HSE Service API Reference Manual

For S32G2XX
v0.2.51.0

Revision 4f9f45a1ba
Apr 2024

NXP Semiconductors

Contents

Title

Section number Page

1 Introduction

This document describes the parameters of the NXP Native services and is an addendum to the HSE Firmware Reference Manual (available at NXP Docstore) which contains details on how to install, configure and use the HSE subsystem.

1.1 HSE Messages Guidelines

- The address parameters can be passed as 32 or 40 bit addresses, depending on HSE firmware support (if 64bit addressing is enabled and if the device supports 40 bit addressing mode)
- A service request can be accessible in one-pass or streaming (SUF) mode. In case of streaming mode "three" steps (calls) are needed: START, UPDATE, FINISH. Note that for each streaming step (START, UPDATE or FINISH), some of the parameters are mandatory or optional.
- The streaming mode operation begins with the START step using a specific HSE interface ID and stream ID. The UPDATES and FINISH steps shall be sent on the same HSE interface ID and stream ID as the START step; otherwise an error will be signaled.
- If a streaming operation produces an error, the stream is to be considered invalid; a stream can always be reset by a new start command.

2 Host Interface

2.1 HSE GPR Status

Macros

Type: (implicit C type)	
Name	**Value**
HSE_GPR_STATUS_ADDRESS	0x4007C928UL

Type: hseTamperConfigStatus_t	
Name	**Value**
HSE_CMU_TAMPER_CONFIG_STATUS	1U << 0U
HSE_PHYSICAL_TAMPER_CONFIG_STATUS	1U << 1U
HSE_TEMP_SENSOR_VIO_CONFIG_STATUS	1U << 2U

Typedefs

- typedef uint32_t hseTamperConfigStatus_t

Macro Definition Documentation

HSE_GPR_STATUS_ADDRESS

```
#define HSE_GPR_STATUS_ADDRESS (0x4007C928UL)
```

HSE-GPR REG3 is in Security subsystem registers Description (refer to hseTamperConfigStatus_t)

HSE_CMU_TAMPER_CONFIG_STATUS

```
#define HSE_CMU_TAMPER_CONFIG_STATUS ((hseTamperConfigStatus_t)1U << 0U)
```

HSE-GPR REG3[0]- this bit is set when the CMU tamper is configured:

- For HSE_H, the clock must be configured in this range: 10Mhz < clock frequency < 420Mhz.
- For HSE_B, the clock must be configured in this range: 3Mhz < clock frequency < 126Mhz.
- For HSE_M, the clock must be configured in this range:
  - s32r41x: 45.6Mhz < clock frequency < 420Mhz.

HSE_PHYSICAL_TAMPER_CONFIG_STATUS

```
#define HSE_PHYSICAL_TAMPER_CONFIG_STATUS ((hseTamperConfigStatus_t)1U << 1U)
```

HSE-GPR REG3[1]- this bit is set when the physical tamper is configured. Note that the application must configure SIUL2 Pads before enabling the tamper.

HSE_TEMP_SENSOR_VIO_CONFIG_STATUS

```
#define HSE_TEMP_SENSOR_VIO_CONFIG_STATUS ((hseTamperConfigStatus_t)1U << 2U)
```

HSE-GPR REG3[2] this bit is set when the temperature Sensor violation is configured.

Typedef Documentation

hseTamperConfigStatus_t

```
typedef uint32_t hseTamperConfigStatus_t
```

HSE Tamper Config Status bits (register address is HSE_ GPR_ STATUS_ ADDRESS)

This status is updated when a tamper is configured by HSE during initialization or via attribute. The host can get the HSE Tamper Config Status reading the HSE_ GPR_ STATUS_ ADDRESS register. In this way, the host to check what tampers are configured. The HSE_ GPR_ STATUS_ ADDRESS register is read-only.

Note

- For HSE_H/S32R41, HSE-GPR REG3 used.
- For HSE_B, CONFIG_REG4 used.

2.2 About the Host Interface

This section contains information on the available services accepted by the firmware.

The firmware accepts commands in the form of service descriptors. Data types and values relevant for the services are also listed. One-time settings or information about the state of the system are accessible via attributes. The attributes are also listed below.

2.3 HSE Service Descriptor

Data Structures

- struct hseSrvDescriptor_t
- union hseSrvDescriptor_t.hseSrv

Macros

Type: hseSrvId_t	
Name	**Value**
HSE_SRV_ID_SET_ATTR	HSE_SRV_VER_0 | 0x00000001UL
HSE_SRV_ID_GET_ATTR	HSE_SRV_VER_0 | 0x00A50002UL
HSE_SRV_ID_SELF_TEST	HSE_SRV_VER_0 | 0x00000003UL
HSE_SRV_ID_CANCEL	HSE_SRV_VER_0 | 0x00A50004UL
HSE_SRV_ID_FIRMWARE_UPDATE	HSE_SRV_VER_0 | 0x00000005UL
HSE_SRV_ID_SYS_AUTH_REQ	HSE_SRV_VER_0 | 0x00000006UL
HSE_SRV_ID_SYS_AUTH_RESP	HSE_SRV_VER_0 | 0x00000007UL
HSE_SRV_ID_BOOT_DATA_IMAGE_SIGN	HSE_SRV_VER_0 | 0x00000008UL
HSE_SRV_ID_BOOT_DATA_IMAGE_VERIFY	HSE_SRV_VER_0 | 0x00000009UL
HSE_SRV_ID_IMPORT_EXPORT_STREAM_CTX	HSE_SRV_VER_0 | 0x00A5000AUL
HSE_SRV_ID_PUBLISH_SYS_IMAGE	HSE_SRV_VER_0 | 0x00000011UL
HSE_SRV_ID_GET_SYS_IMAGE_SIZE	HSE_SRV_VER_0 | 0x00000012UL
HSE_SRV_ID_VERIFY_SYS_IMAGE	HSE_SRV_VER_0 | 0x00000018UL
HSE_SRV_ID_PUBLISH_LOAD_CNT_TBL	HSE_SRV_VER_0 | 0x00000013UL
HSE_SRV_ID_INSTALL_OTFAD_CTX	HSE_SRV_VER_0 | 0x00000014UL
HSE_SRV_ID_ACTIVATE_OTFAD_CTX	HSE_SRV_VER_0 | 0x00000015UL
HSE_SRV_ID_GET_OTFAD_CTX	HSE_SRV_VER_0 | 0x00000016UL
HSE_SRV_ID_PREPARE_FOR_STANDBY	HSE_SRV_VER_0 | 0x00A50017UL
HSE_SRV_ON_DEMAND_ANTI_RBC_UPDATE	HSE_SRV_VER_0 | 0x00000022UL
HSE_SRV_ID_FIRMWARE_VERIFY	HSE_SRV_VER_0 | 0x00000030UL
HSE_SRV_ID_CONFIG_COUNTER	HSE_SRV_VER_0 | 0x00000052UL
HSE_SRV_ID_LOAD_ECC_CURVE	HSE_SRV_VER_0 | 0x00000100UL
HSE_SRV_ID_FORMAT_KEY_CATALOGS	HSE_SRV_VER_0 | 0x00000101UL
HSE_SRV_ID_ERASE_KEY	HSE_SRV_VER_0 | 0x00000102UL
HSE_SRV_ID_GET_KEY_INFO	HSE_SRV_VER_0 | 0x00A50103UL
HSE_SRV_ID_IMPORT_KEY	HSE_SRV_VER_0 | 0x00000104UL
HSE_SRV_ID_EXPORT_KEY	HSE_SRV_VER_0 | 0x00000105UL
HSE_SRV_ID_KEY_GENERATE	HSE_SRV_VER_0 | 0x00000106UL
HSE_SRV_ID_DH_COMPUTE_SHARED_SECRET	HSE_SRV_VER_0 | 0x00000107UL
HSE_SRV_ID_KEY_DERIVE	HSE_SRV_VER_0 | 0x00000108UL
HSE_SRV_ID_KEY_DERIVE_COPY	HSE_SRV_VER_0 | 0x00000109UL
HSE_SRV_ID_KEY_VERIFY	HSE_SRV_VER_0 | 0x0000010BUL
HSE_SRV_ID_EXTEND_KEY_CATALOG	HSE_SRV_VER_0 | 0x0000010CUL

HSE_SRV_ID_SHE_LOAD_KEY	HSE_SRV_VER_0 | 0x0000A101UL
HSE_SRV_ID_SHE_LOAD_PLAIN_KEY	HSE_SRV_VER_0 | 0x0000A102UL
HSE_SRV_ID_SHE_EXPORT_RAM_KEY	HSE_SRV_VER_0 | 0x0000A103UL
HSE_SRV_ID_SHE_GET_ID	HSE_SRV_VER_0 | 0x0000A104UL
HSE_SRV_ID_SHE_BOOT_OK	HSE_SRV_VER_0 | 0x0000A105UL
HSE_SRV_ID_SHE_BOOT_FAILURE	HSE_SRV_VER_0 | 0x0000A106UL
HSE_SRV_ID_HASH	HSE_SRV_VER_0 | 0x00A50200UL
HSE_SRV_ID_MAC	HSE_SRV_VER_0 | 0x00A50201UL
HSE_SRV_ID_FAST_CMAC	HSE_SRV_VER_0 | 0x00A50202UL
HSE_SRV_ID_SYM_CIPHER	HSE_SRV_VER_0 | 0x00A50203UL
HSE_SRV_ID_AEAD	HSE_SRV_VER_0 | 0x00A50204UL
HSE_SRV_ID_XTS_AES_CIPHER	HSE_SRV_VER_0 | 0x00A50205UL
HSE_SRV_ID_SIGN	HSE_SRV_VER_0 | 0x00000206UL
HSE_SRV_ID_RSA_CIPHER	HSE_SRV_VER_0 | 0x00000207UL
HSE_SRV_ID_AUTHENC	HSE_SRV_VER_0 | 0x00A50208UL
HSE_SRV_ID_CRC32	HSE_SRV_VER_0 | 0x00A50209UL
HSE_SRV_ID_SIPHASH	HSE_SRV_VER_0 | 0x0000020AUL
HSE_SRV_ID_CMAC_WITH_COUNTER	HSE_SRV_VER_0 | 0x00A5020BUL
HSE_SRV_ID_GET_RANDOM_NUM	HSE_SRV_VER_0 | 0x00000300UL
HSE_SRV_ID_INCREMENT_COUNTER	HSE_SRV_VER_0 | 0x00A50400UL
HSE_SRV_ID_READ_COUNTER	HSE_SRV_VER_0 | 0x00A50401UL
HSE_SRV_ID_SMR_ENTRY_INSTALL	HSE_SRV_VER_0 | 0x00000501UL
HSE_SRV_ID_SMR_VERIFY	HSE_SRV_VER_0 | 0x00000502UL
HSE_SRV_ID_CORE_RESET_ENTRY_INSTALL	HSE_SRV_VER_0 | 0x00000503UL
HSE_SRV_ID_ON_DEMAND_CORE_RESET	HSE_SRV_VER_0 | 0x00000504UL
HSE_SRV_ID_SMR_ENTRY_ERASE	HSE_SRV_VER_0 | 0x00000505UL
HSE_SRV_ID_CORE_RESET_ENTRY_ERASE	HSE_SRV_VER_0 | 0x00000506UL

HSE service descriptor details

Each service is identified by a unique ID (called service ID). Each service ID identifies a service from the hseSrvDescriptor_t::hseSrv union. The service ID contains 4 bytes that specify the following:
- byte[0]: service index (0..255)
- byte[1]: service class index (0..255)(see more details below)
- byte[2]: 0x00 - service can be canceled; 0xA5 - service can not be canceled
- byte[3]: service version (0..255)

The following service classes are defined:
- Administrative services (e.g set/get an HSE attribute, self-test, cancel service etc.)
- Key management services (e.g key generation, Diffie-Hellman shared secret computation, import/export

key etc.)
- Crypto services (e.g. HASH, MAC generate/verify, encryption/decryption, signature generate/verify)
- Random number
- Monotonic counters
- Secure boot and memory checking services (Secure Memory Regions (SMR) and Core reset(CR) services)
- Network Crypto services (IPsec).

Note

- The services guarded by HSE_SPT_FLASHLESS_DEV macro are available only for HSE_H/M (flashless devices).
- The services guarded by HSE_SPT_INTERNAL_FLASH_DEV macro are available only for HSE_B (devices with internal flash).

Data Structure Documentation

struct hseSrvDescriptor_t

Data Fields

Type	Name	Description
hseSrvId_t	srvId	The service ID of the HSE message.
hseSrvMetaData_t	srvMetaData	The service metadata (e.g. priority)
union hseSrvDescriptor_t.hseSrv	hseSrv	The service ID will identify a service in the following union.

union hseSrvDescriptor_t.hseSrv

The service ID will identify a service in the following union.

Data Fields

Type	Name	Description
hseSetAttrSrv_t	setAttrReq	Request to set a HSE attribute (note that some attributes are read only)
hseGetAttrSrv_t	getAttrReq	Request to get a HSE attribute.
hseSelfTestSrv_t	selfTestReq	Request to execute a HSE self test procedure.
hseCancelSrv_t	cancelSrvReq	Request to cancel a one-pass or streaming service on a specific channel.

Data Fields

Type	Name	Description
hseFirmwareUpdateSrv_t	firmwareUpdateReq	Request to HSE firmware update.
hseFirmwareVerifySrv_t	firmwareVerifyReq	Request to Verify the Blue or Pink FW image in SRAM or external flash.
hseSysAuthorizationReqSrv_t	sysAuthorizationReq	Perform an SYS Authorization Request.
hseSysAuthorizationRespSrv_t	sysAuthorizationResp	Send the SYS Authorization Response.
hseBootDataImageSignSrv_t	bootDataImageSignReq	Request to generate the Signature for Boot Data images (e.g. for HSE_H/M, IVT/DCD/ST/LPDDR4(ZSE devices)/AppBSB image; for HSE_B, IVT/XRDC/AppBSB image)
hseBootDataImageVerifySrv_t	bootDataImageSigVerifyReq	Request to verify the Signature for Boot Data images (e.g. for HSE_H/M, IVT/DCD/ST/LPDDR4(ZSE devices)/AppBSB image; for HSE_B, IVT/XRDC/AppBSB image)
hseImportExportStreamCtxSrv_t	importExportStreamCtx	Request to import/export a streaming context.
hsePublishSysImageSrv_t	publishSysImageReq	Request to Publish a NVM SYS-IMAGE (only for HSE_H/M).
hseGetSysImageSizeSrv_t	getSysImageSizeReq	Request to get SYS-IMAGE size (only for HSE_H/M).
hseVerifySysImageSrv_t	verifySysImageReq	Request to Verify SYS-IMAGE after it is stored in external flash (only for HSE_H/M).
hsePublishLoadCntTblSrv_t	publishLoadCntTblReq	Request to publish/load the NVM container for the Monotonic Counter table (only for HSE_H/M).
hseInstallOtfadContextSrv_t	installOtfadReq	Request to install an OTFAD context (only for HSE_H/M).
hseActivateOtfadContextSrv_t	activateOtfadReq	Request to activate on-demand an already installed OTFAD context (only for HSE_H/M).
hseGetOtfadContextSrv_t	getOtfadCtxReq	Request to get OTFAD context information (only for HSE_H/M).
hsePrepareForStandBySrv_t	prepareForStandByReq	Request HSE to prepare for Stand-By mode (only for HSE_H/M).
hseOnDemandAntiRbcUpdateSrv_t	onDemandAntiRbcUpdateReq	Request on-demand an update of the anti-rollback counter.

Data Fields

Type	Name	Description
hseLoadEccCurveSrv_t	loadEccCurveReq	Request to load an ECC curve.
hseFormatKeyCatalogsSrv_t	formatKeyCatalogsReq	Format the key catalogs.
hseEraseKeySrv_t	eraseKeyReq	Request to erase NVM/RAM key(s).
hseGetKeyInfoSrv_t	getKeyInfoReq	Request to get key information (flags)
hseImportKeySrv_t	importKeyReq	Request to import a key.
hseExportKeySrv_t	exportKeyReq	Request to export a key.
hseKeyVerifySrv_t	verifyKeyReq	Request to verify a key.
hseKeyGenerateSrv_t	keyGenReq	Request to generate a key (e.g. sym random key, rsa key pair etc.) .
hseDHComputeSharedSecretSrv_t	dhComputeSecretReq	Request a ECC Diffie-Hellman Compute shared secret.
hseKeyDeriveSrv_t	keyDeriveReq	Request key derivation function.
hseKeyDeriveCopyKeySrv_t	keyDeriveCopyKeyReq	Request to copy a key from the derived key material.
hseExtendKeyCatalogSrv_t	extendKeyCatalogReq	Request to extend the NVM or RAM key catalog format.
hseSheLoadKeySrv_t	sheLoadKeyReq	Request to load a SHE key using memory update protocol (as per SHE specification)
hseSheLoadPlainKeySrv_t	sheLoadPlainKeyReq	Request to load the SHE RAM key from plain text (as per SHE specification)
hseSheExportRamKeySrv_t	sheExportRamKeyReq	Request to export the SHE RAM key (as per SHE specification)
hseSheGetIdSrv_t	sheGetIdReq	Request to get UID (as per SHE specification)
hseHashSrv_t	hashReq	Request a HASH.
hseMacSrv_t	macReq	Request to generate/verify a MAC.
hseFastCMACSrv_t	fastCmacReq	Request to FAST generate/verify a CMAC.
hseCmacWithCounterSrv_t	cmacWithCounterReq	Request to generate/verify a CMAC with counter.

Data Fields

Type	Name	Description
hseSymCipherSrv_t	symCipherReq	Request a Symmetric Cipher operation.
hseAeadSrv_t	aeadReq	Request an AEAD operation.
hseXtsAesCipherSrv_t	xtsAesCipherReq	Request a XTS AES Cipher operation.
hseSignSrv_t	signReq	Request a Digital Signature Generation/Verification.
hseRsaCipherSrv_t	rsaCipherReq	Request a RSA Cipher (Encryption/Decryption) operation.
hseAuthEncSrv_t	authEncReq	Request an AuthEncryption operation (encrypt/decrypt + authenticate)
hseCrc32Srv_t	crc32Req	Request to initialize an CRC computation.
hseSipHashSrv_t	sipHashReq	Request to generate/verify a SipHash.
hseGetRandomNumSrv_t	getRandomNumReq	Request to random number generation.
hseIncrementCounterSrv_t	incCounterReq	Request to increment a monotonic counter.
hseReadCounterSrv_t	readCounterReq	Request to read a monotonic counter.
hseConfigSecCounterSrv_t	configSecCounter	Request to configure a secure counter.
hseSmrEntryInstallSrv_t	smrEntryInstallReq	Request to install a Secure Memory Region (SMR) table entry.
hseSmrVerifySrv_t	smrVerifyReq	Request to verify a Secure Memory Region (SMR) table entry.
hseSmrEntryEraseSrv_t	smrEntryEraseReq	Request to erase a Secure Memory Region (SMR) table entry.
hseCrEntryInstallSrv_t	crEntryInstallReq	Request to install a Core Reset (CR) table entry.
hseCrOnDemandBootSrv_t	crOnDemandBootReq	Request to release a Core Reset (CR) table entry.
hseCrEntryEraseSrv_t	crEntryEraseReq	Request to erase a Core Reset (CR) table entry.

Macro Definition Documentation

HSE_SRV_ID_SET_ATTR

```
#define HSE_SRV_ID_SET_ATTR ((hseSrvId_t)( HSE_SRV_VER_0| 0x00000001UL))
```

Set HSE attribute. Data structure used: hseSetAttrSrv_t.

HSE_SRV_ID_GET_ATTR

```
#define HSE_SRV_ID_GET_ATTR ((hseSrvId_t)( HSE_SRV_VER_0| 0x00A50002UL))
```

Get HSE attribute. Data structure used: hseGetAttrSrv_t.

HSE_SRV_ID_SELF_TEST

```
#define HSE_SRV_ID_SELF_TEST ((hseSrvId_t)( HSE_SRV_VER_0| 0x00000003UL))
```

Self-test. Data structure used: hseSelfTestSrv_t.

HSE_SRV_ID_CANCEL

```
#define HSE_SRV_ID_CANCEL ((hseSrvId_t)( HSE_SRV_VER_0| 0x00A50004UL))
```

Cancel a one-pass or streaming service on a specific channel. Data structure used: hseCancelSrv_t.

HSE_SRV_ID_FIRMWARE_UPDATE

```
#define HSE_SRV_ID_FIRMWARE_UPDATE ((hseSrvId_t)( HSE_SRV_VER_0| 0x00000005UL))
```

HSE firmware update. Data structure used: hseFirmwareUpdateSrv_t.

HSE_SRV_ID_SYS_AUTH_REQ

```
#define HSE_SRV_ID_SYS_AUTH_REQ ((hseSrvId_t)( HSE_SRV_VER_0| 0x00000006UL))
```

Perform a SYS Authorization request. Data structure used: hseSysAuthorizationReqSrv_t.

HSE_SRV_ID_SYS_AUTH_RESP

```
#define HSE_SRV_ID_SYS_AUTH_RESP ((hseSrvId_t)( HSE_SRV_VER_0| 0x00000007UL))
```

Send the SYS Authorization response. Data structure used: hseSysAuthorizationRespSrv_t.

HSE_SRV_ID_BOOT_DATA_IMAGE_SIGN

```
#define HSE_SRV_ID_BOOT_DATA_IMAGE_SIGN ((hseSrvId_t)( HSE_SRV_VER_0|
0x00000008UL))
```

Boot Data image sign (e.g. for HSE_H/M, IVT/DCD/ST/LPDDR4(S32Z/E devices)/AppBSB image; for HSE_B, IVT/AppBSB image). Data structure used: hseBootDataImageSignSrv_t.

HSE_SRV_ID_BOOT_DATA_IMAGE_VERIFY

```
#define HSE_SRV_ID_BOOT_DATA_IMAGE_VERIFY ((hseSrvId_t)( HSE_SRV_VER_0|
0x00000009UL))
```

Boot Data images verify (e.g. for HSE_H/M, IVT/DCD/ST/LPDDR4(S32Z/E devices)/AppBSB image; for HSE_B, IVT/AppBSB image). Data structure used: hseBootDataImageVerifySrv_t.

HSE_SRV_ID_IMPORT_EXPORT_STREAM_CTX

```
#define HSE_SRV_ID_IMPORT_EXPORT_STREAM_CTX ((hseSrvId_t)( HSE_SRV_VER_0|
0x00A5000AUL))
```

Import/Export Streaming Context. Data structure used: hseImportExportStreamCtxSrv_t.

HSE_SRV_ID_PUBLISH_SYS_IMAGE

```
#define HSE_SRV_ID_PUBLISH_SYS_IMAGE ((hseSrvId_t)( HSE_SRV_VER_0| 0x00000011UL))
```

Publish the NVM SYS-IMAGE. Data structure used: hsePublishSysImageSrv_t.

HSE_SRV_ID_GET_SYS_IMAGE_SIZE

```
#define HSE_SRV_ID_GET_SYS_IMAGE_SIZE ((hseSrvId_t)( HSE_SRV_VER_0| 0x00000012UL))
```

Get the SYS-IMAGE size. Data structure used: hseGetSysImageSizeSrv_t.

HSE_SRV_ID_VERIFY_SYS_IMAGE

```
#define HSE_SRV_ID_VERIFY_SYS_IMAGE ((hseSrvId_t)( HSE_SRV_VER_0| 0x00000018UL))
```

Verify SYS-IMAGE. Data structure used: hseVerifySysImageSrv_t.

HSE_SRV_ID_PUBLISH_LOAD_CNT_TBL

```
#define HSE_SRV_ID_PUBLISH_LOAD_CNT_TBL ((hseSrvId_t)( HSE_SRV_VER_0|
0x00000013UL))
```

Request to publish/load the NVM container for the Monotonic Counter table (only for HSE_H/M). Data structure used: hsePublishLoadCntTblSrv_t.

HSE_SRV_ID_INSTALL_OTFAD_CTX

```
#define HSE_SRV_ID_INSTALL_OTFAD_CTX ((hseSrvId_t)( HSE_SRV_VER_0| 0x00000014UL))
```

Install an On-The-Fly AES Decryption (OTFAD) context (only for HSE_H/M). Data structure used: hseInstallOtfadContextSrv_t.

HSE_SRV_ID_ACTIVATE_OTFAD_CTX

```
#define HSE_SRV_ID_ACTIVATE_OTFAD_CTX ((hseSrvId_t)( HSE_SRV_VER_0| 0x00000015UL))
```

Activate on-demand OTFAD context (only for HSE_H/M). Data structure used: hseActivateOtfadContextSrv_t.

HSE_SRV_ID_GET_OTFAD_CTX

```
#define HSE_SRV_ID_GET_OTFAD_CTX ((hseSrvId_t)( HSE_SRV_VER_0| 0x00000016UL))
```

Get OTFAD context information (only for HSE_H/M). Data structure used: hseGetOtfadContextSrv_t.

HSE_SRV_ID_PREPARE_FOR_STANDBY

```
#define HSE_SRV_ID_PREPARE_FOR_STANDBY ((hseSrvId_t)( HSE_SRV_VER_0| 0x00A50017UL))
```

Prepare HSE before system goes to Stand-By mode (only for HSE_H/M). Data structure used: hsePrepareForStandBySrv_t.

HSE_SRV_ON_DEMAND_ANTI_RBC_UPDATE

```
#define HSE_SRV_ON_DEMAND_ANTI_RBC_UPDATE ((hseSrvId_t)( HSE_SRV_VER_0|
0x00000022UL))
```

Trigger on demand anti-rollback counter updates in fuses. Data structure used: hseOnDemandAntiRbcUpdateSrv_t.

HSE_SRV_ID_FIRMWARE_VERIFY

```
#define HSE_SRV_ID_FIRMWARE_VERIFY ((hseSrvId_t)( HSE_SRV_VER_0| 0x00000030UL))
```

Verifies the Blue or Pink FW image in SRAM or external flash. Data structure used: hseFirmwareVerifySrv_t.

HSE_SRV_ID_CONFIG_COUNTER

```
#define HSE_SRV_ID_CONFIG_COUNTER ((hseSrvId_t)( HSE_SRV_VER_0| 0x00000052UL))
```

Configure the secure counter (only for HSE_B). This service requires SuperUser rights. Data structure used: hseConfigSecCounterSrv_t.

HSE_SRV_ID_LOAD_ECC_CURVE

```
#define HSE_SRV_ID_LOAD_ECC_CURVE ((hseSrvId_t)( HSE_SRV_VER_0| 0x00000100UL))
```

Load the parameters for a Weierstrass ECC curve. Data structure used: hseLoadEccCurveSrv_t.

HSE_SRV_ID_FORMAT_KEY_CATALOGS

```
#define HSE_SRV_ID_FORMAT_KEY_CATALOGS ((hseSrvId_t)( HSE_SRV_VER_0| 0x00000101UL))
```

Format key catalogs (NVM or RAM). Data structure used: hseFormatKeyCatalogsSrv_t.

HSE_SRV_ID_ERASE_KEY

```
#define HSE_SRV_ID_ERASE_KEY ((hseSrvId_t)( HSE_SRV_VER_0| 0x00000102UL))
```

Erase NVM/RAM key(s). Data structure used: hseEraseKeySrv_t.

HSE_SRV_ID_GET_KEY_INFO

```
#define HSE_SRV_ID_GET_KEY_INFO ((hseSrvId_t)( HSE_SRV_VER_0| 0x00A50103UL))
```

Get key information header. Data structure used: hseGetKeyInfoSrv_t.

HSE_SRV_ID_IMPORT_KEY

```
#define HSE_SRV_ID_IMPORT_KEY ((hseSrvId_t)( HSE_SRV_VER_0| 0x00000104UL))
```

Import a key. Data structure used: hseImportKeySrv_t.

HSE_SRV_ID_EXPORT_KEY

```
#define HSE_SRV_ID_EXPORT_KEY ((hseSrvId_t)( HSE_SRV_VER_0| 0x00000105UL))
```

Export a key. Data structure used: hseExportKeySrv_t.

HSE_SRV_ID_KEY_GENERATE

```
#define HSE_SRV_ID_KEY_GENERATE ((hseSrvId_t)( HSE_SRV_VER_0| 0x00000106UL))
```

Key Generation (e.g. rsa key pair, ecc key pair etc.). Data structure used: hseKeyGenerateSrv_t.

HSE_SRV_ID_DH_COMPUTE_SHARED_SECRET

```
#define HSE_SRV_ID_DH_COMPUTE_SHARED_SECRET ((hseSrvId_t)( HSE_SRV_VER_0|
0x00000107UL))
```

ECC Diffie-Hellman Compute Key (shared secret). Data structure used: hseDHComputeSharedSecretSrv_t.

HSE_SRV_ID_KEY_DERIVE

```
#define HSE_SRV_ID_KEY_DERIVE ((hseSrvId_t)( HSE_SRV_VER_0| 0x00000108UL))
```

Perform a key derivation function. Data structure used: hseKeyDeriveSrv_t.

HSE_SRV_ID_KEY_DERIVE_COPY

```
#define HSE_SRV_ID_KEY_DERIVE_COPY ((hseSrvId_t)( HSE_SRV_VER_0| 0x00000109UL))
```

Copy a key from the derived key material. Data structure used: hseKeyDeriveCopyKeySrv_t.

HSE_SRV_ID_KEY_VERIFY

```
#define HSE_SRV_ID_KEY_VERIFY ((hseSrvId_t)( HSE_SRV_VER_0| 0x0000010BUL))
```

Perform a verification for CMAC and SHA256/384/512. Data structure used: hseKeyVerifySrv_t.

HSE_SRV_ID_EXTEND_KEY_CATALOG

```
#define HSE_SRV_ID_EXTEND_KEY_CATALOG ((hseSrvId_t)( HSE_SRV_VER_0| 0x0000010CUL))
```

Update the NVM or RAM key catalogs format. Data structure used: hseExtendKeyCatalogSrv_t.

HSE_SRV_ID_SHE_LOAD_KEY

```
#define HSE_SRV_ID_SHE_LOAD_KEY ((hseSrvId_t)( HSE_SRV_VER_0| 0x0000A101UL))
```

Load a SHE key using the SHE memory update protocol. Data structure used: hseSheLoadKeySrv_t.

HSE_SRV_ID_SHE_LOAD_PLAIN_KEY

```
#define HSE_SRV_ID_SHE_LOAD_PLAIN_KEY ((hseSrvId_t)( HSE_SRV_VER_0| 0x0000A102UL))
```

Load the SHE RAM key as plain text. Data structure used: hseSheLoadPlainKeySrv_t.

