [op_gc_akey_gen_args_t](#)

Functions

- [hsm_err_t hsm_gc_akey_gen](#) ([hsm_hdl_t](#) session_hdl, [op_gc_akey_gen_args_t](#) *args)

3.16.1 Detailed description

3.16.2 Data structure documentation

3.16.2.1 struct op_gc_akey_gen_args_t

Structure detailing the generic crypto asymmetric key generate operation members.

Table 4. Data Fields

| | | |
|----------------------------------|----------------|---|
| uint8_t * | modulus | Pointer to the output buffer of key modulus. |
| uint8_t * | priv_buff | Pointer to the output buffer of key private exponent. |
| uint8_t * | pub_buff | Pointer to the input buffer containing key public exponent. |
| uint16_t | modulus_size | Size in bytes of the modulus buffer. |
| uint16_t | priv_buff_size | Size in bytes of the private exponent buffer. |
| uint16_t | pub_buff_size | Size in bytes of the public exponent buffer. |
| hsm_key_type_t | key_type | Indicates which type of keypair must be generated. |
| hsm_bit_key_sz_t | bit_key_sz | Size in bits of the keypair to be generated. |

3.16.3 Function documentation

3.16.3.1 hsm_gc_akey_gen()

```

hsm\_err\_t hsm_gc_akey_gen (
hsm\_hdl\_t session_hdl,
op\_gc\_akey\_gen\_args\_t * args)
    
```

This command is designed to perform the following operation:

Generate asymmetric keys, without using FW keystore.

Parameters

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

Returns

Error code

3.17 Generic Crypto: Cipher

Data Structures

- struct op_gc_cipher_args_t

Enumerations

- enum hsm_op_gc_cipher_algo_t {
HSM_GC_CIPHER_ALGO_AES_ECB_NOPAD = 0x2,
HSM_GC_CIPHER_ALGO_AES_CBC_NOPAD = 0x3 }
- enum hsm_gc_cipher_op_mode_t {
HSM_GC_CIPHER_OP_MODE_ENCRYPT = 0x41,
HSM_GC_CIPHER_OP_MODE_DECRYPT = 0x42 }

Functions

- hsm_err_t hsm_gc_cipher (hsm_hdl_t session_hdl, op_gc_cipher_args_t *args)

3.17.1 Detailed description

3.17.2 Data structure documentation

3.17.2.1 struct op_gc_cipher_args_t

Structure describing member arguments of generic crypto for cipher ops.

Data Fields

| | | |
|-------------------------|-------------------|-------------------------------------|
| hsm_op_gc_cipher_algo_t | algorithm | Algorithm to use for the operation. |
| hsm_gc_cipher_op_mode_t | op_mode | Indicates the operation mode. |
| uint16_t | byte_key_sz | Key size in bytes. |
| uint8_t * | in_data_buff | Pointer to the input data buffer. |
| uint8_t * | out_data_buff | Pointer to the out data buffer. |
| uint32_t | in_data_buff_sz | Size in bytes of input data buffer. |
| uint8_t * | plain_key_buff | Pointer to the data buffer 1. |
| uint32_t | plain_key_buff_sz | Size in bytes of input data buffer. |
| uint8_t * | iv_buff | Pointer to the data buffer 1. |
| uint32_t | iv_buff_sz | Size in bytes of input data buffer. |

3.17.3 Enumeration type documentation

3.17.3.1 hsm_op_gc_cipher_algo_t

```
enum hsm_op_gc_cipher_algo_t
```

Enum detailing the generic crypto supported cipher algorithms.

3.17.3.2 hsm_gc_cipher_op_mode_t

```
enum hsm_gc_cipher_op_mode_t
```

Enum describing the generic cipher crypto supported operating modes.

3.17.4 Function documentation

3.17.4.1 hsm_gc_cipher ()

```
hsm_err_t hsm_gc_cipher (
hsm_hdl_t session_hdl,
op_gc_cipher_args_t * args)
```

This command is designed to perform the following operations:

- Cipher Operations:
 - AES ECB encryption/decryption
 - AES CBC encryption/decryption

Parameters

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

Returns

Error code

3.18 Generic Crypto: AEAD

Data Structures

- struct op_gc_aead_args_t

Enumerations

- enum hsm_op_gc_aead_algo_t {
 - HSM_GC_AEAD_ALGO_AES_GCM = 0x05,
 - HSM_GC_AEAD_ALGO_AES_CCM = 0x06,
 - HSM_GC_AEAD_ALGO_CHACHA20_POLY1305 = 0x07 }
- enum hsm_gc_aead_op_mode_t {
 - HSM_GC_AEAD_OP_MODE_ENCRYPT = 0x41,
 - HSM_GC_AEAD_OP_MODE_DECRYPT = 0x42 }

Functions

- `hsm_err_t hsm_gc_aead (hsm_hdl_t session_hdl, op_gc_aead_args_t *args)`

3.18.1 Detailed description

3.18.2 Data structure documentation

3.18.2.1 struct op_gc_aead_args_t

Structure describing the member arguments for generic crypto for AEAD Ops.

Data Fields

| | | |
|------------------------------------|--------------------------------|--|
| <code>hsm_op_gc_aead_algo_t</code> | <code>algorithm</code> | Algorithm to use for the operation. |
| <code>hsm_gc_aead_op_mode_t</code> | <code>op_mode</code> | Indicates the operation mode. |
| <code>uint8_t *</code> | <code>in_data_buff</code> | Pointer to the input data buffer. |
| <code>uint8_t *</code> | <code>out_data_buff</code> | Pointer to the out data buffer. |
| <code>uint32_t</code> | <code>in_data_buff_sz</code> | Size in bytes of input data buffer. |
| <code>uint8_t *</code> | <code>plain_key_buff</code> | Pointer to the plain-text key buffer. |
| <code>uint32_t</code> | <code>plain_key_buff_sz</code> | Size in bytes of plain-text key buffer. |
| <code>uint8_t *</code> | <code>nonce_buff</code> | Pointer to the nonce buffer. |
| <code>uint32_t</code> | <code>nonce_sz</code> | Size in bytes of the nonce buffer. |
| <code>uint8_t *</code> | <code>aad_buff</code> | Pointer to the authenticated associated data buffer. |
| <code>uint32_t</code> | <code>aad_sz</code> | Size in bytes of aad buffer. |
| <code>uint8_t *</code> | <code>tag_buff</code> | Pointer to the tag buffer. |
| <code>uint32_t</code> | <code>tag_sz</code> | Size in bytes of input data buffer. |

3.18.3 Enumeration type documentation

3.18.3.1 hsm_op_gc_aead_algo_t

```
enum hsm_op_gc_aead_algo_t
```

Enum detailing the generic aead crypto supported algorithms.

3.18.3.2 hsm_gc_aead_op_mode_t

```
enum hsm_gc_aead_op_mode_t
```

Enum describing the generic AEAD crypto supported operating modes.

3.18.4 Function documentation

3.18.4.1 hsm_gc_aead ()

```
hsm_err_t hsm_gc_aead (
hsm_hdl_t session_hdl,
op_gc_aead_args_t * args)
```

This command is designed to perform the following operations:

- AEAD Cipher generic crypto operations:
 - Encryption/Decryption

Parameters

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

Returns

Error code

3.19 Get Info

Data Structures

- struct [op_get_info_args_t](#)

Functions

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

3.19.1 Detailed description

3.19.2 Data structure documentation

3.19.2.1 struct op_get_info_args_t

Structure describing the get info operation member arguments.

Data Fields

| | | |
|-----------|------------------------|--|
| uint32_t | user_sab_id | Stores User identifier (32 bits). |
| uint8_t * | chip_unique_id | Stores the chip unique identifier. |
| uint16_t | chip_unq_id_sz | Size of the chip unique identifier in bytes. |
| uint16_t | chip_monotonic_counter | Stores the chip monotonic counter value (16 bits). |
| uint16_t | chip_life_cycle | Stores the chip current life cycle bitfield (16 bits). |
| uint32_t | version | Stores the module version (32 bits). |
| uint32_t | version_ext | Stores the module extended version (32 bits). |
| uint8_t | fips_mode | Stores the FIPS mode bitfield (8 bits). Bitmask definition: bit0: FIPS mode of operation: <ul style="list-style-type: none"> • Value 0: part is running in FIPS non-approved mode. • Value 1: part is running in FIPS approved mode. Bit 1: FIPS certified part: <ul style="list-style-type: none"> • Value 0: part is not FIPS certified. • Value 1: part is FIPS certified. Bit 2-7: reserved <ul style="list-style-type: none"> • Value 0. |

3.19.3 Function documentation

3.19.3.1 hsm_get_info()

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

Perform device attestation operation.

Users can call this function only after opening the session.

Parameters

| | |
|-----------------|---|
| <i>sess_hdl</i> | Handle identifying the active session. |
| <i>args</i> | Pointer to the structure containing the function arguments. |

Returns

Error code

3.20 Key exchange

Data Structures

- struct `op_key_exchange_args_t`

Macros

- #define `HSM_OP_KEY_EXCHANGE_FLAGS_INPUT_SIGNED_CONTENT` `((hsm_op_key_exchange_flags_t)(0u << 0))`
- #define `HSM_OP_KEY_EXCHANGE_FLAGS_INPUT_PLAINTEXT_CONTENT` `((hsm_op_key_exchange_flags_t)(1u << 0))`
- #define `HSM_OP_KEY_EXCHANGE_FLAGS_RETURN_KEY_IDS` `((hsm_op_key_exchange_flags_t)(1u << 1))`
- #define `HSM_OP_KEY_EXCHANGE_FLAGS_RETURN_OUTPUT` `((hsm_op_key_exchange_flags_t)(1u << 2))`
- #define `HSM_OP_KEY_EXCHANGE_FLAGS_SALT_ZERO` `((hsm_op_key_exchange_flags_t)(0u << 0))`
- #define `HSM_OP_KEY_EXCHANGE_FLAGS_SALT_PEER_PUBKEY_HASH` `((hsm_op_key_exchange_flags_t)(1u << 0))`
- #define `HSM_OP_KEY_EXCHANGE_FLAGS_MONOTONIC` `((hsm_op_key_exchange_flags_t)(1u << 5))`
- #define `HSM_OP_KEY_EXCHANGE_FLAGS_STRICT_OPERATION` `((hsm_op_key_exchange_flags_t)(1u << 7))`

Typedefs

```
typedef uint16_t hsm_op_key_exchange_flags_t
```

Enumerations

- enum `hsm_op_key_exchange_algo_t` {
 - `HSM_KEY_EX_ECDH_HKDF_SHA256_KEY_IMPORT = 0x09020109,`
 - `HSM_KEY_EX_ECDH_HKDF_SHA384_KEY_IMPORT = 0x0902010A,`
 - `HSM_KEY_EX_ECDH_HKDF_SHA256 = 0x89020109,`
 - `HSM_KEY_EX_ECDH_HKDF_SHA384 = 0x8902010A,`
 - `HSM_KEY_EXCHANGE_ECDH_HKDF_SHA256 = 0x09020109,`
 - `HSM_KEY_EXCHANGE_ECDH_HKDF_SHA384 = 0x0902010A}`

- enum [hsm_op_key_derivation_algo_t](#) {
 HSM_KEY_DERIVATION_HKDF_SHA256 = 0x08000109,
 HSM_KEY_DERIVATION_HKDF_SHA384 = 0x0800010A,
 HSM_KEY_DERIVATION_HKDF_EXTRACT_SHA256 = 0x08000409,
 HSM_KEY_DERIVATION_HKDF_EXTRACT_SHA384 = 0x0800040A,
 HSM_KEY_DERIVATION_HKDF_EXPAND_SHA256 = 0x08000509,
 HSM_KEY_DERIVATION_HKDF_EXPAND_SHA384 = 0x0800050A }
- enum [hsm_op_key_derivation_tls1_3_algo_t](#) {
 HSM_KEY_DERIVATION_TLS1_3_EARLY_SECRET_SHA256 = 0x8800D009,
 HSM_KEY_DERIVATION_TLS1_3_EARLY_SECRET_SHA384 = 0x8800D00A,
 HSM_KEY_DERIVATION_TLS1_3_HANDSHAKE_SECRET_SHA256 = 0x8800D109,
 HSM_KEY_DERIVATION_TLS1_3_HANDSHAKE_SECRET_SHA384 = 0x8800D10A,
 HSM_KEY_DERIVATION_TLS1_3_KEYING_MATERIAL_SHA256 = 0x8800D309,
 HSM_KEY_DERIVATION_TLS1_3_KEYING_MATERIAL_SHA384 = 0x8800D30A,
 HSM_KEY_DERIVATION_TLS1_3_IV_SHA256 = 0x8800D409,
 HSM_KEY_DERIVATION_TLS1_3_IV_SHA384 = 0x8800D40A,
 HSM_KEY_DERIVATION_TLS1_3_MASTER_SECRET_SHA256 = 0x8800D209,
 HSM_KEY_DERIVATION_TLS1_3_MASTER_SECRET_SHA384 = 0x8800D20A }
- enum [hsm_op_key_derivation_tls1_2_algo_t](#) {
 HSM_KEY_DERIVATION_TLS1_2_MASTER_SECRET_SHA256 = 0x8800E009,
 HSM_KEY_DERIVATION_TLS1_2_MASTER_SECRET_SHA384 = 0x8800E00A,
 HSM_KEY_DERIVATION_TLS1_2_KEY_BLOCK_SHA256 = 0x8800E109,
 HSM_KEY_DERIVATION_TLS1_2_KEY_BLOCK_SHA384 = 0x8800E10A,
 HSM_KEY_DERIVATION_TLS1_2_VERIFY_DATA_SHA256 = 0x8800E409,
 HSM_KEY_DERIVATION_TLS1_2_VERIFY_DATA_SHA384 = 0x8800E40A,
 HSM_KEY_DERIVATION_TLS1_2_IV_SHA256 = 0x8800E209,
 HSM_KEY_DERIVATION_TLS1_2_IV_SHA384 = 0x8800E20A}

Functions

[hsm_err_t hsm_key_exchange](#) (hsm_hdl_t key_management_hdl, op_key_exchange_args_t *args)

3.20.1 Detailed description

3.20.2 Data structure documentation

3.20.2.1 struct op_key_exchange_args_t

Structure describing the key exchange operation member arguments.

Table 5. Data Fields

| | | |
|-----------|--------------------|--|
| uint32_t | in_content_sz | Input content buffer size in bytes. |
| uint8_t * | in_content | Input content buffer. |
| uint32_t | in_pub_buffer_sz | Input public buffer size in bytes. |
| uint8_t * | in_pub_buffer | Input public buffer. |
| uint32_t | user_fixed_info_sz | Input user fixed information size in bytes. |
| uint8_t * | user_fixed_info | Input user fixed information buffer. Optional, i.e., can be NULL, depending on the key derivation process. |
| uint32_t | output_sz | Output buffer size in bytes. |

Table 5. Data Fields...continued

| | | |
|-----------------------------|--------------------|---|
| uint8_t * | output | Operation output returned as Output buffer. Optional ,i.e., can be NULL, depending on Flags. |
| uint32_t | out_derived_key_id | Identifier of the derived key, with FW Resp. |
| uint32_t | out_salt_sz | Salt size in bytes, from FW Resp. It is equal to the hash (in bytes) of hash algorithm used in the key exchange algorithm. |
| uint32_t | exp_output_sz | Size in bytes of the exported derived key buffer, or Expected output derived key size (bytes), in case of HSM_OUT_TOO_SMALL error code. |
| hsm_op_key_exchange_flags_t | flags | <p>Bit field indicating the requested operations:</p> <ul style="list-style-type: none"> • Bit 0: Input content type. <ul style="list-style-type: none"> – 0: Input signed content – 1: Input unsigned content • Bit 1: When multiple keys are derived (TLS 1.2), IDs are returned as buffer. • Bit 2: Operation output is returned as buffer (in plain text) and not stored in EdgeLock secure enclave storage. • Bit 3 to 4: Reserved. • Bit 5: Monotonic counter increment (SYNC operation). When used in conjunction with SYNC key group or SYNC key store and storage master, the request is completed only when the monotonic counter has been updated. • Bit 6: Reserved when the modified key store and storage master have been written in the NVM. • Bit 7: SYNC operation. The request is completed only when the new key has been written in NVM. |

3.20.2.2 Field description - Input Content buffer

This API can be used for several operations, by choosing the Input Content buffer accordingly.

The following sections provide the tables having representations and description of required Input Content formats, for each possible operation. The supported operations are as follows:

- Combined key agreement and key derivation operation
- TLS 1.2
 - Master Secret
 - Key Block
 - IV
- TLS 1.3
 - Early Secrets, Handshake Secrets, and Master Secrets
- HMAC-based Key Derivation Function
 - HKDF
 - HKDF Extract
 - HKDF Expand

3.20.2.2.1 Combined key agreement and key derivation operation

Table 6. Key exchange input signed content payload description

| Field | Size (in bits) | Description |
|---------------------------------|----------------|--|
| TAG | 8 | Signed content TAG, must be set to 0x47 . |
| Version | 8 | Signed content version, must be set to 0x07 . |
| Reserved | 16 | Reserved bits. |
| Key store ID | 32 | Key store ID where to store the derived key. It must be the key store ID related to the key management handle set in the command API. |
| Key exchange algorithm | 32 | Algorithm used by the key exchange process. |
| Derived key group | 16 | Indicates the derived key group ID. 100 groups are available per key store. It must be a value in the range [0; 99]. |
| Salt flags | 16 | Bit field indicating the requested salt flags: Bit 0: Salt in step #1 (HKDF-extract) of HMAC based two-step key derivation process. <ul style="list-style-type: none"> 0: Use zeros salt. 1: Use peer public key digest as salt. Bit 1: In case of OEM Master Key generation for Key Import purpose, salt used to derive OEM_IMPORT_WRAP_SK and OEM_IMPORT_CMAC_SK. <ul style="list-style-type: none"> 0: Zeros string. 1: Device SRKH. Bit 2 to 15: Reserved. |
| Derived key type | 16 | Derived key type attribute (Symmetric). Reminder: In case of OEM Master Key generation for Key Import purpose, the value must be set to OEM_IMPORT_MK_SK. |
| Derived key security size bits | 16 | Derived key security size in bits attribute (Symmetric). Reminder: In case of OEM Master Key generation for Key Import purpose, the value must be set to 256 . |
| Derived key lifetime | 32 | Derived key lifetime attribute. |
| Derived key usage | 32 | Derived key usage attribute. Reminder: In case of OEM Master Key generation for Key Import purpose, the value must be set to Derive . |
| Derived key permitted algorithm | 32 | Derived key permitted algorithm attribute. Reminder: In case of OEM Master Key generation for Key Import purpose, the value must be set to HKDF SHA256 . |
| Derived key lifecycle | 32 | Derived key lifecycle attribute. The current device lifecycle could be used by setting this field to 0x0 . |
| Derived key ID | 32 | Derived key ID attribute. <ul style="list-style-type: none"> 0x0 to let EdgeLock secure enclave Firmware choose the key identifier (supported by all Persistence level indicator). Wanted key identifier of the derived key (only supported by Persistent and Permanent Persistence level indicator). In both cases, the key identifier value must respect the range defined. |

Table 6. Key exchange input signed content payload description...continued

| Field | Size (in bits) | Description |
|--------------------------------------|----------------|--|
| Private key ID | 32 | Identifier in the EdgeLock secure enclave Firmware key storage of the private key to use with the peer public key during the key agreement process. |
| Input peer public key digest word #0 | 32 | First 32-bits word (in little endian) of the input peer public key digest buffer. The algorithm used to generate the digest must be SHA256 . |
| ... | 32 | Intermediate 32-bits words (in little endian) of the input peer public key digest buffer. The algorithm used to generate the digest must be SHA256 . |
| Input peer public key digest word #7 | 32 | Last 32-bits word (in little endian) of the input peer public key digest buffer. The algorithm used to generate the digest must be SHA256 . |
| Input user fixed info digest word #0 | 32 | First 32-bits word (in little endian) of the input user fixed info digest buffer. The algorithm used to generate the digest must be SHA256 . <i>Note: If the input user fixed info is not used (set to NULL in command API), this digest value is ignored.</i> |
| ... | 32 | Intermediate 32-bits words (in little endian) of the input user fixed info digest buffer. The algorithm used to generate the digest must be SHA256 . <i>Note: If the input user fixed info is not used (set to NULL in command API), this digest value is ignored.</i> |
| Input user fixed info digest word #7 | 32 | Last 32-bits word (in little endian) of the input user fixed info digest buffer. The algorithm used to generate the digest must be SHA256 . <i>Note: If the input user fixed info is not used (set to NULL in command API), this digest value is ignored.</i> |