HSE_SRV_ID_SHE_EXPORT_RAM_KEY

```
#define HSE_SRV_ID_SHE_EXPORT_RAM_KEY ((hseSrvId_t)( HSE_SRV_VER_0| 0x0000A103UL))
```

Export the SHE RAM key. Data structure used: hseSheExportRamKeySrv_t.

HSE_SRV_ID_SHE_GET_ID

```
#define HSE_SRV_ID_SHE_GET_ID ((hseSrvId_t)( HSE_SRV_VER_0| 0x0000A104UL))
```

Get UID as per SHE specification. Data structure used: hseSheGetIdSrv_t.

HSE_SRV_ID_SHE_BOOT_OK

```
#define HSE_SRV_ID_SHE_BOOT_OK ((hseSrvId_t)( HSE_SRV_VER_0| 0x0000A105UL))
```

BOOT_OK as per SHE specification. No data structure used.

HSE_SRV_ID_SHE_BOOT_FAILURE

```
#define HSE_SRV_ID_SHE_BOOT_FAILURE ((hseSrvId_t)( HSE_SRV_VER_0| 0x0000A106UL))
```

BOOT_FAILURE as per SHE specification. No data structure used.

HSE_SRV_ID_HASH

```
#define HSE_SRV_ID_HASH ((hseSrvId_t)( HSE_SRV_VER_0| 0x00A50200UL))
```

HASH service ID. Data structure used: hseHashSrv_t.

HSE_SRV_ID_MAC

```
#define HSE_SRV_ID_MAC ((hseSrvId_t)( HSE_SRV_VER_0| 0x00A50201UL))
```

MAC generate/verify. Data structure used: hseMacSrv_t.

HSE_SRV_ID_FAST_CMAC

```
#define HSE_SRV_ID_FAST_CMAC ((hseSrvId_t)( HSE_SRV_VER_0| 0x00A50202UL))
```

CMAC fast generate/verify. Data structure used: hseFastCMACSrv_t.

HSE_SRV_ID_SYM_CIPHER

```
#define HSE_SRV_ID_SYM_CIPHER ((hseSrvId_t)( HSE_SRV_VER_0| 0x00A50203UL))
```

Symmetric encryption/decryption. Data structure used: hseSymCipherSrv_t.

HSE_SRV_ID_AEAD

```
#define HSE_SRV_ID_AEAD ((hseSrvId_t)( HSE_SRV_VER_0| 0x00A50204UL))
```

AEAD encryption/decryption. Data structure used: hseAeadSrv_t.

HSE_SRV_ID_XTS_AES_CIPHER

```
#define HSE_SRV_ID_XTS_AES_CIPHER ((hseSrvId_t)( HSE_SRV_VER_0| 0x00A50205UL))
```

XTS AES encryption/decryption. Data structure used: hseXtsAesCipherSrv_t.

HSE_SRV_ID_SIGN

```
#define HSE_SRV_ID_SIGN ((hseSrvId_t)( HSE_SRV_VER_0| 0x00000206UL))
```

Digital Signature. Data structure used: hseSignSrv_t.

HSE_SRV_ID_RSA_CIPHER

```
#define HSE_SRV_ID_RSA_CIPHER ((hseSrvId_t)( HSE_SRV_VER_0| 0x00000207UL))
```

RSA Cipher ID. Data structure used: hseRsaCipherSrv_t.

HSE_SRV_ID_AUTHENC

```
#define HSE_SRV_ID_AUTHENC ((hseSrvId_t)( HSE_SRV_VER_0| 0x00A50208UL))
```

AuthEnc ID. Data structure used: hseAuthEncSrv_t.

HSE_SRV_ID_CRC32

```
#define HSE_SRV_ID_CRC32 ((hseSrvId_t)( HSE_SRV_VER_0| 0x00A50209UL))
```

CRC32 ID. Data structure used: hseCrc32Srv_t.

HSE_SRV_ID_SIPHASH

```
#define HSE_SRV_ID_SIPHASH ((hseSrvId_t)( HSE_SRV_VER_0| 0x0000020AUL))
```

SipHash service ID. Data structure used: hseSipHashSrv_t.

HSE_SRV_ID_CMAC_WITH_COUNTER

```
#define HSE_SRV_ID_CMAC_WITH_COUNTER ((hseSrvId_t)( HSE_SRV_VER_0| 0x00A5020BUL))
```

CMAC with counter service ID. Data structure used: hseCmacWithCounterSrv_t.

HSE_SRV_ID_GET_RANDOM_NUM

```
#define HSE_SRV_ID_GET_RANDOM_NUM ((hseSrvId_t)( HSE_SRV_VER_0| 0x00000300UL))
```

Get random number. Data structure used: hseGetRandomNumSrv_t.

HSE_SRV_ID_INCREMENT_COUNTER

```
#define HSE_SRV_ID_INCREMENT_COUNTER ((hseSrvId_t)( HSE_SRV_VER_0| 0x00A50400UL))
```

Increment a monotonic counter. Data structure used: hseIncrementCounterSrv_t.

HSE_SRV_ID_READ_COUNTER

```
#define HSE_SRV_ID_READ_COUNTER ((hseSrvId_t)( HSE_SRV_VER_0| 0x00A50401UL))
```

Read a monotonic counter. Data structure used: hseReadCounterSrv_t.

HSE_SRV_ID_SMR_ENTRY_INSTALL

```
#define HSE_SRV_ID_SMR_ENTRY_INSTALL ((hseSrvId_t)( HSE_SRV_VER_0| 0x00000501UL))
```

Install a Secure memory region (SMR) table entry. Data structure used: hseSmrEntryInstallSrv_t.

HSE_SRV_ID_SMR_VERIFY

```
#define HSE_SRV_ID_SMR_VERIFY ((hseSrvId_t)( HSE_SRV_VER_0| 0x00000502UL))
```

Verify a Secure memory region (SMR) table entry. Data structure used: hseSmrVerifySrv_t.

HSE_SRV_ID_CORE_RESET_ENTRY_INSTALL

```
#define HSE_SRV_ID_CORE_RESET_ENTRY_INSTALL ((hseSrvId_t)( HSE_SRV_VER_0|
0x00000503UL))
```

Install a Core Reset (CR) table entry. Data structure used: hseCrEntryInstallSrv_t.

HSE_SRV_ID_ON_DEMAND_CORE_RESET

```
#define HSE_SRV_ID_ON_DEMAND_CORE_RESET ((hseSrvId_t)( HSE_SRV_VER_0|
0x00000504UL))
```

On demand release a core from reset after loading and verification. Data structure used: hseCrOnDemandBootSrv_t.

HSE_SRV_ID_SMR_ENTRY_ERASE

```
#define HSE_SRV_ID_SMR_ENTRY_ERASE ((hseSrvId_t)( HSE_SRV_VER_0| 0x00000505UL))
```

Erase a Secure memory region (SMR) table entry. Data structure used: hseSmrEntryEraseSrv_t.

HSE_SRV_ID_CORE_RESET_ENTRY_ERASE

```
#define HSE_SRV_ID_CORE_RESET_ENTRY_ERASE ((hseSrvId_t)( HSE_SRV_VER_0|
0x00000506UL))
```

Erase a Core Reset (CR) table entry. Data structure used: hseCrEntryEraseSrv_t.

2.4 HSE Service Responses

Macros

Type: hseSrvResponse_t	
Name	**Value**
HSE_SRV_RSP_OK	0x55A5AA33UL
HSE_SRV_RSP_VERIFY_FAILED	0x55A5A164UL
HSE_SRV_RSP_INVALID_ADDR	0x55A5A26AUL
HSE_SRV_RSP_INVALID_PARAM	0x55A5A399UL
HSE_SRV_RSP_NOT_SUPPORTED	0xAA55A11EUL
HSE_SRV_RSP_NOT_ALLOWED	0xAA55A21CUL
HSE_SRV_RSP_NOT_ENOUGH_SPACE	0xAA55A371UL
HSE_SRV_RSP_READ_FAILURE	0xAA55A427UL
HSE_SRV_RSP_WRITE_FAILURE	0xAA55A517UL
HSE_SRV_RSP_STREAMING_MODE_FAILURE	0xAA55A6B1UL
HSE_SRV_RSP_KEY_NOT_AVAILABLE	0xA5AA51B2UL
HSE_SRV_RSP_KEY_INVALID	0xA5AA52B4UL
HSE_SRV_RSP_KEY_EMPTY	0xA5AA5317UL
HSE_SRV_RSP_KEY_WRITE_PROTECTED	0xA5AA5436UL
HSE_SRV_RSP_KEY_UPDATE_ERROR	0xA5AA5563UL
HSE_SRV_RSP_MEMORY_FAILURE	0x33D6D136UL
HSE_SRV_RSP_CANCEL_FAILURE	0x33D6D261UL
HSE_SRV_RSP_CANCELED	0x33D6D396UL
HSE_SRV_RSP_GENERAL_ERROR	0x33D6D4F1UL
HSE_SRV_RSP_COUNTER_OVERFLOW	0x33D6D533UL
HSE_SRV_RSP_SHE_NO_SECURE_BOOT	0x33D6D623UL
HSE_SRV_RSP_SHE_BOOT_SEQUENCE_ERROR	0x33D7D83AUL
HSE_SRV_RSP_RNG_INIT_IN_PROGRESS	0x33D7D92AUL
HSE_SRV_RSP_FUSE_WRITE_FAILURE	0xBB4456E7UL
HSE_SRV_RSP_FUSE_VDD_GND	0xBB4457F3UL

HSE_SRV_RSP_SBAF_UPDATE_REQUIRED	0xCC66FEADUL

Typedefs

- typedef uint32_t hseSrvResponse_t

Macro Definition Documentation

HSE_SRV_RSP_OK

```
#define HSE_SRV_RSP_OK ((hseSrvResponse_t)0x55A5AA33UL)
```

HSE service successfully executed with no error.

HSE_SRV_RSP_VERIFY_FAILED

```
#define HSE_SRV_RSP_VERIFY_FAILED ((hseSrvResponse_t)0x55A5A164UL)
```

HSE signals that a verification request fails (e.g. MAC and Signature verification).

HSE_SRV_RSP_INVALID_ADDR

```
#define HSE_SRV_RSP_INVALID_ADDR ((hseSrvResponse_t)0x55A5A26AUL)
```

The address parameters are invalid.

HSE_SRV_RSP_INVALID_PARAM

```
#define HSE_SRV_RSP_INVALID_PARAM ((hseSrvResponse_t)0x55A5A399UL)
```

The HSE request parameters are invalid.

HSE_SRV_RSP_NOT_SUPPORTED

```
#define HSE_SRV_RSP_NOT_SUPPORTED ((hseSrvResponse_t)0xAA55A11EUL)
```

The operation or feature not supported.

HSE_SRV_RSP_NOT_ALLOWED

```
#define HSE_SRV_RSP_NOT_ALLOWED ((hseSrvResponse_t)0xAA55A21CUL)
```

The operation is not allowed because of some restrictions (in attributes, life-cycle dependent operations, key-management, etc.).

HSE_SRV_RSP_NOT_ENOUGH_SPACE

```
#define HSE_SRV_RSP_NOT_ENOUGH_SPACE ((hseSrvResponse_t)0xAA55A371UL)
```

There is no enough space to perform service (e.g. format key store)

HSE_SRV_RSP_READ_FAILURE

```
#define HSE_SRV_RSP_READ_FAILURE ((hseSrvResponse_t)0xAA55A427UL)
```

The service request failed because read access was denied. For HSE_B, it can be returned if Host Flash Programming/Erase operaton was in progress at the time of giving the command.

HSE_SRV_RSP_WRITE_FAILURE

```
#define HSE_SRV_RSP_WRITE_FAILURE ((hseSrvResponse_t)0xAA55A517UL)
```

The service request failed because write access was denied.

HSE_SRV_RSP_STREAMING_MODE_FAILURE

```
#define HSE_SRV_RSP_STREAMING_MODE_FAILURE ((hseSrvResponse_t)0xAA55A6B1UL)
```

The service request that uses streaming mode failed (e.g. UPDATES and FINISH steps do not use the same HSE interface ID and channel ID as START step).

HSE_SRV_RSP_KEY_NOT_AVAILABLE

```
#define HSE_SRV_RSP_KEY_NOT_AVAILABLE ((hseSrvResponse_t)0xA5AA51B2UL)
```

This error code is returned if a key is locked due to failed boot measurement or an active debugger.

HSE_SRV_RSP_KEY_INVALID

```
#define HSE_SRV_RSP_KEY_INVALID ((hseSrvResponse_t)0xA5AA52B4UL)
```

The key usage flags (provided using the key handle) don't allow to perform the requested crypto operation (the key flags don't match the crypto operation; e.g. the key is configured to be used for decryption, and the host requested an encryption). In SHE, the key ID provided is either invalid or non-usable due to some flag restrictions.

HSE_SRV_RSP_KEY_EMPTY

```
#define HSE_SRV_RSP_KEY_EMPTY ((hseSrvResponse_t)0xA5AA5317UL)
```

Specified key slot is empty.

HSE_SRV_RSP_KEY_WRITE_PROTECTED

```
#define HSE_SRV_RSP_KEY_WRITE_PROTECTED ((hseSrvResponse_t)0xA5AA5436UL)
```

Key slot to be loaded is protected with WRITE PROTECTION restriction flag.

HSE_SRV_RSP_KEY_UPDATE_ERROR

```
#define HSE_SRV_RSP_KEY_UPDATE_ERROR ((hseSrvResponse_t)0xA5AA5563UL)
```

Used only in the context of SHE specification: specified key slot cannot be updated due to errors in verification of the parameters.

HSE_SRV_RSP_MEMORY_FAILURE

```
#define HSE_SRV_RSP_MEMORY_FAILURE ((hseSrvResponse_t)0x33D6D136UL)
```

Detect physical errors, flipped bits etc., during memory read or write operations.

HSE_SRV_RSP_CANCEL_FAILURE

```
#define HSE_SRV_RSP_CANCEL_FAILURE ((hseSrvResponse_t)0x33D6D261UL)
```

The service can not be canceled.

HSE_SRV_RSP_CANCELED

```
#define HSE_SRV_RSP_CANCELED ((hseSrvResponse_t)0x33D6D396UL)
```

The service has been canceled.

HSE_SRV_RSP_GENERAL_ERROR

```
#define HSE_SRV_RSP_GENERAL_ERROR ((hseSrvResponse_t)0x33D6D4F1UL)
```

This error code is returned if an error not covered by the error codes above is detected inside HSE.

HSE_SRV_RSP_COUNTER_OVERFLOW

```
#define HSE_SRV_RSP_COUNTER_OVERFLOW ((hseSrvResponse_t)0x33D6D533UL)
```

The monotonic counter overflows.

HSE_SRV_RSP_SHE_NO_SECURE_BOOT

```
#define HSE_SRV_RSP_SHE_NO_SECURE_BOOT ((hseSrvResponse_t)0x33D6D623UL)
```

HSE did not perform SHE based secure Boot.

HSE_SRV_RSP_SHE_BOOT_SEQUENCE_ERROR

```
#define HSE_SRV_RSP_SHE_BOOT_SEQUENCE_ERROR ((hseSrvResponse_t)0x33D7D83AUL)
```

Received SHE_BOOT_OK or SHE_BOOT_FAILURE more then one time.

HSE_SRV_RSP_RNG_INIT_IN_PROGRESS

```
#define HSE_SRV_RSP_RNG_INIT_IN_PROGRESS ((hseSrvResponse_t)0x33D7D92AUL)
```

This error code is returned if RNG Intialization is in Progress.

HSE_SRV_RSP_FUSE_WRITE_FAILURE

```
#define HSE_SRV_RSP_FUSE_WRITE_FAILURE ((hseSrvResponse_t)0xBB4456E7UL)
```

This error code is returned, if fuse write operation fail.

HSE_SRV_RSP_FUSE_VDD_GND

```
#define HSE_SRV_RSP_FUSE_VDD_GND ((hseSrvResponse_t)0xBB4457F3UL)
```

This error code is returned, if EFUSE_VDD connected to ground during fuse write operation.

HSE_SRV_RSP_SBAF_UPDATE_REQUIRED

```
#define HSE_SRV_RSP_SBAF_UPDATE_REQUIRED ((hseSrvResponse_t)0xCC66FEADUL)
```

This error code is returned, if operation is dependent on Secure BAF version, which on the device happens to be old.

Typedef Documentation

hseSrvResponse_t

```
typedef uint32_t hseSrvResponse_t
```

HSE Service response.

The Service response is provided by MUB_RRx register after the service execution.

2.5 HSE Errors

Macros

Type: hseError_t

Name	Value
HSE_ERR_GENERAL	1UL << 0U
HSE_WA_SMR_PERIODIC_CHECK_FAILED	1UL << 8U
HSE_WA_RNG_NOT_INIT	1UL << 10U
HSE_WA_PUBLISH_COUNTER_TBL	1UL << 11U
HSE_WA_OTP_FUSE_WRITE_FAILURE_ON_BOOT	1UL << 12U

Typedefs

- typedef uint32_t hseError_t

HSE Errors Details

These error events are reported when some kind of intrusion/violation is detected in the system. The most significant 16 bits are reserved for NXP internal errors and less significant 16 bits indicate the source of violation as defined below.

Note

- If the MU General Purpose Interrupt is enabled on the host-side, any bit set to "1" (on MUB_GSR register) triggers an interrupt.
- The host must read the MUB_GSR register and write back the register value to clear the bits (W1C - write one to clear).
- The bits[0..7] (listed below) are fatal errors that trigger an HSE shutdown (HSE enters in the secure failure state, all MU are disabled).
- The bits[8..15] (listed below) are warning events (something failed, but it is not fatal).

Macro Definition Documentation

HSE_ERR_GENERAL

```
#define HSE_ERR_GENERAL ((hseError_t)1UL << 0U)
```

Internal fatal error detected by HSE. The HSE system shutdowns.

HSE_WA_SMR_PERIODIC_CHECK_FAILED

```
#define HSE_WA_SMR_PERIODIC_CHECK_FAILED ((hseError_t)1UL << 8U)
```

The verification of periodic check SMR (hseSmrEntry_t::checkPeriod !=0) failed. The application can read HSE_SMR_CORE_BOOT_STATUS_ATTR_ID attribute to see what SMR failed.

HSE_WA_RNG_NOT_INIT

```
#define HSE_WA_RNG_NOT_INIT ((hseError_t)1UL << 10U)
```

RNG is not initialized. Services depending on the RNG may be delayed as HSE attempts RNG re-initialization.

HSE_WA_PUBLISH_COUNTER_TBL

```
#define HSE_WA_PUBLISH_COUNTER_TBL ((hseError_t)1UL << 11U)
```

The application shall publish and store the monotonic counter table.

HSE_WA_OTP_FUSE_WRITE_FAILURE_ON_BOOT

```
#define HSE_WA_OTP_FUSE_WRITE_FAILURE_ON_BOOT ((hseError_t)1UL << 12U)
```

At start-up, the the fuse write operation (anti-rollback counter update) failed. A destructive reset is needed.

Typedef Documentation

hseError_t

```
typedef uint32_t hseError_t
```

2.6 Host Events To HSE

Macros

Type: hseHostEvent_t	
Name	**Value**
HSE_HOST_PERIPH_CONFIG_DONE	1UL << 0U

Typedefs

- typedef uint32_t hseHostEvent_t

Host Events To HSE Details

These events are sent by Host to notify HSE of actions that needs synchronization between the two. In order to signal HSE of these events, the host must write its value to MUB_GCR.

Note

This is applicable only for MU0 instance.

Macro Definition Documentation

HSE_HOST_PERIPH_CONFIG_DONE

```
#define HSE_HOST_PERIPH_CONFIG_DONE ((hseHostEvent_t)1UL << 0U)
```

This event is sent by the host to notify HSE after it configures the external peripherals at init-time.

Note

This host event is applicable only at start-up:

- When BOOT_SEQ == 0, until the HSE sets HSE_STATUS_INIT_OK
- Or, when BOOT_SEQ == 1 and the POST_BOOT SMRs are used, after HSE sets HSE_STATUS_BOOT_OK, until HSE_STATUS_INIT_OK is set.
- In the above cases,for HSE_B/H/M (except SAF85XX), if the HSE_HOST_PERIPH_CONFIG_DONE is not received within 5 seconds (computed at maximum frequency), the HSE execution continues.
- Or, for SAF85XX if the HSE_HOST_PERIPH_CONFIG_DONE is not received within 240 milliseconds (computed at maximum frequency), the HSE execution continues.

Typedef Documentation

hseHostEvent_t

```
typedef uint32_t hseHostEvent_t
```

2.7 HSE Status

Macros

Type: hseStatus_t	
Name	**Value**
HSE_SHE_STATUS_SECURE_BOOT	1U << 1U
HSE_SHE_STATUS_SECURE_BOOT_INIT	1U << 2U

HSE_SHE_STATUS_SECURE_BOOT_FINISHED	1U << 3U
HSE_SHE_STATUS_SECURE_BOOT_OK	1U << 4U
HSE_STATUS_RNG_INIT_OK	1U << 5U
HSE_STATUS_HOST_DEBUGGER_ACTIVE	1U << 6U
HSE_STATUS_HSE_DEBUGGER_ACTIVE	1U << 7U
HSE_STATUS_INIT_OK	1U << 8U
HSE_STATUS_INSTALL_OK	1U << 9U
HSE_STATUS_BOOT_OK	1U << 10U
HSE_STATUS_CUST_SUPER_USER	1U << 11U
HSE_STATUS_OEM_SUPER_USER	1U << 12U
HSE_STATUS_PUBLISH_SYS_IMAGE	1U << 13U
HSE_STATUS_PRIMARY_SYS_IMAGE	1U << 14U
HSE_STATUS_BACKUP_SYS_IMAGE	1U << 15U

Typedefs

- typedef uint16_t hseStatus_t

HSE Status Details

HSE status can be read by the HOST and represents the most significant 16 bits in MUB.FSR register. The least significant 16 bits in MUB.FSR register identifies the status of each channel:
– 0b - channel idle and it can accept service requests
– 1b - channel busy

Macro Definition Documentation

HSE_SHE_STATUS_SECURE_BOOT

```
#define HSE_SHE_STATUS_SECURE_BOOT ((hseStatus_t)1U << 1U)
```

This bit is set when the SHE based secure boot process has been started by HSE firmware. This bit is only set when SMR0 entry has been installed by the user and its authentication key is set as SHE based BOOT_MAC_KEY

HSE_SHE_STATUS_SECURE_BOOT_INIT

```
#define HSE_SHE_STATUS_SECURE_BOOT_INIT ((hseStatus_t)1U << 2U)
```

This bit is set when BOOT_MAC personalization has been completed by HSE firmware. It means that the BOOT_MAC slot was empty and SHE-based secure boot is performed the the first time. In that case, if

BOOT_MAC_KEY is present, then HSE firmware calculates the BOOT_MAC of the SMR image present in the SMR0 (using the BOOT_MAC_KEY) and store it as part of sys image.

HSE_SHE_STATUS_SECURE_BOOT_FINISHED

```
#define HSE_SHE_STATUS_SECURE_BOOT_FINISHED ((hseStatus_t)1U << 3U)
```

This bit is set when the HSE firmware has completed the secure boot process with a failure status. (the image verification failed).

HSE_SHE_STATUS_SECURE_BOOT_OK

```
#define HSE_SHE_STATUS_SECURE_BOOT_OK ((hseStatus_t)1U << 4U)
```

This bit is set when the HSE firmware has successfully completed the secure boot process (the image verification was successful).

HSE_STATUS_RNG_INIT_OK

```
#define HSE_STATUS_RNG_INIT_OK ((hseStatus_t)1U << 5U)
```

This bit is set when HSE FW has successfully initiliazed the RNG.

HSE_STATUS_HOST_DEBUGGER_ACTIVE

```
#define HSE_STATUS_HOST_DEBUGGER_ACTIVE ((hseStatus_t)1U << 6U)
```

This bit is set when debugger on HOST side is active as well as enabled.

HSE_STATUS_HSE_DEBUGGER_ACTIVE

```
#define HSE_STATUS_HSE_DEBUGGER_ACTIVE ((hseStatus_t)1U << 7U)
```

This bit is set when debugger on HSE side is active as well as enabled.

HSE_STATUS_INIT_OK

```
#define HSE_STATUS_INIT_OK ((hseStatus_t)1U << 8U)
```

This bit is set when the HSE initialization has been successfully completed (HSE service requests can be sent over MUs). If this bit is cleared, the host can NOT perform any service request (MUs are disabled).

HSE_STATUS_INSTALL_OK

```
#define HSE_STATUS_INSTALL_OK ((hseStatus_t)1U << 9U)
```

This flag signals the application that needs to format the key catalogs (NVM and RAM).

- When it is clear, the application shall format the key catalogs;
- When it is set, the HSE installation phase has been successfully completed. (e.g HSE is in normal state and the application can install the NVM key, configure the SMR entries etc).

  Note

  This step is MANDATORY.

HSE_STATUS_BOOT_OK

```
#define HSE_STATUS_BOOT_OK ((hseStatus_t)1U << 10U)
```

This bit is set when the HSE booting phase has been successfully completed. This bit is cleared if the HSE booting phase is still in execution or failed.

Note

- HSE set this bit only when the secure boot is configured (BOOT_SEQ = 1).
- This bit represents the status of booting phase which includes the PRE_BOOT SMR verification (without POST_BOOT SMRs) and cores un-gating.
- The HSE FW signals the end of the POST_BOOT phase along with additional peripherals initialization via HSE_STATUS_INIT_OK flag.

HSE_STATUS_CUST_SUPER_USER

```
#define HSE_STATUS_CUST_SUPER_USER ((hseStatus_t)1U << 11U)
```

After reset, if the Life Cycle = CUST_DEL, this bit is set (SuperUser rights are granted).
During run-time:

- it is set if the authorization request for CUST SuperUser rights are granted using an CUST authorization key.
- it is cleared for USER rights.

Note

If CUST START_AS_USER policy attribute is set (TRUE), the device will always start having User rights.

HSE_STATUS_OEM_SUPER_USER

```
#define HSE_STATUS_OEM_SUPER_USER ((hseStatus_t)1U << 12U)
```

After reset: if the Life Cycle = OEM_PROD, this bit is set (SuperUser rights are granted).
During run-time:

- it is set if the authorization request for OEM SuperUser rights are granted using an OEM authorization key.
- it is cleared for USER rights.

Note

If OEM START_AS_USER policy attribute is set (TRUE), the device will always start having User rights.

HSE_STATUS_PUBLISH_SYS_IMAGE

```
#define HSE_STATUS_PUBLISH_SYS_IMAGE ((hseStatus_t)1U << 13U)
```

This flag signals the application to publish the SYS-IMAGE.

- When this flags is set, the host must trigger a PUBLISH_SYS_IMG request.

Note

This flag is set whenever the HSE SYS-IMAGE has been updated in the HSE internal RAM (e.g. after a key update, SMR update, etc.).

- Once SYS-IMG is published to application RAM, this bit is cleared.

HSE_STATUS_PRIMARY_SYS_IMAGE

```
#define HSE_STATUS_PRIMARY_SYS_IMAGE ((hseStatus_t)1U << 14U)
```

This flag signals the application whether HSE FW has loaded or not the SYS-IMAGE from primary address.

- If this flag is set, the primary SYS-IMAGE has been loaded.
- If this flag is cleared, the primary SYS-IMAGE has NOT been loaded. This means that HSE either loaded the SYS-IMAGE from backup address (see HSE_STATUS_BACKUP_SYS_IMAGE flag) or both primary and backup loads failed.

HSE_STATUS_BACKUP_SYS_IMAGE

```
#define HSE_STATUS_BACKUP_SYS_IMAGE ((hseStatus_t)1U << 15U)
```

This flag signals the application whether HSE FW has loaded or not the SYS-IMAGE from backup address.

- If this flag is set, the backup SYS-IMAGE has been loaded.
- If this flag is cleared, the backup SYS-IMAGE has NOT been loaded. This means that HSE either loaded the SYS-IMAGE from primary address (see HSE_STATUS_PRIMARY_SYS_IMAGE flag) or both primary and backup loads failed.

Typedef Documentation

hseStatus_t

```
typedef uint16_t hseStatus_t
```

3 Administration Services

3.1 HSE Utility Services

Data Structures

- struct hseOnDemandAntiRbcUpdateSrv_t
- struct hseCancelSrv_t
- struct hseImportExportStreamCtxSrv_t
- struct hsePrepareForStandBySrv_t

Macros

Type: (implicit C type)	
Name	**Value**
MAX_STREAMING_CONTEXT_SIZE	372UL

Type: hseAntiRbcMask_t	
Name	**Value**
HSE_SYS_IMG_ANTI_RBC_MASK	0x00005F51UL
HSE_FW_IMG_ANTI_RBC_MASK	0xF31C0000UL
HSE_SYS_FW_IMG_ANTI_RBC_MASK	0xF31C5F51UL

Type: hseStreamContextOp_t	
Name	**Value**
HSE_IMPORT_STREAMING_CONTEXT	1U
HSE_EXPORT_STREAMING_CONTEXT	2U

Typedefs

- typedef uint32_t hseAntiRbcMask_t
- typedef uint8_t hseStreamContextOp_t

Data Structure Documentation

struct hseOnDemandAntiRbcUpdateSrv_t

On demand anti-rollback counter update service.

The service can be used only if the anti-rollback counter policy disableOtpRollbackProtect (see hseOtpRollbackProtectionPolicy_t) is set to HSE_ON_DEMAND_ANTI_RBC_UPDATE (otherwise, it returns HSE_SRV_RSP_NOT_ALLOWED). If the anti-rollback counter is already updated, it returns HSE_SRV_RSP_OK.

Note

: The SYS-IMG should be properly stored to external flash, then sucessfully loaded and authenticated by HSE before calling this service. This ensures that the SYS-IMG was not corrupted before burning the fuses. The VDD_EFUSE must be powered before calling this service.

Data Fields

Type	Name	Description
hseAntiRbcMask_t	hseAntiRollbackCounterMask	INPUT: Specifies the mask value that selects the counterpart of the anti-rollback counter that will be updated (SYS-IMG or FW-IMG). Select either one option or both simultaneously, depending on the use case.

struct hseCancelSrv_t

HSE Cancel service.

This service cancels a HSE one-pass and streaming service that was sent on a specific channel.

Note

- The requests with the service ID that starts with 0x00A5XXXX can not be canceled.
- Cancel requests cannot be canceled (by a subsequent request);

Data Fields

Type	Name	Description
uint8_t	muChannelIdx	INPUT: The channel Index of MU interface [0.. HSE_NUM_OF_CHANNELS_PER_MU). The muChannelIdx and the MU channel on which the service is sent, must belong to the same MU Interface. Otherwise an HSE_SRV_RSP_INVALID_PARAM error will be reported.
uint8_t	reserved[3]	

struct hseImportExportStreamCtxSrv_t

HSE Import/Export Streaming Context service.

This service allows import/export of a streaming context used in an on-going streaming operation (e.g. Hash, MAC, Cipher, AEAD, etc).
The streaming context will be imported/exported as a blob (encrypted with a device specific key).

Data Fields

Type	Name	Description
hseStreamContextOp_t	operation	INPUT: Specifies the operation to be performed with the streaming context: Import/Export.
hseStreamId_t	streamId	INPUT: Specifies the stream to be exported or overwritten if imported. Note that each interface supports up to HSE_STREAM_COUNT streams per interface.
uint8_t	reserved[2]	
uint64_t	pStreamContext	OUTPUT/INPUT: The output buffer where the streaming context will be copied (export) or the input buffer from which HSE will copy the streaming context (import). Length of the buffer should be at least MAX_STREAMING_CONTEXT_SIZE bytes. A streaming context can be imported or exported on the same MU instance on which the streaming START step was called (e.g. the steaming context is allocated when the START step is called).".

struct hsePrepareForStandBySrv_t

Prepare the security subsytem (BootROM + HSE) for Stand-By.

This service is used for updating the internal state of HSE before system goes in Stand-By mode. Applicable only for flashless devices (HSE_H/M variants). This service can be called only once per running state, otherwise HSE will return HSE_SRV_RSP_NOT_ALLOWED .

Data Fields

Type	Name	Description
uint8_t	reserved[4U]	

Macro Definition Documentation

HSE_SYS_IMG_ANTI_RBC_MASK

```
#define HSE_SYS_IMG_ANTI_RBC_MASK ((hseAntiRbcMask_t)(0x00005F51UL))
```

SYS-IMG counter mask.

HSE_FW_IMG_ANTI_RBC_MASK

```
#define HSE_FW_IMG_ANTI_RBC_MASK ((hseAntiRbcMask_t)(0xF31C0000UL))
```

FW-IMG counter mask.

HSE_SYS_FW_IMG_ANTI_RBC_MASK

```
#define HSE_SYS_FW_IMG_ANTI_RBC_MASK ((hseAntiRbcMask_t)(0xF31C5F51UL))
```

Both SYS-IMG and FW-IMG counters mask.

MAX_STREAMING_CONTEXT_SIZE

```
#define MAX_STREAMING_CONTEXT_SIZE (372UL)
```

The maximum size of the streaming context for any operation.

HSE_IMPORT_STREAMING_CONTEXT

```
#define HSE_IMPORT_STREAMING_CONTEXT ((hseStreamContextOp_t)1U)
```

Import streaming context.

HSE_EXPORT_STREAMING_CONTEXT

```
#define HSE_EXPORT_STREAMING_CONTEXT ((hseStreamContextOp_t)2U)
```

Export streaming context.

Typedef Documentation

hseAntiRbcMask_t

```
typedef uint32_t hseAntiRbcMask_t
```

Mask value that specifies the counterpart of the anti-rollback counter (SYS-IMG or FW-IMG).

Selects the counterpart of the anti-rollback counter that will be updated when calling hseOnDemandAntiRbcUpdateSrv_t service.

hseStreamContextOp_t

```
typedef uint8_t hseStreamContextOp_t
```

Streaming Context Operation: Import/Export.

3.2 HSE Set/Get Attribute Services

Data Structures

- struct hseSetAttrSrv_t
- struct hseGetAttrSrv_t
- struct hseAttrFwVersion_t
- struct hseFwBuildInfo_t
- struct hseAttrSmrCoreStatus_t
- struct hseAttrMUInstanceConfig_t
- struct hseAttrMUConfig_t
- struct hseAttrMemRegion_t
- struct hseAttrMuMemRegions_t
- struct hseAttrAllMuMemRegions_t
- struct hseAttrExtendCustSecurityPolicy_t
- struct hseAttrExtendOemSecurityPolicy_t
- struct hseAttrPhysicalTamper_t
- struct hseAttrPhysicalTamperConfig_t
- struct hseAppSpecificData_t
- struct hseRbCounterInfo_t
- struct hseOtfadCtxStatus_t
- struct hseOtpRollbackProtectionPolicy_t

Macros

Type: (implicit C type)	
Name	**Value**
HSE_ALGO_CAP_MASK (capIdx)	(1ULL << (capIdx))
HSE_FILTER_DURATION_MAX	((uint32_t)128U)
HSE_OTP_BOOT_SEQ_MARKER	0xCEDEADDAUL
HSE_OTP_BOOT_SEQ_NOT_SET	0x00000000UL

Type: hseMemRegAccess_t	
Name	**Value**
HSE_MEM_REG_ACCESS_MASK_IN	0x00003C96UL
HSE_MEM_REG_ACCESS_MASK_OUT	0x5A690000UL
HSE_MEM_REG_ACCESS_MASK_INOUT	HSE_MEM_REG_ACCESS_MASK_IN | HSE_MEM_REG_ACCESS_MASK_OUT

Type: hseAttrDisableAppDebug_t	
Name	**Value**
HSE_APP_DEBUG_DIS_NONE	0x0U
HSE_APP_DEBUG_DIS_OEM	0x1U
HSE_APP_DEBUG_DIS_FLD	0x2U
HSE_APP_DEBUG_DIS_OEM_FLD	0x3U

Type: hseMUConfig_t	
Name	**Value**
HSE_MU_ACTIVATED	0xA5U
HSE_MU_DEACTIVATED	0x5AU

Type: hseAttrRamPubKeyImportPolicy_t	
Name	**Value**
HSE_KM_POLICY_DEFAULT	0x4E8BD124UL
HSE_KM_POLICY_ALLOW_RAM_PUB_KEY_IMPORT	0xB1742EDBUL

Type: hseAttrCfg_t	
Name	**Value**
HSE_CFG_NO	0x0UL
HSE_CFG_YES	0xB7A5C365UL

Type: hseTamperOutputClock_t	
Name	**Value**
HSE_TAMPER_ACTIVE_CLOCK_16HZ	0U
HSE_TAMPER_ACTIVE_CLOCK_8HZ	1U
HSE_TAMPER_ACTIVE_CLOCK_4HZ	2U
HSE_TAMPER_ACTIVE_CLOCK_2HZ	3U

Type: hseDisableAntiRbcCfg_t	
Name	**Value**
HSE_DIS_ANTI_RBC_CFG_NO	HSE_CFG_NO
HSE_DIS_ANTI_RBC_CFG_YES	HSE_CFG_YES
HSE_ON_DEMAND_ANTI_RBC_UPDATE	0x676E2064UL

Type: hseOtfadContextStatus_t	
Name	**Value**
HSE_OTFAD_CTX_NOT_INSTALLED	0x00U
HSE_OTFAD_CTX_INSTALLED	0xCAU
HSE_OTFAD_CTX_ACTIVE	0xACU
HSE_OTFAD_CTX_INACTIVE	0xDEU

Type: hseAttrCoreResetRelease_t	
Name	**Value**
HSE_CR_RELEASE_ALL_AT_ONCE	0xA5556933UL
HSE_CR_RELEASE_ONE_BY_ONE	0xA5557555UL

Type: hseOutputPinConfig_t	
Name	**Value**
HSE_TAMPER_PASSIVE	0U
HSE_TAMPER_ACTIVE_ONE	1U
HSE_TAMPER_ACTIVE_TWO	2U

Type: hseAttrConfigBootAuth_t	
Name	**Value**
HSE_IVT_NO_AUTH	0x0U
HSE_IVT_AUTH	0x1U

Type: hseAttrDebugAuthMode_t	
Name	**Value**
HSE_DEBUG_AUTH_MODE_PW	0x0U
HSE_DEBUG_AUTH_MODE_CR	0x1U

Type: hseAttrFwPartition_t	
Name	**Value**
HSE_FW_PARTITION_PRIMARY	0x1U
HSE_FW_PARTITION_BACKUP	0x2U

Type: hseAttrId_t	
Name	**Value**
HSE_NONE_ATTR_ID	0U

HSE_FW_VERSION_ATTR_ID	1U
HSE_CAPABILITIES_ATTR_ID	2U
HSE_SMR_CORE_BOOT_STATUS_ATTR_ID	3U
HSE_FW_BUILD_INFO_ATTR_ID	4U
HSE_DEBUG_AUTH_MODE_ATTR_ID	10U
HSE_APP_DEBUG_KEY_ATTR_ID	11U
HSE_SECURE_LIFECYCLE_ATTR_ID	12U
HSE_ENABLE_BOOT_AUTH_ATTR_ID	13U
HSE_EXTEND_CUST_SECURITY_POLICY_ATTR_ID	14U
HSE_MU_CONFIG_ATTR_ID	20U
HSE_EXTEND_OEM_SECURITY_POLICY_ATTR_ID	21U
HSE_FAST_CMAC_MIN_TAG_BIT_LEN_ATTR_ID	22U
HSE_CORE_RESET_RELEASE_ATTR_ID	23U
HSE_RAM_PUB_KEY_IMPORT_POLICY_ATTR_ID	24U
HSE_RESET_SOC_ON_TAMPER_ATTR_ID	25U
HSE_OTP_ROLLBACK_PROTECTION_POLICY_ATTR_ID	26U
HSE_APP_SPECIFIC_DATA_ATTR_ID	27U
HSE_DISABLE_APP_SPECIFIC_DATA_WRITE_ATTR_ID	28U
HSE_PHYSICAL_TAMPER_ATTR_ID	30U
HSE_MEM_REGIONS_PROTECT_ATTR_ID	31U
HSE_DISABLE_PAIRWISE_CONSISTENCY_TEST_ATTR_ID	32U
HSE_FW_SIZE_ATTR_ID	100U
HSE_AVAIL_ANTI_ROLLBACK_COUNTER_ATTR_ID	101U
HSE_FW_PARTITION_ATTR_ID	102U
HSE_OTFAD_CTX_STATUS_ATTR_ID	103U
HSE_RB_COUNTER_INFO_ATTR_ID	105U
HSE_APP_DEBUG_DIS_ATTR_ID	200U
HSE_OTP_BOOT_SEQ_ATTR_ID	201U
HSE_TEMP_SENSOR_VIO_CONFIG_ATTR_ID	400U

Type: hseTempSensVioConfig_t	
Name	**Value**

HSE_TEMP_SENS_VIO_ACTIVATED	0xA5U
HSE_TEMP_SENS_VIO_DEACTIVATED	0x5AU

Type: hseTamperConfig_t	
Name	**Value**
HSE_TAMPER_CONFIG_DEACTIVATE	0U
HSE_TAMPER_CONFIG_ACTIVATE	1U

Type: hseAttrSecureLifecycle_t	
Name	**Value**
HSE_LC_CUST_DEL	0x4U
HSE_LC_OEM_PROD	0x8U
HSE_LC_IN_FIELD	0x10U
HSE_LC_PRE_FA	0x14U
HSE_LC_SIMULATED_OEM_PROD	0xA6U
HSE_LC_SIMULATED_IN_FIELD	0xA7U

Type: hseTamperPolarity_t	
Name	**Value**
HSE_TAMPER_POL_ACTIVE_LOW	0U
HSE_TAMPER_POL_ACTIVE_HIGH	1U

Typedefs

- typedef uint32_t hseAttrCfg_t
- typedef uint16_t hseAttrId_t
- typedef uint64_t hseAttrCapabilities_t
- typedef uint32_t hseAttrCoreResetRelease_t
- typedef uint8_t hseAttrDebugAuthMode_t
- typedef uint8_t hseAttrApplDebugKey_t[16]
- typedef hseKeyHandle_t hseAttrSecureApplDebugKey_t
- typedef uint8_t hseAttrSecureLifecycle_t
- typedef uint8_t hseAttrConfigBootAuth_t
- typedef uint8_t hseMUConfig_t
- typedef uint32_t hseMemRegAccess_t
- typedef uint32_t hseAttrRamPubKeyImportPolicy_t
- typedef uint8_t hseAttrFastCmacMinTagBitLen_t
- typedef uint8_t hseTamperConfig_t
- typedef uint8_t hseTamperPolarity_t
- typedef uint8_t hseOutputPinConfig_t
- typedef uint8_t hseTamperOutputClock_t
- typedef hseAttrCfg_t hseDisableAppSpecificDataWrite_t
- typedef uint32_t hseAttrHseFwSize_t
- typedef uint32_t hseAvailAntiRollbackCounter_t

- typedef uint8_t hseAttrFwPartition_t
- typedef uint8_t hseAttrDisableAppDebug_t
- typedef uint32_t hseAttrOtpBootSeq_t
- typedef uint8_t hseTempSensVioConfig_t
- typedef uint8_t hseOtfadContextStatus_t
- typedef uint32_t hseDisableAntiRbcCfg_t
- typedef hseAttrCfg_t hseResetSocOnTamper_t
- typedef hseAttrCfg_t hseDisablePairWiseConsistencyTest_t

Data Structure Documentation

struct hseSetAttrSrv_t

Set HSE attribute service.

Note

SuperUser rights (for NVM Configuration) are needed to perform this service.

Data Fields

Type	Name	Description
hseAttrId_t	attrId	INPUT: Specifies the HSE attribute ID.
uint8_t	reserved[2]	
uint32_t	attrLen	INPUT: Specifies the attribute length (in bytes). The size of the memory location must be equal to the length of attribute structure.
uint64_t	pAttr	INPUT: The address of the attribute. The attribute must have the format of the corresponding attributes structure (see attributes definition) Note The comment for each attribute ID provides the name for the attribute data structure. E.g:The HSE_MU_CONFIG_ATTR_ID definition includes the following comment: "NVM-RW-ATTR; MU configuration (see #hseAttrMUConfig_t)".

struct hseGetAttrSrv_t

Get HSE attribute service.

Data Fields

Type	Name	Description
hseAttrId_t	attrId	INPUT: Specifies the HSE attribute ID.
uint8_t	reserved[2]	
uint32_t	attrLen	INPUT: Specifies the attribute length (in bytes).The size of the memory location must be bigger than or equal to the length of attribute structure.
uint64_t	pAttr	OUTPUT: The address where the attribute will be stored. The attribute must be stored in the format of the corresponding attribute Id (see the attributes definition). Note The comment for each attribute ID provides the name for the attribute data structure. E.g: The HSE_FW_VERSION_ATTR_ID definition includes the following comment: "RO-ATTR; HSE FW version (see #hseAttrFwVersion_t)".

struct hseAttrFwVersion_t

HSE FW version attribute (HSE_H/M/B attribute). This is a READ-ONLY global attribute.

Data Fields

Type	Name	Description
uint8_t	reserved	For HSE_B, it is used for OTA Config: 0 = Full Mem Config; 1 = AB Swap Config. For other SOC type: Reserved, expected to be 0.
uint8_t	socTypeId	Identifies the SoC Type ID; same as HSE_PLATFORM from hse_target.h.
uint16_t	fwTypeId	Identifies the FW type: – 0 - Standard FW targeting all customers – 1 - Premium FW targeting all customers – 2-7 - Reserved – 8 >= Custom1, Custom2... etc
uint8_t	majorVersion	Major revision. – 0 - Pre-stabilization releases – 1 - at first stable interface release, and increased later if breaking changes were introduced

Data Fields

Type	Name	Description
uint8_t	minorVersion	Minor revision, bumped on new compatible changes added; reset to 0 on majorVersion bump, if majorVersion > 0.
uint16_t	patchVersion	Hotfix release (patch version, bug fix releases). After majorVersion > 0, reset to 0 on majorVersion or minorVersion bump.

struct hseFwBuildInfo_t

HSE FW build information attribute. This is a READ-ONLY global attribute.

Data Fields

Type	Name	Description
uint64_t	buildUniqueIdentifier	Build unique identifier (8 bytes of SHA1 of the HEAD)
uint32_t	buildDate	Build date in hexadecimal (hex(YYYYMMDD))
uint32_t	buildTime	Build time in hexadecimal (hex(HHMMSS))

struct hseAttrSmrCoreStatus_t

The SMR and Core Boot status.

Provides the following infomation:

- SMR entry installation status corresponding to the entries present in SMR table (refer to smrEntryInstallStatus)
- SMR verification status corresponding to the entries present in SMR table (refer to smrStatus[])
- Provides Core Boot status (refer to coreBootStatus[])
- In case Basic Secure Boot (BSB) is performed, it provides the Core Boot status and the location of loaded application (primary/backup, refer to coreBootStatus[])

Data Fields

Type	Name	Description
uint32_t	smrStatus[2U]	0-31 bit will represent 32 SMR table entries (applicable when SMR is present/enabled). – smrStatus[0].bit : 0 - SMR Not verified – smrStatus[0].bit : 1 - SMR verified – smrStatus[1].bit : 0 - SMR verification fail – smrStatus[1].bit : 1 - SMR verification pass

Data Fields

Type	Name	Description
uint32_t	coreBootStatus[2U]	0-31 bit will represent CORE-ID (0-31): – coreBootStatus[0].bit : 1 - Core booted – coreBootStatus[0].bit : 0 - Core Not booted – coreBootStatus[1].bit : 1 - Core booted with pass/primary reset address – coreBootStatus[1].bit : 0 - Core booted with alternate/backup reset address
uint32_t	smrEntryInstallStatus	0-31 bit will represent 32 SMR table entries (applicable when SMR is present/enabled). – bit : 0 - SMR entry not installed – bit : 1 - SMR entry installed

struct hseAttrMUInstanceConfig_t

MU Configuration and XRDC configuration definition for a MU interface.

Configures a MU interface and XRDC configuration for the HOST Interface Memory.

Note

> If the device does have (or use) any Host Interface memory, the xrdcDomainId and sharedMemChunkSize can be set zero.

Data Fields

Type	Name	Description
hseMUConfig_t	muConfig	This value specifies MU interface state. – HSE_MU_ACTIVATED : MU interface activated – HSE_MU_DEACTIVATED : MU interface deactivated Note It is not allowed to deactivate the MU0 interface
uint8_t	xrdcDomainId	Domain Id to access the Host Interface memory chunk reserved for the MU interface. Must have a value between interval [0, 7]. The xrdcDomainId field is not taken into account when the sharedMemChunkSize field is equal to 0.

Data Fields

Type	Name	Description
uint16_t	sharedMemChunkSize	Specifies what chunk of host interface memory to reserve for the specific MU interface. For a value of 0 there is no memory reserved for the MU interface. If the sharedMemChunkSize field is equal to 0 for all MU interfaces, the XRDC is disabled and there are no restrictions on the host interface memory.
uint8_t	reserved[60]	

struct hseAttrMUConfig_t

MU Configurations and XRDC configuration definition.

Configures the MU interfaces and XRDC configurations for the HOST Interface Memory.

Data Fields

Type	Name	Description
hseAttrMUInstanceConfig_t	muInstances[(4U)]	Contains the configurations for all MU interfaces.

struct hseAttrMemRegion_t

HSE Memory region.

Defines base address and length of a region

Data Fields

Type	Name	Description
hseMemRegAccess_t	accessType	INPUT: Access type on which the region applies.
uint32_t	length	INPUT: Length of memory region.
uint64_t	pBaseAddr	INPUT: Start address of memory region.

struct hseAttrMuMemRegions_t

HSE Memory region attribute for a single MU.

Defines the number of regions and their start address and sizes for a single MU

Data Fields

Type	Name	Description
uint8_t	numofMemRegions	INPUT: Specify the number of memory regions for one MU. Note Set to zero if not used
uint8_t	reserved[3]	
hseAttrMemRegion_t	memRegionList[(12U)]	INPUT: Specifies the memory regions for one MU.

struct hseAttrAllMuMemRegions_t

HSE Memory regions protection attribute for all HSE MUs.

HSE Memory regions protection is a service used to prevent memory accesses between disallowed bus masters through HSE MUs. HSE uses these regions to validate the input/output parameters for each service received on the corresponding MU.

Note

- The attribute is not persistent and can only be set once.
  A reset is necessary for this configuration to be settable again.
- Input and output data linked via pointers in the service descriptor (these are typically pointers to SRAM or DRAM) can be isolated between hosts using the HSE_MEM_REGIONS_PROTECT_ATTR_ID attribute. The host can communicate to HSE the memory ranges that are associated with each MU instance. If provided, the HSE dismisses the data that falls outside the ranges for a particular MU instance.

Data Fields

Type	Name	Description
hseAttrMuMemRegions_t	muMemRegions[(4U)]	INPUT: Array with memory regions for all MUs.

struct hseAttrExtendCustSecurityPolicy_t

HSE extend CUST security policies attribute definition.

Determines whether certain security policies are extended in HSE Firmware or not; applies only for CUST_DEL LC.
- Read: Tells which extended security policies are set or not.
- Write:
  - If a given policy is not set to be TRUE, there is no change on security policy extension.
  - If a given policy is set to be TRUE, security policy is extended on successful operation.
  - Write operation is allowed only for users with CUST SU rights in CUST_DEL LC.

Data Fields

Type	Name	Description
bool_t	enableADKm	Application Debug Key/Password (attribute) diversified with UID before being written in fuse. The supplied 128-bit value for ADK/P attribute will be interpreted as ADKPm (customer's master key/ password). If needed, this policy must be set before setting ADK/P attribute. If set, the following logic must be used at customer's end for debug-authorization: – hUID = SHA2_256(UID) – hADKPm = SHA2_256(ADKPm) – ADKP {for debugger} = AES256-ECB(hUID(16 bytes..0 to 15)), key = hADKPm; {ADKPm = customer's master key/ password}. The hash of ADKPm (set using ADKP attribute) will be used as the key in the derivation of the application password. An error will be returned if the value of this attribute is given as 0 from host interface. Note For HSE_H/M, the UID is read by HSE from system fuses. The application must provide read access (xRDC restriction) to HSE in order to be able to read the UID.
bool_t	startAsUser	Host starts with User rights in LC = CUST_DEL. Note Setting this attribute will take effect only after publishing the SYS Image and issuing a reset.
uint8_t	reserved[2]	HSE reserved.

struct hseAttrExtendOemSecurityPolicy_t

HSE extend OEM security policies attribute definition.

Determines whether certain security policies are extended in HSE Firmware or not in OEM_PROD LC.
- Read: Tells which extended security policies are set or not.
- Write:
  - If a given policy is not set to be TRUE, there is no change on security policy extension.
  - If a given policy is set to be TRUE, security policy is extended on successful operation.
  - Write operation is allowed only for users with OEM SU rights in OEM_PROD LC.

Data Fields

Type	Name	Description
bool_t	startAsUser	Host starts with User rights in LC = OEM_PROD. Note Setting this attribute will take effect only after publishing the SYS Image and issuing a reset.
uint8_t	reserved[3]	HSE reserved.

struct hseAttrPhysicalTamper_t

Enables the tamper violation in HSE subsystem for all physical tampers supported by the SOC.

This service only enables the tamper violation in HSE subsystem for all physical tampers supported by the SOC. Once violation is active it cannot be disabled until next reset.
Physical tamper feature can be configured in following two ways:

1. Active Tamper Configuration
2. Passive tamper configuration

   Note

   User must configure the GPIO pins for tamper functionality before calling this service; otherwise, a false violation can be triggered by HSE. User is also recommended to protect the tamper GPIO configuration using register protection, virtual wrapper and XRDC configuration against further modification by any application running on host side.

Data Fields

Type	Name	Description
hseTamperConfig_t	tamperConfig	This field indicates the tamper configuration to be enable or not.

Data Fields

Type	Name	Description
hseOutputPinConfig_t	tamperOutputConfig	This parameter tells which type (Active or Passive) of input is connected to external tamper input. If it is an active input, up to 2 tamper options can be selected as input source for external tamper input. Based on the value of this parameter, the clock will be driven on this pad by HSE.
uint8_t	filterDuration	Configures the length of the digital glitch filter for the external tamper pin between 128 and 32640 SIRC clock cycles. Any assertion on external tamper that is equal to or less than the value of the digital glitch filter is ignored. The length of the glitches filtered out is: – 128 + ((FilterDuration - 1) x 256), where FilterDuration = 1, ... , 128. If the FilterDuration value is 0, then the glitch filter will not be enabled. Filter Duration is a must requirement for Active Tamper and optional for Passive Tamper.
hseTamperPolarity_t	tamperPolarity	This field indicates the polarity of the tamper to be be configured. It can be "Active LOW" or "Active HIGH". This parameter is considered only when the tamper source in tamperOutputConfig is selected as passive.
hseTamperOutputClock_t	tamperActiveClock	Determines the clock to be driven on the output pad of the tamper. This parameter is considered only when the tamper source in tamperOutputConfig is selected as active.
uint8_t	reserved[3]	HSE reserved.

struct hseAttrPhysicalTamperConfig_t

Physical Tamper Configurations.

Configures all available physical tamper instances.

Data Fields

Type	Name	Description
hseAttrPhysicalTamper_t	tamperInstances[(1U)]	Contains the configuration for all the physical temper interfaces.

struct hseAppSpecificData_t

Application-Specific Data stored in SYS-IMG (persistent memory)

It can be used to store persistent application data.

Note

- The attribute can be read at any time without any restriction. If the attribute is not set previously, a read request will fail.
- If one of the following conditions is met the attribute cannot be set:

1. If writing was disabled previously (see hseDisableAppSpecificDataWrite_t)
2. At start-up, when BOOT_SEQ == 1 and the secure boot (SMR or BSB) fails
3. If any of the on-demand/runtime SMR specified in smrPeriodicOnDemand flags failed.

Data Fields

Type	Name	Description
uint8_t	specificData[252U]	The data to be stored in SYS-IMG. Note: All bytes must be specified; if some bytes are not used, those can be set zero. The on-demand or periodic SMR(s) that can be used to restrict the writing of the attribute; if any of the smrPeriodicOnDemand SMR verification fails, the attribute can not be written. Note: Set this field to zero if the on-demand or periodic SMR are not used.
uint32_t	smrPeriodicOnDemand	

struct hseRbCounterInfo_t

Provides details about SYS-IMG and FW Blue image anti-rollback counters (RBC) from fuses and image's headers.

This attribute (read-only) can be read by the host to return:
- the OTP counter for SYS-IMG (from fuses)
- the OTP counter for HSE FW (from fuses)
- the counter from the loaded SYS-IMG header (returns 0xFF if the SYS-IMG was not loaded)
- the counter from the loaded Blue FW image header (returns 0xFF if the FW was loaded from pink FW image)

Data Fields

Type	Name	Description
uint8_t	sysImgOtpCounter	The OTP counter for SYS-IMG (from fuses)
uint8_t	fwImgOtpCounter	The OTP counter for HSE FW (from fuses)

Data Fields

Type	Name	Description
uint8_t	sysImgHeaderCounter	The counter from the loaded SYS-IMG header. If the SYS-IMG was not loaded, it returns 0xFF.
uint8_t	fwImgHeaderCounter	The counter from the loaded Blue FW image header. If the FW was loaded from pink FW image, it returns 0xFF.

struct hseOtfadCtxStatus_t

OTFAD context status.

The OTFAD context status for all OTFAD entries.

Data Fields

Type	Name	Description
hseOtfadContextStatus_t	OtfadCtxStatus[(1U)][(4U)]	Contains the status for all OTFAD region.

struct hseOtpRollbackProtectionPolicy_t

Disable the OTP rollback protection when updating the FW Blue Image and/or SYS-IMG.

After updating the new FW Blue Image or SYS-IMG (NVM keystore) in external flash, a system reset is needed to update the anti-rollback counter in OTP area (fuses). To be able to update the fuse counter, the VDD_EFUSE must be always powered or can be powered at start-up by HSE FW (if the VDD_EFUSE configuration word from IVT is supported; refer to the HSE Firmware Reference Manual).

Note

- Anti-rollback protection on FW-IMG is provided only when it is re-encrypted with a device specific key (only when the so called FW Blue Image is used).
- SuperUser rights are needed to configure this attribute.

Data Fields

Type	Name	Description
hseDisableAntiRbcCfg_t	disableOtpRollbackProtect	This field controls the rollback protection configuration for FW Blue Image and/or SYS-IMG update. Enables or disable the rollback protection; if enabled, it configures if the anti-RBC is updated in fuses at boot time or on-demand. See hseDisableAntiRbcCfg_t for more details.
hseAttrCfg_t	disallowRunningFromPinkFWImage	Prevents the firmware from running if it was loading from a Pink FW Image. This configuration is ignored for non-secure boot. – HSE_CFG_NO (default configuration): allow HSE to run if the firmware was loading from a Pink or Blue FW Image. – HSE_CFG_YES: disallow HSE to run if firmware was loading from a Pink FW Image. Only the FW Blue Image can be used after setting this field to HSE_CFG_YES. Note – Only the Blue FW image provides the rollback protection. Once the disallowRunningFromPinkFWImage attribute is set, no FW pink image shall be used in primary or back-up locations (only Blue FW image must be used). – For secure-boot use case, if disallowRunningFromPinkFWImage == HSE_CFG_YES and the firmware was loaded from PINK FW image , HSE performs a function reset; otherwise, it will continue its execution. – For non-secure boot, this configuration is ignored.

Macro Definition Documentation

HSE_CFG_NO

```
#define HSE_CFG_NO ((hseAttrCfg_t)(0x0UL))
```

NO, deactivate the configuration.

HSE_CFG_YES

```
#define HSE_CFG_YES ((hseAttrCfg_t)(0xB7A5C365UL))
```

YES, activate the configuration

HSE_NONE_ATTR_ID

```
#define HSE_NONE_ATTR_ID ((hseAttrId_t)0U)
```

HSE_FW_VERSION_ATTR_ID

```
#define HSE_FW_VERSION_ATTR_ID ((hseAttrId_t)1U)
```

RO-ATTR; HSE FW version (see hseAttrFwVersion_t)

HSE_CAPABILITIES_ATTR_ID

```
#define HSE_CAPABILITIES_ATTR_ID ((hseAttrId_t)2U)
```

RO-ATTR; HSE capabilities (see hseAttrCapabilities_t)

HSE_SMR_CORE_BOOT_STATUS_ATTR_ID

```
#define HSE_SMR_CORE_BOOT_STATUS_ATTR_ID ((hseAttrId_t)3U)
```

RO-ATTR; SMR verification & Core-boot status (see hseAttrSmrCoreStatus_t)

HSE_FW_BUILD_INFO_ATTR_ID

```
#define HSE_FW_BUILD_INFO_ATTR_ID ((hseAttrId_t)4U)
```

RO-ATTR; HSE FW Build information (see hseFwBuildInfo_t)

HSE_DEBUG_AUTH_MODE_ATTR_ID

```
#define HSE_DEBUG_AUTH_MODE_ATTR_ID ((hseAttrId_t)10U)
```

OTP-ATTR; Debug Authorization mode (see hseAttrDebugAuthMode_t)

HSE_APP_DEBUG_KEY_ATTR_ID

```
#define HSE_APP_DEBUG_KEY_ATTR_ID ((hseAttrId_t)11U)
```

OTP-ATTR; Application Debug Key / Password (see hseAttrApplDebugKey_t and hseAttrSecureApplDebugKey_t)

HSE_SECURE_LIFECYCLE_ATTR_ID

```
#define HSE_SECURE_LIFECYCLE_ATTR_ID ((hseAttrId_t)12U)
```

OTP-ADVANCE-ATTR; Secure Life-cycle (see hseAttrSecureLifecycle_t)

HSE_ENABLE_BOOT_AUTH_ATTR_ID

```
#define HSE_ENABLE_BOOT_AUTH_ATTR_ID ((hseAttrId_t)13U)
```

OTP-ATTR; IVT/ DCD Authentication bit for HSE H and IVT Authentication bit for HSE M (see hseAttrConfigBootAuth_t)

HSE_EXTEND_CUST_SECURITY_POLICY_ATTR_ID

```
#define HSE_EXTEND_CUST_SECURITY_POLICY_ATTR_ID ((hseAttrId_t)14U)
```

OTP-ATTR & NVM-RW-ATTR; HSE security policies extension in CUST_DEL lifecycle for user with CUST SU rights (see hseAttrExtendCustSecurityPolicy_t). \ Note that this attribute also enables the

ADKPm in OTP (ADKP diversified with UID), along with the START_AS_USER setting for CUST_DEL lifecycle.