3.20.2.2.2 TLS 1.2

Table 7. Key exchange input unsigned content TLS 1.2 Master Secret

| Field | Size (in bits) | Description |
|-------------------|----------------|---|
| Version | 16 | This structure version, only 0x1 is supported. |
| Reserved | 16 | Reserved bits. |
| Key store ID | 32 | Key store ID where to store the Master Secret. It must be the key store ID related to the key management handle set in the command API. This is not used and ignored when Master Secret is exported and not stored in the EdgeLock secure enclave storage. |
| TLS 1.2 algorithm | 32 | Algorithm used by the key exchange process, Master Secret values. |
| Private key ID | 32 | Identifier in the EdgeLock secure enclave Firmware key storage of the private key to use with the peer public key during the premaster secret generation. |
| Derived key type | 16 | Ignored by the EdgeLock secure enclave Firmware, any value can be set. |

Table 7. Key exchange input unsigned content TLS 1.2 Master Secret...continued

| Field | Size (in bits) | Description |
|---------------------------------|----------------|---|
| | | When Master Secret is stored in the EdgeLock secure enclave key storage, it is set to DERIVE by the EdgeLock secure enclave Firmware. |
| Derived key security size bits | 16 | Ignored by the EdgeLock secure enclave Firmware as hardcoded to 384 . |
| Derived key lifetime | 32 | Ignored by the EdgeLock secure enclave Firmware, any value can be set. When Master Secret is stored in EdgeLock secure enclave key storage, it is set to VOLATILE by the EdgeLock secure enclave Firmware. |
| Derived key usage | 32 | Ignored by the EdgeLock secure enclave Firmware, any value can be set. When Master Secret is stored in the EdgeLock secure enclave key storage. it is set to DERIVE by the EdgeLock secure enclave Firmware. |
| Derived key permitted algorithm | 32 | Ignored by EdgeLock secure enclave Firmware, any value can be set. When Master Secret is stored in the EdgeLock secure enclave key storage, it is set by the EdgeLock secure enclave Firmware to an internal algorithm value that allows the Master Secret to derive the TLS 1.2 key block and IVs. The hash algorithm is the same as the one set in the <code>TLS 1.2 algorithm</code> field. |
| Derived key lifecycle | 32 | Ignored by the EdgeLock secure enclave Firmware, any value can be set. When Master Secret is stored in the EdgeLock secure enclave key storage, it is set to current device lifecycle by the EdgeLock secure enclave Firmware. |
| Derived key ID | 32 | Ignored by the EdgeLock secure enclave Firmware, any value can be set. |
| Reserved | 32 | Reserved bits. |

Table 8. Key exchange input unsigned content TLS 1.2 Key Block

| Field | Size (in bits) | Description |
|-------------------|----------------|--|
| Version | 16 | This structure version, only 0x1 is supported. |
| Reserved | 16 | Reserved bits. |
| Key store ID | 32 | Key store ID where to store the derived keys. It must be the key store ID related to the key management handle set in the command API. This is not used and ignored when derived keys are exported and not stored in the EdgeLock secure enclave storage. |
| TLS 1.2 algorithm | 32 | Algorithm used by the key exchange process, Key Block values. |
| Private key ID | 32 | Identifier in the EdgeLock secure enclave Firmware key storage of the Master Secret to use to derive the keys. Could be set to 0x0 if Master Secret is used as input plaintext buffer. |

Table 8. Key exchange input unsigned content TLS 1.2 Key Block...continued

| Field | Size (in bits) | Description |
|------------------------------------|----------------|--|
| Derived key type | 16 | Ignored by the EdgeLock secure enclave Firmware, any value can be set. |
| Ciphersuite key security size bits | 16 | The client/server write key security size in bits. |
| Derived key lifetime | 32 | Ignored by the EdgeLock secure enclave Firmware, any value can be set. When keys are stored in the EdgeLock secure enclave key storage, it is set to VOLATILE by the EdgeLock secure enclave Firmware. |
| Derived key usage | 32 | Ignored by the EdgeLock secure enclave Firmware, any value can be set. |
| Ciphersuite algorithm | 32 | The permitted algorithm assigned to client/server write key that defined the ciphersuite. |
| Derived key lifecycle | 32 | Ignored by the EdgeLock secure enclave Firmware, any value can be set. When keys are stored in the EdgeLock secure enclave key storage, it is set to current device lifecycle by the EdgeLock secure enclave Firmware. |
| Derived key ID | 32 | Ignored by the EdgeLock secure enclave Firmware, any value can be set. |
| Master Secret length | 32 | Master Secret size (in bytes) if used as input plaintext buffer. Must be set to 48 in that case. Must be set to 0 if used as stored secret. |
| Master Secret buffer | 384 | (Optional) Master Secret value, in big endian format, if Master Secret length is set. |

Table 9. Key exchange input unsigned content TLS 1.2 IV

| Field | Size (in bits) | Description |
|------------------------------------|----------------|---|
| Version | 16 | This structure version, only 0x1 is supported. |
| Reserved | 16 | Reserved bits. |
| Key store ID | 32 | Ignored by the EdgeLock secure enclave Firmware, any value can be set. |
| TLS 1.2 algorithm | 32 | Algorithm used by the key exchange process, IV values. |
| Private key ID | 32 | Identifier in the EdgeLock secure enclave Firmware key storage of the Master Secret to use to derive the IVs. Could be set to 0x0 if Master Secret is used as input plaintext buffer. |
| Derived key type | 16 | Ignored by the EdgeLock secure enclave Firmware, any value can be set |
| Ciphersuite key security size bits | 16 | The client/server write key security size in bits. |
| Derived key lifetime | 32 | Ignored by the EdgeLock secure enclave Firmware, any value can be set. |

Table 9. Key exchange input unsigned content TLS 1.2 IV...continued

| Field | Size (in bits) | Description |
|-----------------------|----------------|---|
| Derived key usage | 32 | Ignored by the EdgeLock secure enclave Firmware, any value can be set. |
| Ciphersuite algorithm | 32 | The permitted algorithm assigned to the client/server write key that defined the ciphersuite. |
| Derived key lifecycle | 32 | Ignored by the EdgeLock secure enclave Firmware, any value can be set. |
| Derived key ID | 32 | Ignored by the EdgeLock secure enclave Firmware, any value can be set. |
| Master Secret length | 32 | Master Secret size (in bytes) if used as input plaintext buffer. Must be set to EdgeLock® secure enclave Firmware in that case. Must be set to 0 if used as stored secret. |
| Master Secret buffer | 384 | (Optional) Master Secret value, in big endian format, if Master Secret length is set. |

3.20.2.2.3 TLS 1.3

Table 10. Key exchange input unsigned content TLS 1.3

| Field | Size (in bits) | Description |
|---------------------------------|----------------|---|
| Version | 16 | This structure version, only 0x1 is supported. |
| Reserved | 16 | Reserved bits. |
| Key store ID | 32 | Key store ID where to store the derived key. It must be the key store ID related to the key management handle set in the command API. This is not used and ignored when derived key is exported and not stored in EdgeLock secure enclave storage. |
| TLS 1.3 algorithm | 32 | Algorithm used by the key exchange process. |
| Private key ID | 32 | Identifier in the EdgeLock secure enclave Firmware key storage of the private key to use with the peer public key during the key agreement process. Not used and ignored for Early Secret algorithm. |
| Derived key type | 16 | Ignored by the EdgeLock secure enclave Firmware, any value can be set. When key is stored in the EdgeLock secure enclave key storage, it is set to DERIVE by the EdgeLock secure enclave Firmware. |
| Derived key security size bits | 16 | Derived key security size in bits attribute. |
| Derived key lifetime | 32 | Ignored by the EdgeLock secure enclave Firmware, any value can be set. When key is stored in EdgeLock secure enclave key storage, it is set to VOLATILE by the EdgeLock secure enclave Firmware. |
| Derived key usage | 32 | Ignored by the EdgeLock secure enclave Firmware, any value can be set. When key is stored in EdgeLock secure enclave key storage, it is set to DERIVE by the EdgeLock secure enclave Firmware. |
| Derived key permitted algorithm | 32 | Derived key permitted algorithm attribute. Must be a "HKDF Expand SHAx" value. |

Table 10. Key exchange input unsigned content TLS 1.3...continued

| Field | Size (in bits) | Description |
|-----------------------|----------------|--|
| | | Not used and ignored if the derived key is exported. |
| Derived key lifecycle | 32 | Ignored by the EdgeLock secure enclave Firmware, any value can be set. When key is stored in EdgeLock secure enclave key storage, it is set to current device lifecycle by the EdgeLock secure enclave Firmware. |
| Derived key ID | 32 | Ignored by the EdgeLock secure enclave Firmware, any value can be set |
| PSK length | 32 | (Optional) PSK length in bytes . Can be set to 0 if not used. |
| PSK buffer | PSK length * 8 | (Optional) PSK value if PSK length is set. |

3.20.2.2.4 HMAC-based Key Derivation function

Table 11. Key exchange input unsigned content HKDF Extract

| Field | Size (in bits) | Description |
|---------------------------------|----------------|--|
| Version | 16 | This structure version, only 0x1 is supported. |
| Reserved | 16 | Reserved bits. |
| Key store ID | 32 | Key store ID where to store the derived key. It must be the key store ID related to the key management handle set in the command API. This is not used and ignored when derived key is exported and not stored in the EdgeLock secure enclave storage. |
| HKDF Extract algorithm | 32 | Algorithm used by the key exchange process, HKDF Extract . |
| Private key ID | 32 | Identifier in the EdgeLock secure enclave Firmware key storage of the private key to use during the key derivation process. It is used as private key for ECDH operation. The ECDH result is used as IKM for the HKDF Extract operation. Could be set to 0x0 if IKM is used as the input plaintext buffer. |
| Derived key type | 16 | Ignored by the EdgeLock secure enclave Firmware, any value can be set. When key is stored in EdgeLock secure enclave key storage, it is set to DERIVE by the EdgeLock secure enclave Firmware. |
| Derived key security size bits | 16 | Ignored by the EdgeLock secure enclave Firmware, any value can be set. it is set, by the EdgeLock secure enclave Firmware, to hash algorithm length used by this operation. |
| Derived key lifetime | 32 | Derived key lifetime attribute. Not used and ignored if derived key is exported. |
| Derived key usage | 32 | Ignored by the EdgeLock secure enclave Firmware, any value can be set. When key is stored in EdgeLock secure enclave key storage, it is set to DERIVE by the EdgeLock secure enclave Firmware. |
| Derived key permitted algorithm | 32 | Ignored by the EdgeLock secure enclave Firmware, any value can be set. |

Table 11. Key exchange input unsigned content HKDF Extract...continued

| Field | Size (in bits) | Description |
|-----------------------|-----------------|--|
| | | When key is stored in EdgeLock secure enclave key storage, it is set to HKDF_Expand (Hash algorithm used in current Extract operation) by the EdgeLock secure enclave Firmware. |
| Derived key lifecycle | 32 | Derived key lifecycle attribute. Current device lifecycle could be used by setting this field to 0x0 . Not used and ignored if the derived key is exported. |
| Derived key ID | 32 | Derived key ID attribute: <ul style="list-style-type: none"> • 0x0 to let the EdgeLock secure enclave Firmware choose the key identifier (supported by all Persistence level indicator). • Wanted key identifier of the derived key (only supported by Persistent and Permanent Persistence level indicator). In both cases, the key identifier value must respect the range defined. Not used and ignored if derived key is exported. |
| Salt length | 32 | Salt length in bytes . It can be set to 0x0 . In that case a zero's string of hash algorithm (used by this operation) length is used. |
| Salt buffer | Salt length * 8 | Salt value if the Salt length is set. |

Table 12. Key exchange input unsigned content HKDF Expand

| Field | Size (in bits) | Description |
|---------------------------------|----------------|---|
| Version | 16 | This structure version, only 0x1 is supported. |
| Reserved | 16 | Reserved bits. |
| Key store ID | 32 | Key store ID where to store the derived key. It must be the key store ID related to the key management handle set in the command API. This is not used and ignored when the derived key is exported and not stored in EdgeLock secure enclave storage. |
| HKDF Expand algorithm | 32 | Algorithm used by the key exchange process, HKDF Expand . |
| PRK ID | 32 | Identifier in the EdgeLock secure enclave Firmware key storage of the PRK to use during the key derivation process. Could be set to 0x0 if PRK is used as input plaintext buffer. |
| Derived key type | 16 | Derived key type attribute (Symmetric). Not used and ignored if the derived key is exported. |
| Derived key security size bits | 16 | Derived key security size in bits attribute, (Symmetric). |
| Derived key lifetime | 32 | Derived key lifetime attribute. Not used and ignored if derived key is exported. |
| Derived key usage | 32 | Derived key usage attribute. Not used and ignored if derived key is exported. |
| Derived key permitted algorithm | 32 | Derived key permitted algorithm attribute. Not used and ignored if derived key is exported. |

Table 12. Key exchange input unsigned content HKDF Expand...continued

| Field | Size (in bits) | Description |
|-----------------------|--------------------------|---|
| Derived key lifecycle | 32 | Derived key lifecycle attribute. Current device lifecycle could be used by setting this field to 0x0 . Not used and ignored if derived key is exported. |
| Derived key ID | 32 | Derived key ID attribute: <ul style="list-style-type: none"> • 0x0 to let EdgeLock secure enclave Firmware choose the key identifier (supported by all Persistence level indicators). • Wanted key identifier of the derived key (only supported by Persistent and Permanent Persistence level indicators). In both cases, the key identifier value must respect the range defined. Not used and ignored if derived key is exported. |
| Plaintext PRK length | 32 | PRK length in bytes. Must be set to 0x0 if a PRK ID is used. |
| Plaintext PRK buffer | Plaintext PRK length * 8 | Plaintext PRK value if the Plaintext PRK length is set. |

Table 13. Key exchange input unsigned content HKDF

| Field | Size (in bits) | Description |
|---------------------------------|----------------|--|
| Version | 16 | This structure version, only 0x1 is supported. |
| Reserved | 16 | Reserved bits. |
| Key store ID | 32 | Key store ID where to store the derived key. It must be the key store ID related to the key management handle set in the command API. This is not used and ignored when derived key is exported and not stored in EdgeLock secure enclave storage. |
| HKDF algorithm | 32 | HKDF Algorithm used by the key exchange process. |
| Private key ID | 32 | Identifier, in the EdgeLock secure enclave Firmware key storage, of the private key to use during the key derivation process. It is used as private key for ECDH operation. ECDH result is used as IKM for HKDF Extract operation. Could be set to 0x0 if IKM is used as the input plaintext buffer. |
| Derived key type | 16 | Derived key type attribute. Not used and ignored if derived key is exported. |
| Derived key security size bits | 16 | Derived key security size in bits attribute. |
| Derived key lifetime | 32 | Derived key lifetime attribute. Not used and ignored if derived key is exported. |
| Derived key usage | 32 | Derived key usage attribute. Not used and ignored if derived key is exported. |
| Derived key permitted algorithm | 32 | Derived key permitted algorithm attribute. Not used and ignored if derived key is exported. |
| Derived key lifecycle | 32 | Derived key lifecycle attribute. Current device lifecycle could be used by setting this field to 0x0 . Not used and ignored if derived key is exported. |
| Derived key ID | 32 | Derived key ID attribute: <ul style="list-style-type: none"> • 0x0 to let EdgeLock secure enclave Firmware choose the key identifier |

Table 13. Key exchange input unsigned content HKDF...continued

| Field | Size (in bits) | Description |
|-------------|-----------------|--|
| | | <ul style="list-style-type: none"> Wanted key identifier of the derived key (only supported by Persistent and Permanent Persistence level indicators). In both cases, the key identifier value must respect the range defined. Not used and ignored if derived key is exported. |
| Salt length | 32 | Salt length in bytes . It can be set to 0x0. In that case a zero's string of hash algorithm (used by this operation) length is used. |
| Salt buffer | Salt length * 8 | Salt value if Salt length is set. |

3.20.3 Macro definition documentation

3.20.3.1 HSM_OP_KEY_EXCHANGE_FLAGS_INPUT_SIGNED_CONTENT

```
#define HSM_OP_KEY_EXCHANGE_FLAGS_INPUT_SIGNED_CONTENT
((hsm_op_key_exchange_flags_t) (0u << 0))
```

Input signed content.

3.20.3.2 HSM_OP_KEY_EXCHANGE_FLAGS_INPUT_PLAINTEXT_CONTENT

```
#define HSM_OP_KEY_EXCHANGE_FLAGS_INPUT_PLAINTEXT_CONTENT
((hsm_op_key_exchange_flags_t) (1u << 0))
```

Input plaintext content.

3.20.3.3 HSM_OP_KEY_EXCHANGE_FLAGS_RETURN_KEY_IDS

```
#define HSM_OP_KEY_EXCHANGE_FLAGS_RETURN_KEY_IDS ((hsm_op_key_exchange_flags_t)
(1u << 1))
```

Return the multiple derived key IDs as output buffer. Flag only available for TLS 1.2 Key Block operation.

3.20.3.4 HSM_OP_KEY_EXCHANGE_FLAGS_RETURN_OUTPUT

```
#define HSM_OP_KEY_EXCHANGE_FLAGS_RETURN_OUTPUT ((hsm_op_key_exchange_flags_t)
(1u << 2))
```

Return the operation's output as buffer (in plaintext).

3.20.3.5 HSM_OP_KEY_EXCHANGE_FLAGS_SALT_ZERO

```
#define HSM_OP_KEY_EXCHANGE_FLAGS_SALT_ZERO ((hsm_op_key_exchange_flags_t) (0u
<< 0))
```

Use zeros salt.

3.20.3.6 HSM_OP_KEY_EXCHANGE_FLAGS_SALT_PEER_PUBKEY_HASH

```
#define HSM_OP_KEY_EXCHANGE_FLAGS_SALT_PEER_PUBKEY_HASH
((hsm_op_key_exchange_flags_t) (1u << 0))
```

Use peer public key hash salt.

3.20.3.7 HSM_OP_KEY_EXCHANGE_FLAGS_MONOTONIC

```
#define HSM_OP_KEY_EXCHANGE_FLAGS_MONOTONIC ((hsm_op_key_exchange_flags_t) (1u
<< 5))
```

When used in conjunction with the STRICT flag, the request is completed only when the monotonic counter has been updated.