HSE_MU_CONFIG_ATTR_ID

```
#define HSE_MU_CONFIG_ATTR_ID ((hseAttrId_t)20U)
```

NVM-RW-ATTR; MU configuration (see hseAttrMUConfig_t)

HSE_EXTEND_OEM_SECURITY_POLICY_ATTR_ID

```
#define HSE_EXTEND_OEM_SECURITY_POLICY_ATTR_ID ((hseAttrId_t)21U)
```

NVM-RW-ATTR; HSE security policies extension in OEM_PROD lifecycle for user with OEM SU rights (see hseAttrExtendOemSecurityPolicy_t)

HSE_FAST_CMAC_MIN_TAG_BIT_LEN_ATTR_ID

```
#define HSE_FAST_CMAC_MIN_TAG_BIT_LEN_ATTR_ID ((hseAttrId_t)22U)
```

NVM-RW-ATTR; The minimum tag bit length that can be used for Fast CMAC verify/generate (see hseAttrFastCmacMinTagBitLen_t)

HSE_CORE_RESET_RELEASE_ATTR_ID

```
#define HSE_CORE_RESET_RELEASE_ATTR_ID ((hseAttrId_t)23U)
```

NVM-RW-ATTR; Specifies Core Reset table parsing strategy (see hseAttrCoreResetRelease_t)

HSE_RAM_PUB_KEY_IMPORT_POLICY_ATTR_ID

```
#define HSE_RAM_PUB_KEY_IMPORT_POLICY_ATTR_ID ((hseAttrId_t)24U)
```

NVM-RW-ATTR; Specifies RAM public keys import policy in advanced LCs (see hseAttrRamPubKeyImportPolicy_t)

HSE_RESET_SOC_ON_TAMPER_ATTR_ID

```
#define HSE_RESET_SOC_ON_TAMPER_ATTR_ID ((hseAttrId_t)25U)
```

NVM-RW-ATTR; Reset Soc on tamper detection (see hseResetSocOnTamper_t)

HSE_OTP_ROLLBACK_PROTECTION_POLICY_ATTR_ID

```
#define HSE_OTP_ROLLBACK_PROTECTION_POLICY_ATTR_ID ((hseAttrId_t)26U)
```

NVM-RW-ATTR; Disable or enable (default) the OTP rollback protection for FW Blue Image and SYS-IMG (see hseOtpRollbackProtectionPolicy_t)

HSE_APP_SPECIFIC_DATA_ATTR_ID

```
#define HSE_APP_SPECIFIC_DATA_ATTR_ID ((hseAttrId_t)27U)
```

NVM-RW-ATTR; Set an application-specific data of maximum #HSE_APP_SPECIFIC_DATA_MAX_BUFFER_SIZE bytes (see hseAppSpecificData_t)

HSE_DISABLE_APP_SPECIFIC_DATA_WRITE_ATTR_ID

```
#define HSE_DISABLE_APP_SPECIFIC_DATA_WRITE_ATTR_ID ((hseAttrId_t)28U)
```

SET-ONLY-ONCE-ATTR; Disable the write of APP_SPECIFIC_DATA attribute (see hseDisableAppSpecificDataWrite_t)

HSE_PHYSICAL_TAMPER_ATTR_ID

```
#define HSE_PHYSICAL_TAMPER_ATTR_ID ((hseAttrId_t)30U)
```

SET-ONLY-ONCE-ATTR; Enables the physical tamper violation in HSE. Once the violation is enabled in HSE, it can not be cleared until next reset. There are two tamper related functions available on PADs: Input (TAMPER_IN), Output (TAMPER_OUT). To support protection against physical tampering, connect TAMPER_OUT to TAMPER_IN. Any physical tamper that breaks this connectivity sets off an alarm at HSE (if enabled using this attribute). User can optionally lock those pads configuration for further modification using virtual wrapper (refer to hseAttrPhysicalTamper_t). The configuration status is provided by reading the HSE_GPR_STATUS_ADDRESS register (refer to hseTamperConfigStatus_t).

HSE_MEM_REGIONS_PROTECT_ATTR_ID

```
#define HSE_MEM_REGIONS_PROTECT_ATTR_ID ((hseAttrId_t)31U)
```

SET-ONLY-ONCE-ATTR; Configures memory regions accessible through each MU (refer to hseAttrAllMuMemRegions_t)

HSE_DISABLE_PAIRWISE_CONSISTENCY_TEST_ATTR_ID

```
#define HSE_DISABLE_PAIRWISE_CONSISTENCY_TEST_ATTR_ID ((hseAttrId_t)32U)
```

SET-ONLY-ONCE-ATTR; Disable the pair wise consistency test when calling import RSA/ECC key pair (see hseDisablePairWiseConsistencyTest_t)

HSE_FW_SIZE_ATTR_ID

```
#define HSE_FW_SIZE_ATTR_ID ((hseAttrId_t)100U)
```

RO-ATTR; HSE Firmware Size (see hseAttrHseFwSize_t)

HSE_AVAIL_ANTI_ROLLBACK_COUNTER_ATTR_ID

```
#define HSE_AVAIL_ANTI_ROLLBACK_COUNTER_ATTR_ID ((hseAttrId_t)101U)
```

RO-ATTR; The anti-rollback counter updates left (see hseAvailAntiRollbackCounter_t)

HSE_FW_PARTITION_ATTR_ID

```
#define HSE_FW_PARTITION_ATTR_ID ((hseAttrId_t)102U)
```

RO-ATTR; The partition (primary or backup) used by BootRom to load the HSE Firmware (see hseAttrFwPartition_t)

HSE_OTFAD_CTX_STATUS_ATTR_ID

```
#define HSE_OTFAD_CTX_STATUS_ATTR_ID ((hseAttrId_t)103U)
```

RO-ATTR; Otfad contexts status (see hseOtfadContextStatus_t).

HSE_RB_COUNTER_INFO_ATTR_ID

```
#define HSE_RB_COUNTER_INFO_ATTR_ID ((hseAttrId_t)105U)
```

RO-ATTR; Get the anti-rollback counters information (see hseRbCounterInfo_t)

HSE_APP_DEBUG_DIS_ATTR_ID

```
#define HSE_APP_DEBUG_DIS_ATTR_ID ((hseAttrId_t)200U)
```

OTP-ATTR; Disable Application Debug (see hseAttrDisableAppDebug_t)

HSE_OTP_BOOT_SEQ_ATTR_ID

```
#define HSE_OTP_BOOT_SEQ_ATTR_ID ((hseAttrId_t)201U)
```

OTP-ATTR; Configures OTP_BOOT_SEQ==1 (secure boot) (see hseAttrOtpBootSeq_t)

HSE_TEMP_SENSOR_VIO_CONFIG_ATTR_ID

```
#define HSE_TEMP_SENSOR_VIO_CONFIG_ATTR_ID ((hseAttrId_t)400U)
```

SET-ONLY-ONCE-ATTR; Enable the temperature sensor violation in HSE (see hseTempSensVioConfig_t)

HSE_ALGO_CAP_MASK

```
#define HSE_ALGO_CAP_MASK( capIdx ) (1ULL << (capIdx))
```

Provided the bit (used in hseAttrCapabilities_t) based on the algorithm capability index (see hseAlgoCapIdx_t)

HSE_CR_RELEASE_ALL_AT_ONCE

```
#define HSE_CR_RELEASE_ALL_AT_ONCE ((hseAttrCoreResetRelease_t)0xA5556933UL)
```

Cores are released all-at-once after the pre-boot verification phase is over.

HSE_CR_RELEASE_ONE_BY_ONE

```
#define HSE_CR_RELEASE_ONE_BY_ONE ((hseAttrCoreResetRelease_t)0xA5557555UL)
```

Cores are released from reset one-by-one after their respective pre-boot phase has finalized successfully (i.e. the SMR entries linked to the core via CR table have been loaded and verified).
The cores are released in ascending order of their indicies in the Core Reset table.
Flashless devices (e.g. HSE_H/M) limitations:

- Only the first Core Reset entry can be booted from SD/MMC.
- The system clocks and QSPI configurations shall not be changed by the core(s) booted until HSE_STATUS_BOOT_OK status is set.

HSE_DEBUG_AUTH_MODE_PW

```
#define HSE_DEBUG_AUTH_MODE_PW ((hseAttrDebugAuthMode_t)0x0U)
```

Password based application debug authorization mode.

- Read: Application debug authorization will be password based.
- Write: Does not affect application debug authorization mode at all.

HSE_DEBUG_AUTH_MODE_CR

```
#define HSE_DEBUG_AUTH_MODE_CR ((hseAttrDebugAuthMode_t)0x1U)
```

Challenge-Response based application debug authorization mode.

- Read: Application debug authorization will be challenge-response based.
- Write: Enables challenge-response application debug authorization mode. Once this mode is enabled, it cannot be disabled. Operation allowed in CUST_DEL, OEM_PROD and IN_FIELD LCs only.

HSE_LC_CUST_DEL

```
#define HSE_LC_CUST_DEL ((hseAttrSecureLifecycle_t)0x4U)
```

Customer Delivery Lifecycle.

- Read: The current LC is CUST_DEL.
- Write: Advancement to this LC is not allowed (through HSE Firmware).

HSE_LC_OEM_PROD

```
#define HSE_LC_OEM_PROD ((hseAttrSecureLifecycle_t)0x8U)
```

OEM Production Lifecycle.
- Read: The current LC is OEM_PROD.
- Write: Advancement to this LC is allowed only once (from CUST_DEL LC). The key catalogs MUST be configured before advancing to this lifecycle.

HSE_LC_IN_FIELD

```
#define HSE_LC_IN_FIELD ((hseAttrSecureLifecycle_t)0x10U)
```

In-Field Lifecycle.
- Read: The current LC is IN_FIELD.
- Write: Advancement to this LC is allowed only once (from CUST_DEL, OEM_PROD LCs). The key catalogs MUST be configured before advancing to this lifecycle.

HSE_LC_PRE_FA

```
#define HSE_LC_PRE_FA ((hseAttrSecureLifecycle_t)0x14U)
```

Pre-Failure Analysis Lifecycle.
- Read: The current LC is Pre-FA.
- Write: Advancement from/to this LC is NOT allowed (through HSE Firmware).

HSE_LC_SIMULATED_OEM_PROD

```
#define HSE_LC_SIMULATED_OEM_PROD ((hseAttrSecureLifecycle_t)0xA6U)
```

Simulated OEM_PROD to avoid writing in FUSE/UTEST. A system reset will revert LC to FUSE/UTEST value.
- Read: The current LC is OEM_PROD.
- Write: Advancement to this LC is allowed only once (from CUST_DEL LC). The key catalogs MUST be configured before advancing to this lifecycle.

HSE_LC_SIMULATED_IN_FIELD

```
#define HSE_LC_SIMULATED_IN_FIELD ((hseAttrSecureLifecycle_t)0xA7U)
```

Simulated IN_FIELD to avoid writing in FUSE/UTEST. A system reset will revert LC to FUSE/UTEST value.
- Read: The current LC is IN_FIELD.
- Write: Advancement to this LC is allowed only once (from CUST_DEL, SIMULATED_OEM_PROD LCs). The key catalogs MUST be configured before advancing to this lifecycle.

HSE_IVT_NO_AUTH

```
#define HSE_IVT_NO_AUTH ((hseAttrConfigBootAuth_t)0x0U)
```

For HSE_H/M, the IVT/DCD/ST is not authenticated by BootROM:
- Read: IVT/DCD/ST is not authenticated by BootROM.
- Write: Does not affect IVT/ DCD authentication value at all.

For HSE_B, the IVT configuration is not authenticated by Secure BAF:
- Read: IVT is not authenticated by Secure BAF.
- Write: Does not affect IVT configuration authentication value at all.

HSE_IVT_AUTH

```
#define HSE_IVT_AUTH ((hseAttrConfigBootAuth_t)0x1U)
```

For HSE_H/M, the IVT/DCD/ST to be authenticated by BootROM:
- Read: IVT/DCD/ST is authenticated by BootROM.
- Write: Sets IVT/DCD/ST authentication value. Once this value is set, it cannot be cleared back. Operation allowed in CUST_DEL, OEM_PROD & IN_FIELD LCs only.

For HSE_B, the IVT to be authenticated by Secure BAF:
- Read: IVT will be authenticated by Secure BAF.
- Write: Sets IVT authentication value. Once this value is set, it cannot be cleared back. Operation allowed in CUST_DEL, OEM_PROD & IN_FIELD LCs only.

HSE_MU_ACTIVATED

```
#define HSE_MU_ACTIVATED ((hseMUConfig_t)(0xA5U))
```

HSE enables the receive interrupt on the MU interface.

HSE_MU_DEACTIVATED

```
#define HSE_MU_DEACTIVATED ((hseMUConfig_t)(0x5AU))
```

HSE disables the receive interrupt on the MU interface.

HSE_MEM_REG_ACCESS_MASK_IN

```
#define HSE_MEM_REG_ACCESS_MASK_IN ((hseMemRegAccess_t)(0x00003C96UL))
```

HSE_MEM_REG_ACCESS_MASK_OUT

```
#define HSE_MEM_REG_ACCESS_MASK_OUT ((hseMemRegAccess_t)(0x5A690000UL))
```

HSE_MEM_REG_ACCESS_MASK_INOUT

```
#define HSE_MEM_REG_ACCESS_MASK_INOUT ((hseMemRegAccess_t)( HSE_MEM_REG_ACCESS_
MASK_IN| HSE_MEM_REG_ACCESS_MASK_OUT))
```

HSE_KM_POLICY_DEFAULT

```
#define HSE_KM_POLICY_DEFAULT ((hseAttrRamPubKeyImportPolicy_t)(0x4E8BD124UL))
```

HSE_KM_POLICY_ALLOW_RAM_PUB_KEY_IMPORT

```
#define
HSE_KM_POLICY_ALLOW_RAM_PUB_KEY_IMPORT ((hseAttrRamPubKeyImportPolicy_t)(0xB1742EDBUL))
```

HSE_TAMPER_CONFIG_DEACTIVATE

```
#define HSE_TAMPER_CONFIG_DEACTIVATE ((hseTamperConfig_t)(0U))
```

HSE Tamper Deactivate.

HSE_TAMPER_CONFIG_ACTIVATE

```
#define HSE_TAMPER_CONFIG_ACTIVATE ((hseTamperConfig_t)(1U))
```

HSE Tamper Activate.

HSE_TAMPER_POL_ACTIVE_LOW

```
#define HSE_TAMPER_POL_ACTIVE_LOW ((hseTamperPolarity_t)(0U))
```

HSE Tamper Active low polarity.

HSE_TAMPER_POL_ACTIVE_HIGH

```
#define HSE_TAMPER_POL_ACTIVE_HIGH ((hseTamperPolarity_t)(1U))
```

HSE Tamper Active high polarity.

HSE_FILTER_DURATION_MAX

```
#define HSE_FILTER_DURATION_MAX ((uint32_t)128U)
```

Filter Duration.

This macro describes the maximum filter duration that is possible for the physical tamper. The clock frequency used in the glitch filter is 32 KHz.

HSE_TAMPER_PASSIVE

```
#define HSE_TAMPER_PASSIVE ((hseOutputPinConfig_t)(0U))
```

HSE_TAMPER_ACTIVE_ONE

```
#define HSE_TAMPER_ACTIVE_ONE ((hseOutputPinConfig_t)(1U))
```

HSE_TAMPER_ACTIVE_TWO

```
#define HSE_TAMPER_ACTIVE_TWO ((hseOutputPinConfig_t)(2U))
```

HSE_TAMPER_ACTIVE_CLOCK_16HZ

```
#define HSE_TAMPER_ACTIVE_CLOCK_16HZ ((hseTamperOutputClock_t)(0U))
```

HSE_TAMPER_ACTIVE_CLOCK_8HZ

```
#define HSE_TAMPER_ACTIVE_CLOCK_8HZ ((hseTamperOutputClock_t)(1U))
```

HSE_TAMPER_ACTIVE_CLOCK_4HZ

```
#define HSE_TAMPER_ACTIVE_CLOCK_4HZ ((hseTamperOutputClock_t)(2U))
```

HSE_TAMPER_ACTIVE_CLOCK_2HZ

```
#define HSE_TAMPER_ACTIVE_CLOCK_2HZ ((hseTamperOutputClock_t)(3U))
```

HSE_FW_PARTITION_PRIMARY

```
#define HSE_FW_PARTITION_PRIMARY ((hseAttrFwPartition_t)0x1U)
```

HSE firmware was loaded from primary partition.

HSE_FW_PARTITION_BACKUP

```
#define HSE_FW_PARTITION_BACKUP ((hseAttrFwPartition_t)0x2U)
```

HSE firmware was loaded from back-up partition.

HSE_APP_DEBUG_DIS_NONE

```
#define HSE_APP_DEBUG_DIS_NONE ((hseAttrDisableAppDebug_t)0x0U)
```

Application Debug not disabled.

– Read: Application Debug is not disabled for OEM_PROD/ IN_FIELD LC. Application debug can be opened in OEM_PROD/ IN_FIELD LC using the debug authorization mechanism.
– Write: Does not disable the application debug.

HSE_APP_DEBUG_DIS_OEM

```
#define HSE_APP_DEBUG_DIS_OEM ((hseAttrDisableAppDebug_t)0x1U)
```

Application Debug disabled for OEM_PROD LC.

– Read: Application Debug is disabled for OEM_PROD LC. Application debug can never be opened in OEM_PROD LC.
– Write: Disables application debug for OEM_PROD LC only. Operation allowed in CUST_DEL, OEM_PROD & IN_FIELD LCs only.

HSE_APP_DEBUG_DIS_FLD

```
#define HSE_APP_DEBUG_DIS_FLD ((hseAttrDisableAppDebug_t)0x2U)
```

Application Debug disabled for IN_FIELD LC.

– Read: Application Debug is disabled for IN_FIELD LC. Application debug can never be opened in IN_FIELD LC.
– Write: Disables application debug for IN_FIELD LC only. Operation allowed in CUST_DEL, OEM_PROD & IN_FIELD LCs only.

HSE_APP_DEBUG_DIS_OEM_FLD

```
#define HSE_APP_DEBUG_DIS_OEM_FLD ((hseAttrDisableAppDebug_t)0x3U)
```

Application Debug disabled for both OEM_PROD & IN_FIELD LCs.

– Read: Application Debug is disabled for both OEM_PROD & IN_FIELD LCs. Application debug can never be opened in OEM_PROD & IN_FIELD LCs. -Write: Disables application debug for both OEM_PROD & IN_FIELD LCs. Operation allowed in CUST_DEL, OEM_PROD & IN_FIELD LCs only.

HSE_OTP_BOOT_SEQ_MARKER

```
#define HSE_OTP_BOOT_SEQ_MARKER (0xCEDEADDAUL)
```

The marker that must be used when setting hseAttrOtpBootSeq_t attribute.

HSE_OTP_BOOT_SEQ_NOT_SET

```
#define HSE_OTP_BOOT_SEQ_NOT_SET (0x00000000UL)
```

The OTP_BOOT_SEQ is not set in fuses.

HSE_TEMP_SENS_VIO_ACTIVATED

```
#define HSE_TEMP_SENS_VIO_ACTIVATED ((hseTempSensVioConfig_t)(0xA5U))
```

HSE enables the temperature sensor violation in SNVS.

HSE_TEMP_SENS_VIO_DEACTIVATED

```
#define HSE_TEMP_SENS_VIO_DEACTIVATED ((hseTempSensVioConfig_t)(0x5AU))
```

HSE disables the temperature sensor violation in SNVS.

HSE_OTFAD_CTX_NOT_INSTALLED

```
#define HSE_OTFAD_CTX_NOT_INSTALLED ((hseOtfadContextStatus_t)(0x00U))
```

OTFAD context not installed.

HSE_OTFAD_CTX_INSTALLED

```
#define HSE_OTFAD_CTX_INSTALLED ((hseOtfadContextStatus_t)(0xCAU))
```

OTFAD context installed but not configured.

HSE_OTFAD_CTX_ACTIVE

```
#define HSE_OTFAD_CTX_ACTIVE ((hseOtfadContextStatus_t)(0xACU))
```

OTFAD context configured and active.

HSE_OTFAD_CTX_INACTIVE

```
#define HSE_OTFAD_CTX_INACTIVE ((hseOtfadContextStatus_t)(0xDEU))
```

OTFAD context configured but not active.

HSE_DIS_ANTI_RBC_CFG_NO

```
#define HSE_DIS_ANTI_RBC_CFG_NO ((hseDisableAntiRbcCfg_t)( HSE_CFG_NO))
```

The rollback protection mechanism is enabled, \ and the anti-RBC counter is updated automatically at start-up \ if LC != CUST_DEL or BOOT_SEQ == 1 and at least one core is booted. \ This is the default configuration.

HSE_DIS_ANTI_RBC_CFG_YES

```
#define HSE_DIS_ANTI_RBC_CFG_YES ((hseDisableAntiRbcCfg_t)( HSE_CFG_YES))
```

The rollback protection is disabled (the rollback protection mechanism is disabled and anti-RBC is not updated in fuses).

HSE_ON_DEMAND_ANTI_RBC_UPDATE

```
#define HSE_ON_DEMAND_ANTI_RBC_UPDATE ((hseDisableAntiRbcCfg_t)(0x676E2064UL))
```

The rollback protection mechanism is enabled, \ and the anti-RBC is updated in fuses only \ on-demand by calling the hseOnDemandAntiRbcUpdateSrv_t service.

Typedef Documentation

hseAttrCfg_t

```
typedef uint32_t hseAttrCfg_t
```

Activate or not a specific configuration.

Tells whether the HSE activate or not a specific configuration.

hseAttrId_t

```
typedef uint16_t hseAttrId_t
```

HSE attribute IDs.

The following attribute types are defined:
- RO-ATTR - Read-Only attribute
- OTP-ATTR - One Time Programmable; can be written only once (set FUSE/UTEST area)
- OTP-ADVANCE-ATTR - One Time Programmable attribute that can only be advanced (e.g. LifeCycle)
- NVM-RW-ATTR - System NVM attributes; can be read or written
- SET-ONCE-ATTR- Once the attribute is set, it can not be changed until next reset (e.g. can be set once at initialization time)

Note

- For HSE_H/M devices, if the NVM-RW attributes were updated, the SYS-IMAGE must be published and stored in external flash.
- For HSE_H/M devices, in order to program fuses (OTP-ATTR and OTP-ADVANCE-ATTR) during development and potentially in the field, the application must supply power to the VDD_EFUSE pin of the device. This allows the programming operation to take place. If theVDD_EFUSE is not powered and the application tries to write a fuse, the HSE returns an error (HSE_SRV_RSP_FUSE_VDD_GND). The VDD_EFUSE state is checked before the fuse write by reading the NCSPD_STAT register of the on-chip PMC module. The application must provide read access (xRDC restriction) to HSE in order to be able to read the NCSPD_STAT register. For more details, refer to HSE FW Reference Manual.
- To set/update the OTP or NVM attributes (except SET-ONCE-ATTR and OTP-ADVANCE-ATTR), the host needs SuperUser rights.
- After setting the HSE_SECURE_LIFECYCLE_ATTR_ID attribute, a destructive reset is needed.

hseAttrCapabilities_t

```
typedef uint64_t hseAttrCapabilities_t
```

HSE capabilities bits definition.

Provides information about the capabilities of HSE security blocks (list of what algorithms are supported). Each bit specifies an supported algorithm. The index for each bit in the attribute is defined by hseAlgoCapIdx_t.

hseAttrCoreResetRelease_t

```
typedef uint32_t hseAttrCoreResetRelease_t
```

The Core Reset release from reset method.

Specifies the startup method for releasing the application core from reset.

hseAttrDebugAuthMode_t

```
typedef uint8_t hseAttrDebugAuthMode_t
```

Debug Authorization Mode bit (HSE_H/M/B attribute).

Tells whether the Application debug authorization will be password based or challenge-response based.

hseAttrApplDebugKey_t

```
typedef uint8_t hseAttrApplDebugKey_t[16]
```

Application Debug Key/ Password definition (HSE_H/M/B attribute).

It is an 128-bit Application Debug Key/ Password to be set by the host in CUST_DEL LifeCycle.

- Read: Not allowed if ADKP has not been written yet. After it has been written, first 16 bytes of SHA2_224(ADKP) can be requested via get ADKP attribute service.
- Write: ADKP can be updated only once. The operation allowed only in CUST_DEL LifeCycle.

Note

> Provided ADKP, if containing all 0x00 bytes or all 0xFF bytes will be rejected by HSE Firmware with the response HSE_ SRV_ RSP_ INVALID_ PARAM .

hseAttrSecureApplDebugKey_t

```
typedef hseKeyHandle_t hseAttrSecureApplDebugKey_t
```

Secure Application Debug Key/ Password definition (HSE_H/M/B attribute).

It is the key handle referencing a key already installed in HSE. It must be an AES 128-bits key from RAM or NVM key catalogs.

- Read: Allowed only as the hash over the ADKP (see Read from hseAttrApplDebugKey_t).
- Write:
  - ADKP can be updated only once. The operation allowed only in CUST_DEL LifeCycle.
  - The key referenced must be installed in HSE a priori. After the key is written successfully in the fuse as ADK/P, it will be erased from the RAM/NVM key catalog.

hseAttrSecureLifecycle_t

```
typedef uint8_t hseAttrSecureLifecycle_t
```

HSE secure lifecycle definition.

Represents HSE secure lifecycle. The lifecycle can be advanced only in forward direction. Warnings:

- The lifecycle is read/scanned by hardware during the reset phase. Hence, a reset is recommended after each LC write-advance operation.
- The lifecycle can be advanced to OEM_PROD/IN_FIELD only if the HSE_APP_DEBUG_KEY_ATTR_ID attribute was set before.

hseAttrConfigBootAuth_t

```
typedef uint8_t hseAttrConfigBootAuth_t
```

Boot Authentication bit.

Value used by Boot ROM to check whether the IVT data needs be authenticated.

hseMUConfig_t

```
typedef uint8_t hseMUConfig_t
```

MU configuration byte (HSE_H/M/B attribute).

Tells whether the HSE enables the receive interrupt on the configured MU interface.

hseMemRegAccess_t

```
typedef uint32_t hseMemRegAccess_t
```

Access types for HSE_SPT_MEM_REGION_PROTECT service regions.

hseAttrRamPubKeyImportPolicy_t

```
typedef uint32_t hseAttrRamPubKeyImportPolicy_t
```

HSE key management policy regarding RAM public keys import.

Determines whether public keys can be imported without authentication in advanced LCs.
Default value is HSE_KM_POLICY_DEFAULT, i.e. HSE does not allow public key import in RAM, when having User rights, if they are not an authenticated key container.
Otherwise, if set to HSE_KM_POLICY_ALLOW_RAM_PUB_KEY_IMPORT, RAM public keys are allowed to be imported without authentication, regardless of the access rights.
SU access rights with configuration privileges are required to update this attribute value.

hseAttrFastCmacMinTagBitLen_t

```
typedef uint8_t hseAttrFastCmacMinTagBitLen_t
```

Minimal tag bit length for Fast CMAC service.

By default, the minimal tag bit length that can be used for the Fast CMAC service (see hseFastCMACSrv_t) is HSE_DEFAULT_MIN_FAST_CMAC_TAG_BITLEN bits. This attribute can be set to be able to use the Fast CMAC service with the tag bit length less than HSE_DEFAULT_MIN_FAST_CMAC_TAG_BITLEN bits. The value to be set must be provided in bits.

hseTamperConfig_t

```
typedef uint8_t hseTamperConfig_t
```

Activate or Deactivate a tamper.

Tells whether tamper needs to be activated or deactivated.

hseTamperPolarity_t

```
typedef uint8_t hseTamperPolarity_t
```

Tamper Polarity.

Specifies the polarity to activate the tamper. This configuration is applicable only for passive tamper configuration. User must set the default state of the tamper input pin accordingly on the board. For example: If the tamper polarity is set "ACTIVE_HIGH" then the default state on the tamper input pin must be "ACTIVE LOW".

hseOutputPinConfig_t

```
typedef uint8_t hseOutputPinConfig_t
```

Tamper routing configuration.

This configuration defines the type of tamper (i.e. active or passive).

- In case of active tamper, the clock is derived on GPIO pad which should be routed back to the input tamper pin on the ECU. User must configure the alternate functionality of GPIO pin to tamper output so that the clock can be routed on that pin.
- In case of passive tamper, HSE senses the change in polarity of the input pin. In this case, there is no need to configure the active tamper pin. Only external tamper pin should be configured.
- User is recommended to refer the SIUL chapter in SOC reference manual to configure the correct GPIO pin. For some SOC types, only one active tamper can be supported. Please refer to #HSE_NUM_OF_PHYSICAL_TAMPER_INSTANCES to see how many active tamper are supported.

Note

HSE_TAMPER_ACTIVE_TWO is not valid for devices - S32G2, S32K3xx

hseTamperOutputClock_t

```
typedef uint8_t hseTamperOutputClock_t
```

Tamper clock that needs to be driven on the tamper output pad.

Tamper clock that needs to be driven on the tamper output pad. Please note that the alternate functionality of GPIO pin must be configured (for the tamper functionality) so that below the mentioned clock can be driven on that pad. Not applicable for passive tamper configuration

hseDisableAppSpecificDataWrite_t

```
typedef hseAttrCfg_t hseDisableAppSpecificDataWrite_t
```

Disable the writing of Application-Specific Data attribute.