3.20.3.8 HSM_OP_KEY_EXCHANGE_FLAGS_STRICT_OPERATION

```
#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 is applicable for the persistent key only.

Note: In the latest ELE FW API guide, "STRICT" has been replaced with "SYNC".

3.20.4 Typedef documentation

3.20.4.1 hsm_op_key_exchange_flags_t

```
typedef uint16_t hsm_op_key_exchange_flags_t
```

Bitmap specifying the key exchange operation properties.

3.20.5 Enumeration type documentation

3.20.5.1 hsm_op_key_exchange_algo_t

```
enum hsm_op_key_exchange_algo_t
```

Enum describing Key Exchange algorithms supported.

Table 14. Enumerators

| | |
|--|---|
| HSM_KEY_EX_ECDH_HKDF_SHA256_KEY_IMPORT | ECDH HKDF SHA256 KEY IMPORT. |
| HSM_KEY_EX_ECDH_HKDF_SHA384_KEY_IMPORT | ECDH HKDF SHA384 KEY IMPORT. |
| HSM_KEY_EX_ECDH_HKDF_SHA256 | ECDH HKDF SHA256. |
| HSM_KEY_EX_ECDH_HKDF_SHA384 | ECDH HKDF SHA384. |
| HSM_KEY_EXCHANGE_ECDH_HKDF_SHA256 | ECDH HKDF SHA256. This will be deprecated soon. |
| HSM_KEY_EXCHANGE_ECDH_HKDF_SHA384 | ECDH HKDF SHA384. This will be deprecated soon. |

3.20.5.2 hsm_op_key_derivation_algo_t

```
enum hsm_op_key_derivation_algo_t
```

Enum describing the Key Derivation algorithms supported.

Table 15. Enumerators

| | |
|--|-----------------------------|
| HSM_KEY_DERIVATION_HKDF_SHA256 | HKDF SHA256 (HMAC two-step) |
| HSM_KEY_DERIVATION_HKDF_SHA384 | HKDF SHA384 (HMAC two-step) |
| HSM_KEY_DERIVATION_HKDF_EXTRACT_SHA256 | HKDF Extract SHA256 |
| HSM_KEY_DERIVATION_HKDF_EXTRACT_SHA384 | HKDF Extract SHA384 |
| HSM_KEY_DERIVATION_HKDF_EXPAND_SHA256 | HKDF Expand SHA256 |
| HSM_KEY_DERIVATION_HKDF_EXPAND_SHA384 | HKDF Expand SHA384 |

3.20.5.3 hsm_op_key_derivation_tls1_3_algo_t

```
enum hsm_op_key_derivation_tls1_3_algo_t
```

Enum describing the TLS 1.3 Key Derivation algorithms supported.

Table 16. Enumerators

| | |
|---|---------------------------------|
| HSM_KEY_DERIVATION_TLS1_3_EARLY_SECRET_SHA256 | TLS 1.3 Early Secret SHA256 |
| HSM_KEY_DERIVATION_TLS1_3_EARLY_SECRET_SHA384 | TLS 1.3 Early Secret SHA384 |
| HSM_KEY_DERIVATION_TLS1_3_HANDSHAKE_SECRET_SHA256 | TLS 1.3 Handshake Secret SHA256 |
| HSM_KEY_DERIVATION_TLS1_3_HANDSHAKE_SECRET_SHA384 | TLS 1.3 Handshake Secret SHA384 |
| HSM_KEY_DERIVATION_TLS1_3_MASTER_SECRET_SHA256 | TLS 1.3 Master Secret SHA256 |
| HSM_KEY_DERIVATION_TLS1_3_MASTER_SECRET_SHA384 | TLS 1.3 Master Secret SHA384 |
| HSM_KEY_DERIVATION_TLS1_3_KEYING_MATERIAL_SHA256 | TLS 1.3 Keying Material SHA256 |
| HSM_KEY_DERIVATION_TLS1_3_KEYING_MATERIAL_SHA384 | TLS 1.3 Keying Material SHA384 |
| HSM_KEY_DERIVATION_TLS1_3_IV_SHA256 | TLS 1.3 IV SHA256 |
| HSM_KEY_DERIVATION_TLS1_3_IV_SHA384 | TLS 1.3 IV SHA384 |

3.20.5.4 hsm_op_key_derivation_tls1_2_algo_t

```
enum hsm_op_key_derivation_tls1_2_algo_t
```

Enum describing TLS 1.2 Key Derivation algorithms supported.

Table 17. Enumerators

| | |
|--|------------------------------|
| HSM_KEY_DERIVATION_TLS1_2_MASTER_SECRET_SHA256 | TLS 1.2 Master Secret SHA256 |
| HSM_KEY_DERIVATION_TLS1_2_MASTER_SECRET_SHA384 | TLS 1.2 Master Secret SHA384 |
| HSM_KEY_DERIVATION_TLS1_2_KEY_BLOCK_SHA256 | TLS 1.2 Key Block SHA256 |
| HSM_KEY_DERIVATION_TLS1_2_KEY_BLOCK_SHA384 | TLS 1.2 Key Block SHA384 |
| HSM_KEY_DERIVATION_TLS1_2_VERIFY_DATA_SHA256 | TLS 1.2 VERIFY DATA SHA256 |
| HSM_KEY_DERIVATION_TLS1_2_VERIFY_DATA_SHA384 | TLS 1.2 VERIFY DATA SHA384 |
| HSM_KEY_DERIVATION_TLS1_2_IV_SHA256 | TLS 1.2 IV SHA256 |
| HSM_KEY_DERIVATION_TLS1_2_IV_SHA384 | TLS 1.2 IV SHA384 |

3.20.6 Function documentation

3.20.6.1 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.

This API supports the following use cases:

- Combined key agreement and key derivation operation
 - This option is used to perform combined key agreement and key derivation op.
 - Only symmetric keys can be created with this operation.
- TLS 1.2

This option is used to derive TLS 1.2 secrets and IVs:

 - Master Secret
 - Client/Server write MAC keys
 - Client/Server write keys
 - Client/Server IVs
- TLS 1.3

This option is used to derive:

 - Early Secrets
 - Handshake Secrets
 - Master Secrets
- HMAC-based Key Derivation Function

This option is used to derive key using HKDF algorithm. It can be either the entire process or one specific step.

 1. Extract
 2. Expand

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

Table 18. Parameters

| | |
|---------------------------|---|
| <i>key_management_hdl</i> | Handle identifying the key store management service flow. |
|---------------------------|---|

Table 18. Parameters...continued

| | |
|-------------------|---|
| <code>args</code> | Pointer to the structure containing the function arguments. |
|-------------------|---|

Returns

Error code

3.21 Public key recovery

Public Key Recovery is also known as Public Key Exportation, in PSA compliant APIs. The naming here has been kept unchanged, for backward compatibility and Non-PSA compliant APIs.

Data Structures

- struct [op_pub_key_recovery_args_t](#)

Functions

- [hsm_err_t hsm_pub_key_recovery](#) ([hsm_hdl_t](#) key_store_hdl, [op_pub_key_recovery_args_t](#) *args)

3.21.1 Detailed description

Public Key Recovery is now known as Public Key Exportation, in PSA compliant APIs. The naming here has been kept unchanged, for backward compatibility and Non-PSA compliant APIs.

3.21.2 Data structure documentation

3.21.2.1 struct op_pub_key_recovery_args_t

Structure detailing the public key recovery operation member arguments.

Data Fields

| | | |
|------------------------|-------------------------------|--|
| <code>uint32_t</code> | <code>key_identifier</code> | Pointer to the identifier of the key to be used for the operation. |
| <code>uint8_t *</code> | <code>out_key</code> | Pointer to the output area where the generated public key must be written. |
| <code>uint16_t</code> | <code>out_key_size</code> | Length in bytes of the output key. |
| <code>uint16_t</code> | <code>exp_out_key_size</code> | Expected output key buffer size, valid in case of <code>HSM_OUT_TOO_SMALL</code> (0x1D) resp code. |

3.21.3 Function documentation

3.21.3.1 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.

Users can call this function only after opening a key store.

Parameters

| | |
|----------------------------|---|
| <code>key_store_hdl</code> | Handle identifying the current key store. |
|----------------------------|---|

Parameters...continued

| | |
|------|---|
| args | Pointer to the structure containing the function arguments. |
|------|---|

Returns

Error code

3.22 Key store

User must open a key store service flow to perform the following operations.

Data Structures

- struct [open_svc_key_store_args_t](#)
- struct [op_key_store_reprov_en_args_t](#)

Macros

- #define HSM_SVC_KEY_STORE_FLAGS_LOAD ((hsm_svc_key_store_flags_t)(0u << 0))
It must be specified to load a previously created key store.
- #define HSM_SVC_KEY_STORE_FLAGS_CREATE ((hsm_svc_key_store_flags_t)(1u << 0))
- #define HSM_SVC_KEY_STORE_FLAGS_SHARED ((hsm_svc_key_store_flags_t)(1u << 2))
- #define HSM_SVC_KEY_STORE_FLAGS_SET_MAC_LEN ((hsm_svc_key_store_flags_t)(1u << 3))
- #define HSM_SVC_KEY_STORE_FLAGS_MONOTONIC ((hsm_svc_key_store_flags_t)(1u << 5))
- #define HSM_SVC_KEY_STORE_FLAGS_STRICT_OPERATION ((hsm_svc_key_store_flags_t)(1u << 7))

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)
- [hsm_err_t hsm_key_store_reprov_en](#) (hsm_hdl_t session_hdl, [op_key_store_reprov_en_args_t](#) *args)

3.22.1 Detailed description

Users must open a key store service flow 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.

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

3.22.2 Data structure documentation

3.22.2.1 struct open_svc_key_store_args_t

Structure specifying the open key store service member arguments.

Data Fields

| | | |
|----------|---------------|--|
| uint32_t | key_store_hdl | Handle identifying the key store service flow. |
|----------|---------------|--|

Data Fields...continued

| | | |
|----------|----------------------|--|
| uint32_t | key_store_identifier | User defined ID identifying the key store. Only one key store service can be opened on a given <code>key_store_identifier</code> . |
| uint32_t | authentication_nonce | User defined nonce used as authentication proof for accessing the key store. |
| uint8_t | flags | Bitmap specifying the services properties. |

3.22.2.2 struct op_key_store_reprov_en_args_t

Structure describing the key store reprovisioning enable operation arguments.

Data Fields

| | | |
|-----------|-----------------|---------------------------------------|
| uint8_t * | signed_message | Signed content payload. |
| uint32_t | signed_msg_size | Signed content payload size in bytes. |

3.22.3 Macro definition documentation

3.22.3.1 HSM_SVC_KEY_STORE_FLAGS_CREATE

```
#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 is stored in the NVM only if the STRICT OPERATION flag is set.

3.22.3.2 HSM_SVC_KEY_STORE_FLAGS_SHARED

```
#define HSM_SVC_KEY_STORE_FLAGS_SHARED ((hsm_svc_key_store_flags_t) (1u << 2))
```

It must be specified to create or load a shared key store.

3.22.3.3 HSM_SVC_KEY_STORE_FLAGS_SET_MAC_LEN

```
#define HSM_SVC_KEY_STORE_FLAGS_SET_MAC_LEN ((hsm_svc_key_store_flags_t) (1u << 3))
```

If set, the minimum MAC length specified in the `min_mac_length` field is stored in the key store when creating the key store. It must only be set at key store creation.

3.22.3.4 HSM_SVC_KEY_STORE_FLAGS_MONOTONIC

```
#define HSM_SVC_KEY_STORE_FLAGS_MONOTONIC ((hsm_svc_key_store_flags_t) (1u << 5))
```

When used in conjunction with the STRICT flag, the request is completed only when the monotonic counter has been updated.

3.22.3.5 HSM_SVC_KEY_STORE_FLAGS_STRICT_OPERATION

```
#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 is written in the NVM. This is applicable for CREATE operations only.

3.22.4 Typedef documentation

3.22.4.1 hsm_svc_key_store_flags_t

```
typedef uint8_t hsm_svc_key_store_flags_t
```

Bitmap specifying the open key store service supported attributes.

3.22.5 Function documentation

3.22.5.1 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.

3.22.5.2 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>key_store_hdl</i> | Handle identifying the key store service flow to be closed. |
|----------------------|---|

Returns

Error code.

3.22.5.3 hsm_key_store_reprov_en()

```
hsm_err_t hsm_key_store_reprov_en (
hsm_hdl_t session_hdl,
op_key_store_reprov_en_args_t * args)
```

Table 19. Parameters

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

Returns

Error code.

3.23 Life cycle update

Data Structures

- struct [op_lc_update_msg_args_t](#)

Enumerations

- enum [hsm_lc_new_state_t](#) {
 - HSM_NXP_PROVISIONED_STATE = (1u << 0),
 - HSM_OEM_OPEN_STATE = (1u << 1),
 - HSM_OEM_CLOSE_STATE = (1u << 3),
 - HSM_OEM_FIELD_RET_STATE = (1u << 4),
 - HSM_NXP_FIELD_RET_STATE = (1u << 5),
 - HSM_OEM_LOCKED_STATE = (1u << 7) }

Functions

- [hsm_err_t hsm_lc_update](#) ([hsm_hdl_t](#) session_hdl, [op_lc_update_msg_args_t](#) *args)

3.23.1 Detailed description

None.

3.23.2 Data structure documentation

3.23.2.1 struct op_lc_update_msg_args_t

Structure specifying the life cycle update message arguments.

Data Fields

| | | |
|------------------------------------|--------------|---|
| hsm_lc_new_state_t | new_lc_state | - |
|------------------------------------|--------------|---|

3.23.3 Enumeration type documentation

3.23.3.1 hsm_lc_new_state_t

```
enum hsm\_lc\_new\_state\_t
```

Enum specifying the Life Cycle state.

3.23.4 Function documentation

3.23.4.1 hsm_lc_update()

```
hsm_err_t hsm_lc_update (
hsm_hdl_t session_hdl,
op_lc_update_msg_args_t * args)
```

This API performs the Life Cycle update.

Parameters

| | |
|-------------|---|
| session_hdl | Handle identifying the session handle. |
| args | Pointer to the structure containing the function arguments. |

Returns

Error code

3.24 Configure monotonic counter

Data Structures

- struct op_dev_mono_counter_args_t

Functions

- hsm_err_t hsm_dev_mono_counter (hsm_hdl_t sess_hdl, op_dev_mono_counter_args_t *args)

3.24.1 Detailed description

This API is used to configure the monotonic counter fuse value for V2X HSM, V2X SHE, and ELE HSM. This API can be called only once with the sum of all three fuse word values equal to the available fuse words for the monotonic counter for the SoC.

3.24.2 Data structure documentation

3.24.2.1 struct op_dev_mono_counter_args_t

Structure detailing the device mono counter operation member arguments.

Data Fields

| | | |
|---------|-------------------|---|
| uint8_t | ele_hsm_fuse_word | Count of ELE HSM fuse word. |
| uint8_t | v2x_hsm_fuse_word | Count of V2X HSM fuse word. |
| uint8_t | v2x_she_fuse_word | Count of V2X SHE fuse word. The sum of the three above should not exceed the total available monotonic counter fuse words on the board. |
| uint8_t | enforce_fuse | Set this flag to configure any ELE/V2X counter to zero. |
| uint8_t | warning | Warning returned by the mono counter operation. |

3.24.3 Function documentation

3.24.3.1 hsm_err_t hsm_dev_mono_counter ()

```
hsm_err_t hsm_dev_mono_counter (hsm_hdl_t sess_hdl, op_dev_mono_counter_args_t * args)
```

Perform a mono counter operation. Users can call this function only after having opened the session. This function is used to partition monotonic counter fuse words between HSMs. The configuration can only be issued once per device. If no configuration is present in OTP and a monotonic counter operation is triggered, a warning will be issued and no rollback protection will be applied.

If a valid configuration has already been issued, the written configuration is returned. All available monotonic counter fuse words must be used. If the sum of the configured fuse words is different from the total amount available, the configuration will be considered invalid.

A null configuration (all HSM words at 0) is considered as a read configuration operation and will succeed on a provisioned device. If there is no existing configuration, it will fail.

Table 20. Parameters

| | |
|----------|---|
| sess_hdl | Handle identifying the active session. |
| args | Pointer to the structure containing the function arguments. |

Returns

Error code.

3.25 Boot mode

Data Structures

- struct op_get_boot_mode_args_t
- struct op_set_boot_mode_args_t

Functions

- crrm_err_t crrm_get_boot_mode (cr_rm_hdl_t cr_rm_hdl, op_get_boot_mode_args_t *args)
- crrm_err_t crrm_set_boot_mode (cr_rm_hdl_t cr_rm_hdl, op_set_boot_mode_args_t *args)

3.25.1 Detailed description

CRRM APIs to get and set the boot mode.

3.25.2 Data structure documentation

3.25.2.1 struct op_get_boot_mode_args_t

Structure describing the member arguments for the getting boot mode operation.

Data Fields

| | | |
|---------|-----------|---|
| uint8_t | boot_mode | Stores the boot mode. Indicates the reason for the previous boot: <ul style="list-style-type: none"> • 0x69 = Normal (no CRRM events) • 0x3c = Recovery download (timeout of CRRM watchdog) • 0x5a = Recovery install (recovery image available) |
|---------|-----------|---|

Data Fields...continued

| | | |
|---------|----------|---|
| | | • Other values reserved |
| uint8_t | timer_id | If the current mode was entered due to timer expiry, provides the ID of the timer. Value 0xFF indicates no timer expired. |

3.25.2.2 struct op_set_boot_mode_args_t

Structure describing the member arguments for the setting boot mode operation.

Data Fields

| | | |
|---------|------------------|--|
| uint8_t | new_boot_mode | Stores the new boot mode to be set: <ul style="list-style-type: none"> • 0x69 = Normal • 0x3c = Recovery download • 0x5a = Recovery install • All other values reserved |
| uint8_t | stored_boot_mode | Indicates the stored boot mode (may be different from requested mode if forbidden transition requested): <ul style="list-style-type: none"> • 0x69 = Normal • 0x3c = Recovery download • 0x5a = Recovery install • All other values reserved |

3.25.3 Function documentation

3.25.3.1 crrm_err_t crrm_get_boot_mode ()

```
crrm_err_t crrm_get_boot_mode (crrm_hdl_t crrm_hdl, op_get_boot_mode_args_t * args)
```

Perform the operation of getting boot mode. Users can call this function only after CRRM initiation has been done. Get CRRM handle before calling this API. This API requests the boot mode information from the BBSM. For the case that the BBSM has just been initialized, the boot mode is indicated as normal boot.

Table 21. Parameters

| | |
|----------|---|
| crrm_hdl | Handle identifying the current session. |
| args | Pointer to the structure containing the function arguments. |

Returns

Error code.

3.25.3.2 crrm_err_t crrm_set_boot_mode ()

```
crrm_err_t crrm_set_boot_mode (crrm_hdl_t crrm_hdl, op_set_boot_mode_args_t * args)
```

Perform the operation of setting boot mode. Users can call this function only after CRRM initiation has been done. Get CRRM handle before calling this API.

This API sets the boot mode information in the BBSM. It is used by the recovery OS to request a mode transition. Only specific transitions are accepted, and others are ignored. Accepted transitions are:

- Recovery download -> recovery install (after recovery image successfully downloaded)
- Recovery install -> normal (after recovery image successfully installed)
- Recovery install -> recovery download (encountered error in recovery image installation)

Only one transition can be requested per boot cycle.

Table 22. Parameters

| | |
|-----------------|---|
| <i>crrm_hdl</i> | Handle identifying the current session. |
| <i>args</i> | Pointer to the structure containing the function arguments. |

Returns

Error code.

3.26 Get AWDT status

Data Structures

- struct AWDT_timer
- struct op_get_AWDT_status_args_t

Macros

- #define TIMER_INFO_SIZE 16
- #define AWDT_TIMER_STATUS_SIZE sizeof(AWDT_timer)

Functions

- crrm_err_t crrm_get_AWDT_status (crrm_hdl_t crrm_hdl, op_get_AWDT_status_args_t *args)

3.26.1 Detailed description

CRRM API to retrieve information about the current status of the AWDT timer.

3.26.2 Data structure documentation

3.26.2.1 struct AWDT_timer

Structure describing the AWDT timer status parameters.

Table 23. Data Fields

| | | |
|----------|------------------|--|
| uint8_t | initialized | 0 = not initialized, 1 = initialized |
| uint8_t | running | 0 = not running, 1 = running |
| uint16_t | BBSM_running | 0 = not running, 1 = running |
| uint32_t | curr_timer_value | Current timer value in ms |
| uint32_t | safety_period | Safety period in ms |
| uint32_t | max_timer_value | Maximum timer value in ms |
| uint32_t | BBSM_timer_value | Stored timer value in ms |
| uint16_t | timer_info_len | Length of timer information |
| uint8_t | timer_info[] | Variable-length timer information data |

3.26.2.2 struct op_get_AWDT_status_args_t

Structure describing the member arguments for the getting status operation.

Table 24. Data Fields

| | | |
|--------------|-------------|---|
| uint8_t | timer_id | Timer ID |
| uint8_t * | Addr | AWDT status report address |
| uint32_t | input_size | AWDT status report size |
| uint32_t | output_size | AWDT status report size provided by the FW |
| AWDT_timer * | status | Status of the timer with ID <code>timer_id</code> |

3.26.3 Function documentation

3.26.3.1 crrm_err_t crrm_get_AWDT_status ()

```
crrm_err_t crrm_get_AWDT_status (crrm_hdl_t crrm_hdl, op_get_AWDT_status_args_t * args)
```

Perform the operation of getting AWDT status. Users can call this function only after CRRM initiation has been done. Get CRRM handle before calling this API.

This API request a status report for a given timer. The status data is output to the memory address provided. If the buffer to copy this data is too small, no data will be copied.

Table 25. Parameters

| | |
|-----------------|---|
| <i>crrm_hdl</i> | Handle identifying the current session. |
| <i>args</i> | Pointer to the structure containing the function arguments. |

Returns

Error code.

3.27 Get nonce

Data Structures

- struct nonce_info
- struct op_get_nonce_args_t

Macros

- #define NONCE_DATA_SIZE 32

Functions

- crrm_err_t crrm_get_nonce (crrm_hdl_t crrm_hdl, op_get_nonce_args_t *args)

3.27.1 Detailed description

CRRM API to get nonce information for the AWDT timer.

3.27.2 Data structure documentation

3.27.2.1 struct nonce_info

Structure describing the nonce information parameters.

Table 26. Data Fields

| | | |
|---------|-----------------------------|--|
| uint8_t | nonce_data[NONCE_DATA_SIZE] | Nonce data |
| uint8_t | timer_info[] | Variable-length timer information data |

3.27.2.2 struct op_get_nonce_args_t

Structure describing the member arguments for the getting nonce operation.

Table 27. Data Fields

| | | |
|--------------|-------------|--|
| uint8_t | timer_id | Timer ID |
| uint8_t * | nonce_addr | Nonce information address |
| uint32_t | nonce_size | Nonce information input size |
| uint32_t | output_size | Nonce information size provided by FW |
| nonce_info * | Info | Nonce information (nonce data + timer information) |

3.27.3 Function documentation

3.27.3.1 crrm_err_t crrm_get_nonce ()