The write of Application-Specific Data attribute can be disabled by setting this attribute as HSE_CFG_YES .

Note

- When BOOT_SEQ ==1, if the secure boot at start-up and the on-demand or periodic SMR check (configured using hseAppSpecificData_t service) fails, this attribute is automatically set to HSE_CFG_YES (application specific data can not be written).
- At start-up, the default value of the attribute is HSE_CFG_NO . The attribute can only be modified from HSE_CFG_NO to HSE_CFG_YES , using the set attribute service. The attribute can be set only once, meaning that once the attribute is set to HSE_CFG_YES , it cannot be set back to HSE_CFG_NO in the current power cycle.

hseAttrHseFwSize_t

```
typedef uint32_t hseAttrHseFwSize_t
```

HSE-Firmware Size.

Size of HSE-Firmware in bytes.

hseAvailAntiRollbackCounter_t

```
typedef uint32_t hseAvailAntiRollbackCounter_t
```

Anti-rollback counter updates left.

There are available 158 anti-rollback counter updates (fuses) for the key store and HSE firmware. After 158 updates, the key store and HSE firmware are not protected against rollbacks.

hseAttrFwPartition_t

```
typedef uint8_t hseAttrFwPartition_t
```

HSE-Firmware used partition on load.

Specified the partition (primary or backup) used by BootRom to load the HSE Firmware.

hseAttrDisableAppDebug_t

```
typedef uint8_t hseAttrDisableAppDebug_t
```

Application debug disable.

Tells if the Application debug is disabled or not for OEM_PROD and/or IN_FIELD life-cycles.

hseAttrOtpBootSeq_t

```
typedef uint32_t hseAttrOtpBootSeq_t
```

Set BOOT_SEQ flag in OTP (called OTP_BOOT_SEQ below).

This attribute complements the BOOT_SEQ flag in IVT with a flag in OTP. Setting this attribute enforces the IVT's BOOT_SEQ (IVT_BOOT_SEQ) to 1. When setting this attribute:
- the HSE_OTP_BOOT_SEQ_MARKER marker must be used
- the IVT_BOOT_SEQ must be 1 On read, if the OTP_BOOT_SEQ in fuses is set, HSE returns the HSE_OTP_BOOT_SEQ_MARKER marker; otherwise, it returns HSE_OTP_BOOT_SEQ_NOT_SET.

When the HSE FW starts, it verifies the value of OTP_BOOT_SEQ (from fuses) against the value of IVT_BOOT_SEQ as follows:
- If(LC == OEM_PROD or IN_FIELD) and (OTP_BOOT_SEQ == 1) and (OTP_BOOT_SEQ != IVT_BOOT_SEQ), apply a functional reset; Otherwise, continue the boot sequence.

Note

:
- After 8 functional resets, BootROM enters serial boot mode.

hseTempSensVioConfig_t

```
typedef uint8_t hseTempSensVioConfig_t
```

Temperature Sensor violation configuration byte.

Once the violation is enabled in HSE, it can not be cleared until next reset. User must configure the Temperature Monitoring Unit (TMU) before giving the attribute. It can also be configured via DCD configuration. The HSE Firmware signals an Fatal error (see hseError_t bits) if this tamper is detected. User is recommended to protect the TMU Registers (see REG_PROT on Soc) after the configuration. The tamper configuration status is available by reading the HSE_GPR_STATUS_ADDRESS register (refer to hseTamperConfigStatus_t). Four TMU Monitors are mapped to HSE: Average High Critical Temperature TMU Monitor, Average Low Critical Temperature TMU Monitor, Rising Rate Critical Temperature TMU Monitor, Falling Rate Critical Temperature TMU Monitor.

hseOtfadContextStatus_t

```
typedef uint8_t hseOtfadContextStatus_t
```

OTFAD context status.

After installation of the otfad context, the OTFAD region may be activated or deactivated. The OTFAD region may be deactivated because the HSE_OTFAD_CTX_ACTIVE_ON_BOOT flag is not set during installation or due to a configuration error.

hseDisableAntiRbcCfg_t

```
typedef uint32_t hseDisableAntiRbcCfg_t
```

Configuration option for anti-rollback counter (anti-RBC) handling.

Tells if the rollback protection mechanism is active or not, or if the anti-RBC is updated in fuses at boot time or on-demand by calling the hseOnDemandAntiRbcUpdateSrv_t service.

hseResetSocOnTamper_t

```
typedef hseAttrCfg_t hseResetSocOnTamper_t
```

HSE Reset Soc on tamper detection.

By default HSE does not reset the Soc on tamper detection (this attribute is configured as HSE_CFG_NO), Instead it signals an HSE error (see hseError_t) and enter shutdown mode. To reset the Soc, the host application must set this attribute to HSE_CFG_YES and publish SYS-IMG. On next boot, HSE configures the attribute.

Note

> The Application must have SU rights to configure this attribute.

hseDisablePairWiseConsistencyTest_t

```
typedef hseAttrCfg_t hseDisablePairWiseConsistencyTest_t
```

Disable the pair wise consistency test at import for RSA/ECC/DH key pairs.

By default, when importing a RSA/ECC/DH key pair, HSE checks the pair wise consistency of the provided public and private keys. Since this operation is time consuming, the host can disable this check to speed up the key import operation. To disable the pair wise consistency test, the host must set this attribute to HSE_CFG_YES . By default, this attribute is set to HSE_CFG_NO (pair wise consistency is checked).

Note

: This is a "SET-ONLY-ONCE-ATTR" attribute. The attribute can only be set from HSE_CFG_NO to HSE_CFG_YES , using the set attribute service. Once the attribute is set to HSE_CFG_YES , it cannot be set back to HSE_CFG_NO in the current power cycle.

3.3 HSE System Authorization Services

Data Structures

- struct hseSysAuthorizationReqSrv_t
- struct hseSysAuthorizationRespSrv_t

Macros

Type: (implicit C type)	
Name	**Value**
HSE_SYS_AUTH_ALL	(HSE_SYS_AUTH_KEY_MGMT) | (HSE_SYS_AUTH_NVM_CONFIG)
HSE_SYS_AUTH_CHALLENGE_LENGTH	32UL

Type: hseSysRights_t	
Name	**Value**
HSE_RIGHTS_SUPER_USER	1U
HSE_RIGHTS_USER	2U

Type: hseSysAuthOption_t	
Name	**Value**
HSE_SYS_AUTH_KEY_MGMT	1U << 0U
HSE_SYS_AUTH_NVM_CONFIG	1U << 1U

Typedefs

- typedef uint8_t hseSysRights_t
- typedef uint8_t hseSysAuthOption_t

Data Structure Documentation

struct hseSysAuthorizationReqSrv_t

HSE SYS Authorization Request service.

During run-time (IN_FIELD Life cycle), the User rights can be temporarily elevated to SuperUser(CUST/OEM) using HSE Authorization Request/Response.

- CUST SuperUser rights are granted using an authorization key owned by CUST.
- OEM SuperUser rights are granted using an authorization key owned by OEM.
- The User rights (non privilege rights) can be requested without authorization. In this case, HSE_SYS_Authorization_Resp shall not be used.

Note

- After reset, the default access rights are used (see hseSysRights_t).
- If no authorization key is installed during CUST_DEL or OEM_PROD life cycle, the keys can be updated only having USER rights.
- HSE FW can perform only one SYS Authorization Request at a time. A second request will overwrite the first request.
- An authorization key is a NVM key that can only be used for verify.
- If authorization succeeds, it will be opened on the MU Interface on which the request was performed, and the services that needs authorization (e.g. key import/generate/derive/export) must be performed on the same MU Interface.
- The system authorization procedure can be used to emulate the SHE CMD_DEBUG using the MASTER_ECU_KEY key (as per SHE specification). In this case, if SU access rights are requested for Key Management services (see hseSysAuthOption_t), the authorization using MASTER_ECU_KEY cannot be performed if any SHE key has the WRITE_PROTECTED flag set.
  Access rights requested only for NVM Configuration services (see hseSysAuthOption_t) are not bound to this condition. Note that SHE keys can be erased only if the authorization was performed with the MASTER_ECU_KEY (refer to hseEraseKeySrv_t).

Data Fields

Type	Name	Description
hseSysAuthOption_t	sysAuthOption	INPUT: Authorization option: Key management/NVM configuration/Both.
hseSysRights_t	sysRights	INPUT: Requested system rights: SuperUser (CUST/OEM) or User rights.

Data Fields

Type	Name	Description
uint8_t	reserved[2]	
hseKeyHandle_t	ownerKeyHandle	INPUT: The owner key handle: – if sysRights = HSE_RIGHTS_SUPER_USER, it shall be a CUST or OEM key used for only for signature verification. – if sysRights = HSE_RIGHTS_USER, the key handle is not used.
hseAuthScheme_t	authScheme	INPUT: Authentication scheme. ONLY RSA, ECDSA, EDDSA and CMAC schemes are supported. If sysRights = HSE_RIGHTS_USER, authScheme is not used. Note – EDDSA scheme with user provided context (eddsa.contextLength != 0) is NOT supported.
uint64_t	pChallenge	OUTPUT: The output challenge that needs to be signed by the HOST. In case SHE MASTER_ECU_KEY is used, the returned challenge is HSE_SYS_AUTH_CHALLENGE_LENGTH - 1 byte long and is formed from 16 random bytes concatenated with SHE UID: (RANDOM(16 bytes) || SHE_UID(15 bytes)). Otherwise, for any other key type, the challenge size is HSE_SYS_AUTH_CHALLENGE_LENGTH bytes. If sysRights = HSE_RIGHTS_USER, pChallenge is not used. Note For HSE_H/M, the UID is read by HSE from system fuses. The application must provide read-only access (xRDC restriction) to HSE to read the UID.

struct hseSysAuthorizationRespSrv_t

HSE SYS Authorization Response service.

Provides the signature for the requested challenge (using hseSysAuthorizationReqSrv_t service).

Note

- In case SHE MASTER_ECU key is used, the HSE will return the HSE_SRV_RSP_VERIFY_FAILED status as the equivalent of ERC_NO_DEBUGGING status as specified by the SHE spec (returned when the tag over the challenge is not correct).

Data Fields

Type	Name	Description
uint16_t	authLen[2]	INPUT: Byte length(s) of the authentication tag(s). Note – For RSA signature and CMAC only authLen[0] is used. – Both lengths are used for (R,S) (ECC). – The MAC tag size must be minimum 16 bytes. – RSA signature size must be HSE_BYTES_TO_BITS(keyBitLength); – R or S size for ECDSA/EDDSA signature must be HSE_BYTES_TO_BITS(keyBitLength)
uint64_t	pAuth[2]	INPUT: Address(es) to authentication tag. Note – For RSA signature and CMAC only pAuth[0] is used. – Both pointers are used for (R,S) (ECC). – If SHE MASTER_ECU_KEY is used, the CMAC must be computed over the challenge (31 bytes) using a derived key (as per SHE specification).

Macro Definition Documentation

HSE_RIGHTS_SUPER_USER

```
#define HSE_RIGHTS_SUPER_USER ((hseSysRights_t)1U)
```

SuperUser rights: can install/update CUST/OEM NVM keys or RAM keys using less restrictions. CUST/OEM SuperUser restrictions are specific to CUST_DEL/OEM_PROD Life cycle.

HSE_RIGHTS_USER

```
#define HSE_RIGHTS_USER ((hseSysRights_t)2U)
```

User rights: can install/update NVM/RAM keys using high restrictions.
User restrictions are specific to IN_FILED life cycle.

HSE_SYS_AUTH_KEY_MGMT

```
#define HSE_SYS_AUTH_KEY_MGMT ((hseSysAuthOption_t)(1U << 0U))
```

Request SuperUser rights for Key Management services (e.g. import/export/erase/key generate/key derive).
If SuperUser rights are granted, Key Management services can be performed using less restrictions.

HSE_SYS_AUTH_NVM_CONFIG

```
#define HSE_SYS_AUTH_NVM_CONFIG ((hseSysAuthOption_t)(1U << 1U))
```

Request SuperUser rights to update/install the HSE NVM tables/attributes which are stored in SYS-IMAGE(HSE_H/M)/internal flash(HSE_B) (e.g. SMR, CR, OTFAD, NVM attributes).
If SuperUser rights are granted, updates of NVM configuration will be permitted.

HSE_SYS_AUTH_ALL

```
#define HSE_SYS_AUTH_ALL (( HSE_SYS_AUTH_KEY_MGMT) | ( HSE_SYS_AUTH_NVM_CONFIG))
```

Request SuperUser rights for both Key Management services and NVM configuration updates.

HSE_SYS_AUTH_CHALLENGE_LENGTH

```
#define HSE_SYS_AUTH_CHALLENGE_LENGTH (32UL)
```

Challenge length: Length of the challenge (in bytes) returned by a successful authorization request.

Typedef Documentation

hseSysRights_t

```
typedef uint8_t hseSysRights_t
```

HSE System Access rights.

After reset (default access rights):

Life Cycle	NVM CUST keys	NVM OEM keys	RAM keys	NVM config
CUST_DEL	SU/U∗	U	SU/U∗	SU/U∗
OEM_PROD	U	SU/U∗	SU/U∗	SU/U∗
IN_FIELD	U	U	U	U

After reset, the SYS rights are synchronized with Life cycle (LC) and CUST/OEM START_AS_USER policy attributes (see CUST/OEM policy attributes).

- if LC = CUST_DEL:
  - if CUST_START_AS_USER policy = FALSE, CUST SuperUser rights are granted (CUST NVM Keys / NVM configuration updates)
  - otherwise User rights are granted (U∗ in the above table)
- if LC = OEM_DEL:
  - if OEM_START_AS_USER policy = FALSE, OEM SuperUser rights are granted (OEM NVM Keys / NVM configuration updates)
  - otherwise User rights are granted (U∗ in the above table)
- if LC = IN_FIELD, User rights are granted.

hseSysAuthOption_t

```
typedef uint8_t hseSysAuthOption_t
```

HSE System Authorization options.

Specifies the services for which the system authorization is performed.

3.4 HSE Boot Images Signature Generate/Verify

Data Structures

- struct hseAppHeader_t
- struct hseBootDataImageSignSrv_t
- struct hseBootDataImageVerifySrv_t

Data Structure Documentation

struct hseAppHeader_t

The Application Image header that keeps information about the Basic Secure Booting (BSB) (e.g. header information, source and destination addresses, app code length, tag location).

Note

> If both SMR and BSB are configured, HSE executes the secure boot using SMR only. If the SYS-IMG is not loaded because it is corrupted (the SMRs are not present), HSE executes the secure boot using BSB. In this case, the App BSB can be seen as a recovery image (to recover the SYS-IMG). Note that the App image can be booted without loading the SYS-IMG.

Data Fields

Type	Name	Description
uint8_t	hdrTag	App header tag shall be 0xD5.
uint8_t	reserved1[2]	Reserved field has no impact. Set to all zeroes.
uint8_t	hdrVersion	App header version shall be 0x60.
uint32_t	pAppDestAddres	The destination address where the application is copied. Note For HSE_B, it is NULL (the code is executed from flash)
uint32_t	pAppStartEntry	The address of the first instruction to be executed.
uint32_t	codeLength	Length of application image.
hseAppCore_t	coreId	The application core ID that is un-gated. Note Valid for HSE_B devices only. For HSE_H/M core id defined in IVT
uint8_t	reserved2[47]	Reserved field has no impact. Set to all zeroes.

struct hseBootDataImageSignSrv_t

HSE Boot Data Image GMAC generation.

This service is used to generate the GMAC tag along with the random IV (for new device revisions; see notes below) for different Boot Data images.
For HSE_H/M, the following Boot Data Images can be signed:

– IVT, DCD, SELF-TEST and Application Image (also referred below as App BSB Image).

- LPDDR4 Flash image for S32Z/E (HSE_H) devices. For HSE_B, the following Boot Data Images can be signed:
- IVT and Application Image (also referred below as App BSB Image). The computed random IV and GMAC tag must be placed/copied at the end of the image. The 12 bytes of random IV and 16 bytes of GMAC are generated by HSE Firmware. The random IV is also part of GMAC calculation (for images format, refer to HSE FW Reference Manual).

Note

SuperUser rights (for NVM Configuration) are needed to perform this service.

- For new device revisions (see table below), the service provides in the pOutTagAddr a random IV (12byte) followed by the GMAC tag (16 bytes). The IV and GMAC tag must be placed at the end of the image (with one exception for LPDDR4 FLash image; see below). Note that the GMAC tag generation is also done over the random IV. The GMAC tag and random IV offsets in the image are specified in the HSE Firmware Reference Manual.
- For older device revisions (the part revision is smaller than the revision specified in the table below), a static IV is used that is not placed in the image. For the static IV value, refer to HSE Firmware Reference Manual.
- The application can check the device revision information reading the MAJOR_MASK and MINOR_MASK fields of SIUL2_4 for S32ZE or SIUL2_0 for the others devices
- S32K3XX devices support only random IV

Device	New part revision
S32G2	rev2.1 or higher
S32G3	rev1.1 or higher
S32ZE	rev1.1 or higher
S32R45	rev2.1 or higher
S32R41	rev1.1 or higher
SAF85XX	rev2.0 or higher
S32K3XX	new rev only

Data Fields

Type	Name	Description
uint64_t	pInImage	INPUT: The address of the Boot Data Image. The Boot Data Image can be: – For HSE_H/M, IVT or DCD or SELF-TEST or App BSB or LPDDR4(for S32Z/E devices) image; the address may be a QSPI-FLASH (external flash) or system RAM address. – For HSE_B, the IVT or App BSB image; the address can be a flash or system RAM address. The length of the pInImage is not provided. HSE uses the information from the provided pInImage to compute the image length. The length of each image is computed in the below manner: 1. For HSE_H/M new device revisions: – the IVT Image length must be 256 bytes (IVT Image header (4bytes) + IVT Image data (224 bytes) + Random IV (12bytes) + GMAC(16 bytes)) – For S32Z/E devices (HSE_H), DCD/SELF-TEST Image length must be maximum 32768 bytes (DCD/ST Image header(4 bytes) + maximum DCD/ST Image data (32764 byte)) – For other devices,DCD/SELF-TEST Image length must be maximum 8192 bytes (DCD/ST Image header(4 bytes) + maximum DCD/ST Image data (8188 byte)) – For S32Z/E devices (HSE_H), the maximum length of the LPDDR4 Flash image must be smaller or equal to (7MB + 336bytes)(Image header(336 bytes) + code length(maximum 7MB)) – pInImage can point to the App BSB Image that contains the App header and App code: · App image header shall be specified as hseAppHeader_t. It has a fixed size of 64 bytes. · App image code shall follow the App image header and has a variable length specified by "codelength" parameter. · The computed GMAC tag for App BSB Image includes both App header, App code and Random IV. ...

Data Fields

Type	Name	Description
uint64_t	pInImage	For old device revision (static IV): – For IVT, the IV bytes are reserved (set to zero) – For DCD/SELF-TEST and APP BSB, the IV bytes are not included at all. – For S32Z/E devices, the image does not include any IV in the image header. – For S32Z/E devices (HSE_H), DCD/SELF-TEST Image length must be maximum 8192 bytes (DCD/ST Image header(4 bytes) + maximum DCD/ST Image data (8188 byte)) 2. For HSE_B: – The IVT image length must be 256 bytes (IVT Image header (4bytes) + IVT Image data (224 bytes) + IV (12 bytes) + GMAC(16 bytes)). The computed GMAC tag is over IVT Image header and data (228 bytes) and IV (12 bytes). – pInImage can point to the App BSB Image that contains the App header and App code: · App image header shall be specified as hseAppHeader_t. It has a fixed size of 64 bytes. · App image code shall follow the App image header and has a variable length specified by "codelength" parameter. · The computed GMAC tag for App BSB Image includes App header, App code and IV (12 bytes)
uint32_t	inTagLength	INPUT: The length in bytes of the IV + GMAC tag. This length must be equal to or greater than. 1. For HSE_H/M: – new device revisions (random IV): 28 bytes – old device revisions (static IV): 16 bytes 2. For HSE_B, 28 bytes

Data Fields

Type	Name	Description
uint64_t	pOutTagAddr	OUTPUT: 1. For HSE_H/M, the output address: – new device revisions: random IV (12 bytes) followed by the GMAC tag (16 bytes) – old device revisions: GMAC tag (16 bytes) For HSE_B: The address where the random IV (12 bytes), followed by the GMAC tag (16 bytes) are generated. It must be a system RAM address. Note The computed output data shall be copied at the end of boot data image. Exception: For S32Z/E devices (HSE_H), computed random IV shall be copied to LPDDR4 image header (at 0x144 offset) and the computed GMAC tag shall be copied at the end of boot data image.

struct hseBootDataImageVerifySrv_t

HSE Boot Data Image GMAC verification.

This service can be used to verify the GMAC tag generated using the hseBootDataImageSignSrv_t service.

Data Fields

Type	Name	Description
uint64_t	pInImage	INPUT: The address of the HSE Boot Data Image (for more details about the HSE Boot Data Images refer to pInImage parameter from hseBootDataImageSignSrv_t service). Note – HSE uses the Boot Data Image information (provided by pInImage) to compute the length of the image and to verify the authentication TAG.

3.5 HSE Self Test Service

Data Structures

- struct hseSelfTestSrv_t

Macros

Type: hseSelfTestMask_t	
Name	**Value**
HSE_ST_FW_INTEGRITY	1ULL << 0U
HSE_ST_RNG_ENGINE	1ULL << 1U
HSE_ST_AES_ENGINE	1ULL << 2U
HSE_ST_HASH_ENGINE	1ULL << 3U
HSE_ST_CRC_ENGINE	1ULL << 4U
HSE_ST_RSA_ENGINE	1ULL << 5U
HSE_ST_ECC_ENGINE	1ULL << 6U
HSE_ST_HMAC_ENGINE	1ULL << 7U

Typedefs

- typedef uint64_t hseSelfTestMask_t

Data Structure Documentation

struct hseSelfTestSrv_t

Self Test service.

Performs a self-test on a specific security block or a full self-test.

Note

- During the self-test operation, the HSE firmware cannot be interrupted by another request (until the operation is completed).
– The requested self-tests must be supported; otherwise, the self-test service returns the HSE_SRV_RSP_NOT_ALLOWED status (no requested self-tests will be executed).
– If one of the check fails, the HSE firmware returns HSE_SRV_RSP_GENERAL_ERROR and goes to shutdown (a fatal error occurred). In this case, the application must perform a system reset.
– The HSE_ST_FW_INTEGRITY flag checks the integrity of HSE FW and runtime SYS-IMG (if present) inside HSE.
– At first request for HSE_ST_FW_INTEGRITY the hash over HSE FW will be computed, verification being done in the subsequent requests.

Data Fields

Type	Name	Description
hseSelfTestMask_t	selfTest	INPUT: Select bits to run a specific self-test. (note that the selected bits should map the supported self-tests). All bits zero means that a full self-test will be performed.
uint64_t	pTestResultsBitMask	OUTPUT: The address where the self-tests results bit mask is returned (points to a hseSelfTestMask_t type). If one of the requested self-tests failed, HSE returns HSE_SRV_RSP_GENERAL_ERROR and the corresponding bit for the failing test is set to one. If all the self-tests passed, HSE returns HSE_SRV_RSP_OK and this field can be ignored.

Macro Definition Documentation

HSE_ST_FW_INTEGRITY

```
#define HSE_ST_FW_INTEGRITY ((hseSelfTestMask_t)(1ULL << 0U))
```

Verify HSE Firmware Integrity.

HSE_ST_RNG_ENGINE

```
#define HSE_ST_RNG_ENGINE ((hseSelfTestMask_t)(1ULL << 1U))
```

Random Number Generator.

HSE_ST_AES_ENGINE

```
#define HSE_ST_AES_ENGINE ((hseSelfTestMask_t)(1ULL << 2U))
```

Advanced Encryption Standard Hardware Accelerator.

HSE_ST_HASH_ENGINE

```
#define HSE_ST_HASH_ENGINE ((hseSelfTestMask_t)(1ULL << 3U))
```

Hash Generator.

HSE_ST_CRC_ENGINE

```
#define HSE_ST_CRC_ENGINE ((hseSelfTestMask_t)(1ULL << 4U))
```

Cyclic-Redundancy Check Hardware Accelerator.

HSE_ST_RSA_ENGINE

```
#define HSE_ST_RSA_ENGINE ((hseSelfTestMask_t)(1ULL << 5U))
```

Rivest-Shamir-Adleman Hardware Accelerator.

HSE_ST_ECC_ENGINE

```
#define HSE_ST_ECC_ENGINE ((hseSelfTestMask_t)(1ULL << 6U))
```

Elliptic Curve Cryptography Hardware Accelerator.

HSE_ST_HMAC_ENGINE

```
#define HSE_ST_HMAC_ENGINE ((hseSelfTestMask_t)(1ULL << 7U))
```

Hash-Based Message Authentication Code Hardware Accelerator.

Typedef Documentation

hseSelfTestMask_t

```
typedef uint64_t hseSelfTestMask_t
```

Self-test capabilities mask.

<

3.6 HSE Firmware Update Service

Data Structures

- struct hseFirmwareUpdateSrv_t
- struct hseFirmwareVerifySrv_t

Data Structure Documentation

struct hseFirmwareUpdateSrv_t

HSE_H/M Firmware Update Service.

This service is used to re-encrypt the current running HSE FW image or the HSE firmware delivered by NXP (pink image) with a device-specific key.
The re-encrypted image (blue image) is published back on system RAM. The re-encryption operation can be performed in place by overwriting the pink image (the application can use the same pink image buffer for the output).

Note

- The HSE firmware boot can be protected against rollback attack only if it boots a blue image. This protection at boot does not exist if IVT is pointing to a pink image. It is possible to enforce a boot to blue image via HSE system attributes (refer to hseOtpRollbackProtectionPolicy_t attribute). As long as the HSE SYS-IMG is available, it can prevent the HSE executing a pink image, hence bypassing the rollback protection. The HSE always ensure that the rollback counter value in the blue image is above or equal to the rollback counter infuse
- If the OTP rollback protection is not disabled (refer to hseOtpRollbackProtectionPolicy_t attribute), to be able to update fuse counter, the VDD_EFUSE supply must be powered at start-up before fuses are written (refer to HSE FW Reference Manual). The anti-rollback counter is incremented in fuses at start-up. After writing the updated current/blue FW image in the external flash, a reset is needed. The VDD_EFUSE state is checked before the fuse write by reading the NCSPD_STAT register of the on-chip PMC module. The application shall provide read-only access (xRDC restriction) to HSE to read the NCSPD_STAT register. The rollback counter is NOT updated in fuses when LC == CUST_DEL and BOOT_SEQ == 0. If the rollback counter is saturated, the HSE firmware can still be updated, but without rollback protection.

Data Fields

Type	Name	Description
uint32_t	inFwFileLength	INPUT: The length in bytes of the new NXP Firmware file. It represents the length of new NXP Firmware file to be re-encrypted with a device-specific key. – If "#inFwFileLength == 0", then the pInFwFile parameter is ignored and an encrypted version of the currently running HSE FW image will be generated with a device-specific key (generate the blue firmware image of the currently running HSE FW image). – If "#inFwFileLength != 0", then inFwFileLength must be equal with the new NXP firmware image (pink image) size.

Data Fields

Type	Name	Description
uint64_t	pInFwFile	INPUT: The address of new version of HSE Firmware file to be re-encrypted with a device specific key (inFwFileLength != 0).
uint64_t	pFwBufferLength	INPUT: The address where the length (an uint32_t value) of the buffer will be provided. – If "#inFwFileLength == 0", then the buffer length must be equal to at least the size obtained by getting the attribute HSE_FW_SIZE_ATTR_ID . – If "#inFwFileLength != 0", then the buffer length must be equal to or greater than inFwFileLength. – If the size of the buffer is less than the expected size of HSE_H/M FW file an error will be returned. OUTPUT: The HSE FW will return the total length of the image which have been published.
uint64_t	pOutFwBuffer	INPUT: It is the address of the buffer where the encrypted version of HSE_H/M FW file (with a device specific key) will be stored.

struct hseFirmwareVerifySrv_t

HSE_H/M Firmware Verify Service.

This service can be used to verify the pink or blue FW image (in SRAM or external flash)

Data Fields

Type	Name	Description
uint64_t	pInFwFile	INPUT: The address of HSE Firmware file.

3.7 HSE Publish SYS-IMG Service

Data Structures

- struct hsePublishSysImageSrv_t
- struct hseGetSysImageSizeSrv_t
- struct hseVerifySysImageSrv_t

Macros

Type: hsePublishOptions_t	
Name	**Value**
HSE_PUBLISH_RESERVED	1U << 0U
HSE_PUBLISH_ALL_DATA_SETS	1U << 1U

Typedefs

- typedef uint8_t hsePublishOptions_t

Data Structure Documentation

struct hsePublishSysImageSrv_t

HSE Publish SYS-IMAGE (only for HSE_H/M devices).

Publish the SYS-IMAGE to be stored on an external RAM memory (controlled by application). The host application uses this service to request the SYS-IMAGE. The SYS-IMAGE is built from three Data Sets:

- a Main Header (one flash page is allocated)
- SMR/CR/OTFAD/NVM/IEE_DDR(if supported) attributes Data set; max size is max(8KB, flashPageSize).
- NVM Key Store Data Set; max size is 32KB.
  The last two data sets are protected against reply attacks using a version counter stored in fuses. The SYS-IMAGE size depends on the flash page size configured in the IVT (if set zero in IVT, HSE used 4KB as the default flash page size); it can be calculated as "flashPageSize + max(8KB, flashPageSize) + 32KB" (e.g for 4KB flash sector size, the SYS-IMAGE size is 44KB). The application can request to publish only all data sets at the same time.

Note

- The rollback protection for SYS-IMG can be disabled using HSE_OTP_ROLLBACK_PROTECTION_POLICY_ATTR_ID attribute
- If the OTP rollback protection is not disabled (refer to HSE_OTP_ROLLBACK_PROTECTION_POLICY_ATTR_ID attribute), to be able to update fuse counter, the VDD_EFUSE supply must be powered before fuses are written (refer to HSE FW Reference Manual).
- If the OTP rollback protection is not disabled
  - if the host requests to publish the Data Sets, and none of the Data Sets wasn't updated, the anti-rollback counter will not be updated in fuses. If one of the Data Sets is updated, the counter is incremented (fuse counter+1) in image header. The counter will be updated also in fuses (when sys-image is loaded during start-up).
  - During a power cycle, the value of the anti-rollback counter is incremented with a maximum value of 1. (multiple update-publish requests in one power cycle will not burn more than one fuse). The VDD_EFUSE state is checked before the fuse write by reading the NCSPD_STAT

register of the on-chip PMC module. The application shall provide read-only access (xRDC restriction) to HSE to read the NCSPD_STAT register.
- For S32ZE, the SYS-IMG size is 48KB.
- Depending on the size of the DataSet, not the entire flash page(s) is used. An empty space is reserved at the end of the DataSet (end of the last flash page of the DataSet) for further extention of the DataSet. These reserved empty spaces are not authenticated during SYS-IMG loading (e.g only relevant data is authenticated and encrypted).

Data Fields

Type	Name	Description
hsePublishOptions_t	publishOptions	INPUT: Publish SYS-IMAGE options: – HSE_PUBLISH_RESERVED - unsupported publish method. – HSE_PUBLISH_ALL_DATA_SETS - publish all Data Sets.
uint8_t	reserved[3]	
uint64_t	pPublishOffset	OUTPUT: The address where to store the Data Set offset (a uint32_t value). This offset specifies where the provided output buffer needs to be stored in the external flash SYS-IMAGE (e.g. the buffer of size pBuffLength shall be copied in the external flash starting from address specified by "SYS_IMAGE_BASE_ADDR + PublishOffset"). Since the only supported publish method is for all data sets at the same time, the offset is always set to zero.
uint64_t	pBuffLength	INPUT/OUTPUT: As input, it specifies the length (a uint32_t value) of the output buffer provided by the application. This needs to be at least greater or equal to the size returned by get the SYS_IMG size request (see hseGetSysImageSizeSrv_t). The uint32_t value pointed by pBuffLength will be overwritten by HSE with the number of bytes that were written into the pBuff buffer.
uint64_t	pBuff	OUTPUT: The address of the output buffer.

struct hseGetSysImageSizeSrv_t

HSE Get SYS_IMAGE size (only for HSE_H/M devices).

Return the total length of SYS_IMAGE in bytes.

Data Fields

Type	Name	Description
uint64_t	pSysImageSize	OUTPUT: The address where to store the size of the SYS_IMAGE (a uint32_t value).

struct hseVerifySysImageSrv_t

HSE Verify SYS_IMAGE (only for HSE_H/M devices).

This service can be used to verify the SYS_IMAGE integrity after it is written in external flash.

Data Fields

Type	Name	Description
uint64_t	pSysImageAddr	INPUT: The address where SYS_IMAGE is stored. Must point to the start of the entire SYS_IMAGE. Cannot be in SD/eMMC external flash.

Macro Definition Documentation

HSE_PUBLISH_RESERVED

```
#define HSE_PUBLISH_RESERVED ((hsePublishOptions_t)1U << 0U)
```

Unsupported publish method.

HSE_PUBLISH_ALL_DATA_SETS

```
#define HSE_PUBLISH_ALL_DATA_SETS ((hsePublishOptions_t)1U << 1U)
```

Publish all data sets.

Typedef Documentation

hsePublishOptions_t

```
typedef uint8_t hsePublishOptions_t
```

Publish SYS-IMAGE options.

3.8 HSE On-The-Fly AES Decryption (OTFAD) Services

Data Structures

- struct hseOtfadContext_t
- struct hseInstallOtfadContextSrv_t
- struct hseActivateOtfadContextSrv_t
- struct hseGetOtfadContextSrv_t

Macros

Type: (implicit C type)	
Name	**Value**
HSE_OTFAD_INSTANCE_0	0U

Type: hseOtfadActivateFlag_t	
Name	**Value**
HSE_OTFAD_CTX_ACTIVE_ON_BOOT	0xAB65U
HSE_OTFAD_CTX_INACTIVE_ON_BOOT	0x375AU

Typedefs

- typedef uint8_t hseOtfadInstance_t
- typedef uint16_t hseOtfadActivateFlag_t

Data Structure Documentation

struct hseOtfadContext_t

Define the parameters of OTFAD/IEE_DDR context entry.

Note

- The OTFAD/IEE_DDR configuration is part of SYS-IMG. Once configured, the host must publish and save the SYS_IMG to Flash.
- The OTFAD/IEE_DDR cannot be used for IVT, DCD/ST-DCD, FW_IMG/FW_IMG∗ and SYS-IMG images: these images must be stored unencrypted in external Flash.

Data Fields

Type	Name	Description
hseKeyHandle_t	keyHandle	The key handle of the OTFAD key (AES 128bit), the OTFAD key handle must always and only have HSE_KF_USAGE_OTFAD_DECRYPT flag set.
uint8_t	iv[4]	– OTFAD Instance[0, 1]: Byte array defining the user's part of the initial vector (counter) used by the AES-CTR mode algorithm. To avoid possible attack scenarios, diversification of the IV for each updated version of application code is highly recommended. – IEE_DDR Instance[2]: Fixed upper 32-bits of the Counter value when using PRINCE-CTR mode. Counter mode uses a 64-bit counter value. It is composed of two halves, an upper fixed value stored as a 32-bit integer, and a lower half formed from the ((address - startAddress) >> 3) (e.g offset in the defined memory region) at which Ciphertext data is stored in memory. This counter value increments for each 8-byte block of encrypted data. Note The lower half of the counter always starts at 0 for a block of encrypted data that starts from startAddress.
uint32_t	startAddress	– OTFAD Instance[0, 1]: The start address of the memory region. Must be aligned on a 1KB boundary. – IEE_DDR Instance[2]: The start address of the memory region. Must be aligned on a 1MB boundary.
uint32_t	endAddress	– OTFAD Instance[0, 1]: The end address of the memory region. Must be aligned on a 1KB boundary. – IEE_DDR Instance[2]: The end address of the memory region. Must be aligned on a 1MB boundary.

Data Fields

Type	Name	Description
hseSmrFlags_t	smrFlags	– When BOOT_SEQ == 1 (Secure boot), it specifies the SMR entries (bit field) that should be verified before the activation of the OTFAD/IEE_DDR entry. – When BOOT_SEQ == 0 (Un-secure boot), if there is any SMR linked with OTFAD/IEE_DDR entry, the application should trigger the verification at run-time (activate the OTFAD/IEE_DDR context using the service structure hseActivateOtfadContextSrv_t); in this case, the SMR must NOT be in the QSPI flash region configured using OTFAD/IEE_DDR.
hseOtfadActivateFlag_t	activateOnBoot	If activateOnBoot == HSE_OTFAD_CTX_ACTIVE_ON_BOOT , the configured OTFAD/IEE_DDR context will automatically activate while booting. otherwise, the hseOtfadActivateContextSrv_t service must be called to activate the OTFAD/IEE_DDR context.
uint8_t	reserved[2]	

struct hseInstallOtfadContextSrv_t

HSE OTFAD/IEE_DDR Install Context service (update or add new entry).

This service installs an existing OTFAD/IEE_DDR context or add a new one.

Note

SuperUser rights (for NVM Configuration) are needed to perform this service.

Data Fields

Type	Name	Description
uint8_t	otfadIdx	INPUT: Identifies the index of OTFAD configuration table which has to be installed/updated. I can be defined up to HSE_NUM_OF_OTFAD_ENTRIES contexts (per OTFAD instance) and up to #HSE_NUM_OF_IEE_DDR_ENTRIES for IEE_DDR instance (if supported)

Data Fields

Type	Name	Description
hseOtfadInstance_t	otfadInstance	INPUT: Identifies the OTFAD or IEE_DDR instance (refer to hseOtfadInstance_t). it shall be between 0 and HSE_NUM_OF_OTFAD_INSTANCES . Note – S32ZE devices have more than one OTFAD instances. – If IEE_DDR for flash decrypt is supported (see #HSE_SPT_IEE_DDR_FLASH), the last instance (see #HSE_IEE_DDR_INSTANCE_2) is used to configure the IEE DDR. This instance contains #HSE_NUM_OF_IEE_DDR_ENTRIES contexts.
uint8_t	reserved[2]	
uint64_t	pOtfadCtxEntry	INPUT: Address to hseOtfadContext_t that contains the configuration properties of OTFAD/IEE_DDR context.

struct hseActivateOtfadContextSrv_t

HSE Activate Otfad Context service.

This service is used to configure the hardware using an already installed OTFAD/IEE_DDR entry. The SMR flag used in the OTFAD/IEE_DDR entry must be verified before calling this service.

Note

- For S32ZE, IEE regions can be activated only once, per run-time session after installation. If an IEE region is updated during run-time, the activation of the updated IEE entry will be possible, only after the next reset.

Data Fields

Type	Name	Description
uint8_t	otfadIdx	INPUT: Identifies the entry in the OTFAD/IEE_DDR configuration table. Note – The OTFAD instance contains HSE_NUM_OF_OTFAD_ENTRIES (otfadIdx < HSE_NUM_OF_OTFAD_ENTRIES) – The IEE_DDR instance contains #HSE_NUM_OF_IEE_DDR_ENTRIES (otfadIdx < HSE_NUM_OF_IEE_DDR_ENTRIES)
hseOtfadInstance_t	otfadInstance	INPUT: Identifies the OTFAD or IEE_DDR instance (refer to hseOtfadInstance_t). it shall be between 0 and HSE_NUM_OF_OTFAD_INSTANCES . Note – S32ZE devices have more than one OTFAD instances. – if IEE_DDR for flash decrypt is supported (see #HSE_SPT_IEE_DDR_FLASH), the last instance (see #HSE_IEE_DDR_INSTANCE_2) is used to configure the IEE_DDR. .
uint8_t	reserved[2]	

struct hseGetOtfadContextSrv_t

HSE Get OTFAD/IEE_DDR Context Info service.

This service is used to extract the context parameters previously set in the OTFAD/IEE_DDR table.

Data Fields

Type	Name	Description
uint8_t	otfadIdx	INPUT: Identifies the entry in the OTFAD/IEE_DDR configuration table whose parameters need to be extracted. Note – The OTFAD instance contains HSE_NUM_OF_OTFAD_ENTRIES (otfadIdx < HSE_NUM_OF_OTFAD_ENTRIES) – The IEE_DDR instance contains #HSE_NUM_OF_IEE_DDR_ENTRIES (otfadIdx < HSE_NUM_OF_IEE_DDR_ENTRIES)
hseOtfadInstance_t	otfadInstance	INPUT: Identifies the OTFAD or IEE_DDR instance (refer to hseOtfadInstance_t). it shall be between 0 and HSE_NUM_OF_OTFAD_INSTANCES . Note – S32ZE devices have more than one OTFAD instances. – if IEE DDR for flash decrypt is supported (see #HSE_SPT_IEE_DDR_FLASH), the last instance (see #HSE_IEE_DDR_INSTANCE_2) is used to configure the IEE DDR.
uint8_t	reserved[2]	
uint64_t	pOtfadContext	OUTPUT: Address where the configuration parameters of the selected OTFAD context need to be stored. It's up to the user to allocate memory of sizeof(hseOtfadContext_t) in the application memory space.

Macro Definition Documentation

HSE_OTFAD_INSTANCE_0

```
#define HSE_OTFAD_INSTANCE_0 (0U)
```

OTFAD Instance 0.

HSE_OTFAD_CTX_ACTIVE_ON_BOOT

```
#define HSE_OTFAD_CTX_ACTIVE_ON_BOOT ((hseOtfadActivateFlag_t)0xAB65U)
```

Activate context on boot.

HSE_OTFAD_CTX_INACTIVE_ON_BOOT

```
#define HSE_OTFAD_CTX_INACTIVE_ON_BOOT ((hseOtfadActivateFlag_t)0x375AU)
```

Inactive context on boot.

Typedef Documentation

hseOtfadInstance_t

```
typedef uint8_t hseOtfadInstance_t
```

OTFAD instance type.

hseOtfadActivateFlag_t

```
typedef uint16_t hseOtfadActivateFlag_t
```

OTFAD/IEE_DDR context activation flag.

4 Cryptographic Services

4.1 HSE MAC Service

Data Structures

- struct hseMacSrv_t
- struct hseFastCMACSrv_t

Data Structure Documentation

struct hseMacSrv_t

MAC service.

MAC algorithms are symmetric key cryptographic techniques to provide message authentication codes (MACs), also known as tags. These can be used to verify both the integrity and authenticity of a message.

This service can be accessible in one-pass or streaming (SUF) mode. In case of streaming mode, three steps (calls) will be used: START, UPDATE, FINISH. START and FINISH are mandatory; UPDATE is optional. Not all fields are used by each access mode.

The table below summarizes which fields are used by each access mode. Unused fields are ignored by the HSE.

Field \ Mode	One-pass	Start	Update	Finish
accessMode	*	*	*	*
streamId	*	*	*	*
authDir	*	*		
sgtOption	*	*	*	*
macScheme	*	*		
keyHandle	*	*		
inputLength	*	*	*	*
pInput	*	*	*	*
pTagLength	*			*
pTag	*			*

Data Fields

Type	Name	Description
hseAccessMode_t	accessMode	INPUT: Specifies the access mode: ONE-PASS, START, UPDATE, FINISH. STREAMING USAGE: Used in all steps.

Data Fields

Type	Name	Description
hseStreamId_t	streamId	INPUT: Specifies the stream to use for START, UPDATE, FINISH access modes. Each interface supports a limited number of streams per interface, up to HSE_STREAM_COUNT . STREAMING USAGE: Used in all steps.
hseAuthDir_t	authDir	INPUT: Specifies the direction: generate/verify. STREAMING USAGE: Used in START.
hseSGTOption_t	sgtOption	INPUT: Specify if pInput is provided as hseScatterList_t list (the host address points to a hseScatterList_t list). Ignored if SGT is not supported. Note – ONLY HSE_SGT_OPTION_INPUT can be used. – For HSE_B devices: · The SGT for the HMAC scheme is not available for the following hash algorithms: · SHA2_384/512 (not available in HW) · If the HMAC is requested with above hash algorithms, the HSE_SRV_RSP_NOT_SUPPORTED error is returned. – If scatter option is selected (set), the length (e.g. inputLength) shall specified the entire message length (sum of all hseScatterList_t lengths). – The number for SGT entries shall be less then HSE_MAX_NUM_OF_SGT_ENTRIES . STREAMING USAGE: Used in all steps.
hseMacScheme_t	macScheme	INPUT: Specifies the MAC scheme. STREAMING USAGE: Used in START.
hseKeyHandle_t	keyHandle	INPUT: The key to be used for the operation. STREAMING USAGE: Used in START.

Data Fields

Type	Name	Description
uint32_t	inputLength	INPUT: Length of the input message. Can be zero. STREAMING USAGE: Used in all steps. – START: Must be a multiple of block length (for HMAC-hash or AES), or zero. Cannot be zero for HMAC. – UPDATE: Must be a multiple of block length (for HMAC-hash or AES). Cannot be zero. Refrain from issuing the service request, instead of passing zero. – FINISH: Can be any value (For CMAC & XCBC-MAC, zero length is invalid). Algorithm block lengths (for STREAMING USAGE): – CMAC, GMAC, XCBC-MAC: 16 – HMAC, depends on underlying hash: · SHA1, SHA2_224, SHA2_256: 64 · SHA2_512_224, SHA2_512_256, SHA2_384, SHA2_512: 128 · SHA3: not supported for HMAC · Miyaguchi-Preneel: not supported for HMAC
uint64_t	pInput	INPUT: The input message. Note The input message for GMAC is the AAD (as specified by AEAD-GCM). STREAMING USAGE: Used in all steps, but ignored when inputLength is zero

Data Fields

Type	Name	Description
uint64_t	pTagLength	INPUT/OUTPUT: Holds the address to a memory location (an uint32_t variable) in which the tag length in bytes is stored. – GENERATE: · On calling service (input), this parameter shall contain the size of the buffer provided by pTag. · For GMAC, valid tag lengths are 8, 12, 13, 14, 15 and 16. Tag-lengths greater than 16 will be truncated to 16. · For HMAC, valid tag lengths are [8, hash-length]. Tag-lengths greater than hash-length will be truncated to hash-length. · For CMAC & XCBC-MAC, valid tag lengths are [8, cipher-block-length]. Tag-lengths greater than cipher-block-length will be truncated to cipher-block-length. · When the request has finished (output), the actual length of the returned value shall be stored. – VERIFY: · On calling service (input), this parameter shall contain the tag-length to be verified. · For GMAC, valid tag lengths are 8, 12, 13, 14, 15 and 16. · For HMAC, valid tag lengths are [8, hash-length]. · For CMAC & XCBC-MAC, valid tag lengths are [8, cipher block-length]. STREAMING USAGE: Used in FINISH.
uint64_t	pTag	OUTPUT/INPUT: The output tag for "generate"; the input tag for "verify". STREAMING USAGE: Used in FINISH.

struct hseFastCMACSrv_t

Fast CMAC service.

CMAC algorithms are symmetric key cryptographic techniques to provide message authentication codes (MACs), also known as tags. These can be used to verify both the integrity and authenticity of a message.

This FAST CMAC version can provide improved performance for CAN frames and compared to the other MAC implementation is using bits representation for pInput and pTag.

Note

Bits are represented from left to right at byte level.

Data Fields

Type	Name	Description
hseKeyHandle_t	keyHandle	INPUT: The key to be used for the operation.
uint64_t	pInput	INPUT: The input message.
uint32_t	inputBitLength	INPUT: Length of the input message.(in bits)
hseAuthDir_t	authDir	INPUT: Specifies the direction: generate/verify.
uint8_t	tagBitLength	INPUT/OUTPUT: Holds tag length in bits. – GENERATE: · On calling service (input), this parameter shall contain the size of the buffer provided by pTag. · Recommended tag lengths are [32, 128]. Tag-lengths greater than 128 will be truncated to 128. – VERIFY: · On calling service (input), this parameter shall contain the tag-length to be verified. · Recommended tag lengths are [32, 128]. · The HSE_FAST_CMAC_MIN_TAG_BIT_LEN_ATTR_ID attribute can be used to overwrite the lower recommended tag bit length limit (minimum is 1).
uint8_t	reserved[2]	
uint64_t	pTag	OUTPUT/INPUT: The output tag for "generate"; the input tag for "verify".

4.2 HSE Symmetric Cipher Service

Data Structures

- struct hseSymCipherSrv_t
- struct hseXtsAesCipherSrv_t

Data Structure Documentation

struct hseSymCipherSrv_t

Symmetric Cipher service.

To perform encryption/decryption with a block cipher in ECB or CBC mode, the length of the input must be an exact multiple of the block size. For all AES variants it is 16 bytes (128 bits). If the input plaintext is not an exact multiple of block size, it must be padded by application (by adding a padding string). For other modes, such as counter mode (CTR) or OFB or CFB, padding is not required. In these cases, the ciphertext is always the same length as the plaintext. If the plaintext is always an exact multiple of the block length, padding can be avoided.

This service can be accessible in one-pass or streaming (SUF) mode. In case of streaming mode, three steps (calls) will be used: START, UPDATE, FINISH. START and FINISH are mandatory; UPDATE is optional. Not all fields are used by each access mode.

The table below summarizes which fields are used by each access mode. Unused fields are ignored by the HSE.

Field \ Mode	One-pass	Start	Update	Finish
accessMode	*	*	*	*
streamId		*	*	*
cipherAlgo	*	*		
cipherBlockMode	*	*		
cipherDir	*	*		
sgtOption	*	*	*	*
keyHandle	*	*		
pIV	*	*		
inputLength	*	*	*	*
pInput	*	*	*	*
pOutput	*	*	*	*

Data Fields

Type	Name	Description
hseAccessMode_t	accessMode	INPUT: Specifies the access mode: ONE-PASS, START, UPDATE, FINISH. STREAMING USAGE: Used in all steps.
hseStreamId_t	streamId	INPUT: Specifies the stream to use for START, UPDATE, FINISH access modes. Each interface supports a limited number of streams per interface, up to HSE_STREAM_COUNT . STREAMING USAGE: Used in all steps.
hseCipherAlgo_t	cipherAlgo	INPUT: Specifies the cipher algorithm . STREAMING USAGE: Used in START.
hseCipherBlockMode_t	cipherBlockMode	INPUT: Specifies the cipher mode. STREAMING USAGE: Used in START.
hseCipherDir_t	cipherDir	INPUT: Specifies the cipher direction: encryption/decryption. STREAMING USAGE: Used in START.

Data Fields

Type	Name	Description
hseSGTOption_t	sgtOption	INPUT: Specify if pInput/pOutput are provided as hseScatterList_t list (the host address points to a hseScatterList_t list). Ignored if SGT is not supported. Note – If scatter option is selected (set), the length (e.g. inputLength) shall specified the entire message length (sum of all hseScatterList_t lengths). – The number for SGT entries shall be less then HSE_MAX_NUM_OF_SGT_ENTRIES . STREAMING USAGE: Used in all steps.
uint8_t	reserved[2]	
hseKeyHandle_t	keyHandle	INPUT: The key to be used for the operation. STREAMING USAGE: Used in START step.
uint64_t	pIV	INPUT: Initialization Vector/Nonce. Ignored for NULL & ECB cipher block modes. IV length is 16 bytes. (AES cipher block size). STREAMING USAGE: Used in START.
uint32_t	inputLength	INPUT: The plaintext and ciphertext length. For ECB, CBC & CFB cipher block modes, must be a multiple of block length. Cannot be zero. STREAMING USAGE: MANDATORY for all steps. – START: Must be a multiple of block length. Can be zero. – UPDATE: Must be a multiple of block length. Cannot be zero. Refrain from issuing the service request, instead of passing zero. – FINISH: For ECB, CBC & CFB cipher block modes, must be a multiple of block length. Cannot be zero. For remaining cipher block modes, can be any value except zero. AES block lengths: 16
uint64_t	pInput	INPUT: The plaintext for encryption or the ciphertext for decryption. STREAMING USAGE: Used in START, UPDATE and FINISH. Ignored in START if inputLength is zero.
uint64_t	pOutput	OUTPUT: The plaintext for decryption or ciphertext for encryption. STREAMING USAGE: Used in START, UPDATE and FINISH. Ignored in START if inputLength is zero.

struct hseXtsAesCipherSrv_t

XTS AES Cipher service.

To perform XTS AES encryption/decryption.

Note

ONLY AES128 and AES256 keys shall be used.

Data Fields

Type	Name	Description
hseCipherDir_t	cipherDir	INPUT: Specifies the cipher direction: encryption/decryption.
uint8_t	reserved0[3]	
hseKeyHandle_t	cipherKeyHandle	INPUT: The key to be used for the operation. Note The cipher key must have the encryption or decryption usage flags set, depending on the desired operations.
hseKeyHandle_t	tweakKeyHandle	INPUT: The XTS Tweak key. Note The XTS Tweak key must always and only have the HSE_KF_USAGE_XTS_TWEAK flag set, as it is used internally to encrypt the tweak value during both the encryption and the decryption operations.
uint64_t	sectorNumber	INPUT: The sector number.
uint16_t	sectorSize	INPUT: Sector size. Must be a multiple of 16 bytes.
hseSGTOption_t	sgtOption	INPUT: Specify if pInput/pOutput are provided as hseScatterList_t list (the host address points to a hseScatterList_t list). Note – If scatter option is selected (set), the length (e.g. inputLength) shall specified the entire message length (sum of all hseScatterList_t lengths). – The number for SGT entries shall be less than HSE_MAX_NUM_OF_SGT_ENTRIES .

Data Fields

Type	Name	Description
uint8_t	reserved1	
uint32_t	inputLength	INPUT: The plaintext and ciphertext length. Must be above or equal to 16.
uint64_t	pInput	INPUT: The plaintext for encryption or the ciphertext for decryption.
uint64_t	pOutput	OUTPUT: The plaintext for decryption or ciphertext for encryption.

4.3 HSE CMAC With Counter Service

Data Structures

- struct hseCmacWithCounterSrv_t

Note

For HSE_B (devices with internal flash), the first service request after reset that depends on Monotonic Counters, i.e., hseConfigSecCounterSrv_t, hseReadCounterSrv_t, hseIncrementCounterSrv_t and hseCmacWithCounterSrv_t will take more time because of Monotonic Counters Initialization. Further requests will take usual time.

Data Structure Documentation

struct hseCmacWithCounterSrv_t

CMAC With Counter service.

This service calculates/verifies the CMAC of a given input message concatenated with a selected secure counter.

Note

- The secure counter must be configured before (refer to hseConfigSecCounterSrv_t)
- Bits are represented from left to right at byte level.
- In the description below, the following notation is used:
  - SC - 64bit secure counter
  - RP - The Rollover Protection bits of the secure counter (refer to hseConfigSecCounterSrv_t)
  - VC - The Volatile Counter bits of the secure counter (refer to hseConfigSecCounterSrv_t)
  - SC_counterIdx is the secure counter identified by the counterIdx (counter index)
  - VC_counterIdx is the volatile part of the secure counter (volatile counter) identified by the counterIdx

- RP_counterIdx is the Rollover Protection value of the secure counter identified by the counterIdx (the volatile counter bits are all zeros)
- "||" means concatenation
- VCI is the Volatile Counter provide as input parameter by the service (pVolatileCounter parameter)
- RPO is the Rollover Protection Offset (RPOffset parameter for CMAC verify) added to Rollover Protection value to adjust the RP bits.
- ISC - the implied value of the SC computed by HSE concatenating the optionally adjusted RP bits with the VCI bits (refer to CMAC verify sequence below)

For CMAC generate, the HSE firmware performs the following sequence:

```
SC_counterIdx = SC_counterIdx + 1
TAG = CMAC_GENERATE(KeyHandle, input || SC_counterIdx)
VC_counterIdx = SC_counterIdx - RP_counterIdx
if(VC_counterIdx == 0) then update RP_counterIdx in NVM
return  TAG, VC_counterIdx & RSP_STATUS_OK
```

For CMAC verify, the HSE firmware performs the following sequence:

```
if(VCI > VC_counterIdx) then ISC = (RP_counterIdx + RPO) || VCI
if(VCI <= VC_counterIdx) then ISC = (RP_counterIdx + 1 + RPO) || VCI
if(CMAC_VERIFY(KEY_HANDLE, input || ISC)) then
{
     SC_counterIdx = ISC
     if((RPO != 0) or (VCI <= VC_counterIdx)) then update RP_counterIdx in NVM
     rsp_status =  HSE_ SRV_ RSP_ OK
}
else
{
     rsp_status =   HSE_ SRV_ RSP_ VERIFY_ FAILED
}
return rsp_status
```

Data Fields

Type	Name	Description
hseAuthDir_t	authDir	INPUT: Specifies the direction: generate/verify.
uint8_t	reserved1[3U]	
hseKeyHandle_t	keyHandle	INPUT: The key to be used for the operation.
uint32_t	counterIdx	INPUT: The counter Index of the secure counter.
uint8_t	RPOffset	INPUT: The Rollover protection offset used to adjust the Rollover protection bits of the secure counter in the CMAC verify operation. It is ignored for CMAC generate. If the CMAC verification fails, the application can try with a different RPOffset.

Data Fields

Type	Name	Description
hseSGTOption_t	sgtOption	INPUT: Specify if pInput is provided as hseScatterList_t list (the host address points to a hseScatterList_t list). Ignored if SGT is not supported. Note – ONLY HSE_SGT_OPTION_INPUT can be used. – If scatter option is selected (set), the length (e.g. inputBitLength) shall specified the entire message length (sum of all hseScatterList_t lengths in bits). – If scatter option is selected, the number of input SGT entries shall be 2.
uint8_t	reserved2[2U]	
uint32_t	inputBitLength	INPUT: Length of the input message.(in bits)
uint64_t	pInput	INPUT: The input message.
uint8_t	tagBitLength	INPUT: Holds tag length in bits. – CMAC GENERATE: · On calling service (input), this parameter shall contain the length of the buffer (in bits) provided by pTag. · Recommended tag lengths are [32, 128]. Tag-lengths greater than 128 are truncated to 128. – CMAC VERIFY: · On calling service (input), this parameter shall contain the bit-length to be verified. · Recommended tag lengths are [32, 128]. · The HSE_FAST_CMAC_MIN_TAG_BIT_LEN_ATTR_ID attribute can be used to overwrite the lower recommended tag bit length limit (minimum is 1).
uint8_t	reserved3[3U]	
uint64_t	pTag	OUTPUT/INPUT: The output tag for "generate"; the input tag for "verify".

Data Fields

Type	Name	Description
uint64_t	pVolatileCounter	OUTPUT/INPUT: The address of the volatile counter. HSE reads/writes HSE_BITS_TO_BYTES (64-RPBitSize) bytes at pVolatileCounter address: – CMAC GENERATE: Specifies the address where to provide the Volatile Counter (Output parameter). – CMAC VERIFY: Input parameter that specifies the Volatile Counter to be used for the CMAC verify operation.

4.4 HSE HASH Service

Data Structures

- struct hseHashSrv_t

Data Structure Documentation

struct hseHashSrv_t

HASH service.

The HASH service is used to map data of arbitrary size to data of fixed size. The values returned by a hash function are called hash values, hash codes, digests, or simply hashes.

The HASH service can be accessible in one-pass or streaming (SUF) mode. In case of streaming mode, three steps (calls) will be used: START, UPDATE, FINISH. START and FINISH are mandatory; UPDATE is optional. Not all fields are used by each access mode.

The table below summarizes which fields are used by each access mode. Unused fields are ignored by the HSE.

Field \ Mode	One-pass	Start	Update	Finish
accessMode	*	*	*	*
streamId		*	*	*
hashAlgo	*	*		
sgtOption	*	*	*	*
inputLength	*	*	*	*
pInput	*	*	*	*
pHashLength	*			*

Field \ Mode	One-pass	Start	Update	Finish
pHash	*			*

Data Fields

Type	Name	Description
hseAccessMode_t	accessMode	INPUT: Specifies the access mode: ONE-PASS, START, UPDATE, FINISH. Note – Miyaguchi-Preneel does not support streaming. For MP this parameter is ignored and considered default ONE-PASS. STREAMING USAGE: Used in all steps.
hseStreamId_t	streamId	INPUT: Specifies the stream to use for START, UPDATE, FINISH access modes. Each interface supports a limited number of streams per interface, up to HSE_STREAM_COUNT . Note – Miyaguchi-Preneel does not support streaming. For MP this parameter is ignored. STREAMING USAGE: Used in all steps.
hseHashAlgo_t	hashAlgo	INPUT: Specifies the hash algorithm. STREAMING USAGE: Used in START.

Data Fields

Type	Name	Description
hseSGTOption_t	sgtOption	INPUT: Specify if pInput is provided as hseScatterList_t list (the host address points to a hseScatterList_t list). Ignored if SGT is not supported. Note – SGT is not available for the following hash algorithms and the parameter is ignored: · Miyaguchi-Preneel · SHA3 (unless the targeted platform has #HSE_SPT_HW_SHA3 defined) · SHA2_384/512 for HSE_B devices (not available in hardware) – ONLY HSE_SGT_OPTION_INPUT can be used. HSE_SGT_OPTION_OUTPUT will be ignored if used, as output is always considered a buffer. – If scatter option is selected (set), the length (e.g. inputLength) shall specified the entire message length (sum of all hseScatterList_t lengths). – The number for SGT entries shall be less then HSE_MAX_NUM_OF_SGT_ENTRIES . STREAMING USAGE: Used in all steps.