```
cr_rm_err_t cr_rm_get_nonce (cr_rm_hdl_t cr_rm_hdl, op_get_nonce_args_t * args)
```

Perform the operation of getting nonce. Users can call this function only after CRRM initiation has been done. Get CRRM handle before calling this API.

This API request a new nonce value for a given timer. This nonce value invalidates any previous nonce values for the given timer. The nonce plus programmed timer information will be output.

If the buffer to copy this data is too small, no data will be copied, but the nonce will still be updated inside the Sentinel.

Table 28. Parameters

| | |
|-----------|---|
| cr_rm_hdl | Handle identifying the current session. |
| args | Pointer to the structure containing the function arguments. |

Returns

Error code.

3.28 Handle

Typedefs

- typedef uint32_t cr_rm_hdl_t
- typedef hsm_err_t cr_rm_err_t

Functions

- `crmm_hdl_t get_session_handle (void)`

3.28.1 Detailed description

First API to be called before calling any other CRM API. This API returns the handle to be used in calling the CRRM API.

3.28.2 Typedef documentation

3.28.2.1 typedef `uint32_t crmm_hdl_t`

Defines the CRRM handle type.

3.28.2.2 typedef `hsm_err_t crmm_err_t`

Defines the CRRM error type.

3.28.3 Function documentation

3.28.3.1 `crmm_hdl_t get_session_handle (void)`

Returns the CRRM handle to be used while requesting CRRM services.

Check if you have received a valid (non-zero) session handle before continuing.

This is the first API to be called.

Returns

CRRM handle.

3.29 Refresh AWDT

Data Structures

- struct `signature_data`
- struct `request_data`
- struct `op_refresh_AWDT_args_t`

Macros

- `#define NONCE_DATA_SIZE 32`

Functions

- `crmm_err_t crmm_refresh_AWDT (crmm_hdl_t crmm_hdl, op_refresh_AWDT_args_t *args)`

3.29.1 Detailed description

CRRM API to refresh the timeout value of the AWDT timer for the provided timer ID.

3.29.2 Data structure documentation

3.29.2.1 struct `signature_data`

Structure describing the signature data parameters to be signed.

Table 29. Data Fields

| | | |
|----------|------------------------------|--|
| uint8_t | timer_id | Timer ID |
| uint8_t | pub_key_index | Public key index |
| uint16_t | reserved | Reserved value |
| uint32_t | new_timer_value | New value for timer |
| uint8_t | nonce_data [NONCE_DATA_SIZE] | Nonce data |
| uint8_t | timer_info [] | Variable-length timer information data |

3.29.2.2 struct request_data

Structure describing the request data parameters.

Table 30. Data Fields

| | | |
|----------|------------------|--|
| uint32_t | new_timer_value | New value for timer |
| uint16_t | signature_length | Length of the signature field |
| uint8_t | signature [] | Signature calculated on the signature data |

3.29.2.3 struct op_refresh_AWDT_args_t

Structure describing the refresh AWDT operation member arguments.

Table 31. Data Fields

| | | |
|----------|-----------------|------------------------------|
| uint8_t | timer_id | Timer ID |
| uint8_t | pub_key_index | Public key index |
| uint8_t* | req_data_addr | Nonce information address |
| uint32_t | req_data_size | Nonce information input size |
| uint32_t | new_timer_value | New value for timer |

3.29.3 Function documentation

3.29.3.1 crm_err_t crm_refresh_AWDT ()

```
crm_err_t crm_refresh_AWDT (crm_hdl_t crm_hdl, op_refresh_AWDT_args_t * args)
```

Perform the operation of refreshing AWDT. Users can call this function only after CRRM initiation has been done. Get CRRM handle before calling this API.

This API is used to refresh the specified AWDT.

The message contains a signature, which is calculated over the concatenation of the following information:

- Word 1 of the message below (public key index and timer ID)
- The new TimerValue from the request structure
- Current nonce value for the requested timer
- Timer information for the requested timer if present

If the signature is valid, the corresponding AWDT timer is refreshed.

Table 32. Parameters

| | |
|-----------------|---|
| <i>crrm_hdl</i> | Handle identifying the current session. |
| <i>args</i> | Pointer to the structure containing the function arguments. |

Returns

Error code.

3.30 Get event

Data Structures

- struct hsm_event_report_info_t
- struct op_get_event_msg_args_t

Macros

- #define MAX_NUM_EVENT_REPORT 8

Functions

- hsm_err_t hsm_get_event (hsm_hdl_t session_hdl, op_get_event_msg_args_t *args)

3.30.1 Detailed description

None.

3.30.2 Data structure documentation

3.30.2.1 struct hsm_event_report_info_t

Structure describing the event report information parameters.

Table 33. Data Fields

| | | |
|----------|----------------|--|
| uint16_t | event_rsp_code | Response code helps identify the reason for reporting the event. |
| uint8_t | event_cmd_id | Command ID, for which event is reported. |
| uint8_t | event_ipc_id | IPC ID (0x01 = MU RTD, 0x02 = MU APPS), for which event is reported. |

3.30.2.2 struct op_get_event_msg_args_t

Structure specifying the members that will enable reporting the first *n* events from the FW, where *n* is defined by the macro MAX_NUM_EVENT_REPORT.

Table 34. Data Fields

| | | |
|-------------------------|--|---|
| uint16_t | num_event_report | Number of events reported since the last power cycle. |
| hsm_event_report_info_t | hsm_event_report_list [MAX_NUM_EVENT_REPORT] | List of events reported since the last power cycle. |

3.30.3 Function documentation

3.30.3.1 hsm_get_event ()

```
hsm_err_t hsm_get_event (hsm_hdl_t session_hdl, op_get_event_msg_args_t * args)
```

This API reports the first *n* reported events from the FW, where *n* is defined by the macro `MAX_NUM_EVENT_REPORT`.

Table 35. Parameters

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

Returns

Error code.

3.31 Blob Unblob

Data Structures

- struct `op_blob_unblob_msg_args_t`

Macros

- `#define HSM_OP_BLOB_UNBLOB_FLAG_USE_MU_IN_DER_CTX ((hsm_op_key_gen_flags_t)(1u << 0))`
- `#define HSM_OP_BLOB_UNBLOB_FLAG_USE_BOOT_STAGE_IN_DER_CTX ((hsm_op_key_gen_flags_t)(1u << 1))`
- `#define HSM_OP_BLOB_UNBLOB_FLAG_USE_UUID_IN_DER_CTX ((hsm_op_key_gen_flags_t)(1u << 2))`

Typedefs

- `typedef uint8_t hsm_op_blob_unblob_flags_t`

Enumerations

- `enum hsm_op_blob_unblob_mode_t {
HSM_OP_BLOB_UNBLOB_MODE_WRAP = 0x55,
HSM_OP_BLOB_UNBLOB_MODE_UNWRAP = 0xAA, }`

Functions

- `hsm_err_t hsm_blob_unblob (hsm_hdl_t session_hdl, op_blob_unblob_msg_args_t *args)`

3.31.1 Detailed description

None.

3.31.2 Data structure documentation

3.31.2.1 struct `op_blob_unblob_msg_args_t`

Structure specifying the blob-unblob request.

Table 36. Data Fields

| | | |
|-------------------------|----------------------|--|
| <code>uint32_t *</code> | <code>in_addr</code> | Input address where blob or plain-text can be found. |
|-------------------------|----------------------|--|

Table 36. Data Fields...continued

| | | |
|-----------|--------------|--|
| uint32_t* | out_addr | Output address where blob or plain-text can be found. |
| uint32_t* | usr_ctx_addr | Address space, where user input to blob-key derivation context, can be found. |
| uint32_t | in_size | Size in bytes of Input. |
| uint32_t | out_size | Size in bytes of Output. |
| uint32_t | usr_ctx_size | Size in bytes of user-input. Can be zero. |
| uint8_t | flags | Bitmap specifying the operation properties as defined by <code>hsm_op_key_gen_flags_t</code> . |
| uint8_t | mode | Input mode to specify: HSM_OP_BLOB_UNBLOB_MODE_WRAP or HSM_OP_BLOB_UNBLOB_MODE_UNWRAP. |
| uint32_t | exp_out_size | Expected size in bytes of Output. |

3.31.3 Macro definition documentation

3.31.3.1 HSM_OP_BLOB_UNBLOB_FLAG_USE_MU_IN_DER_CTX

```
#define HSM_OP_BLOB_UNBLOB_FLAG_USE_MU_IN_DER_CTX ((hsm_op_key_gen_flags_t)(1u << 0))
```

Use MU Attributes in the blob-key derivation context. Blob is binded to the MU attributes (sec/priv, masternum, did, and MU index).

3.31.3.2 HSM_OP_BLOB_UNBLOB_FLAG_USE_BOOT_STAGE_IN_DER_CTX

```
#define HSM_OP_BLOB_UNBLOB_FLAG_USE_BOOT_STAGE_IN_DER_CTX ((hsm_op_key_gen_flags_t)(1u << 1))
```

Use the boot stage value in the derivation context.

3.31.3.3 HSM_OP_BLOB_UNBLOB_FLAG_USE_UUID_IN_DER_CTX

```
#define HSM_OP_BLOB_UNBLOB_FLAG_USE_UUID_IN_DER_CTX ((hsm_op_key_gen_flags_t)(1u << 2))
```

Use the unique ID in the derivation context.

3.31.4 Typedef documentation

3.31.4.1 hsm_op_blob_unblob_flags_t

```
typedef uint8_t hsm_op_blob_unblob_flags_t
```

3.31.5 Enumeration type documentation

3.31.5.1 hsm_op_blob_unblob_mode_t

Enum describing the mode of the operation: wrap or unwrap.

3.31.6 Function documentation

3.31.6.1 hsm_blob_unblob ()

```
hsm_err_t hsm_blob_unblob (hsm_hdl_t session_hdl, op_blob_unblob_msg_args_t *args)
```

This API enables:

- Creating blob by wrapping a plain-text, using the wrapping key.
- Un-blob, by un-wrapping the blob to plain-text, using the same wrapping key.

Table 37. Parameters

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

Returns

Error code.

3.32 Boot stage

Data Structures

- struct op_boot_stage_ctx_args_t

Enumerations

- enum hsm_op_code_bs_ctx_t {
 HSM_OP_BOOT_STAGE_CTX_SET_VAL = 0x55,
 HSM_OP_BOOT_STAGE_CTX_GET_VAL = 0xAA, }

Functions

- hsm_err_t hsm_bsctx_ops (hsm_hdl_t session_hdl, op_boot_stage_ctx_args_t *args)

3.32.1 Detailed description

None.

3.32.2 Data structure documentation

3.32.2.1 struct op_boot_stage_ctx_args_t

Structure specifying the boot stage context operation request.

Data Fields

| | | |
|---------|---------|--|
| uint8_t | op_code | Operation code to specify if HSM_OP_BOOT_STAGE_CTX_SET_VAL or HSM_OP_BOOT_STAGE_CTX_GET_VAL. |
|---------|---------|--|

Data Fields...continued

| | | |
|----------|----------------|---|
| uint32_t | bs_ctx_val | Value to be written in context when HSM_OP_BOOT_STAGE_CTX_SET_VAL operation is set. |
| uint32_t | exp_bs_ctx_val | Value read from the context when HSM_OP_BOOT_STAGE_CTX_GET_VAL operation is set. |

3.32.3 Enumeration type documentation

3.32.3.1 hsm_op_code_bs_ctx_t

Enum describing the mode of operation: set or get the boot-stage context value.

3.32.4 Function documentation

3.32.4.1 hsm_err_t hsm_bsctx_ops ()

```
hsm_err_t hsm_bsctx_ops (hsm_hdl_t session_hdl, op_boot_stage_ctx_args_t * args)
```

This API reports the first *n* reported events from the FW, where *n* is defined by the macro MAX_NUM_EVENT_REPORT.

Table 38. Parameters

| | |
|-------------|---|
| session_hdl | Handle identifying the session handle. |
| args | Pointer to the structure containing the function arguments. |

Returns

Error code.

3.33 Asymmetric crypto encryption

Data Structures

- struct op_asymmetric_enc_open_args_t
- struct op_asymmetric_enc_args_t

Macros

- #define HSM_ASYM_FLAGS_DECRYPT (0u << 1)
- #define HSM_ASYM_FLAGS_ENCRYPT BIT(0)
- #define HSM_ASYM_FLAGS_PLAINTEXT_KEY BIT(3)
- #define HSM_ASYM_FLAGS_OPAQUE_KEY (0u << 3)

Enumerations

- enum hsm_op_asymmetric_algo_t { HSM_ASYM_ALGO_RSA_PKCS1_V15_CRYPT = ALGO_RSA_PKCS1_V15_CRYPT, HSM_ASYM_ALGO_RSA_PKCS1_OAEP_SHA1 = ALGO_RSA_PKCS1_OAEP_SHA1, HSM_ASYM_ALGO_RSA_PKCS1_OAEP_SHA224 = ALGO_RSA_PKCS1_OAEP_SHA224, HSM_ASYM_ALGO_RSA_PKCS1_OAEP_SHA256 = ALGO_RSA_PKCS1_OAEP_SHA256, HSM_ASYM_ALGO_RSA_PKCS1_OAEP_SHA384 = ALGO_RSA_PKCS1_OAEP_SHA384, HSM_ASYM_ALGO_RSA_PKCS1_OAEP_SHA512 = ALGO_RSA_PKCS1_OAEP_SHA512 }

Functions

- hsm_err_t hsm_asymmetric_enc_open (hsm_hdl_t session_hdl, op_asymmetric_enc_open_args_t *args)
- hsm_err_t hsm_asymmetric_enc_close (hsm_hdl_t asymmetric_encryption_handle)
- hsm_err_t hsm_asymmetric_enc (hsm_hdl_t session_hdl, op_asymmetric_enc_args_t *args)

3.33.1 Detailed description

None.

3.33.2 Data structure documentation

3.33.2.1 struct op_asymmetric_enc_open_args_t

Structure representing the Asymmetric Encryption Open (0x90) command. This command is used to open an asymmetric encryption service flow on a specified key store. It is only available at runtime in the EdgeLock Secure Enclave Firmware.

Table 39. Data Fields

| | | |
|----------|-----------------------|---|
| uint32_t | key_store_handle | Handle identifying the key store service flow. |
| uint32_t | input_pointer_msb | Most significant 32 bits of the input pointer address. Must be set to 0x0. 64-bit addressing is not supported yet. |
| uint32_t | output_pointer_msb | Most significant 32 bits of the output pointer address. Must be set to 0x0. 64-bit addressing is not supported yet. |
| uint8_t | flags | Flags: reserved for future use. |
| uint32_t | asymmetric_enc_handle | Asymmetric encryption handle returned by the API. |

3.33.2.2 struct op_asymmetric_enc_args_t

Structure representing the Asymmetric Encryption Command API.

Table 40. Data Fields

| | | |
|-----------|-----------------------|---|
| uint32_t | asymmetric_enc_handle | OPAQUE KEY: Handle identifying the Asymmetric encryption service flow. PLAINTEXT KEY: Reserved. |
| uint32_t | key_id | OPAQUE KEY: Key identifier of the key to be used for the operation. |
| uint8_t * | key_addr | PLAINTEXT KEY: Private or public key address (LSB). Encryption: Public key buffer must be filled with the modulus only (default public exponent is used by EdgeLock Secure Enclave Firmware). Decryption: Private key buffer must be filled with the concatenation of the private exponent followed by the modulus. |
| uint8_t * | plaintext_addr | LSB address of the plaintext in requestor space. |
| uint8_t * | ciphertext_addr | LSB address of the ciphertext in requestor space. |
| uint8_t * | label_addr | LSB address of the label (optional, only for RSA PKCS1 OAEP). |
| uint32_t | plaintext_size | Size of the plaintext in bytes. |
| uint32_t | ciphertext_size | Size of the ciphertext in bytes. |
| uint32_t | label_size | Size of the label in bytes. |
| uint8_t | flags | Bit field indicating requested operations. |

Table 40. Data Fields...continued

| | | |
|------------------------------|------------------------------|---|
| | | <ul style="list-style-type: none"> • Bit 0: 0 = Decryption, 1 = Encryption. • Bit 3: 0 = Opaque key, 1 = Plaintext key. |
| hsm_asymmetric_crypto_algo_t | algorithm | Encryption algorithm to use. |
| uint32_t | input_plainkey_size | PLAINTEXT KEY ONLY: Input plaintext key size in bytes. |
| uint16_t | input_plainkey_type | PLAINTEXT KEY ONLY: Input plaintext key type. |
| uint16_t | input_plainkey_security_size | PLAINTEXT KEY ONLY: Input plaintext key security size in bits. |
| uint32_t | exp_plaintext_size | For decryption operation, plaintext size cannot be known in advance. Operation can be called with the output plaintext size set to 0. In that case, 0x1D29 error is returned with the maximum plaintext size. Setting an invalid size (smaller than expected) will result in 0x1D29 error and the expected size will be returned. |
| uint32_t | exp_ciphertext_size | For encryption operation, the ciphertext size is not known in advance. Setting an invalid size (smaller than expected) will result in 0x1D29 error and the expected size will be returned. |

3.33.3 Macro definition documentation

- #define HSM_ASYM_FLAGS_DECRYPT (0u << 1)
Bit indicating the decrypt operation.
- #define HSM_ASYM_FLAGS_ENCRYPT BIT(0)
Bit indicating the encrypt operation.
- #define HSM_ASYM_FLAGS_PLAINTEXT_KEY BIT(3)
Bit indicating the PLAINTEXT key user input.
- #define HSM_ASYM_FLAGS_OPAQUE_KEY (0u << 3)
Bit indicating the OPAQUE key user input.

3.33.4 Enumeration type documentation

3.33.4.1 hsm_op_asymmetric_algo_t

```
enum hsm_op_asymmetric_algo_t
```

Enum describing the asymmetric operation algorithm.

3.33.5 Function documentation

3.33.5.1 hsm_err_t hsm_asymmetric_enc_open ()

```
hsm_err_t hsm_asymmetric_enc_open (hsm_hdl_t session_hdl,
op_asymmetric_enc_open_args_t * args)
```

This command is designed to open the Asymmetric crypto service flow.

Table 41. Parameters

| | |
|-------------|---|
| session_hdl | Handle identifying the current session. |
| args | Pointer to the structure containing the function arguments. |

Returns

Error code.

3.33.5.2 hsm_err_t hsm_asymmetric_enc_close ()

```
hsm_err_t hsm_asymmetric_enc_close (hsm_hdl_t asymmetric_encryption_handle)
```

This command is designed to close the Asymmetric crypto service flow.

Table 42. Parameters

| | |
|-------------------------------------|---|
| <i>asymmetric_encryption_handle</i> | Handle identifying the Asymmetric service flow. |
|-------------------------------------|---|

Returns

Error code.

3.33.5.3 hsm_err_t hsm_asymmetric_enc ()