Data Fields

Type	Name	Description
uint32_t	inputLength	INPUT: Length of the input message. Can be zero (except Miyaguchi-Preneel). For Miyaguchi-Preneel, inputLength must be multiple of 16 bytes and not equal to zero. STREAMING USAGE: Used in all steps. – START: Must be a multiple of block length, or zero. – UPDATE: Must be a multiple of block length. Cannot be zero. Refrain from issuing the service request, instead of passing zero. – FINISH: Can be any value. Algorithm block lengths: – Miyaguchi-Preneel: not supported in streaming mode – SHA1, SHA2_224, SHA2_256: 64 – SHA2_384, SHA2_512, SHA2_512_224, SHA2_512_256: 128 – SHA3-224: 144 – SHA3-256: 136 – SHA3-384: 104 – SHA3-512: 72 – SHA3: If the targeted platform does NOT have #HSE_SPT_HW_SHA3 defined, there is no limitation (input can be any size)
uint64_t	pInput	INPUT: Address of the input message. For Miyaguchi-Preneel, according to SHE specification, the input shall be (K | C | padding). Ignored if inputLength is zero. STREAMING USAGE: Used in all steps (except if inputLength is zero). Note If the HOST_ADDR is on 64 bits and the hash algorithm is not supported in HW (#HSE_SPT_HW_SHA3 macro is not defined), the address must fall within the 32-bit address range.

Data Fields

Type	Name	Description
uint64_t	pHashLength	INPUT/OUTPUT: Pointer to a uint32_t location in which the hash length in bytes is stored. On calling this service, this parameter shall contain the size of the buffer provided by host. When the request has finished, the actual length of the returned value shall be stored. If the buffer is smaller than the size of the hash, the hash will be truncated (not applicable for Miyaguchi Preneel). For Miyaguchi-Preneel, if the buffer is smaller than the size of the hash (16 bytes), parameter will be considered invalid. If the buffer is larger, pHashLength is adjusted to the size of the hash. A hash buffer length (i.e. a pHashLength) of zero makes no sense, and is considered invalid. STREAMING USAGE: MANDATORY for FINISH.
uint64_t	pHash	OUTPUT: The address of the output buffer where the resulting hash will be stored. STREAMING USAGE: MANDATORY for FINISH.

4.5 HSE SipHash Service

Data Structures

- struct hseSipHashSrv_t

Macros

Type: hseSipHashVariant_t	
Name	**Value**
HSE_SIPHASH_VARIANT_64	0x2U
HSE_SIPHASH_VARIANT_128	0x4U

Typedefs

- typedef uint8_t hseSipHashVariant_t

Data Structure Documentation

struct hseSipHashSrv_t

SipHash service.

SipHash is a method to provide message authentication codes (MACs), also known as tags. These can be used to verify both the integrity and authenticity of a message. SipHash is optimized for fast processing speeds when used to authenticate small messages.

This service is only accessible in one-pass.

Data Fields

Type	Name	Description
hseAuthDir_t	authDir	INPUT: Specifies the direction: generate/verify.
hseSipHashVariant_t	sipHashVariant	INPUT: Specifies the SipHash variant: 64 or 128 bits. This is also the tag length in bits. The classic SipHash_2_4 is selected by picking HSE_SIPHASH_VARIANT_64 . Note At present we do not support a variable number of SipRounds
uint16_t	inputLength	INPUT: The length of the input message. Can be zero.
hseKeyHandle_t	keyHandle	INPUT: The key to be used for the operation. Must be a slot of type HSE_KEY_TYPE_SIPHASH , of the appropriate size for the variant. Key sizes for each SipHash variant are: – HSE_SIPHASH_VARIANT_64 : 128 bits (16 bytes) – HSE_SIPHASH_VARIANT_128 : 128 bits (16 bytes)
uint64_t	pInput	INPUT: The input message. Note If the HOST_ADDR is on 64 bits, the address must fall within the 32-bit address range.

Data Fields

Type	Name	Description
uint8_t	tagLength	INPUT: The tag length in bytes is stored. Zero length is invalid. – GENERATE: · On calling service, this parameter shall contain the size of the buffer provided by pTag. · If a truncated tag is desired, provide a shorter tag length. · Requested tag lengths greater than the algorithm variant output will be truncated to algorithm output (e.g. SipHash64 tag lengths greater than 8 will be truncated to 8) – VERIFY: · On calling service, this parameter shall contain the tag length to be verified. · Tag lengths less than the variant output assume a truncated tag. · Tag lengths greater than the variant output are invalid. (e.g. SipHash64 tag lengths greater than 8 are invalid).
uint8_t	reserved[3]	
uint64_t	pTag	OUTPUT/INPUT: The output tag for "generate"; the input tag for "verify".

Macro Definition Documentation

HSE_SIPHASH_VARIANT_64

```
#define HSE_SIPHASH_VARIANT_64 ((hseSipHashVariant_t)0x2U)
```

64 bit SipHash - the classic. Perform a SipHash_2_4 calculation.

HSE_SIPHASH_VARIANT_128

```
#define HSE_SIPHASH_VARIANT_128 ((hseSipHashVariant_t)0x4U)
```

128 bit SipHash

Typedef Documentation

hseSipHashVariant_t

```
typedef uint8_t hseSipHashVariant_t
```

HSE SipHash algorithm.

4.6 HSE AEAD Service

Data Structures

- struct hseAeadSrv_t

Data Structure Documentation

struct hseAeadSrv_t

AEAD service.

Authenticated Encryption with Associated Data (AEAD, also known as Authenticated Encryption) is a block cipher mode of operation which also allows integrity checks (e.g. AES-GCM). Additional authenticated data (AAD) is optional additional input header which is authenticated, but not encrypted. Both confidentiality and message authentication is provided on the input plaintext.

This service can be accessible in one-pass or streaming (SUF) mode. In case of streaming mode, three steps (calls) will be used: START, UPDATE, FINISH. START and FINISH are mandatory; UPDATE is optional. Not all fields are used by each access mode.

Note

1. Streaming mode is not supported for CCM.
2. The key usage flags used with AEAD operations:
   - HSE_KF_USAGE_ENCRYPT specifies that the key can be used for encryption and tag computation (note that the HSE_KF_USAGE_SIGN flag is not used).
   - HSE_KF_USAGE_DECRYPT specifies that the key can be used for decryption and tag verification (note that HSE_KF_USAGE_VERIFY flag is not used).

The table below summarizes which fields are used by each access mode. Unused fields are ignored by the HSE.

Field \ Mode	One-pass	Start	Update	Finish
accessMode	*	*	*	*
streamId		*	*	*
authCipherMode	*	*		
cipherDir	*	*		
keyHandle	*	*		

Field \ Mode	One-pass	Start	Update	Finish
ivLength	*	*		
pIV	*	*		
aadLength	*	*		
pAAD	*	*		
sgtOption	*	*	*	*
inputLength	*		*	*
pInput	*		*	*
tagLength	*			*
pTag	*			*
pOutput	*		*	*

Data Fields

Type	Name	Description
hseAccessMode_t	accessMode	INPUT: Specifies the access mode: ONE-PASS, START, UPDATE, FINISH. STREAMING USAGE: Used in all steps.
hseStreamId_t	streamId	INPUT: Specifies the stream to use for START, UPDATE, FINISH access modes. Each interface supports a limited number of streams per interface, up to HSE_STREAM_COUNT . STREAMING USAGE: Used in all steps.
hseAuthCipherMode_t	authCipherMode	INPUT: Specifies the authenticated cipher mode. STREAMING USAGE: Used in all steps.
hseCipherDir_t	cipherDir	INPUT: Specifies the cipher direction: encryption/decryption. STREAMING USAGE: Used in all steps.
hseKeyHandle_t	keyHandle	INPUT: The key to be used for the operation. STREAMING USAGE: Used in START step.
uint32_t	ivLength	INPUT: The length of the IV/Nonce (in bytes). – CCM valid IV sizes 7, 8, 9, 10, 11, 12, 13 bytes – GCM: 1<= ivLength <= 2^32-1. Recommended 12 bytes or greater. STREAMING USAGE: Used in START.
uint64_t	pIV	INPUT: Initialization Vector/Nonce. STREAMING USAGE: Used in START.

Data Fields

Type	Name	Description
uint32_t	aadLength	INPUT: The length of AAD Header data (in bytes). Can be zero. – CCM: Restricted to lengths less than or equal to (2^16 - 2^8) bytes. STREAMING USAGE: Used in START. Any AAD is ignored in UPDATE or FINISH, and must be passed to the HSE in START.
uint64_t	pAAD	INPUT: The AAD Header data. Ignored if aadLength is zero. STREAMING USAGE: Used in START. Any AAD is ignored in UPDATE or FINISH, and must be passed to the HSE in START.
hseSGTOption_t	sgtOption	INPUT: Specify if pInput/pOutput are provided as hseScatterList_t list (the host address points to a hseScatterList_t list). Ignored if SGT is not supported. Note – If scatter option is selected (set), the length (e.g. inputLength) shall specified the entire message length (sum of all hseScatterList_t lengths). – The number for SGT entries shall be less then HSE_MAX_NUM_OF_SGT_ENTRIES . STREAMING USAGE: Used in all steps.
uint8_t	reserved[3]	
uint32_t	inputLength	INPUT: The length of the plaintext and ciphertext (in bytes). Can be zero (compute/verify the tag without input message). STREAMING USAGE: – START: The input length is ignored. – UPDATE: Must be a multiple of block length. Cannot be zero. Refrain from issuing the service request instead of passing zero. – FINISH: All lengths are allowed.
uint64_t	pInput	INPUT: The plaintext for "authenticated encryption" or the ciphertext for "authenticated decryption". STREAMING USAGE: Used in UPDATE and FINISH step. Ignored for START step or if inputLength is zero.

Data Fields

Type	Name	Description
uint32_t	tagLength	INPUT: The length of tag (in bytes). – CCM valid Tag sizes 4, 6, 8, 10, 12, 14, 16 bytes – GCM valid Tag sizes 4, 8, 12, 13, 14, 15, 16 bytes STREAMING USAGE: Used in FINISH step.
uint64_t	pTag	OUTPUT/INPUT: The output tag for "authenticated encryption" or the input tag for "authenticated decryption". STREAMING USAGE: Used in FINISH step.
uint64_t	pOutput	OUTPUT: The ciphertext for "authenticated encryption" or the plaintext for "authenticated decryption". STREAMING USAGE: Used in UPDATE and FINISH step.

4.7 HSE Digital Signature Service

Data Structures

- struct hseSignSrv_t

Data Structure Documentation

struct hseSignSrv_t

Digital Signature service.

Uses the input parameters to perform the signature calculation and stores the signature in the memory location pointed by the output parameter.

This service can be accessible in one-pass or streaming (SUF) mode. In case of streaming mode, three steps (calls) will be used: START, UPDATE, FINISH. START and FINISH are mandatory; UPDATE is optional. Not all fields are used by each access mode.

The table below summarizes which fields are used by each access mode. Unused fields are ignored by the HSE.

Field \ Mode	One-pass	Start	Update	Finish
accessMode	*	*	*	*
streamId		*	*	*
signScheme	*	*		
authDir	*	*		

Field \ Mode	One-pass	Start	Update	Finish
keyHandle	*	*		
sgtOption	*	*	*	*
inputLength	*	*	*	*
pInput	*	*	*	*
pSignatureLength	*			*
pSignature	*			*

Data Fields

Type	Name	Description
hseAccessMode_t	accessMode	INPUT: Specifies the access mode: ONE-PASS, START, UPDATE, FINISH. STREAMING USAGE: Used in all steps.
hseStreamId_t	streamId	INPUT: Specifies the stream to use for START, UPDATE, FINISH access modes. Each interface supports a limited number of streams per interface, up to HSE_STREAM_COUNT . STREAMING USAGE: Used in all steps.
hseAuthDir_t	authDir	INPUT: Specifies the direction: generate/verify. STREAMING USAGE: Used in START.
bool_t	bInputIsHashed	INPUT: Specifies that the input is already hashed with the algorithm in specified in the sign scheme. Not valid for any signing scheme that does not perform prehashing (i.e. PureEDDSA) Note The hashing algorithm from signScheme and the input length must be: – for RSA scheme: hashAlgo != HSE_HASH_ALGO_NULL, where the inputLength must be equal to the hash ouput length; – for ECDSA scheme: hashAlgo != HSE_HASH_ALGO_NULL, where the inputLength must be equal to the hash ouput length; hashAlgo == HSE_HASH_ALGO_NULL, where the inputLength must be within (0, 64]; – for EDDSA scheme: hashAlgo is not used and the inputLength should be the output length of the hash associated with the curve; STREAMING USAGE: Not supported in streaming mode.

Data Fields

Type	Name	Description
hseSignScheme_t	signScheme	INPUT: Scheme for selected Signature algo. STREAMING USAGE: Used in START.
hseKeyHandle_t	keyHandle	INPUT: The key to be used for the operation. STREAMING USAGE: Used in START.
hseSGTOption_t	sgtOption	INPUT: Specify if pInput is provided as hseScatterList_t list (the host address points to a hseScatterList_t list). Ignored if SGT is not supported. Note – ONLY HSE_SGT_OPTION_INPUT can be used (the rest of the bits are ignored) – Scatter option is supported only for RSA and ECDSA. For EDDSA, it is not supported. – Scatter option is not supported for SHA3 hashes (unless the targeted platform has #HSE_SPT_HW_SHA3 defined). – Scatter option is not supported for pre-hashed inputs – If scatter option is selected (set), the length (e.g. inputLength) shall specified the entire message length (sum of all hseScatterList_t lengths). – The number for SGT entries shall be less then HSE_MAX_NUM_OF_SGT_ENTRIES . STREAMING USAGE: Used in all steps.
uint8_t	reserved[3U]	

Data Fields

Type	Name	Description
uint32_t	inputLength	INPUT: The length of the message. For RSA schemes, this must be the length of the original (not pre-hashed) input. STREAMING USAGE: Used in all steps. – START: Must be a multiple of block length of the hash, or zero. – UPDATE: Must be a multiple of block length of the hash. Cannot be zero. Refrain from issuing the service request, instead of passing zero. – FINISH: Can be any value. Algorithm block lengths: – SHA1, SHA2_224, SHA2_256: 64 – SHA2_512_224, SHA2_512_256, SHA2_384, SHA2_512: 128 – SHA3-224: 144 – SHA3-256: 136 – SHA3-384: 104 – SHA3-512: 72 – SHA3: If the targeted platform does NOT have #HSE_SPT_HW_SHA3 defined, there is no limitation (input can be any size) Note EDDSA does not support streaming
uint64_t	pInput	INPUT: The address of the message to be signed/verify. For RSA schemes, this is the actual (not pre-hashed) input. STREAMING USAGE: Used in all steps. Note If the HOST_ADDR is on 64 bits: – For PureEDDSA: the address must fall within the 32-bit address range. – If hash algorithm is not supported in HW (#HSE_SPT_HW_SHA3 macro is not defined) and input is not hashed, the address must fall within the 32-bit address range.

Data Fields

Type	Name	Description
uint64_t	pSignatureLength[2]	INPUT/OUTPUT: An array of two addresses of two uint32_t values containing signature lengths. It is input/output for "generate" and input for "verify". On calling "generate" service, these parameter shall contain the size of the signature buffers provided by the application. When the request has finished, the actual lengths of the signature components. STREAMING USAGE: Used in FINISH.
uint64_t	pSignature[2]	OUTPUT: Where the signature components must be stored. It is output for "generate" and input for "verify". – RSA has a single signature component, at index 0, and the size of buffer must be at least the byteLength(public modulus n) – ECDSA and EDDSA signature format as (r,s), with r at index 0, and s at index 1. The buffer size for each component must be at least the length of the used curve in bytes (e.g. 32 bytes for a 256 bit curve). STREAMING USAGE: Used in FINISH.

4.8 HSE RSA Cipher Service

Data Structures

- struct hseRsaCipherSrv_t

Data Structure Documentation

struct hseRsaCipherSrv_t

RSA Cipher service.

Performs the RSA Cipher (Encryption/Decryption) (RSAEP) operation.

Data Fields

Type	Name	Description
hseRsaCipherScheme_t	rsaScheme	INPUT: The RSA cipher scheme.

Data Fields

Type	Name	Description
hseCipherDir_t	cipherDir	INPUT: Specifies the cipher direction: encryption/decryption.
uint8_t	reserved[3]	
hseKeyHandle_t	keyHandle	INPUT: The key to be used for the operation.
uint32_t	inputLength	INPUT: The input length (plaintext or ciphertext): – The length of the ciphertext should be HSE_BITS_TO_BYTES(keyBitLen) . – The length of the plaintext (in bytes): · For RSAES NO PADDING, the Input Length must be less than or equal to HSE_BITS_TO_BYTES(keyBitLen), and pInput is considered a big-endian integer. · For RSAES-PKCS1-v1_5, the Input Length shall not be greater than HSE_BITS_TO_BYTES(keyBitLen) -11 bytes. · For RSAES-OAEP, Input Length shall not be greater than HSE_BITS_TO_BYTES(keyBitLen) - 2 ∗ hashLen - 2 bytes.
uint64_t	pInput	INPUT: The plaintext for encryption or the ciphertext for decryption.
uint64_t	pOutputLength	INPUT/OUTPUT: Holds the address to a location (an uint32_t variable) in which the output length in bytes is stored. On calling this service, this parameter shall contain the size of the buffer provided by the application. When the request has finished, the actual length of the returned value shall be stored.
uint64_t	pOutput	OUTPUT: The address of the Output. The plaintext for decryption or ciphertext for encryption. The size of output must be at least the HSE_BITS_TO_BYTES(keyBitLen)

4.9 HSE Combined Authenticated Encryption Service

Data Structures

- struct hseAuthEncSrv_t

Data Structure Documentation

struct hseAuthEncSrv_t

HSE Authenticated Encryption.

This service allows to perform in parallel the Encrypt-then-MAC operation using NULL/AES cipher and MAC algorithms. HSE Authenticated Encryption uses two keys: one for encryption/decryption and another for MAC generate/verify.
The authenticated encryption service (hseAuthEncSrv_t) supports the following combinations:

- AES_(ECB, CBC, CTR, CFB, OFB) and HMAC_ (SHA1, SHA2_224, SHA2_256, SHA2_384, SHA2_512); CMAC/GMAC/XCBC_MAC are not supported with AES encryption.
- NULL cipher with all MAC algorithms (CMAC, GMAC, XCBC_MAC, HMAC_(SHA1, all SHA2))

Data Fields

Type	Name	Description
hseCipherAlgo_t	cipherAlgo	INPUT: Specifies the cipher algorithm. Can be either HSE_CIPHER_ALGO_NULL or HSE_CIPHER_ALGO_AES .
hseCipherBlockMode_t	cipherBlockMode	INPUT: Specifies the block cipher mode. All cipher block modes are supported. Ignored if HSE_CIPHER_ALGO_NULL is used.
hseCipherDir_t	cipherDir	INPUT: Specifies the cipher direction: encryption/decryption or MAC generate/verify.
hseSGTOption_t	sgtOption	INPUT: Specify if pInput/pOutput are provided as hseScatterList_t list (the host address points to a hseScatterList_t list). Ignored if SGT is not supported. Note – If scatter option is selected (set), the length (e.g. inputLength) shall specified the entire message length (sum of all hseScatterList_t lengths). – The number for SGT entries shall be less then HSE_MAX_NUM_OF_SGT_ENTRIES .
hseKeyHandle_t	cipherKeyHandle	INPUT: The key to be used for the cipher operation. This parameter is ignored if HSE_CIPHER_ALGO_NULL is used.
hseMacScheme_t	macScheme	INPUT: Specifies the authentication scheme. All MAC schemes are supported. Note The IV from GMAC is ignored (the IV from this structure is used).

Data Fields

Type	Name	Description
hseKeyHandle_t	authKeyHandle	INPUT: The key to be used for the MAC operation. Note HMAC key size shall be less than hash block size (e.g. 64bytes for SHA2_256).
uint32_t	inputLength	INPUT: The length of the plaintext and ciphertext (in bytes).
uint64_t	pInput	INPUT: The plaintext for "authenticated encryption"; the ciphertext for "authenticated decryption".
uint32_t	ivLength	INPUT: The length of the IV/Nonce (in bytes)used for AES cipher. Not used for ECB mode.
uint64_t	pIV	INPUT: Initialization Vector/Nonce used for AES cipher. Not used for ECB mode. Note The IV is also used for GMAC authentication scheme when processed.
uint32_t	aadLength	INPUT: Length of Additional Authenticated Data (AAD). Optional (can be 0)
uint64_t	pAAD	INPUT: Pointer to Additional Authenticated Data (AAD) buffer. Optional (can be NULL)
uint64_t	pOutput	OUTPUT: The ciphertext for "authenticated encryption" or the plaintext for "authenticated decryption".

Data Fields

Type	Name	Description
uint64_t	pTagLength	INPUT/OUTPUT: Holds the address to a memory location (an uint32_t variable) in which the tag length in bytes is stored. GENERATE: – On calling service (input), this parameter shall contain the size of the buffer provided by pTag. – For GMAC, valid tag lengths are 4, 8, 12, 13, 14, 15 and 16. Tag-lengths greater than 16 will be truncated to 16. – For HMAC, valid tag lengths are [1, hash-length]. Tag-lengths greater than hash-length will be truncated to hash-length. – For CMAC & XCBC-MAC, valid tag lengths are [4, cipher-block-length]. Tag-lengths greater than cipher-block-length will be truncated to cipher-block-length. – When the request has finished (output), the actual length of the returned value shall be stored. VERIFY: – On calling service (input), this parameter shall contain the tag-length to be verified. – For GMAC, valid tag lengths are 4, 8, 12, 13, 14, 15 and 16. – For HMAC, valid tag lengths are [1, hash-length]. – For CMAC & XCBC-MAC, valid tag lengths are [4, cipher block-length].
uint64_t	pTag	OUTPUT/INPUT: The output tag for "authenticated encryption" or the input tag for "authenticated decryption". Ignored if tagLength is zero.

4.10 HSE CRC32 service

Data Structures

- struct hseCrc32Srv_t

Macros

Type: hseCrc32Mode_t	
Name	**Value**
HSE_CRC32_MODE_IEEE_802	0x00000010U

HSE_CRC32_MODE_IETF_3385	0x00000020U
HSE_CRC32_MODE_DONT_INPUT_SWAP	0x00000100U
HSE_CRC32_MODE_DONT_OUTPUT_SWAP	0x00000200U
HSE_CRC32_MODE_DONT_OUTPUT_COMP	0x00000400U
HSE_CRC32_MODE_INITIAL_VALUE_ZERO	0x00000800U

Typedefs

- typedef uint32_t hseCrc32Mode_t

Data Structure Documentation

struct hseCrc32Srv_t

CRC service.

CRC32 is an error-detecting code commonly used in network protocols(such as IPsec). Can be used as an separate service.
This service support two standards:
- HSE_CRC32_MODE_IEEE_802 ; Polynomial: 0x04c11db7 (names IEEE 802.3 or CRC-32)
- HSE_CRC32_MODE_IETF_3385 ; Polynomial: 0x1edc6f41 (names iSCSI, CRC-32C or CRC-32/4)

With additional flags depending on the format of input/output/desired CRC variant:
- HSE_CRC32_MODE_DONT_INPUT_SWAP
- HSE_CRC32_MODE_DONT_OUTPUT_SWAP
- HSE_CRC32_MODE_DONT_OUTPUT_COMP
- HSE_CRC32_MODE_INITIAL_VALUE_ZERO

Data Fields

Type	Name	Description
uint32_t	crcOpMode	INPUT: Specify te operation mode for CRC32 computation. Supported operation: – HSE_CRC32_MODE_IEEE_802 – HSE_CRC32_MODE_IETF_3385 Additional flags that can be used depending on the format of input/output/desired CRC variant. In general, the CRC variant may be simply OR-ed with the desired flags e.g.: crcOperation = HSE_CRC32_MODE_IEEE_802 | HSE_CRC32_MODE_DONT_INPUT_SWAP | HSE_CRC32_MODE_DONT_OUTPUT_SWAP ;
hseSGTOption_t	sgtOption	INPUT: Specify if the pInput are provided as hseScatterList_t list (the host address points to a hseScatterList_t list). Ignored if SGT is not supported. Note – ONLY HSE_SGT_OPTION_INPUT can be used (the rest of the bits are ignored) – If scatter option is selected (set), the length (e.g. inputLength) shall specified the entire message length (sum of all hseScatterList_t lengths). – The number for SGT entries shall be less then HSE_MAX_NUM_OF_SGT_ENTRIES .
uint8_t	reserved[3]	
uint32_t	inputLength	INPUT: Length of the input data(in bytes).
uint64_t	pInput	INPUT: The input data that is used to calculate CRC32.
uint64_t	pOutput	OUTPUT: The address where output CRC32 (an uint32_t value) will be stored.

Macro Definition Documentation

HSE_CRC32_MODE_IEEE_802

```
#define HSE_CRC32_MODE_IEEE_802 ((hseCrc32Mode_t)0x00000010U)
```

CRC32_IEEE_802 standard.

HSE_CRC32_MODE_IETF_3385

```
#define HSE_CRC32_MODE_IETF_3385 ((hseCrc32Mode_t)0x00000020U)
```

CRC32_IETF_3385 standard.

HSE_CRC32_MODE_DONT_INPUT_SWAP

```
#define HSE_CRC32_MODE_DONT_INPUT_SWAP ((hseCrc32Mode_t)0x00000100U)
```

HSE CRC32 additional flags for CRC32 computation .

The input is not bit-swapped within each byte (the difference between with- and without- this flag is equivalent to bitswap within each byte of the input).

HSE_CRC32_MODE_DONT_OUTPUT_SWAP

```
#define HSE_CRC32_MODE_DONT_OUTPUT_SWAP ((hseCrc32Mode_t)0x00000200U)
```

The output is not bit-swapped within each byte (the difference between with- and without- this flag is equivalent to bitswap within each byte of the output).

HSE_CRC32_MODE_DONT_OUTPUT_COMP

```
#define HSE_CRC32_MODE_DONT_OUTPUT_COMP ((hseCrc32Mode_t)0x00000400U)
```

The output is not complimented (xored with all-ones) (the difference between with- and without- this flag is equivalent to xor of the output with all-ones).

HSE_CRC32_MODE_INITIAL_VALUE_ZERO

```
#define HSE_CRC32_MODE_INITIAL_VALUE_ZERO ((hseCrc32Mode_t)0x00000800U)
```

The initial crc value is 0 instead of all-ones.

Typedef Documentation

hseCrc32Mode_t

```
typedef uint32_t hseCrc32Mode_t
```

HSE CRC32 supported standards algorithms .