```
hsm_err_t hsm_asymmetric_enc (hsm_hdl_t session_hdl, op_asymmetric_enc_args_t * args)
```

This command is designed to perform the Asymmetric encryption/decryption.

Table 43. Parameters

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

Returns

Error code.

3.34 Generate ELE Key Blob

Data Structures

- struct op_gen_ele_key_blob_otfad_key_conf_t
- struct op_gen_ele_key_blob_format_t
- struct op_gen_ele_key_blob_msg_args_t

Macros

- #define GEN_ELE_KEY_BLOB_OVERHEAD_SZ 0x30
- #define GEN_ELE_KEY_BLOB_FORMAT_TAG 0x81
- #define GEN_ELE_KEY_BLOB_FORMAT_VERS 0x00
- #define HSM_OP_GEN_ELE_KEY_BLOB_FLAG_BLOB_TYPE_DEK ((hsm_op_gen_ele_key_blob_flags_t)(1u << 0))
Use flag for blob type as DEK.
- #define HSM_OP_GEN_ELE_KEY_BLOB_FLAG_BLOB_TYPE_OTFAD_CONF ((hsm_op_gen_ele_key_blob_flags_t)(1u << 1))
Use flag for blob type for OTFAD configuration.
- #define OP_GEN_ELE_KEY_BLOB_PAYLOAD_OTFAD_SZ 0x28
OP_GEN_ELE_KEY_BLOB_PAYLOAD_OTFAD_SZ equal to sizeof(op_gen_ele_key_blob_otfad_key_conf_t)
- #define OP_GEN_ELE_KEY_BLOB_PAYLOAD_DEK_AES128_CBC_SZ 0x10

- #define OP_GEN_ELE_KEY_BLOB_PAYLOAD_DEK_AES192_CBC_SZ 0x18
- #define OP_GEN_ELE_KEY_BLOB_PAYLOAD_DEK_AES256_CBC_SZ 0x20
- #define OP_GEN_ELE_KEY_BLOB_PAYLOAD_DEK_SM4_CBC_SZ 0x10
- #define OP_GEN_ELE_KEY_BLOB_PAYLOAD_MAX_SZ OP_GEN_ELE_KEY_BLOB_PAYLOAD_OTFAD_SZ
- #define OP_GEN_ELE_KEY_BLOB_FORMAT_SZ_WO_PAYLOAD (sizeof(op_gen_ele_key_blob_format_t) - OP_GEN_ELE_KEY_BLOB_PAYLOAD_MAX_SZ)

Typedefs

- typedef uint8_t hsm_op_gen_ele_key_blob_flags_t

Enumerations

- enum hsm_op_gen_ele_key_blob_algo_t {
 HSM_OP_GEN_ELE_KEY_BLOB_ALGO_AES_CBC = 0x03,
 HSM_OP_GEN_ELE_KEY_BLOB_ALGO_AES_CTR = 0x04,
 HSM_OP_GEN_ELE_KEY_BLOB_ALGO_SM4_CBC = 0x2b,
 HSM_OP_GEN_ELE_KEY_BLOB_ALGO_AES_XTS = 0x37 }

Functions

- hsm_err_t hsm_gen_ele_key_blob (hsm_hdl_t session_hdl, op_gen_ele_key_blob_msg_args_t *args)

3.34.1 Detailed description

None.

3.34.2 Data structure documentation

3.34.2.1 struct op_gen_ele_key_blob_otfad_key_conf_t

Structure specifying the OTFAD Key configuration format.

Table 44. Data Fields

| | | |
|----------|-----------------|-------------------------------------|
| uint8_t | aes_key[16] | AES key to use for OTFAD Key blob. |
| uint64_t | aes_key_counter | AES key counter for OTFAD Key blob. |
| uint32_t | key_start_addr | AES key buffer start address. |
| uint32_t | key_end_addr | AES key buffer end address. |
| uint32_t | reserved | Reserved. |
| uint32_t | crc | CRC. |

3.34.2.2 struct op_gen_ele_key_blob_format_t

Structure specifying the Generate ELE key blob format.

Table 45. Data Fields

| | | |
|---------|---------|---|
| uint8_t | vers | Version. Hard coded as GEN_ELE_KEY_BLOB_FORMAT_VERS. |
| uint8_t | lsb_len | LSB of total length in bytes of the header and its payload. |
| uint8_t | msb_len | MSB of total length in bytes of the header and its payload. |

Table 45. Data Fields...continued

| | | |
|---------------------------------|---|---|
| uint8_t | tag | Tag identifier. Hard coded as GEN_ELE_KEY_BLOB_FORMAT_TAG. |
| hsm_op_gen_ele_key_blob_flags_t | flag | Usage flag to specify the type of ELE key blob to generate. |
| uint8_t | raw_data_sz | Size of the raw data (without the blob overhead and the header). Depends on the flag used. |
| uint8_t | algo | Symmetric cipher algorithm used to generate the blob. |
| uint8_t | resrv | Reserved. |
| uint8_t | payload[OP_GEN_ELE_KEY_BLOB_PAYLOAD_OTFAD_SZ] | Payload as per flag and algo. |

3.34.2.3 struct op_gen_ele_key_blob_msg_args_t

Structure specifying the Generate ELE key blob request.

Table 46. Data Fields

| | | |
|-----------|---------------|---|
| uint32_t | key_id | Depends on the blob type: Non-OTFAD or OTFAD. For Non-OTFAD: Identifier of the input key (defined by the user). The same value has to be provided again when decrypting the generated blob. For OTFAD: Byte 0: Index of the OTFAD key struct (0 .. 3). Important when the key scrambling is enabled. Byte 1: 0x1: FlexSPI 1, 0x2: FlexSPI 2 Byte 2-3: Reserved. |
| uint8_t * | load_addr | Address where the input data can be loaded from. Must be 64 bit aligned. Input data consists in the blob header as defined by the Blob format, filled with relevant info and followed by plaintext data to be encrypted. |
| uint32_t | load_addr_sz | Size of data to be loaded. |
| uint8_t * | store_addr | Address space, where user input to blob-key derivation context, can be found. |
| uint32_t | store_addr_sz | Size of data to be stored. |
| uint16_t | max_export_sz | Maximum memory available in bytes at the provided address. Note: The blob needs 48 bytes more than the plaintext key. |

3.34.3 Macro definition documentation

3.34.3.1 GEN_ELE_KEY_BLOB_OVERHEAD_SZ

```
#define GEN_ELE_KEY_BLOB_OVERHEAD_SZ 0x30
```

ELE key blob overhead size.

3.34.3.2 GEN_ELE_KEY_BLOB_FORMAT_TAG

```
#define GEN_ELE_KEY_BLOB_FORMAT_TAG 0x81
```

ELE key blob format tag identifier.

3.34.3.3 GEN_ELE_KEY_BLOB_FORMAT_VERS

```
#define GEN_ELE_KEY_BLOB_FORMAT_VERS 0x00
```

ELE key blob format version.

3.34.3.4 HSM_OP_GEN_ELE_KEY_BLOB_FLAG_BLOB_TYPE_DEK

```
#define HSM_OP_GEN_ELE_KEY_BLOB_FLAG_BLOB_TYPE_DEK  
((hsm_op_gen_ele_key_blob_flags_t)(1u << 0))
```

Use flag for blob type as DEK.

3.34.3.5 HSM_OP_GEN_ELE_KEY_BLOB_FLAG_BLOB_TYPE_OTFAD_CONF

```
#define HSM_OP_GEN_ELE_KEY_BLOB_FLAG_BLOB_TYPE_OTFAD_CONF  
((hsm_op_gen_ele_key_blob_flags_t)(1u << 1))
```

Use flag for blob type for OTFAD configuration.

3.34.3.6 OP_GEN_ELE_KEY_BLOB_PAYLOAD_OTFAD_SZ

```
#define OP_GEN_ELE_KEY_BLOB_PAYLOAD_OTFAD_SZ 0x28
```

OP_GEN_ELE_KEY_BLOB_PAYLOAD_OTFAD_SZ must be equal to sizeof(op_gen_ele_key_blob_otfad_key_conf_t).

3.34.3.7 OP_GEN_ELE_KEY_BLOB_PAYLOAD_DEK_AES128_CBC_SZ

```
#define OP_GEN_ELE_KEY_BLOB_PAYLOAD_DEK_AES128_CBC_SZ 0x10
```

Payload size, in case the Generate ELE Key Blob configuration is DEK flag, AES CBC algorithm, and 128 bits AES key.

3.34.3.8 OP_GEN_ELE_KEY_BLOB_PAYLOAD_DEK_AES192_CBC_SZ

```
#define OP_GEN_ELE_KEY_BLOB_PAYLOAD_DEK_AES192_CBC_SZ 0x18
```

Payload size, in case the Generate ELE Key Blob configuration is DEK flag, AES CBC algorithm, and 192 bits AES key.

3.34.3.9 OP_GEN_ELE_KEY_BLOB_PAYLOAD_DEK_AES256_CBC_SZ

```
#define OP_GEN_ELE_KEY_BLOB_PAYLOAD_DEK_AES256_CBC_SZ 0x20
```

Payload size, in case the Generate ELE Key Blob configuration is DEK flag, AES CBC algorithm, and 256 bits AES key.

3.34.3.10 OP_GEN_ELE_KEY_BLOB_PAYLOAD_DEK_SM4_CBC_SZ

```
#define OP_GEN_ELE_KEY_BLOB_PAYLOAD_DEK_SM4_CBC_SZ 0x10
```

Payload size, in case the Generate ELE Key Blob configuration is DEK flag, SM4 CBC algorithm, and 128 bits SM4 key.

3.34.3.11 OP_GEN_ELE_KEY_BLOB_PAYLOAD_MAX_SZ

```
#define OP_GEN_ELE_KEY_BLOB_PAYLOAD_MAX_SZ OP_GEN_ELE_KEY_BLOB_PAYLOAD_OTFAD_SZ
```

Maximum payload size of the ELE key blob.

3.34.3.12 OP_GEN_ELE_KEY_BLOB_FORMAT_SZ_WO_PAYLOAD

```
#define OP_GEN_ELE_KEY_BLOB_FORMAT_SZ_WO_PAYLOAD  
(sizeof(op_gen_ele_key_blob_format_t) - OP_GEN_ELE_KEY_BLOB_PAYLOAD_MAX_SZ)
```

Generate the ELE key blob format size without including the payload size.

3.34.4 Typedef documentation

3.34.4.1 hsm_op_gen_ele_key_blob_flags_t

```
typedef uint8_t hsm_op_gen_ele_key_blob_flags_t
```

Define ELE Key blob generate operation flags type, to specify the type of the key blob to generate.

3.34.5 Enumeration type documentation

3.34.5.1 hsm_op_gen_ele_key_blob_algo_t

Enum describing the encryption algos to be used for ELE key blob generation.

3.34.6 Function documentation

3.34.6.1 hsm_gen_ele_key_blob

```
hsm_err_t hsm_gen_ele_key_blob (hsm_hdl_t session_hdl,  
op_gen_ele_key_blob_msg_args_t * args)
```

This API enables:

- Generating the ELE Key blob based on the provided ELE-Key-Blob-Format at `args->load_addr`.
- Generated ELE Key blob is stored at `args->store_addr`.

Table 47. Parameters

| | |
|--------------------|--|
| <i>session_hdl</i> | Handle identifying the session handle. |
|--------------------|--|

Table 47. Parameters...continued

| | |
|-------------------|---|
| <code>args</code> | Pointer to the structure containing the function arguments. |
|-------------------|---|

Returns

Error code.

3.35 SE lib service**Data Structures**

- `struct open_se_lib_serv_args_t`

Typedefs

- `typedef uint32_t se_lib_hdl_t`

Functions

- `uint32_t se_lib_open_service (open_se_lib_serv_args_t *args, se_lib_hdl_t *se_lib_serv_hdl)`
- `uint32_t se_lib_close_service (se_lib_hdl_t se_lib_serv_hdl)`

3.35.1 Detailed description

None.

3.35.2 Data structure documentation**3.35.2.1 struct open_se_lib_serv_args_t**

Structure detailing the open SE library service operation member arguments.

Table 48. Data Fields

| | | |
|---------------------------|------------------------------|---|
| <code>se_lib_hdl_t</code> | <code>se_lib_serv_hdl</code> | SE library service handle. |
| <code>uint8_t</code> | <code>mu_type</code> | MU with associated functionality on which the SE library service is opened. |

3.35.3 Typedef documentation**3.35.3.1 se_lib_hdl_t**

```
typedef uint32_t se_lib_hdl_t
```

Define the SE library service handle type.

3.35.4 Function documentation**3.35.4.1 se_lib_open_service ()**

```
uint32_t se_lib_open_service (open_se_lib_serv_args_t *args, se_lib_hdl_t *se_lib_serv_hdl)
```

This API enables users to exchange message with the ELE firmware through the SE library service handle, without opening a session with ELE, in cases where the session handle is not required.

Table 49. Parameters

| | |
|------------------------|---|
| <i>args</i> | Pointer to the structure containing the function arguments. |
| <i>se_lib_serv_hdl</i> | Pointer to where the SE library service handle must be written. |

Returns

Error code

3.35.4.2 se_lib_close_service ()

```
uint32_t se_lib_close_service (se_lib_hdl_t se_lib_serv_hdl)
```

Close a previously opened SE library service.

Table 50. Parameters

| | |
|------------------------|---|
| <i>se_lib_serv_hdl</i> | Handle identifying the SE library service to be closed. |
|------------------------|---|

Returns

Error code

3.36 Global information

Data Structures

- struct [global_info_s](#)

Macros

- #define SOC_IMX8ULP 0x84d
- #define SOC_IMX91 0x9100
- #define SOC_IMX93 0x9300
- #define SOC_IMX95 0x9500
- #define SOC_IMX943 0x9430
- #define SOC_REV_A0 0xa000
- #define SOC_REV_A1 0xa100
- #define SOC_REV_A2 0xa200
- #define SOC_REV_B0 0xb000
- #define SOC_REV_B1 0xb100
- #define HSM_API_VERSION_1 0x1
- #define HSM_API_VERSION_2 0x2
- #define HSM_API_VERSION_3 0x3
- #define SOC_LF_OPEN 0x1
- #define SOC_LF_CLOSED 0x2
- #define SOC_LF_CLOSED_LOCKED 0x4

Functions

- void [populate_global_info](#) ([hsm_hdl_t](#) hsm_session_hdl)
- void [show_global_info](#) (void)
- bool [is_global_info_populated](#) (void)
- [uint8_t](#) [hsm_get_dev_attest_api_ver](#) (void)
- [uint8_t](#) [se_get_board_type](#)(void)

- uint16_t se_get_soc_id (void)
- uint16_t se_get_soc_rev (void)
- uint16_t se_get_chip_lifecycle (void)
- uint8_t se_get_fips_mode (void)
- uint8_t se_get_lib_newness_ver (void)
- uint8_t se_get_lib_major_ver (void)
- uint8_t se_get_lib_minor_ver (void)
- uint8_t se_get_nvm_newness_ver (void)
- uint8_t se_get_nvm_major_ver (void)
- uint8_t se_get_nvm_minor_ver (void)
- const char * se_get_commit_id (void)
- const char * se_get_lib_version (void)
- const char * se_get_nvm_version (void)
- const char * [get_soc_id_str](#) (uint16_t soc_id)
- const char * get_board_type_str (uint8_t board_type)
- const char * [get_soc_rev_str](#) (uint16_t soc_rev)
- const char * [get_soc_lf_str](#) (uint16_t lifecycle)
- uint32_t se_get_info (uint32_t session_hdl, op_get_info_args_t *args)
- void se_get_soc_info (uint32_t session_hdl, uint16_t *soc_id, uint16_t *soc_rev)

Variables

- struct [global_info_s global_info](#)

3.36.1 Detailed description

3.36.2 Data structure documentation

3.36.2.1 struct global_info_s

Global Information structure contains the information about SoC and the Library. It is used globally to take platform specific decisions.

Table 51. Data Fields

| | | |
|----------|-----------------|---|
| bool | is_populated | To ensure global information is populated once. |
| uint8_t | ver | Supported version of HSM APIs. |
| uint8_t | fips_mode | Fips mode. |
| uint16_t | soc_id | SoC ID. |
| uint16_t | soc_rev | SoC revision. |
| uint16_t | lifecycle | Device lifecycle. |
| uint32_t | lib_newness_ver | Secure Enclave Library Newness version. |
| uint32_t | lib_major_ver | Secure Enclave Library major version. |
| uint32_t | lib_minor_ver | Secure Enclave Library minor version. |
| uint32_t | nvm_newness_ver | NVM Library Newness version. |
| uint32_t | nvm_major_ver | NVM Library major version. |
| uint32_t | nvm_minor_ver | NVM Library minor version. |

Table 51. Data Fields...continued

| | | |
|------|------------------------------------|--|
| char | se_commit_id[GINFO_COMMIT_ID_SZ] | Secure Enclave Build Commit ID. |
| char | lib_version[GINFO_LIB_VERSION_LEN] | Secure Enclave Library version string. |
| char | nvm_version[GINFO_NVM_VERSION_LEN] | NVM version string. |

3.36.3 Function documentation

3.36.3.1 populate_global_info()

```
void populate_global_info (hsm_hdl_t hsm_session_hdl)
```

This function is called to populate the Global Information structure.

Parameters

| | |
|------------------------|---------------------------------|
| <i>hsm_session_hdl</i> | Identifying the active session. |
|------------------------|---------------------------------|

3.36.3.2 show_global_info()

```
void show_global_info (void )
```

This function prints the Global Information of the library.

3.36.3.3 is_global_info_populated()

```
bool is_global_info_populated (void )
```

This function gets the status of Global Info, if populated or not.

3.36.3.4 hsm_get_dev_attest_api_ver()

```
uint8_t hsm_get_dev_attest_api_ver (void )
```

This function returns the version supported for Device Attestation.

3.36.3.5 se_get_board_type()

```
uint8_t se_get_board_type (void)
```

Get the board type.

Note: This API is useful when there are multiple variations of boards e.g., i.MX 8DXL, which has three different types of boards, DL1, DL2 and DL3.

3.36.3.6 se_get_soc_id()

```
uint16_t se_get_soc_id (void )
```

Get the SoC ID.

3.36.3.7 se_get_soc_rev()

```
uint16_t se_get_soc_rev (void )
```

Get the SoC revision.

3.36.3.8 se_get_chip_lifecycle()

```
uint16_t se_get_chip_lifecycle (void )
```

Get the chip lifecycle.

3.36.3.9 se_get_fips_mode()

```
uint8_t se_get_fips_mode (void )
```

Get the Fips mode.

3.36.3.10 se_get_lib_newness_ver()

```
uint8_t se_get_lib_newness_ver (void )
```

Get the library newness version.

3.36.3.11 se_get_lib_major_ver()

```
uint8_t se_get_lib_major_ver (void )
```

Get the library major version.

3.36.3.12 se_get_lib_minor_ver()

```
uint8_t se_get_lib_minor_ver (void )
```

Get the library minor version.

3.36.3.13 se_get_nvm_newness_ver()

```
uint8_t se_get_nvm_newness_ver (void )
```

Get the NVM newness version.

3.36.3.14 se_get_nvm_major_ver()

```
uint8_t se_get_nvm_major_ver (void )
```

Get the NVM major version.

3.36.3.15 se_get_nvm_minor_ver()

```
uint8_t se_get_nvm_minor_ver (void )
```

Get the NVM minor version.

3.36.3.16 se_get_commit_id()

```
const char* se_get_commit_id (void )
```

Get the build commit ID.

3.36.3.17 se_get_lib_version()

```
const char* se_get_lib_version (void )
```

Get the library version string.

3.36.3.18 se_get_nvm_version()

```
const char* se_get_nvm_version (void )
```

Get the NVM version string.

3.36.3.19 get_board_type_str()

```
const char* get_board_type_str (uint8_t board_type)
```

Get the string representating board type.

Parameters

| | |
|-------------------|--------------------------------------|
| <i>board_type</i> | Board Type fetched from Global Info. |
|-------------------|--------------------------------------|

Returns

String representation of the board type.

3.36.3.20 get_soc_id_str()

```
const char* get_soc_id_str (uint16_t soc_id)
```

This function returns a string representating SoC ID.

Parameters

| | |
|---------------|---|
| <i>soc_id</i> | SoC ID fetched from Global Information. |
|---------------|---|

Returns

String representation of the SoC ID.

3.36.3.21 get_soc_rev_str()

```
const char* get_soc_rev_str (uint16_t soc_rev)
```

This function returns a string representing SoC Revision.

Table 52. Parameters

| | |
|----------------|---|
| <i>soc_rev</i> | SoC Revision fetched from Global Information. |
|----------------|---|

Returns

String representation of the SoC revision.

3.36.3.22 get_soc_lf_str()

```
const char* get_soc_lf_str (uint16_t lifecycle)
```

This function returns a string representing Lifecycle.

Table 53. Parameters

| | |
|------------------|--|
| <i>lifecycle</i> | Value fetched from Global Information. |
|------------------|--|

Returns

A string representation of Lifecycle

3.36.3.23 se_get_info()

```
uint32_t se_get_info (
uint32_t session_hdl,
op_get_info_args_t * args )
```

Get the information for Global Info setup.

3.36.3.24 se_get_soc_info()

```
void se_get_soc_info (
uint32_t session_hdl,
uint16_t * soc_id,
uint16_t * soc_rev,
uint8_t * board_type)
```

Get the SoC information for Global Info setup generated.

3.36.4 Variable documentation

3.36.4.1 global_info

```
struct global\_info\_s global_info
```

Global Information structure instance, which is populated and then is used for getting the required platform or library details.

3.37 Common algorithms

Enumerations

- enum hsm_sha_algo_t {
ALGO_HASH_SHA224 = 0x02000008,
ALGO_HASH_SHA256 = 0x02000009,
ALGO_HASH_SHA384 = 0x0200000A,
ALGO_HASH_SHA512 = 0x0200000B,
ALGO_HASH_SM3 = 0x02000014,
};
- enum hsm_hmac_algo_t {
ALGO_HMAC_SHA256 = 0x03800009,
ALGO_HMAC_SHA384 = 0x0380000A,
ALGO_CMAC = 0x03C00200,
ALGO_CBCMAC = 0x03C00100,
};
- enum hsm_cipher_algo_t {
ALGO_CIPHER_CTR = 0x04C01000,
ALGO_CIPHER_CFB = 0x04C01100,
ALGO_CIPHER_OFB = 0x04C01200,
ALGO_CIPHER_ECB_NO_PAD = 0x04404400,
ALGO_CIPHER_CBC_NO_PAD = 0x04404000,
ALGO_CIPHER_ALL = 0x84C0FF00,
ALGO_CIPHER_KEK_CBC = 0x84404000,
};
- enum hsm_signature_algo_t {
ALGO_ECDSA_SHA224 = 0x06000608,
ALGO_ECDSA_SHA256 = 0x06000609,
ALGO_ECDSA_SHA384 = 0x0600060A,
ALGO_ECDSA_SHA512 = 0x0600060B,
ALGO_ED25519PH = 0x0600090B,
ALGO_ED448PH = 0x06000915,
ALGO_PURE_EDDSA = 0x06000800,
ALGO_EDDSA_ALL = 0x86000800,
ALGO_RSA_PKCS1_V15_SHA224 = 0x06000208,
ALGO_RSA_PKCS1_V15_SHA256 = 0x06000209,
ALGO_RSA_PKCS1_V15_SHA384 = 0x0600020A,
ALGO_RSA_PKCS1_V15_SHA512 = 0x0600020B,
ALGO_RSA_PKCS1_V15_SHA_ANY = 0x060002FF,
ALGO_RSA_PKCS1_PSS_MGF1_SHA224 = 0x06000308,
ALGO_RSA_PKCS1_PSS_MGF1_SHA256 = 0x06000309,
ALGO_RSA_PKCS1_PSS_MGF1_SHA384 = 0x0600030A,
ALGO_RSA_PKCS1_PSS_MGF1_SHA512 = 0x0600030B,
ALGO_RSA_PKCS1_PSS_MGF1_SHA_ANY = 0x060003FF,
ALGO_RSA_SIGNATURE_ALL = 0x8600FF00,
};
- enum hsm_pub_key_attest_algo_t {
ALGO_ATTEST_CMAC = 0x83C00200,
ALGO_ATTEST_ECDSA_SHA224 = 0x86000608,
ALGO_ATTEST_ECDSA_SHA256 = 0x86000609,
ALGO_ATTEST_ECDSA_SHA384 = 0x8600060A,
ALGO_ATTEST_ECDSA_SHA512 = 0x8600060B,
};

```

};
• enum hsm_asymmetric_crypto_algo_t {
  ALGO_RSA_PKCS1_V15_CRYPT = 0x07000200,
  ALGO_RSA_PKCS1_OAEP_SHA1 = 0x07000305,
  ALGO_RSA_PKCS1_OAEP_SHA224 = 0x07000308,
  ALGO_RSA_PKCS1_OAEP_SHA256 = 0x07000309,
  ALGO_RSA_PKCS1_OAEP_SHA384 = 0x0700030A,
  ALGO_RSA_PKCS1_OAEP_SHA512 = 0x0700030B,
};
• enum hsm_aead_algo_t {
  ALGO_CCM = 0x05500100,
  ALGO_GCM = 0x05500200,
  ALGO_CHACHA20_POLY1305 = 0x05100500,
};
• enum hsm_key_exchange_algo_t {
  ALGO_ECDH_HKDF_SHA256_KEY_IMPORT = 0x09020109,
  ALGO_ECDH_HKDF_SHA384_KEY_IMPORT = 0x0902010A,
  ALGO_ECDH_HKDF_SHA_ANY_KEY_IMPORT = 0x090201FF,
  ALGO_ECDH_HKDF_SHA256 = 0x89020109,
  ALGO_ECDH_HKDF_SHA384 = 0x8902010A,
  ALGO_ECDH_HKDF_SHA_ANY = 0x890201FF,
};
• enum hsm_key_derivation_algo_t {
  ALGO_HKDF_SHA256 = 0x08000109,
  ALGO_HKDF_SHA384 = 0x0800010A,
  ALGO_HKDF_EXTRACT_SHA256 = 0x08000409,
  ALGO_HKDF_EXTRACT_SHA384 = 0x0800040A,
  ALGO_HKDF_EXTRACT_SHA_ANY = 0x080004FF,
  ALGO_HKDF_EXPAND_SHA256 = 0x08000509,
  ALGO_HKDF_EXPAND_SHA384 = 0x0800050A,
};
• enum hsm_key_derivation_tls1_3_algo_t {
  ALGO_TLS1_3_ES_SHA256 = 0x8800D009,
  ALGO_TLS1_3_ES_SHA384 = 0x8800D00A,
  ALGO_TLS1_3_HS_SHA256 = 0x8800D109,
  ALGO_TLS1_3_HS_SHA384 = 0x8800D10A,
  ALGO_TLS1_3_MS_SHA256 = 0x8800D209,
  ALGO_TLS1_3_MS_SHA384 = 0x8800D20A,
  ALGO_TLS1_3_MS_SHA_ANY = 0x8800D2FF,
  ALGO_TLS1_3_KM_SHA256 = 0x8800D309,
  ALGO_TLS1_3_KM_SHA384 = 0x8800D30A,
  ALGO_TLS1_3_IV_SHA256 = 0x8800D409,
  ALGO_TLS1_3_IV_SHA384 = 0x8800D40A,
};
• enum hsm_key_derivation_tls1_2_algo_t {
  ALGO_TLS1_2_MASTER_SECRET_SHA256 = 0x8800E009,
  ALGO_TLS1_2_MASTER_SECRET_SHA384 = 0x8800E00A,
  ALGO_TLS1_2_MASTER_SECRET_SHA_ANY = 0x8800E0FF,
  ALGO_TLS1_2_KEY_BLOCK_SHA256 = 0x8800E109,
  ALGO_TLS1_2_KEY_BLOCK_SHA384 = 0x8800E10A,
  ALGO_TLS1_2_IV_SHA256 = 0x8800E209,
  ALGO_TLS1_2_IV_SHA384 = 0x8800E20A,

```

```

ALGO_TLS1_2_VERIFY_DATA_SHA256 = 0x8800E409,
ALGO_TLS1_2_VERIFY_DATA_SHA384 = 0x8800E40A,
};
• enum hsm_algo_t {
  ALGO_HMAC_KDF_SHA256 = 0x08000109,
  ALGO_ALL_CIPHER = 0x84C0FF00,
  ALGO_ALL_AEAD = 0x8550FF00,
};

```

3.38 Error codes

Enumerations

```

• enum hsm\_err\_t {
  HSM_NO_ERROR = SE_LIB_NO_ERROR,
  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 = SE_LIB_UNKNOWN_HANDLE,
  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_KEY_GROUP_FULL = 0x19,
  HSM_CANNOT_RETRIEVE_KEY_GROUP = 0x1A,
  HSM_KEY_NOT_SUPPORTED = 0x1B,
  HSM_CANNOT_DELETE_PERMANENT_KEY = 0x1C,
  HSM_OUT_TOO_SMALL = 0x1D,
  HSM_INVALID_SIGNATURE_VERIFICATION_HANDLE = 0x21,
  HSM_CRC_CHECK_ERR = 0xB9,
  HSM_INCONSISTENT_PARAM_ERR = 0xBC,
  HSM_OEM_CLOSED_LC_SIGNED_MSG_VERIFICATION_FAIL = 0xF0,
  HSM_OEM_OPEN_LC_SIGNED_MSG_VERIFICATION_FAIL = 0xF0,
  HSM_FATAL_FAILURE = 0x29,
  HSM_LIB_ERROR = SE_LIB_ERROR,
  HSM_INVALID_LIFECYCLE_OP = 0xF2,
  HSM_SERVICES_DISABLED = 0xF4,
  HSM_UNKNOWN_WARNING = 0xFC,
  HSM_SIGNATURE_INVALID = 0xFD,
  HSM_UNKNOWN_ERROR = 0xFE,
  HSM_GENERAL_ERROR = 0xFF,

```

```
HSM_LIB_ERR_IO_BUF_SETUP_OUT_OF_MEM = HSM_LIB_ERROR | IO_BUF_OUT_OF_MEM }
```

- enum se_lib_err_t {
SE_LIB_NO_ERROR = 0x0,
SE_LIB_UNKNOWN_HANDLE = 0x7,
SE_LIB_ERROR = 0xEF }
- enum lib_err_t {
IO_BUF_OUT_OF_MEM = 0x1400 }

3.38.1 Detailed description

3.38.2 Enumeration type documentation

3.38.2.1 enum hsm_err_t

Error codes returned by HSM functions.

Enumerator

| | |
|---------------------------|---|
| HSM_NO_ERROR | Success. |
| HSM_INVALID_MESSAGE | The received message is invalid or unknown. |
| HSM_INVALID_ADDRESS | The provided address is invalid or does not respect the API requirements. |
| HSM_UNKNOWN_ID | The provided identifier is not known. |
| HSM_INVALID_PARAM | One of the parameters 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" does not exist and the "create" flag is not set. |
| HSM_KEY_STORE_AUTH | Key store authentication fails. |
| HSM_KEY_STORE_ERROR | An error occurred in the key store internal processing. |
| HSM_ID_CONFLICT | An element (key store, key...) with the provided ID already exists. |
| HSM_RNG_NOT_STARTED | The internal RNG is not started. |
| HSM_CMD_NOT_SUPPORTED | The functionality is not supported for the current session/service/key store configuration. |
| HSM_INVALID_LIFECYCLE | Invalid lifecycle for requested operation. |
| HSM_KEY_STORE_CONFLICT | A key store with the same attributes already exists. |
| HSM_KEY_STORE_COUNTER | The current key store reaches the maximum 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. |

Enumerator...continued

| | |
|--|---|
| HSM_KEY_GROUP_FULL | Not enough space to store the key in the key group. |
| HSM_CANNOT_RETRIEVE_KEY_GROUP | Impossible to retrieve key group. |
| HSM_KEY_NOT_SUPPORTED | Key not supported. |
| HSM_CANNOT_DELETE_PERMANENT_KEY | Trying to delete a permanent key. |
| HSM_OUT_TOO_SMALL | Output buffer size is too small. |
| HSM_INVALID_SIGNATURE_VERIFICATION_HANDLE | Invalid Signature verification handle: Opaque key cannot be used. |
| HSM_CRC_CHECK_ERR | Command CRC check error. |
| HSM_INCONSISTENT_PARAM_ERR | Inconsistent parameter provided in the message request. |
| HSM_OEM_CLOSED_LC_SIGNED_MSG_VERIFICATION_FAIL | In OEM closed lifecycle, Signed message signature verification failure. |
| HSM_OEM_OPEN_LC_SIGNED_MSG_VERIFICATION_FAIL | In OEM open lifecycles, Signed message signature verification failure. |
| HSM_FATAL_FAILURE | A fatal failure occurs. The HSM goes in unrecoverable error state not replying to further requests. |
| HSM_SERVICES_DISABLED | Message neither handled by ROM nor FW. |
| HSM_LIB_ERROR | HSM library failure. |
| HSM_INVALID_LIFECYCLE_OP | Invalid Lifecycle operation (ROM). |
| HSM_UNKNOWN_WARNING | Unknown warnings. |
| HSM_SIGNATURE_INVALID | Failure in verification status of operations such as MAC verification, Signature verification. |
| HSM_UNKNOWN_ERROR | Unknown errors. |
| HSM_GENERAL_ERROR | Error in case General Error is received. |
| HSM_LIB_ERR_IO_BUF_SETUP_OUT_OF_MEM | HSM library failure, due to IO Buffer setup error. |

3.38.2.2 enum se_lib_err_t

Error codes returned by SE LIB functions.

Table 54. Enumerator

| | |
|-----------------------|--------------------------------|
| SE_LIB_NO_ERROR | Success |
| SE_LIB_UNKNOWN_HANDLE | Unknown session/service handle |
| SE_LIB_ERROR | SE library failure |

3.38.2.3 enum lib_err_t

Type of library error. When SE_LIB_ERROR (0xEF) is reported, the value on the second byte provides the reason of the library error.

Table 55. Enumerator

| | |
|-------------------|-------------------------|
| IO_BUF_OUT_OF_MEM | IO Buffer setup failure |
|-------------------|-------------------------|

3.39 i.MX 8ULP

[Cipherring](#)

- HSM_CIPHER_ONE_GO_ALGO_OFB is not supported.
- HSM_AEAD_ALGO_GCM is not supported.
- HSM_AEAD_ALGO_ALL_AEAD is not supported.
- ALGO_RSA_PKCS1_V15_SHA224 is not supported.
- ALGO_RSA_PKCS1_V15_SHA256 is not supported.
- ALGO_RSA_PKCS1_V15_SHA384 is not supported.
- ALGO_RSA_PKCS1_V15_SHA512 is not supported.
- ALGO_RSA_PKCS1_V15_SHA_ANY is not supported.
- ALGO_RSA_PKCS1_PSS_MGF1_SHA_ANY is not supported.
- ALGO_RSA_PKCS1_PSS_MGF1_SHA224 is not supported.
- ALGO_RSA_PKCS1_PSS_MGF1_SHA256 is not supported.
- ALGO_RSA_PKCS1_PSS_MGF1_SHA384 is not supported.
- ALGO_RSA_PKCS1_PSS_MGF1_SHA512 is not supported.

[Generic Crypto: AEAD](#)

- HSM_GC_AEAD_ALGO_AES_GCM is not supported.
- HSM_GC_AEAD_ALGO_CHACHA20_POLY1305 is not supported.

Key exchange: TLS 1.2, TLS 1.3, and HKDF options are not supported.

Signature Verification operation using Opaque key is not supported.

SM3 and SM4 are not supported.

3.40 i.MX 91

Signature Verification operation using Opaque key is not supported.

SM3 and SM4 are not supported.

3.41 i.MX 93

Signature Verification operation using Opaque key is not supported.

SM3 and SM4 are not supported.

3.42 i.MX 95

Key exchange: Key Exchange for Key Import and Key Import ELE option is not supported.

Generic Crypto APIs are not supported.

SM3 and SM4 are not supported.

3.43 i.MX 943

Key Exchange API is not supported.

Generic Crypto APIs are not supported.

4 Note About the Source Code in the Document

Example code shown in this document has the following copyright and BSD-3-Clause license:

Copyright 2026 NXP Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

1. Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.
2. Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.
3. Neither the name of the copyright holder nor the names of its contributors may be used to endorse or promote products derived from this software without specific prior written permission.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT HOLDER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

5 Revision History

This table provides the revision history.

Revision history

| Document ID | Release date | Description |
|--------------|-------------------|---|
| RM00284 v4.3 | 26 March 2026 | <ul style="list-style-type: none"> • Added support for Generate ELE Key Blob API (0xAF) • Added SM3 and SM4 support for i.MX 943. • Added SE library service support, to work with SE library handles. • New HSM library error code <code>HSM_LIB_ERR_IO_BUF_SETUP_OUT_OF_MEM</code>. |
| RM00284 v4.2 | 12 December 2025 | Added support of Asymmetric Crypto Encryption API. Supported on i.MX 91/93/95. |
| RM00284 v4.1 | 25 September 2025 | <ul style="list-style-type: none"> • Added support for Signature Verification using the Opaque Key on i.MX 95 and i.MX 943. • Added Get Event (0xA2) API support. • Added Blob Unblob (0xBF) API support. • Added Boot Stage (0xBE) API support. • Updated Key Exchange - ECDH-HKDF algorithms. |
| RM00284 v4.0 | 26 June 2025 | Added supports: <ul style="list-style-type: none"> • TLS 1.3 x448. • Ed448 key gen, sign, verify (eddsa and eddsaPH). • New algorithms to replace HKDF Expand: <code>TLS_13_KM</code> and <code>TLS_13_IV</code> (only <code>TLS_13_IV</code> allows plaintext output). • SHAKE256 HASH algorithm. • ChaCha20_Poly1305 key type. |

Revision history...continued

| Document ID | Release date | Description |
|--------------|-------------------|--|
| | | Removed supports: <ul style="list-style-type: none">• Key Exchange API - HKDF operation.• Key Exchange API - TLS output plaintext flag.• i.MX 95 B0, i.MX 943: The legacy AEAD API (<code>hsm_auth_enc</code>) support removed. Replaced by new AEAD API (<code>hsm_auth_enc_new</code>).• i.MX 95 B0, i.MX 943: All Generic crypto APIs. |
| RM00284 v3.3 | 31 March 2025 | Upgraded for the Linux LF6.12.3_1.0.0 release. |
| RM00284 v3.2 | 16 December 2024 | Upgraded for the Linux LF6.6.52_2.2.0 release. |
| RM00284 v3.1 | 30 September 2024 | Upgraded for the Linux LF6.6.36_2.1.0 release. |
| RM00284 v3.0 | 28 June 2024 | Upgraded for the Linux LF6.6.23_2.0.0 release. |
| RM00284 v2.0 | 29 March 2024 | Upgraded for the Linux LF6.6.3_1.0.0 release. |
| RM00284 v1.0 | 15 December 2023 | Initial release. |

Legal information

Definitions

Draft — A draft status on a document indicates that the content is still under internal review and subject to formal approval, which may result in modifications or additions. NXP Semiconductors does not give any representations or warranties as to the accuracy or completeness of information included in a draft version of a document and shall have no liability for the consequences of use of such information.

Disclaimers

Limited warranty and liability — Information in this document is believed to be accurate and reliable. However, NXP Semiconductors does not give any representations or warranties, expressed or implied, as to the accuracy or completeness of such information and shall have no liability for the consequences of use of such information. NXP Semiconductors takes no responsibility for the content in this document if provided by an information source outside of NXP Semiconductors.

In no event shall NXP Semiconductors be liable for any indirect, incidental, punitive, special or consequential damages (including - without limitation - lost profits, lost savings, business interruption, costs related to the removal or replacement of any products or rework charges) whether or not such damages are based on tort (including negligence), warranty, breach of contract or any other legal theory.

Notwithstanding any damages that customer might incur for any reason whatsoever, NXP Semiconductors' aggregate and cumulative liability towards customer for the products described herein shall be limited in accordance with the Terms and conditions of commercial sale of NXP Semiconductors.

Right to make changes — NXP Semiconductors reserves the right to make changes to information published in this document, including without limitation specifications and product descriptions, at any time and without notice. This document supersedes and replaces all information supplied prior to the publication hereof.

Suitability for use — NXP Semiconductors products are not designed, authorized or warranted to be suitable for use in life support, life-critical or safety-critical systems or equipment, nor in applications where failure or malfunction of an NXP Semiconductors product can reasonably be expected to result in personal injury, death or severe property or environmental damage. NXP Semiconductors and its suppliers accept no liability for inclusion and/or use of NXP Semiconductors products in such equipment or applications and therefore such inclusion and/or use is at the customer's own risk.

Applications — Applications that are described herein for any of these products are for illustrative purposes only. NXP Semiconductors makes no representation or warranty that such applications will be suitable for the specified use without further testing or modification.

Customers are responsible for the design and operation of their applications and products using NXP Semiconductors products, and NXP Semiconductors accepts no liability for any assistance with applications or customer product design. It is customer's sole responsibility to determine whether the NXP Semiconductors product is suitable and fit for the customer's applications and products planned, as well as for the planned application and use of customer's third party customer(s). Customers should provide appropriate design and operating safeguards to minimize the risks associated with their applications and products.

NXP Semiconductors does not accept any liability related to any default, damage, costs or problem which is based on any weakness or default in the customer's applications or products, or the application or use by customer's third party customer(s). Customer is responsible for doing all necessary testing for the customer's applications and products using NXP Semiconductors products in order to avoid a default of the applications and the products or of the application or use by customer's third party customer(s). NXP does not accept any liability in this respect.

Terms and conditions of commercial sale — NXP Semiconductors products are sold subject to the general terms and conditions of commercial sale, as published at <https://www.nxp.com/profile/terms>, unless otherwise agreed in a valid written individual agreement. In case an individual agreement is concluded only the terms and conditions of the respective agreement shall apply. NXP Semiconductors hereby expressly objects to applying the customer's general terms and conditions with regard to the purchase of NXP Semiconductors products by customer.

Export control — This document as well as the item(s) described herein may be subject to export control regulations. Export might require a prior authorization from competent authorities.

Suitability for use in non-automotive qualified products — Unless this document expressly states that this specific NXP Semiconductors product is automotive qualified, the product is not suitable for automotive use. It is neither qualified nor tested in accordance with automotive testing or application requirements. NXP Semiconductors accepts no liability for inclusion and/or use of non-automotive qualified products in automotive equipment or applications.

In the event that customer uses the product for design-in and use in automotive applications to automotive specifications and standards, customer (a) shall use the product without NXP Semiconductors' warranty of the product for such automotive applications, use and specifications, and (b) whenever customer uses the product for automotive applications beyond NXP Semiconductors' specifications such use shall be solely at customer's own risk, and (c) customer fully indemnifies NXP Semiconductors for any liability, damages or failed product claims resulting from customer design and use of the product for automotive applications beyond NXP Semiconductors' standard warranty and NXP Semiconductors' product specifications.

HTML publications — An HTML version, if available, of this document is provided as a courtesy. Definitive information is contained in the applicable document in PDF format. If there is a discrepancy between the HTML document and the PDF document, the PDF document has priority.

Translations — A non-English (translated) version of a document, including the legal information in that document, is for reference only. The English version shall prevail in case of any discrepancy between the translated and English versions.

Security — Customer understands that all NXP products may be subject to unidentified vulnerabilities or may support established security standards or specifications with known limitations. Customer is responsible for the design and operation of its applications and products throughout their lifecycles to reduce the effect of these vulnerabilities on customer's applications and products. Customer's responsibility also extends to other open and/or proprietary technologies supported by NXP products for use in customer's applications. NXP accepts no liability for any vulnerability. Customer should regularly check security updates from NXP and follow up appropriately. Customer shall select products with security features that best meet rules, regulations, and standards of the intended application and make the ultimate design decisions regarding its products and is solely responsible for compliance with all legal, regulatory, and security related requirements concerning its products, regardless of any information or support that may be provided by NXP.

NXP has a Product Security Incident Response Team (PSIRT) (reachable at PSIRT@nxp.com) that manages the investigation, reporting, and solution release to security vulnerabilities of NXP products.

NXP B.V. — NXP B.V. is not an operating company and it does not distribute or sell products.

Trademarks

Notice: All referenced brands, product names, service names, and trademarks are the property of their respective owners.

NXP — wordmark and logo are trademarks of NXP B.V.

EdgeLock — is a trademark of NXP B.V.

Contents

| | | | | |
|----------|---|----------|--|--|
| 1 | Overview | 2 | | |
| 2 | General Concepts Related to the API | 2 | | |
| 2.1 | Session | 2 | | |
| 2.2 | Service flow | 2 | | |
| 2.3 | Example | 3 | | |
| 2.4 | Key store | 3 | | |
| 2.4.1 | Key management | 3 | | |
| 2.4.2 | NVM writing | 3 | | |
| 2.5 | Cyber-Resilient Recovery Module (CRRM) | 4 | | |
| 2.6 | Implementation specificities | 4 | | |
| 3 | Module Documentation | 4 | | |
| 3.1 | Session | 4 | | |
| 3.1.1 | Detailed description | 5 | | |
| 3.1.2 | Data structure documentation | 5 | | |
| 3.1.2.1 | struct hsm_session_hdl_s | 5 | | |
| 3.1.2.2 | struct hsm_service_hdl_s | 5 | | |
| 3.1.2.3 | struct open_session_args_t | 6 | | |
| 3.1.3 | Typedef documentation | 6 | | |
| 3.1.3.1 | hsm_hdl_t | 6 | | |
| 3.1.4 | Function documentation | 6 | | |
| 3.1.4.1 | hsm_open_session() | 6 | | |
| 3.1.4.2 | hsm_close_session() | 6 | | |
| 3.1.4.3 | session_hdl_to_ptr() | 6 | | |
| 3.1.4.4 | service_hdl_to_ptr() | 7 | | |
| 3.1.4.5 | delete_session() | 7 | | |
| 3.1.4.6 | delete_service() | 7 | | |
| 3.1.4.7 | add_session() | 7 | | |
| 3.1.4.8 | add_service() | 7 | | |
| 3.2 | Key management | 8 | | |
| 3.2.1 | Detailed description | 12 | | |
| 3.2.2 | Data structure documentation | 12 | | |
| 3.2.2.1 | struct op_delete_key_args_t | 12 | | |
| 3.2.2.2 | struct op_get_key_attr_args_t | 12 | | |
| 3.2.2.3 | struct op_import_key_args_t | 13 | | |
| 3.2.2.4 | struct kek_enc_key_hdr_t | 13 | | |
| 3.2.2.5 | struct op_generate_key_args_t | 13 | | |
| 3.2.2.6 | struct open_svc_key_management_args_t | 14 | | |
| 3.2.2.7 | struct op_manage_key_group_args_t | 14 | | |
| 3.2.3 | Macro definition documentation | 15 | | |
| 3.2.3.1 | HSM_OP_DEL_KEY_FLAGS_STRICT_OPERATION | 15 | | |
| 3.2.3.2 | HSM_KEY_USAGE_ENCRYPT | 15 | | |
| 3.2.3.3 | HSM_KEY_USAGE_DECRYPT | 15 | | |
| 3.2.3.4 | HSM_KEY_USAGE_SIGN_MSG | 15 | | |
| 3.2.3.5 | HSM_KEY_USAGE_VERIFY_MSG | 15 | | |
| 3.2.3.6 | HSM_KEY_USAGE_SIGN_HASH | 15 | | |
| 3.2.3.7 | HSM_KEY_USAGE_VERIFY_HASH | 15 | | |
| 3.2.3.8 | HSM_KEY_USAGE_DERIVE | 16 | | |
| 3.2.3.9 | HSM_OP_KEY_GENERATION_FLAGS_PLAINTEXT_KEY | 16 | | |
| 3.2.3.10 | HSM_OP_KEY_GENERATION_FLAGS_MONOTONIC_COUNTER | 16 | | |
| 3.2.3.11 | HSM_OP_KEY_GENERATION_FLAGS_STRICT_OPERATION | 16 | | |
| 3.2.3.12 | HSM_OP_MANAGE_KEY_GROUP_FLAGS_CACHE_LOCKDOWN | 16 | | |
| 3.2.3.13 | HSM_OP_MANAGE_KEY_GROUP_FLAGS_EXPORT | 16 | | |
| 3.2.3.14 | HSM_OP_MANAGE_KEY_GROUP_FLAGS_MONOTONIC | 16 | | |
| 3.2.3.15 | HSM_OP_MANAGE_KEY_GROUP_FLAGS_SYNC_KEYSTORE | 17 | | |
| 3.2.4 | Typedef documentation | 17 | | |
| 3.2.4.1 | hsm_op_delete_key_flags_t | 17 | | |
| 3.2.4.2 | hsm_op_import_key_flags_t | 17 | | |
| 3.2.4.3 | hsm_key_usage_t | 17 | | |
| 3.2.4.4 | hsm_key_group_t | 17 | | |
| 3.2.4.5 | hsm_key_info_t | 17 | | |
| 3.2.4.6 | hsm_op_key_gen_flags_t | 18 | | |
| 3.2.4.7 | hsm_svc_key_management_flags_t | 18 | | |
| 3.2.5 | Enumeration type documentation | 18 | | |
| 3.2.5.1 | hsm_storage_loc_t | 18 | | |
| 3.2.5.2 | hsm_storage_persist_lvl_t | 18 | | |
| 3.2.5.3 | hsm_key_lifetime_t | 18 | | |
| 3.2.5.4 | hsm_pubkey_type_t | 18 | | |
| 3.2.5.5 | hsm_key_type_t | 18 | | |
| 3.2.5.6 | hsm_bit_key_sz_t | 18 | | |
| 3.2.5.7 | hsm_permitted_algo_t | 19 | | |
| 3.2.5.8 | hsm_key_lifecycle_t | 19 | | |
| 3.2.6 | Function documentation | 19 | | |
| 3.2.6.1 | hsm_delete_key() | 19 | | |
| 3.2.6.2 | hsm_get_key_attr() | 19 | | |
| 3.2.6.3 | hsm_import_key() | 20 | | |
| 3.2.6.4 | hsm_generate_key() | 20 | | |
| 3.2.6.5 | hsm_open_key_management_service() | 20 | | |
| 3.2.6.6 | hsm_close_key_management_service() | 21 | | |
| 3.2.6.7 | hsm_manage_key_group() | 21 | | |
| 3.3 | Ciphering | 21 | | |
| 3.3.1 | Detailed description | 23 | | |
| 3.3.2 | Data structure documentation | 23 | | |
| 3.3.2.1 | struct op_auth_enc_args_t | 23 | | |
| 3.3.2.2 | struct op_auth_enc_new_args_t | 24 | | |
| 3.3.2.3 | struct op_cipher_args_t | 25 | | |
| 3.3.2.4 | struct open_svc_cipher_args_t | 26 | | |
| 3.3.3 | Macro definition documentation | 27 | | |
| 3.3.3.1 | HSM_AUTH_ENC_FLAGS_DECRYPT | 27 | | |
| 3.3.3.2 | HSM_AUTH_ENC_FLAGS_ENCRYPT | 27 | | |
| 3.3.3.3 | HSM_AUTH_ENC_FLAGS_GENERATE_FULL_IV | 27 | | |
| 3.3.3.4 | HSM_AUTH_ENC_FLAGS_GENERATE_COUNTER_IV | 27 | | |
| 3.3.3.5 | HSM_AUTH_ENC_FLAGS_PLAINTEXT_KEY | 27 | | |
| 3.3.3.6 | HSM_AUTH_ENC_FLAGS_ONE_SHOT | 27 | | |
| 3.3.3.7 | HSM_AUTH_ENC_FLAGS_INIT | 27 | | |
| 3.3.3.8 | HSM_AUTH_ENC_FLAGS_UPDATE_AAD | 28 | | |
| 3.3.3.9 | HSM_AUTH_ENC_FLAGS_UPDATE_DATA | 28 | | |
| 3.3.3.10 | HSM_AUTH_ENC_FLAGS_FINALIZE | 28 | | |
| 3.3.3.11 | HSM_AUTH_ENC_FLAGS_FINALIZE_VERIFY | 28 | | |
| 3.3.3.12 | HSM_AUTH_ENC_FLAGS_ABORT | 28 | | |

| | | | | | |
|----------|--|----|---------|--|----|
| 3.3.3.13 | HSM_AUTH_ENC_FLAGS_GET_CTX_SIZE | 28 | 3.4 | Function documentation | 39 |
| 3.3.3.14 | HSM_AEAD_VERIFICATION_STATUS_SUCCESS | 28 | 3.4.6.1 | hsm_do_sign() | 39 |
| 3.3.3.15 | HSM_AEAD_VERIFICATION_STATUS_FAILURE | 29 | 3.4.6.2 | hsm_do_pub_key_attest() | 39 |
| 3.3.3.16 | HSM_AUTH_ENC_IV_OUT_SIZE | 29 | 3.4.6.3 | hsm_pub_key_attest() | 39 |
| 3.3.3.17 | HSM_CIPHER_ONE_GO_FLAGS_DECRYPT | 29 | 3.4.6.4 | hsm_open_signature_generation_service() | 40 |
| 3.3.3.18 | HSM_CIPHER_ONE_GO_FLAGS_ENCRYPT | 29 | 3.4.6.5 | hsm_close_signature_generation_service() | 40 |
| 3.3.3.19 | HSM_CIPHER_FLAGS_DECRYPT | 29 | 3.4.6.6 | hsm_generate_signature() | 40 |
| 3.3.3.20 | HSM_CIPHER_FLAGS_ENCRYPT | 29 | 3.4.6.7 | hsm_prepare_signature() | 41 |
| 3.3.3.21 | HSM_CIPHER_FLAGS_PLAINTEXT_KEY | 29 | 3.5 | Signature verification | 41 |
| 3.3.3.22 | HSM_CIPHER_FLAGS_INIT | 30 | 3.5.1 | Detailed description | 42 |
| 3.3.3.23 | HSM_CIPHER_FLAGS_UPDATE_DATA | 30 | 3.5.2 | Data structure documentation | 42 |
| 3.3.3.24 | HSM_CIPHER_FLAGS_FINALIZE | 30 | 3.5.2.1 | struct open_svc_sign_ver_args_t | 42 |
| 3.3.3.25 | HSM_CIPHER_FLAGS_ABORT | 30 | 3.5.2.2 | struct op_verify_sign_args_t | 42 |
| 3.3.3.26 | HSM_CIPHER_FLAGS_GET_CTX_SIZE | 30 | 3.5.3 | Macro Definition Documentation | 43 |
| 3.3.4 | Typedef documentation | 30 | 3.5.3.1 | HSM_OP_VERIFY_SIGN_FLAGS_INPUT_DIGEST | 43 |
| 3.3.4.1 | hsm_op_auth_enc_flags_t | 30 | 3.5.3.2 | HSM_OP_VERIFY_SIGN_FLAGS_INPUT_MESSAGE | 43 |
| 3.3.4.2 | hsm_op_auth_enc_new_flags_t | 30 | 3.5.3.3 | HSM_OP_VERIFY_SIGN_FLAGS_COMPRESSED_POINT | 43 |
| 3.3.4.3 | hsm_aead_verification_status_t | 30 | 3.5.3.4 | HSM_OP_VERIFY_SIGN_FLAGS_KEY_INTERNAL | 43 |
| 3.3.4.4 | hsm_svc_cipher_flags_t | 31 | 3.5.3.5 | HSM_OP_VERIFY_SIGN_FLAGS_PLAINTEXT_KEY | 43 |
| 3.3.4.5 | hsm_op_cipher_one_go_flags_t | 31 | 3.5.3.6 | HSM_OP_VERIFY_SIGN_FLAGS_OPAQUE_KEY | 44 |
| 3.3.4.6 | hsm_op_cipher_flags_t | 31 | 3.5.3.7 | HSM_VERIFICATION_STATUS_SUCCESS | 44 |
| 3.3.4.7 | op_cipher_one_go_args_t | 31 | 3.5.3.8 | HSM_VERIFICATION_STATUS_FAILURE | 44 |
| 3.3.5 | Enumeration type documentation | 31 | 3.5.4 | Typedef documentation | 44 |
| 3.3.5.1 | hsm_op_auth_enc_algo_t | 31 | 3.5.4.1 | hsm_verification_status_t | 44 |
| 3.3.5.2 | hsm_op_cipher_one_go_algo_t | 31 | 3.5.4.2 | hsm_op_verify_sign_flags_t | 44 |
| 3.3.5.3 | hsm_op_cipher_algo_t | 32 | 3.5.5 | Function documentation | 44 |
| 3.3.6 | Function documentation | 32 | 3.5.5.1 | hsm_verify_sign() | 44 |
| 3.3.6.1 | hsm_do_cipher() | 32 | 3.5.5.2 | hsm_open_signature_verification_service() | 45 |
| 3.3.6.2 | hsm_auth_enc() | 32 | 3.5.5.3 | hsm_close_signature_verification_service() | 45 |
| 3.3.6.3 | hsm_auth_enc_new() | 33 | 3.5.5.4 | hsm_verify_signature() | 46 |
| 3.3.6.4 | hsm_open_cipher_service() | 33 | 3.6 | Random number generation | 46 |
| 3.3.6.5 | hsm_cipher_one_go() | 34 | 3.6.1 | Detailed description | 46 |
| 3.3.6.6 | hsm_cipher() | 34 | 3.6.2 | Data structure documentation | 46 |
| 3.3.6.7 | hsm_close_cipher_service() | 34 | 3.6.2.1 | struct op_get_random_args_t | 46 |
| 3.4 | Signature generation | 35 | 3.6.3 | Function documentation | 47 |
| 3.4.1 | Detailed description | 36 | 3.6.3.1 | hsm_do_rng() | 47 |
| 3.4.2 | Data structure documentation | 36 | 3.6.3.2 | hsm_get_random() | 47 |
| 3.4.2.1 | struct op_pub_key_attest_args_t | 36 | 3.7 | Hashing | 47 |
| 3.4.2.2 | struct open_svc_sign_gen_args_t | 36 | 3.7.1 | Detailed description | 48 |
| 3.4.2.3 | struct op_generate_sign_args_t | 36 | 3.7.2 | Data structure documentation | 48 |
| 3.4.2.4 | struct op_prepare_sign_args_t | 37 | 3.7.2.1 | struct op_hash_one_go_args_t | 48 |
| 3.4.3 | Macro definition documentation | 37 | 3.7.3 | Macro definition documentation | 49 |
| 3.4.3.1 | HSM_OP_GENERATE_SIGN_FLAGS_INPUT_DIGEST | 37 | 3.7.3.1 | HSM_HASH_FLAG_ALLOWED | 49 |
| 3.4.3.2 | HSM_OP_GENERATE_SIGN_FLAGS_INPUT_MESSAGE | 38 | 3.7.4 | Enumeration type documentation | 49 |
| 3.4.3.3 | HSM_OP_GENERATE_SIGN_FLAGS_PLAINTEXT_KEY | 38 | 3.7.4.1 | hsm_hash_algo_t | 49 |
| 3.4.4 | Typedef documentation | 38 | 3.7.4.2 | hsm_hash_svc_flags_t | 49 |
| 3.4.4.1 | hsm_op_generate_sign_flags_t | 38 | 3.7.5 | Function documentation | 49 |
| 3.4.4.2 | hsm_op_prepare_signature_flags_t | 38 | 3.7.5.1 | hsm_do_hash() | 49 |
| 3.4.5 | Enumeration type documentation | 38 | 3.7.5.2 | hsm_hash_one_go() | 50 |
| 3.4.5.1 | hsm_op_pub_key_attest_algo_t | 38 | 3.8 | Data storage | 50 |
| 3.4.5.2 | hsm_signature_scheme_id_t | 38 | 3.8.1 | Detailed description | 51 |
| | | | 3.8.2 | Data structure documentation | 51 |

| | | | | | |
|-----------|--|----|----------|--|----|
| 3.8.2.1 | struct open_svc_data_storage_args_t | 51 | 3.10.5 | Function documentation | 62 |
| 3.8.2.2 | struct op_data_storage_args_t | 51 | 3.10.5.1 | hsm_do_mac() | 62 |
| 3.8.2.3 | struct op_enc_data_storage_args_t | 52 | 3.10.5.2 | hsm_open_mac_service() | 63 |
| 3.8.2.4 | struct op_data_storage_delete_args_t | 52 | 3.10.5.3 | hsm_mac_one_go() | 63 |
| 3.8.3 | Macro definition documentation | 53 | 3.10.5.4 | hsm_mac() | 63 |
| 3.8.3.1 | ENC_DATA_TLV_DEV_UUID_TAG | 53 | 3.10.5.5 | hsm_close_mac_service() | 64 |
| 3.8.3.2 | ENC_DATA_TLV_DEV_UUID_TAG_LEN | 53 | 3.11 | Dump firmware log | 64 |
| 3.8.4 | Typedef documentation | 53 | 3.11.1 | Detailed description | 64 |
| 3.8.4.1 | hsm_svc_data_storage_flags_t | 53 | 3.11.2 | Data structure documentation | 64 |
| 3.8.4.2 | hsm_op_data_storage_flags_t | 53 | 3.11.2.1 | struct op_debug_dump_args_t | 64 |
| 3.8.4.3 | hsm_op_enc_data_storage_flags_t | 53 | 3.11.3 | Function documentation | 65 |
| 3.8.5 | Function Documentation | 53 | 3.11.3.1 | dump_firmware_log() | 65 |
| 3.8.5.1 | hsm_data_ops() | 53 | 3.12 | Dev attest | 65 |
| 3.8.5.2 | hsm_data_delete_ops() | 54 | 3.12.1 | Detailed description | 65 |
| 3.8.5.3 | hsm_enc_data_ops() | 54 | 3.12.2 | Data structure documentation | 65 |
| 3.8.5.4 | hsm_open_data_storage_service() | 55 | 3.12.2.1 | struct op_dev_attest_args_t | 65 |
| 3.8.5.5 | hsm_data_storage() | 55 | 3.12.3 | Macro definition documentation | 66 |
| 3.8.5.6 | hsm_data_storage_delete() | 55 | 3.12.3.1 | DEV_ATTEST_NOUNCE_SIZE_V1 | 66 |
| 3.8.5.7 | hsm_enc_data_storage() | 56 | 3.12.4 | Function documentation | 66 |
| 3.8.5.8 | decode_enc_data_tlv() | 56 | 3.12.4.1 | hsm_dev_attest() | 66 |
| 3.8.5.9 | hsm_close_data_storage_service() | 56 | 3.13 | Dev Info | 67 |
| 3.9 | Authenticated encryption | 56 | 3.13.1 | Detailed description | 67 |
| 3.9.1 | Detailed description | 57 | 3.13.2 | Data structure documentation | 67 |
| 3.9.2 | Function documentation | 57 | 3.13.2.1 | struct op_dev_getinfo_args_t | 67 |
| 3.9.2.1 | hsm_do_auth_enc() | 57 | 3.13.2.2 | hsm_lmnda_val_t | 68 |
| 3.9.2.2 | hsm_do_auth_enc_new() | 57 | 3.13.3 | Function documentation | 68 |
| 3.10 | MAC | 58 | 3.13.3.1 | hsm_dev_getinfo() | 68 |
| 3.10.1 | Detailed description | 58 | 3.13.3.2 | hsm_get_lc_from_lmnda() | 68 |
| 3.10.2 | Data structure documentation | 58 | 3.14 | Write Fuse | 68 |
| 3.10.2.1 | struct open_svc_mac_args_t | 58 | 3.14.1 | Detailed description | 69 |
| 3.10.2.2 | struct op_mac_args_t | 59 | 3.14.2 | Data structure documentation | 69 |
| 3.10.3 | Macro definition documentation | 60 | 3.14.2.1 | struct op_dev_write_fuse_args_t | 69 |
| 3.10.3.1 | HSM_OP_MAC_FLAGS_MAC_VERIFICATION | 60 | 3.14.3 | Function documentation | 69 |
| 3.10.3.2 | HSM_OP_MAC_FLAGS_MAC_GENERATION | 60 | 3.14.3.1 | hsm_dev_write_fuse() | 69 |
| 3.10.3.3 | HSM_OP_MAC_FLAGS_PLAINTEXT_KEY | 60 | 3.15 | Generic Crypto: Asymmetric Crypto | 69 |
| 3.10.3.4 | HSM_OP_MAC_FLAGS_INIT | 60 | 3.15.1 | Detailed description | 70 |
| 3.10.3.5 | HSM_OP_MAC_FLAGS_UPDATE_DATA | 60 | 3.15.2 | Data structure documentation | 70 |
| 3.10.3.6 | HSM_OP_MAC_FLAGS_FINALIZE | 60 | 3.15.2.1 | struct op_gc_acrypto_args_t | 70 |
| 3.10.3.7 | HSM_OP_MAC_FLAGS_FINALIZE_VERIFY | 60 | 3.15.3 | Macro definition documentation | 71 |
| 3.10.3.8 | HSM_OP_MAC_FLAGS_ABORT | 60 | 3.15.3.1 | HSM_OP_GC_ACRYPTO_FLAGS_INPUT_DIGEST | 71 |
| 3.10.3.9 | HSM_OP_MAC_FLAGS_GET_CTX_SIZE | 61 | 3.15.3.2 | HSM_OP_GC_ACRYPTO_FLAGS_INPUT_MESSAGE | 71 |
| 3.10.3.10 | HSM_MAC_VERIFICATION_STATUS_FAILURE | 61 | 3.15.3.3 | HSM_GC_ACRYPTO_VERIFICATION_SUCCESS | 72 |
| 3.10.3.11 | HSM_OP_MAC_ONE_GO_FLAGS_MAC_VERIFICATION | 61 | 3.15.3.4 | HSM_GC_ACRYPTO_VERIFICATION_FAILURE | 72 |
| 3.10.3.12 | HSM_OP_MAC_ONE_GO_FLAGS_MAC_GENERATION | 61 | 3.15.4 | Typedef documentation | 72 |
| 3.10.3.13 | HSM_MAC_VERIFICATION_STATUS_SUCCESS | 61 | 3.15.4.1 | hsm_op_gc_acrypto_flags_t | 72 |
| 3.10.4 | Typedef documentation | 61 | 3.15.4.2 | hsm_gc_acrypto_verification_status_t | 72 |
| 3.10.4.1 | hsm_op_mac_flags_t | 61 | 3.15.5 | Enumeration type documentation | 72 |
| 3.10.4.2 | hsm_op_mac_one_go_flags_t | 61 | 3.15.5.1 | hsm_op_gc_acrypto_algo_t | 72 |
| 3.10.4.3 | hsm_mac_verification_status_t | 62 | 3.15.5.2 | hsm_gc_acrypto_op_mode_t | 72 |
| 3.10.4.4 | hsm_op_mac_algo_t | 62 | 3.15.6 | Function documentation | 72 |
| 3.10.4.5 | hsm_op_mac_one_go_algo_t | 62 | 3.15.6.1 | hsm_gc_acrypto() | 72 |
| 3.10.4.6 | op_mac_one_go_args_t | 62 | 3.16 | Generic Crypto Asymmetric Key Generate | 73 |
| | | | 3.16.1 | Detailed description | 73 |
| | | | 3.16.2 | Data structure documentation | 73 |
| | | | 3.16.2.1 | struct op_gc_akey_gen_args_t | 73 |

| | | | | | |
|----------|---|----|----------|--|-----|
| 3.16.3 | Function documentation | 73 | 3.21.2 | Data structure documentation | 93 |
| 3.16.3.1 | hsm_gc_akey_gen() | 73 | 3.21.2.1 | struct op_pub_key_recovery_args_t | 93 |
| 3.17 | Generic Crypto: Cipher | 74 | 3.21.3 | Function documentation | 93 |
| 3.17.1 | Detailed description | 74 | 3.21.3.1 | hsm_pub_key_recovery() | 93 |
| 3.17.2 | Data structure documentation | 74 | 3.22 | Key store | 94 |
| 3.17.2.1 | struct op_gc_cipher_args_t | 74 | 3.22.1 | Detailed description | 94 |
| 3.17.3 | Enumeration type documentation | 75 | 3.22.2 | Data structure documentation | 94 |
| 3.17.3.1 | hsm_op_gc_cipher_algo_t | 75 | 3.22.2.1 | struct open_svc_key_store_args_t | 94 |
| 3.17.3.2 | hsm_gc_cipher_op_mode_t | 75 | 3.22.2.2 | struct op_key_store_reprov_en_args_t | 95 |
| 3.17.4 | Function documentation | 75 | 3.22.3 | Macro definition documentation | 95 |
| 3.17.4.1 | hsm_gc_cipher () | 75 | 3.22.3.1 | HSM_SVC_KEY_STORE_FLAGS_CREATE | 95 |
| 3.18 | Generic Crypto: AEAD | 75 | 3.22.3.2 | HSM_SVC_KEY_STORE_FLAGS_SHARED | 95 |
| 3.18.1 | Detailed description | 76 | 3.22.3.3 | HSM_SVC_KEY_STORE_FLAGS_SET_MAC_LEN | 95 |
| 3.18.2 | Data structure documentation | 76 | 3.22.3.4 | HSM_SVC_KEY_STORE_FLAGS_MONOTONIC | 95 |
| 3.18.2.1 | struct op_gc_aead_args_t | 76 | 3.22.3.5 | HSM_SVC_KEY_STORE_FLAGS_STRICT_OPERATION | 96 |
| 3.18.3 | Enumeration type documentation | 76 | 3.22.4 | Typedef documentation | 96 |
| 3.18.3.1 | hsm_op_gc_aead_algo_t | 76 | 3.22.4.1 | hsm_svc_key_store_flags_t | 96 |
| 3.18.3.2 | hsm_gc_aead_op_mode_t | 76 | 3.22.5 | Function documentation | 96 |
| 3.18.4 | Function documentation | 76 | 3.22.5.1 | hsm_open_key_store_service() | 96 |
| 3.18.4.1 | hsm_gc_aead () | 76 | 3.22.5.2 | hsm_close_key_store_service() | 96 |
| 3.19 | Get Info | 77 | 3.22.5.3 | hsm_key_store_reprov_en() | 97 |
| 3.19.1 | Detailed description | 77 | 3.23 | Life cycle update | 97 |
| 3.19.2 | Data structure documentation | 77 | 3.23.1 | Detailed description | 97 |
| 3.19.2.1 | struct op_get_info_args_t | 77 | 3.23.2 | Data structure documentation | 97 |
| 3.19.3 | Function documentation | 78 | 3.23.2.1 | struct op_lc_update_msg_args_t | 97 |
| 3.19.3.1 | hsm_get_info() | 78 | 3.23.3 | Enumeration type documentation | 97 |
| 3.20 | Key exchange | 78 | 3.23.3.1 | hsm_lc_new_state_t | 97 |
| 3.20.1 | Detailed description | 79 | 3.23.4 | Function documentation | 98 |
| 3.20.2 | Data structure documentation | 79 | 3.23.4.1 | hsm_lc_update() | 98 |
| 3.20.2.1 | struct op_key_exchange_args_t | 79 | 3.24 | Configure monotonic counter | 98 |
| 3.20.2.2 | Field description - Input Content buffer | 80 | 3.24.1 | Detailed description | 98 |
| 3.20.3 | Macro definition documentation | 89 | 3.24.2 | Data structure documentation | 98 |
| 3.20.3.1 | HSM_OP_KEY_EXCHANGE_FLAGS_INPUT_SIGNED_CONTENT | 89 | 3.24.2.1 | struct op_dev_mono_counter_args_t | 98 |
| 3.20.3.2 | HSM_OP_KEY_EXCHANGE_FLAGS_INPUT_PLAINTEXT_CONTENT | 89 | 3.24.3 | Function documentation | 99 |
| 3.20.3.3 | HSM_OP_KEY_EXCHANGE_FLAGS_RETURN_KEY_IDS | 89 | 3.24.3.1 | hsm_err_t hsm_dev_mono_counter () | 99 |
| 3.20.3.4 | HSM_OP_KEY_EXCHANGE_FLAGS_RETURN_OUTPUT | 89 | 3.25 | Boot mode | 99 |
| 3.20.3.5 | HSM_OP_KEY_EXCHANGE_FLAGS_SALT_ZERO | 89 | 3.25.1 | Detailed description | 99 |
| 3.20.3.6 | HSM_OP_KEY_EXCHANGE_FLAGS_SALT_PEER_PUBKEY_HASH | 90 | 3.25.2 | Data structure documentation | 99 |
| 3.20.3.7 | HSM_OP_KEY_EXCHANGE_FLAGS_MONOTONIC | 90 | 3.25.2.1 | struct op_get_boot_mode_args_t | 99 |
| 3.20.3.8 | HSM_OP_KEY_EXCHANGE_FLAGS_STRICT_OPERATION | 90 | 3.25.2.2 | struct op_set_boot_mode_args_t | 100 |
| 3.20.4 | Typedef documentation | 90 | 3.25.3 | Function documentation | 100 |
| 3.20.4.1 | hsm_op_key_exchange_flags_t | 90 | 3.25.3.1 | crrm_err_t crrm_get_boot_mode () | 100 |
| 3.20.5 | Enumeration type documentation | 90 | 3.25.3.2 | crrm_err_t crrm_set_boot_mode () | 100 |
| 3.20.5.1 | hsm_op_key_exchange_algo_t | 90 | 3.26 | Get AWDT status | 101 |
| 3.20.5.2 | hsm_op_key_derivation_algo_t | 91 | 3.26.1 | Detailed description | 101 |
| 3.20.5.3 | hsm_op_key_derivation_tls1_3_algo_t | 91 | 3.26.2 | Data structure documentation | 101 |
| 3.20.5.4 | hsm_op_key_derivation_tls1_2_algo_t | 91 | 3.26.2.1 | struct AWDT_timer | 101 |
| 3.20.6 | Function documentation | 92 | 3.26.2.2 | struct op_get_AWDT_status_args_t | 102 |
| 3.20.6.1 | hsm_key_exchange | 92 | 3.26.3 | Function documentation | 102 |
| 3.21 | Public key recovery | 93 | 3.26.3.1 | crrm_err_t crrm_get_AWDT_status () | 102 |
| 3.21.1 | Detailed description | 93 | 3.27 | Get nonce | 102 |
| | | | 3.27.1 | Detailed description | 102 |
| | | | 3.27.2 | Data structure documentation | 103 |
| | | | 3.27.2.1 | struct nonce_info | 103 |

| | | | | | |
|-----------------------------|--|---------------------|---------------------------------------|---|-----|
| 3.27.2.2 | struct op_get_nonce_args_t | 103 | 3.33.5.1 | hsm_err_t hsm_asymmetric_enc_open () | 112 |
| 3.27.3 | Function documentation | 103 | 3.33.5.2 | hsm_err_t hsm_asymmetric_enc_close () | 113 |
| 3.27.3.1 | crmm_err_t crmm_get_nonce () | 103 | 3.33.5.3 | hsm_err_t hsm_asymmetric_enc () | 113 |
| 3.28 | Handle | 103 | 3.34 | Generate ELE Key Blob | 113 |
| 3.28.1 | Detailed description | 104 | 3.34.1 | Detailed description | 114 |
| 3.28.2 | Typedef documentation | 104 | 3.34.2 | Data structure documentation | 114 |
| 3.28.2.1 | typedef uint32_t crmm_hdl_t | 104 | 3.34.2.1 | struct op_gen_ele_key_blob_otfad_key_ | |
| 3.28.2.2 | typedef hsm_err_t crmm_err_t | 104 | conf_t | 114 | |
| 3.28.3 | Function documentation | 104 | 3.34.2.2 | struct op_gen_ele_key_blob_format_t | 114 |
| 3.28.3.1 | crmm_hdl_t get_session_handle (void) | 104 | 3.34.2.3 | struct op_gen_ele_key_blob_msg_args_t | 115 |
| 3.29 | Refresh AWDT | 104 | 3.34.3 | Macro definition documentation | 115 |
| 3.29.1 | Detailed description | 104 | 3.34.3.1 | GEN_ELE_KEY_BLOB_OVERHEAD_SZ | 115 |
| 3.29.2 | Data structure documentation | 104 | 3.34.3.2 | GEN_ELE_KEY_BLOB_FORMAT_TAG | 116 |
| 3.29.2.1 | struct signature_data | 104 | 3.34.3.3 | GEN_ELE_KEY_BLOB_FORMAT_VERS | 116 |
| 3.29.2.2 | struct request_data | 105 | 3.34.3.4 | HSM_OP_GEN_ELE_KEY_BLOB_FLAG_ | |
| 3.29.2.3 | struct op_refresh_AWDT_args_t | 105 | BLOB_TYPE_DEK | 116 | |
| 3.29.3 | Function documentation | 105 | 3.34.3.5 | HSM_OP_GEN_ELE_KEY_BLOB_FLAG_ | |
| 3.29.3.1 | crmm_err_t crmm_refresh_AWDT () | 105 | BLOB_TYPE_OTFAD_CONF | 116 | |
| 3.30 | Get event | 106 | 3.34.3.6 | OP_GEN_ELE_KEY_BLOB_PAYLOAD_ | |
| 3.30.1 | Detailed description | 106 | OTFAD_SZ | 116 | |
| 3.30.2 | Data structure documentation | 106 | 3.34.3.7 | OP_GEN_ELE_KEY_BLOB_PAYLOAD_ | |
| 3.30.2.1 | struct hsm_event_report_info_t | 106 | DEK_AES128_CBC_SZ | 116 | |
| 3.30.2.2 | struct op_get_event_msg_args_t | 106 | 3.34.3.8 | OP_GEN_ELE_KEY_BLOB_PAYLOAD_ | |
| 3.30.3 | Function documentation | 107 | DEK_AES192_CBC_SZ | 116 | |
| 3.30.3.1 | hsm_get_event () | 107 | 3.34.3.9 | OP_GEN_ELE_KEY_BLOB_PAYLOAD_ | |
| 3.31 | Blob Unblob | 107 | DEK_AES256_CBC_SZ | 116 | |
| 3.31.1 | Detailed description | 107 | 3.34.3.10 | OP_GEN_ELE_KEY_BLOB_PAYLOAD_ | |
| 3.31.2 | Data structure documentation | 107 | DEK_SM4_CBC_SZ | 117 | |
| 3.31.2.1 | struct op_blob_unblob_msg_args_t | 107 | 3.34.3.11 | OP_GEN_ELE_KEY_BLOB_PAYLOAD_ | |
| 3.31.3 | Macro definition documentation | 108 | MAX_SZ | 117 | |
| 3.31.3.1 | HSM_OP_BLOB_UNBLOB_FLAG_USE_ | | 3.34.3.12 | OP_GEN_ELE_KEY_BLOB_FORMAT_ | |
| MU_IN_DER_CTX | 108 | SZ_WO_PAYLOAD | 117 | | |
| 3.31.3.2 | HSM_OP_BLOB_UNBLOB_FLAG_USE_ | | 3.34.4 | Typedef documentation | 117 |
| BOOT_STAGE_IN_DER_CTX | 108 | 3.34.4.1 | hsm_op_gen_ele_key_blob_flags_t | 117 | |
| 3.31.3.3 | HSM_OP_BLOB_UNBLOB_FLAG_USE_ | | 3.34.5 | Enumeration type documentation | 117 |
| UUID_IN_DER_CTX | 108 | 3.34.5.1 | hsm_op_gen_ele_key_blob_algo_t | 117 | |
| 3.31.4 | Typedef documentation | 108 | 3.34.6 | Function documentation | 117 |
| 3.31.4.1 | hsm_op_blob_unblob_flags_t | 108 | 3.34.6.1 | hsm_gen_ele_key_blob | 117 |
| 3.31.5 | Enumeration type documentation | 109 | 3.35 | SE lib service | 118 |
| 3.31.5.1 | hsm_op_blob_unblob_mode_t | 109 | 3.35.1 | Detailed description | 118 |
| 3.31.6 | Function documentation | 109 | 3.35.2 | Data structure documentation | 118 |
| 3.31.6.1 | hsm_blob_unblob () | 109 | 3.35.2.1 | struct open_se_lib_serv_args_t | 118 |
| 3.32 | Boot stage | 109 | 3.35.3 | Typedef documentation | 118 |
| 3.32.1 | Detailed description | 109 | 3.35.3.1 | se_lib_hdl_t | 118 |
| 3.32.2 | Data structure documentation | 109 | 3.35.4 | Function documentation | 118 |
| 3.32.2.1 | struct op_boot_stage_ctx_args_t | 109 | 3.35.4.1 | se_lib_open_service () | 118 |
| 3.32.3 | Enumeration type documentation | 110 | 3.35.4.2 | se_lib_close_service () | 119 |
| 3.32.3.1 | hsm_op_code_bs_ctx_t | 110 | 3.36 | Global information | 119 |
| 3.32.4 | Function documentation | 110 | 3.36.1 | Detailed description | 120 |
| 3.32.4.1 | hsm_err_t hsm_bsctx_ops () | 110 | 3.36.2 | Data structure documentation | 120 |
| 3.33 | Asymmetric crypto encryption | 110 | 3.36.2.1 | struct global_info_s | 120 |
| 3.33.1 | Detailed description | 111 | 3.36.3 | Function documentation | 121 |
| 3.33.2 | Data structure documentation | 111 | 3.36.3.1 | populate_global_info() | 121 |
| 3.33.2.1 | struct op_asymmetric_enc_open_args_t | 111 | 3.36.3.2 | show_global_info() | 121 |
| 3.33.2.2 | struct op_asymmetric_enc_args_t | 111 | 3.36.3.3 | is_global_info_populated() | 121 |
| 3.33.3 | Macro definition documentation | 112 | 3.36.3.4 | hsm_get_dev_attest_api_ver() | 121 |
| 3.33.4 | Enumeration type documentation | 112 | 3.36.3.5 | se_get_board_type() | 121 |
| 3.33.4.1 | hsm_op_asymmetric_algo_t | 112 | 3.36.3.6 | se_get_soc_id() | 121 |
| 3.33.5 | Function documentation | 112 | 3.36.3.7 | se_get_soc_rev() | 122 |

| | | |
|-----------|---|------------|
| 3.36.3.8 | se_get_chip_lifecycle() | 122 |
| 3.36.3.9 | se_get_fips_mode() | 122 |
| 3.36.3.10 | se_get_lib_newness_ver() | 122 |
| 3.36.3.11 | se_get_lib_major_ver() | 122 |
| 3.36.3.12 | se_get_lib_minor_ver() | 122 |
| 3.36.3.13 | se_get_nvm_newness_ver() | 122 |
| 3.36.3.14 | se_get_nvm_major_ver() | 122 |
| 3.36.3.15 | se_get_nvm_minor_ver() | 123 |
| 3.36.3.16 | se_get_commit_id() | 123 |
| 3.36.3.17 | se_get_lib_version() | 123 |
| 3.36.3.18 | se_get_nvm_version() | 123 |
| 3.36.3.19 | get_board_type_str() | 123 |
| 3.36.3.20 | get_soc_id_str() | 123 |
| 3.36.3.21 | get_soc_rev_str() | 124 |
| 3.36.3.22 | get_soc_lf_str() | 124 |
| 3.36.3.23 | se_get_info() | 124 |
| 3.36.3.24 | se_get_soc_info() | 124 |
| 3.36.4 | Variable documentation | 124 |
| 3.36.4.1 | global_info | 124 |
| 3.37 | Common algorithms | 125 |
| 3.38 | Error codes | 127 |
| 3.38.1 | Detailed description | 128 |
| 3.38.2 | Enumeration type documentation | 128 |
| 3.38.2.1 | enum hsm_err_t | 128 |
| 3.38.2.2 | enum se_lib_err_t | 129 |
| 3.38.2.3 | enum lib_err_t | 129 |
| 3.39 | i.MX 8ULP | 130 |
| 3.40 | i.MX 91 | 130 |
| 3.41 | i.MX 93 | 130 |
| 3.42 | i.MX 95 | 130 |
| 3.43 | i.MX 943 | 130 |
| 4 | Note About the Source Code in the Document | 131 |
| 5 | Revision History | 131 |
| | Legal information | 133 |

Please be aware that important notices concerning this document and the product(s) described herein, have been included in section 'Legal information'.