5 Key Management Services

5.1 HSE Key Management Common Types

Data Structures

- struct hseKeyGroupCfgEntry_t
- struct hseKeyInfo_t
- union hseKeyInfo_t.specific

Macros

Type: (implicit C type)	
Name	**Value**
GET_KEY_HANDLE (catalogId, groupIdx, slotIdx)	-
HSE_KF_USAGE_MASK	-
HSE_KF_MSC_USAGE_MASK	HSE_KF_USAGE_ENCRYPT | HSE_KF_USAGE_DECRYPT | HSE_KF_USAGE_SIGN | HSE_KF_USAGE_VERIFY
HSE_KF_ACCESS_MASK	HSE_KF_ACCESS_WRITE_PROT | HSE_KF_ACCESS_DEBUG_PROT | HSE_KF_ACCESS_EXPORTABLE
HSE_KF_MAX_KEY_COUNTER_VALUE	((uint32_t)0xFFFFFFFFUL - 1UL)

Type: hseKeyHandle_t	
Name	**Value**
HSE_INVALID_KEY_HANDLE	0xFFFFFFFFUL
HSE_ROM_KEY_AES256_KEY0	0x00000000UL
HSE_ROM_KEY_AES256_KEY1	0x00000001UL
HSE_ROM_KEY_RSA3072_PUB_KEY0	0x00000100UL
HSE_ROM_KEY_ECC256_PUB_KEY0	0x00000200UL

Type: hseKeyGroupIdx_t	
Name	**Value**
GET_GROUP_IDX (keyHandle)	((keyHandle) >> 8U) & 0xFFU
HSE_INVALID_GROUP_IDX	0xFFU

Type: hseKeySlotIdx_t	
Name	**Value**
GET_SLOT_IDX (keyHandle)	(keyHandle) & 0xFFU

HSE_INVALID_SLOT_IDX	0xFFU

Type: hseSmrFlags_t	
Name	**Value**
HSE_KF_SMR_0	1UL << 0UL
HSE_KF_SMR_1	1UL << 1UL
HSE_KF_SMR_2	1UL << 2UL
HSE_KF_SMR_3	1UL << 3UL
HSE_KF_SMR_4	1UL << 4UL
HSE_KF_SMR_5	1UL << 5UL
HSE_KF_SMR_6	1UL << 6UL
HSE_KF_SMR_7	1UL << 7UL
HSE_KF_SMR_8	1UL << 8UL
HSE_KF_SMR_9	1UL << 9UL
HSE_KF_SMR_10	1UL << 10UL
HSE_KF_SMR_11	1UL << 11UL
HSE_KF_SMR_12	1UL << 12UL
HSE_KF_SMR_13	1UL << 13UL
HSE_KF_SMR_14	1UL << 14UL
HSE_KF_SMR_15	1UL << 15UL
HSE_KF_SMR_16	1UL << 16UL
HSE_KF_SMR_17	1UL << 17UL
HSE_KF_SMR_18	1UL << 18UL
HSE_KF_SMR_19	1UL << 19UL
HSE_KF_SMR_20	1UL << 20UL
HSE_KF_SMR_21	1UL << 21UL
HSE_KF_SMR_22	1UL << 22UL
HSE_KF_SMR_23	1UL << 23UL
HSE_KF_SMR_24	1UL << 24UL
HSE_KF_SMR_25	1UL << 25UL
HSE_KF_SMR_26	1UL << 26UL
HSE_KF_SMR_27	1UL << 27UL
HSE_KF_SMR_28	1UL << 28UL
HSE_KF_SMR_29	1UL << 29UL
HSE_KF_SMR_30	1UL << 30UL
HSE_KF_SMR_31	1UL << 31UL

Type: hseEccCurveId_t	
Name	**Value**
HSE_EC_CURVE_NONE	0U

HSE_EC_SEC_SECP256R1	1U
HSE_EC_SEC_SECP384R1	2U
HSE_EC_SEC_SECP521R1	3U
HSE_EC_BRAINPOOL_BRAINPOOLP256R1	4U
HSE_EC_BRAINPOOL_BRAINPOOLP320R1	5U
HSE_EC_BRAINPOOL_BRAINPOOLP384R1	6U
HSE_EC_BRAINPOOL_BRAINPOOLP512R1	7U
HSE_EC_25519_ED25519	9U
HSE_EC_25519_CURVE25519	10U
HSE_EC_448_ED448	11U
HSE_EC_448_CURVE448	12U
HSE_EC_USER_CURVE1	101U
HSE_EC_USER_CURVE2	102U
HSE_EC_USER_CURVE3	103U

Type: hseKeyBits_t

Name	Value
HSE_KEY_BITS_INVALID	0xFFFFU
HSE_KEY_BITS_ZERO	0U
HSE_KEY64_BITS	64U
HSE_KEY128_BITS	128U
HSE_KEY160_BITS	160U
HSE_KEY192_BITS	192U
HSE_KEY224_BITS	224U
HSE_KEY240_BITS	240U
HSE_KEY256_BITS	256U
HSE_KEY320_BITS	320U
HSE_KEY384_BITS	384U
HSE_KEY512_BITS	512U
HSE_KEY521_BITS	521U
HSE_KEY638_BITS	638U
HSE_KEY1024_BITS	1024U
HSE_KEY2048_BITS	2048U
HSE_KEY3072_BITS	3072U
HSE_KEY4096_BITS	4096U

Type: hseKeyType_t

Name	Value
HSE_KEY_TYPE_SHE	0x11U
HSE_KEY_TYPE_AES	0x12U

HSE_KEY_TYPE_HMAC	0x20U
HSE_KEY_TYPE_SHARED_SECRET	0x30U
HSE_KEY_TYPE_SIPHASH	0x40U
HSE_KEY_TYPE_ECC_PAIR	0x87U
HSE_KEY_TYPE_ECC_PUB	0x88U
HSE_KEY_TYPE_ECC_PUB_EXT	0x89U
HSE_KEY_TYPE_RSA_PAIR	0x97U
HSE_KEY_TYPE_RSA_PUB	0x98U
HSE_KEY_TYPE_RSA_PUB_EXT	0x99U
HSE_KEY_TYPE_DH_PAIR	0xA7U
HSE_KEY_TYPE_DH_PUB	0xA8U

Type: hseKeyGroupOwner_t	
Name	**Value**
HSE_KEY_OWNER_ANY	0U
HSE_KEY_OWNER_CUST	1U
HSE_KEY_OWNER_OEM	2U

Type: hseKeyFlags_t	
Name	**Value**
HSE_KF_USAGE_ENCRYPT	1U << 0U
HSE_KF_USAGE_DECRYPT	1U << 1U
HSE_KF_USAGE_SIGN	1U << 2U
HSE_KF_USAGE_VERIFY	1U << 3U
HSE_KF_USAGE_EXCHANGE	1U << 4U
HSE_KF_USAGE_DERIVE	1U << 5U
HSE_KF_USAGE_KEY_PROVISION	1U << 6U
HSE_KF_USAGE_AUTHORIZATION	1U << 7U
HSE_KF_USAGE_SMR_DECRYPT	1U << 8U
HSE_KF_ACCESS_WRITE_PROT	1U << 9U
HSE_KF_ACCESS_DEBUG_PROT	1U << 10U
HSE_KF_ACCESS_EXPORTABLE	1U << 11U
HSE_KF_USAGE_XTS_TWEAK	1U << 12U
HSE_KF_USAGE_OTFAD_DECRYPT	1U << 13U

Type: hseAesBlockModeMask_t	
Name	**Value**
HSE_KU_AES_BLOCK_MODE_ANY	0U
HSE_KU_AES_BLOCK_MODE_XTS	1U << 0U

HSE_KU_AES_BLOCK_MODE_CTR	1U << HSE_CIPHER_BLOCK_MODE_CTR
HSE_KU_AES_BLOCK_MODE_CBC	1U << HSE_CIPHER_BLOCK_MODE_CBC
HSE_KU_AES_BLOCK_MODE_ECB	1U << HSE_CIPHER_BLOCK_MODE_ECB
HSE_KU_AES_BLOCK_MODE_CFB	1U << HSE_CIPHER_BLOCK_MODE_CFB
HSE_KU_AES_BLOCK_MODE_OFB	1U << HSE_CIPHER_BLOCK_MODE_OFB
HSE_KU_AES_BLOCK_MODE_CCM	1U << 6U
HSE_KU_AES_BLOCK_MODE_GCM	1U << 7U

Type: hseKeyCatalogId_t	
Name	**Value**
HSE_KEY_CATALOG_ID_ROM	0U
HSE_KEY_CATALOG_ID_NVM	1U
HSE_KEY_CATALOG_ID_RAM	2U
GET_CATALOG_ID (keyHandle)	((keyHandle) >> 16U) & 0xFFU

Typedefs

- typedef uint8_t hseKeyCatalogId_t
- typedef uint8_t hseKeyGroupOwner_t
- typedef uint8_t hseKeyType_t
- typedef uint16_t hseKeyFlags_t
- typedef uint32_t hseSmrFlags_t
- typedef uint8_t hseEccCurveId_t
- typedef uint16_t hseKeyBits_t
- typedef uint8_t hseAesBlockModeMask_t

Data Structure Documentation

struct hseKeyGroupCfgEntry_t

The entry of the Key Catalog Configuration.

The size of a key slot is computed internally based on keytype and maxKeyBitLen.

Note

> A key group (catalog entry) contains keys that have the same key type and the keybitLen <= maxKeyBitLen.

Data Fields

Type	Name	Description
hseMuMask_t	muMask	Specifies the MU Instance(s) for the key group. A key group can belong to one ore more MUs.
hseKeyGroupOwner_t	groupOwner	Specifies the key group owner.
hseKeyType_t	keyType	The key type (see hseKeyType_t).
uint8_t	numOfKeySlots	The number of key slots.
uint16_t	maxKeyBitLen	The maximum length of the key (in bits). All stored keys have keyBitLen <= maxKeyBitLen.
uint8_t	hseReserved[2]	HSE reserved.

struct hseKeyInfo_t

Key properties.

Each cryptographic key material will be based on key properties (info) and key data

Data Fields

Type	Name	Description
hseKeyFlags_t	keyFlags	The key flags (see hseKeyFlags_t)
uint16_t	keyBitLen	The length of key in bits. – For RSA, bit length of modulus n – For ECC, the bit length of the base point order. – Any other key, the bit length of the key.
uint32_t	keyCounter	The key counter used to prevent the rollback attacks on the key. For NVM keys, the key counter must be between 0 and HSE_KF_MAX_KEY_COUNTER_VALUE For RAM keys, the key counter is forced to 0xFFFFFFFF (not used). Note The key counter for SHE keys follows the SHE specification (e.g. key counter is 28bits; for SHE RAM keys, the key counter is forced to zero).

Data Fields

Type	Name	Description
hseSmrFlags_t	smrFlags	A set of flags that define which secure memory region (SMR), indexed from 0 to 31, should be verified before the key can be used. Set to zero means not used. For RAM keys, the SMR flags are forced to zero (not used). Keys linked with SMR(s) that are not yet present in the system will be available until these SMR(s) are successfully installed.
hseKeyType_t	keyType	The key type (see hseKeyType_t).
union hseKeyInfo_t.specific	specific	
uint8_t	hseReserved[2U]	

union hseKeyInfo_t.specific

Data Fields

Type	Name	Description
hseEccCurveId_t	eccCurveId	The ECC curve Id used with this key. This is used only for ECC key type.
uint8_t	pubExponentSize	The size (in bytes) of the RSA public exponent (e); it should be less than 16 bytes.
hseAesBlockModeMask_t	aesBlockModeMask	The cipher mode usage for an AES key. This is used only for AES key type If aesBlockModeMask == 0, any AES block mode can be used.

Macro Definition Documentation

HSE_KEY_CATALOG_ID_ROM

```
#define HSE_KEY_CATALOG_ID_ROM ((hseKeyCatalogId_t)0U)
```

ROM key catalog (NXP keys)

HSE_KEY_CATALOG_ID_NVM

```
#define HSE_KEY_CATALOG_ID_NVM ((hseKeyCatalogId_t)1U)
```

NVM key catalog.

HSE_KEY_CATALOG_ID_RAM

```
#define HSE_KEY_CATALOG_ID_RAM ((hseKeyCatalogId_t)2U)
```

RAM key catalog.

GET_KEY_HANDLE

```
#define GET_KEY_HANDLE( catalogId, groupIdx, slotIdx )
```

Value:

```
(((hseKeyHandle_t)((hseKeyCatalogId_t)(catalogId))) « 16U) | \
 (((hseKeyHandle_t)((hseKeyGroupIdx_t)(groupIdx))) « 8U) | \
 (((hseKeyHandle_t)((hseKeySlotIdx_t)(slotIdx)))))
```

All keys used in cryptographic operations are referenced by a unique key handle. The key handle is a 32-bit integer: the key catalog(byte2), group index in catalog (byte1) and key slot index (byte0). It can be retrieved based on the catalog ID, the group index and its slot index within the group. The group index is between 0 and (n-1), where n is the maximum number of groups defined in the catalog The slot index is between 0 and (p-1), where p is the maximum number of keys defined in the group.

GET_CATALOG_ID

```
#define GET_CATALOG_ID( keyHandle ) ((hseKeyCatalogId_t)(((keyHandle) >> 16U) &
0xFFU))
```

Get key catalog Id.

GET_GROUP_IDX

```
#define GET_GROUP_IDX( keyHandle ) ((hseKeyGroupIdx_t)(((keyHandle) >> 8U) &
0xFFU))
```

Get key group index.

GET_SLOT_IDX

```
#define GET_SLOT_IDX( keyHandle ) ((hseKeySlotIdx_t)((keyHandle) & 0xFFU))
```

Get key slot index.

HSE_INVALID_KEY_HANDLE

```
#define HSE_INVALID_KEY_HANDLE ((hseKeyHandle_t)0xFFFFFFFFUL)
```

HSE invalid key .

HSE_INVALID_GROUP_IDX

```
#define HSE_INVALID_GROUP_IDX ((hseKeyGroupIdx_t)0xFFU)
```

HSE invalid key group index.

HSE_INVALID_SLOT_IDX

```
#define HSE_INVALID_SLOT_IDX ((hseKeySlotIdx_t)0xFFU)
```

HSE invalid key slot index.

HSE_KEY_OWNER_ANY

```
#define HSE_KEY_OWNER_ANY ((hseKeyGroupOwner_t)0U)
```

The key are owned by ANY owner. This applies only for RAM key groups. The RAM keys can be installed/updated by any owner (CUST or OEM) having SuperUser or User rights.

HSE_KEY_OWNER_CUST

```
#define HSE_KEY_OWNER_CUST ((hseKeyGroupOwner_t)1U)
```

The key are owned by OWNER_CUST. This applies only for NVM key groups.
The CUST keys can be installed/updated as follow:

- using CUST SuperUser rights (if Life Cycle = CUST_DEL or if the host was granted with CUST SuperUser rights).
- using User rights (Life Cycle = IN_FIELD)

HSE_KEY_OWNER_OEM

```
#define HSE_KEY_OWNER_OEM ((hseKeyGroupOwner_t)2U)
```

The key groups owned by OWNER_OEM. This applies only for NVM key groups.
The OEM keys can be installed/updated as follow:

- using OEM SuperUser rights (if Life Cycle = OEM_PROD or if the host was granted with OEM SuperUser rights).
- using User rights (Life Cycle = IN_FIELD)

HSE_KEY_TYPE_SHE

```
#define HSE_KEY_TYPE_SHE ((hseKeyType_t)0x11U)
```

Symmetric AES128 key used with SHE specification commands. It can be used with any AES block ciphering mode and AES MACs (same as any AES128 key).

HSE_KEY_TYPE_AES

```
#define HSE_KEY_TYPE_AES ((hseKeyType_t)0x12U)
```

Symmetric AES key or AES OTFAD key.

HSE_KEY_TYPE_HMAC

```
#define HSE_KEY_TYPE_HMAC ((hseKeyType_t)0x20U)
```

Symmetric HMAC key.

HSE_KEY_TYPE_SHARED_SECRET

```
#define HSE_KEY_TYPE_SHARED_SECRET ((hseKeyType_t)0x30U)
```

Shared secret used by DH key exchange protocols.

HSE_KEY_TYPE_SIPHASH

```
#define HSE_KEY_TYPE_SIPHASH ((hseKeyType_t)0x40U)
```

Symmetric SipHash key.

HSE_KEY_TYPE_ECC_PAIR

```
#define HSE_KEY_TYPE_ECC_PAIR ((hseKeyType_t)0x87U)
```

ECC key pair (private and public)

HSE_KEY_TYPE_ECC_PUB

```
#define HSE_KEY_TYPE_ECC_PUB ((hseKeyType_t)0x88U)
```

ECC Public key.

HSE_KEY_TYPE_ECC_PUB_EXT

```
#define HSE_KEY_TYPE_ECC_PUB_EXT ((hseKeyType_t)0x89U)
```

ECC public keys, where the key value is stored in the application area (e.g. certificate)

HSE_KEY_TYPE_RSA_PAIR

```
#define HSE_KEY_TYPE_RSA_PAIR ((hseKeyType_t)0x97U)
```

RSA key pair (private and public key)

HSE_KEY_TYPE_RSA_PUB

```
#define HSE_KEY_TYPE_RSA_PUB ((hseKeyType_t)0x98U)
```

RSA Public key.

HSE_KEY_TYPE_RSA_PUB_EXT

```
#define HSE_KEY_TYPE_RSA_PUB_EXT ((hseKeyType_t)0x99U)
```

RSA public keys, where the key value is stored in the application area (e.g. certificate)

HSE_KEY_TYPE_DH_PAIR

```
#define HSE_KEY_TYPE_DH_PAIR ((hseKeyType_t)0xA7U)
```

DH key pair.

HSE_KEY_TYPE_DH_PUB

```
#define HSE_KEY_TYPE_DH_PUB ((hseKeyType_t)0xA8U)
```

DH public key.

HSE_KF_USAGE_ENCRYPT

```
#define HSE_KF_USAGE_ENCRYPT ((hseKeyFlags_t)1U << 0U)
```

Key is used to encrypt data. If the HSE_KF_USAGE_KEY_PROVISION flag is set, the key can not be used for data encryption (only for key encryption).

HSE_KF_USAGE_DECRYPT

```
#define HSE_KF_USAGE_DECRYPT ((hseKeyFlags_t)1U << 1U)
```

Key is used to decrypt data. If the HSE_KF_USAGE_KEY_PROVISION flag is set, the key can not be used for data decryption (only for key decryption).

HSE_KF_USAGE_SIGN

```
#define HSE_KF_USAGE_SIGN ((hseKeyFlags_t)1U << 2U)
```

Key is used to generate digital signatures or MACs of any data (including keys if HSE_KF_USAGE_KEY_PROVISION is set).

HSE_KF_USAGE_VERIFY

```
#define HSE_KF_USAGE_VERIFY ((hseKeyFlags_t)1U << 3U)
```

Key is used to verify digital signatures or MACs of any data (including keys if HSE_KF_USAGE_KEY_PROVISION is set).

HSE_KF_USAGE_EXCHANGE

```
#define HSE_KF_USAGE_EXCHANGE ((hseKeyFlags_t)1U << 4U)
```

Key is used for key exchange protocol (e.g. DH).

HSE_KF_USAGE_DERIVE

```
#define HSE_KF_USAGE_DERIVE ((hseKeyFlags_t)1U << 5U)
```

Key may be use as a base key for deriving other keys.

HSE_KF_USAGE_KEY_PROVISION

```
#define HSE_KF_USAGE_KEY_PROVISION ((hseKeyFlags_t)1U << 6U)
```

Key used for key provisioning operation. The provision keys can only be NVM keys. This bit (if it is set) along with the encrypt/decrypt/sign/verify flags specifies which operations can be performed on a key using this key (provisioning key).

HSE_KF_USAGE_AUTHORIZATION

```
#define HSE_KF_USAGE_AUTHORIZATION ((hseKeyFlags_t)1U << 7U)
```

Key can be used for system authorization. Can be set only for NVM keys. This key should have the verify flag set, but the sign flag NOT set.

HSE_KF_USAGE_SMR_DECRYPT

```
#define HSE_KF_USAGE_SMR_DECRYPT ((hseKeyFlags_t)1U << 8U)
```

The key is used for SMR decryption. If this bit is set during key installation, the HSE will set the HSE_KF_USAGE_DECRYPT flag to zero.

HSE_KF_ACCESS_WRITE_PROT

```
#define HSE_KF_ACCESS_WRITE_PROT ((hseKeyFlags_t)1U << 9U)
```

The key is write protected and cannot change anymore. For RAM keys, this flag is forced to zero.

HSE_KF_ACCESS_DEBUG_PROT

```
#define HSE_KF_ACCESS_DEBUG_PROT ((hseKeyFlags_t)1U << 10U)
```

The key is disabled when a debugger is attached. For RAM keys, this flag is forced to zero.

HSE_KF_ACCESS_EXPORTABLE

```
#define HSE_KF_ACCESS_EXPORTABLE ((hseKeyFlags_t)1U << 11U)
```

The key can be exported or not in any format. Ignored when used in combination with HSE_KF_USAGE_KEY_PROVISION or HSE_KF_USAGE_AUTHORIZATION (provision/authorization keys are NOT exportable).

HSE_KF_USAGE_XTS_TWEAK

```
#define HSE_KF_USAGE_XTS_TWEAK ((hseKeyFlags_t)1U << 12U)
```

This is used as a tweak key in xts aes encryption; no other flag shall be set.

HSE_KF_USAGE_OTFAD_DECRYPT

```
#define HSE_KF_USAGE_OTFAD_DECRYPT ((hseKeyFlags_t)1U << 13U)
```

The key is used just in OTFAD decryption; no other flag shall be set.

HSE_KF_USAGE_MASK

```
#define HSE_KF_USAGE_MASK
```

Value:

```
( HSE_ KF_ USAGE_ ENCRYPT | HSE_ KF_ USAGE_ DECRYPT | HSE_ KF_ USAGE_ SIGN | HSE_ KF_
USAGE_ VERIFY | HSE_ KF_ USAGE_ EXCHANGE | \
```

```
HSE_KF_USAGE_DERIVE | HSE_ KF_ USAGE_ KEY_ PROVISION | HSE_ KF_ USAGE_ AUTHORIZATION | HSE_
KF_ USAGE_ SMR_ DECRYPT |      \
HSE_KF_USAGE_XTS_TWEAK | HSE_ KF_ USAGE_ OTFAD_ DECRYPT)
```

The Key Usage flags mask.

HSE_KF_MSC_USAGE_MASK

```
#define HSE_KF_MSC_USAGE_MASK  ( HSE_KF_USAGE_ENCRYPT| HSE_KF_USAGE_DECRYPT| HSE_
KF_USAGE_SIGN| HSE_KF_USAGE_VERIFY)
```

The Key Usage flags mask for Managed Security Component (MSC) targeted keys. The accepted key flags for keys that are configured in Key Handle Translation Table (KHTT) for MSC keystore. These key flags can be configured only with HSE_KF_USAGE_ENCRYPT, HSE_KF_USAGE_DECRYPT, HSE_KF_USAGE_SIGN or HSE_KF_USAGE_VERIFY.

HSE_KF_ACCESS_MASK

```
#define HSE_KF_ACCESS_MASK ( HSE_KF_ACCESS_WRITE_PROT| HSE_KF_ACCESS_DEBUG_PROT|
HSE_KF_ACCESS_EXPORTABLE)
```

The Key Access flags mask.

HSE_KF_MAX_KEY_COUNTER_VALUE

```
#define HSE_KF_MAX_KEY_COUNTER_VALUE ((uint32_t)0xFFFFFFFFUL - 1UL)
```

The maximum value of key counter. Note that 0xFFFFFFFF is reserved for RAM keys.

HSE_ROM_KEY_AES256_KEY0

```
#define HSE_ROM_KEY_AES256_KEY0 ((hseKeyHandle_t)0x00000000UL)
```

This key can be used for data encryption/decryption, having the following usage restrictions:

HSE ROM key handles. The ROM key catalog references keys that are provisioned by NXP and can be used by the host.

Note

- The ROM keys have the following access restriction flags set:
(# HSE_ KF_ ACCESS_ WRITE_ PROT | # HSE_ KF_ ACCESS_ DEBUG_ PROT)
- This key is a device-specific secret
- This key can be used to encrypt/decrypt application data with a device-specific key
(# HSE_ KF_ USAGE_ ENCRYPT | # HSE_ KF_ USAGE_ DECRYPT)

HSE_ROM_KEY_AES256_KEY1

```
#define HSE_ROM_KEY_AES256_KEY1 ((hseKeyHandle_t)0x00000001UL)
```

This key can be used for key derivation and key provisioning, having the following usage restrictions:

Note

- This key is a shared secret owned by NXP
- It can be used during key provision to import an application key encrypted with an NXP secret
- This NXP key can be used to encrypt a customer key using an email service provided by NXP. In this way, the customer key can be injected in HSE sub-system in a secure manner. Contact NXP support team for more details.
- The service is used in pair with another RSA key. The email service provides a signature which is verified using the RSA key.
(# HSE_ KF_ USAGE_ DERIVE | # HSE_ KF_ USAGE_ VERIFY | # HSE_ KF_ USAGE_ DECRYPT | # HSE_ KF_ USAGE_ KEY_ PROVISION)

HSE_ROM_KEY_RSA3072_PUB_KEY0

```
#define HSE_ROM_KEY_RSA3072_PUB_KEY0 ((hseKeyHandle_t)0x00000100UL)
```

This key can be used for RSA decryption and signature verify, having the following usage restrictions:

Note

- This key is a public RSA key owned by NXP; the corresponding private key is owned by NXP.
- It can be used during key provision to import an application key signed.
- This NXP key can be used to verify a signature on a customer key which is signed using an email service provided by NXP. In this way, the customer key can be injected in HSE sub-system in a secure manner. Contact NXP support team for more details.
- The service is used in pair with another ROM key i.e HSE_ROM_KEY_AES256_KEY1.
(# HSE_ KF_ USAGE_ DECRYPT | # HSE_ KF_ USAGE_ VERIFY | # HSE_ KF_ USAGE_ KEY_ PROVISION)

HSE_ROM_KEY_ECC256_PUB_KEY0

```
#define HSE_ROM_KEY_ECC256_PUB_KEY0 ((hseKeyHandle_t)0x00000200UL)
```

This key can be used for key provisioning having the following usage restrictions:

Note

- This key is a public ECC key owned by NXP; the corresponding private key owned by NXP.
- It can be used during key provision to import an application key signed using an NXP ECC public key.
- This NXP key can be used to sign a customer key using an email service provided by NXP. In this way, the customer key can be injected in HSE sub-system in a secure manner. Contact NXP for more details.
  (# HSE_ KF_ USAGE_ VERIFY | # HSE_ KF_ USAGE_ KEY_ PROVISION)

HSE_KF_SMR_0

```
#define HSE_KF_SMR_0 ((hseSmrFlags_t)1UL << 0UL)
```

HSE_KF_SMR_1

```
#define HSE_KF_SMR_1 ((hseSmrFlags_t)1UL << 1UL)
```

HSE_KF_SMR_2

```
#define HSE_KF_SMR_2 ((hseSmrFlags_t)1UL << 2UL)
```

HSE_KF_SMR_3

```
#define HSE_KF_SMR_3 ((hseSmrFlags_t)1UL << 3UL)
```

HSE_KF_SMR_4

```
#define HSE_KF_SMR_4 ((hseSmrFlags_t)1UL << 4UL)
```

HSE_KF_SMR_5

```
#define HSE_KF_SMR_5 ((hseSmrFlags_t)1UL << 5UL)
```

HSE_KF_SMR_6

```
#define HSE_KF_SMR_6 ((hseSmrFlags_t)1UL << 6UL)
```

HSE_KF_SMR_7

```
#define HSE_KF_SMR_7 ((hseSmrFlags_t)1UL << 7UL)
```

HSE_KF_SMR_8

```
#define HSE_KF_SMR_8 ((hseSmrFlags_t)1UL << 8UL)
```

HSE_KF_SMR_9

```
#define HSE_KF_SMR_9 ((hseSmrFlags_t)1UL << 9UL)
```

HSE_KF_SMR_10

```
#define HSE_KF_SMR_10 ((hseSmrFlags_t)1UL << 10UL)
```

HSE_KF_SMR_11

```
#define HSE_KF_SMR_11 ((hseSmrFlags_t)1UL << 11UL)
```

HSE_KF_SMR_12

```
#define HSE_KF_SMR_12 ((hseSmrFlags_t)1UL << 12UL)
```

HSE_KF_SMR_13

```
#define HSE_KF_SMR_13 ((hseSmrFlags_t)1UL << 13UL)
```

HSE_KF_SMR_14

```
#define HSE_KF_SMR_14 ((hseSmrFlags_t)1UL << 14UL)
```

HSE_KF_SMR_15

```
#define HSE_KF_SMR_15 ((hseSmrFlags_t)1UL << 15UL)
```

HSE_KF_SMR_16

```
#define HSE_KF_SMR_16 ((hseSmrFlags_t)1UL << 16UL)
```

HSE_KF_SMR_17

```
#define HSE_KF_SMR_17 ((hseSmrFlags_t)1UL << 17UL)
```

HSE_KF_SMR_18

```
#define HSE_KF_SMR_18 ((hseSmrFlags_t)1UL << 18UL)
```

HSE_KF_SMR_19

```
#define HSE_KF_SMR_19 ((hseSmrFlags_t)1UL << 19UL)
```

HSE_KF_SMR_20

```
#define HSE_KF_SMR_20 ((hseSmrFlags_t)1UL << 20UL)
```

HSE_KF_SMR_21

```
#define HSE_KF_SMR_21 ((hseSmrFlags_t)1UL << 21UL)
```

HSE_KF_SMR_22

```
#define HSE_KF_SMR_22 ((hseSmrFlags_t)1UL << 22UL)
```

HSE_KF_SMR_23

```
#define HSE_KF_SMR_23 ((hseSmrFlags_t)1UL << 23UL)
```

HSE_KF_SMR_24

```
#define HSE_KF_SMR_24 ((hseSmrFlags_t)1UL << 24UL)
```

HSE_KF_SMR_25

```
#define HSE_KF_SMR_25 ((hseSmrFlags_t)1UL << 25UL)
```

HSE_KF_SMR_26

```
#define HSE_KF_SMR_26 ((hseSmrFlags_t)1UL << 26UL)
```

HSE_KF_SMR_27

```
#define HSE_KF_SMR_27 ((hseSmrFlags_t)1UL << 27UL)
```

HSE_KF_SMR_28

```
#define HSE_KF_SMR_28 ((hseSmrFlags_t)1UL << 28UL)
```

HSE_KF_SMR_29

```
#define HSE_KF_SMR_29 ((hseSmrFlags_t)1UL << 29UL)
```

HSE_KF_SMR_30

```
#define HSE_KF_SMR_30 ((hseSmrFlags_t)1UL << 30UL)
```

HSE_KF_SMR_31

```
#define HSE_KF_SMR_31 ((hseSmrFlags_t)1UL << 31UL)
```

HSE_EC_CURVE_NONE

```
#define HSE_EC_CURVE_NONE ((hseEccCurveId_t)0U)
```

HSE_EC_SEC_SECP256R1

```
#define HSE_EC_SEC_SECP256R1 ((hseEccCurveId_t)1U)
```

HSE_EC_SEC_SECP384R1

```
#define HSE_EC_SEC_SECP384R1 ((hseEccCurveId_t)2U)
```

HSE_EC_SEC_SECP521R1

```
#define HSE_EC_SEC_SECP521R1 ((hseEccCurveId_t)3U)
```

HSE_EC_BRAINPOOL_BRAINPOOLP256R1

```
#define HSE_EC_BRAINPOOL_BRAINPOOLP256R1 ((hseEccCurveId_t)4U)
```

HSE_EC_BRAINPOOL_BRAINPOOLP320R1

```
#define HSE_EC_BRAINPOOL_BRAINPOOLP320R1 ((hseEccCurveId_t)5U)
```

HSE_EC_BRAINPOOL_BRAINPOOLP384R1

```
#define HSE_EC_BRAINPOOL_BRAINPOOLP384R1 ((hseEccCurveId_t)6U)
```

HSE_EC_BRAINPOOL_BRAINPOOLP512R1

```
#define HSE_EC_BRAINPOOL_BRAINPOOLP512R1 ((hseEccCurveId_t)7U)
```

HSE_EC_25519_ED25519

```
#define HSE_EC_25519_ED25519 ((hseEccCurveId_t)9U)
```

HSE_EC_25519_CURVE25519

```
#define HSE_EC_25519_CURVE25519 ((hseEccCurveId_t)10U)
```

HSE_EC_448_ED448

```
#define HSE_EC_448_ED448 ((hseEccCurveId_t)11U)
```

HSE_EC_448_CURVE448

```
#define HSE_EC_448_CURVE448 ((hseEccCurveId_t)12U)
```

HSE_EC_USER_CURVE1

```
#define HSE_EC_USER_CURVE1 ((hseEccCurveId_t)101U)
```

HSE_EC_USER_CURVE2

```
#define HSE_EC_USER_CURVE2 ((hseEccCurveId_t)102U)
```

HSE_EC_USER_CURVE3

```
#define HSE_EC_USER_CURVE3 ((hseEccCurveId_t)103U)
```

HSE_KEY_BITS_INVALID

```
#define HSE_KEY_BITS_INVALID ((hseKeyBits_t)0xFFFFU)
```

HSE_KEY_BITS_ZERO

```
#define HSE_KEY_BITS_ZERO ((hseKeyBits_t)0U)
```

HSE_KEY64_BITS

```
#define HSE_KEY64_BITS ((hseKeyBits_t)64U)
```

HSE_KEY128_BITS

```
#define HSE_KEY128_BITS ((hseKeyBits_t)128U)
```

HSE_KEY160_BITS

```
#define HSE_KEY160_BITS ((hseKeyBits_t)160U)
```

HSE_KEY192_BITS

```
#define HSE_KEY192_BITS ((hseKeyBits_t)192U)
```

HSE_KEY224_BITS

```
#define HSE_KEY224_BITS ((hseKeyBits_t)224U)
```

HSE_KEY240_BITS

```
#define HSE_KEY240_BITS ((hseKeyBits_t)240U)
```

HSE_KEY256_BITS

```
#define HSE_KEY256_BITS ((hseKeyBits_t)256U)
```

HSE_KEY320_BITS

```
#define HSE_KEY320_BITS ((hseKeyBits_t)320U)
```

HSE_KEY384_BITS

```
#define HSE_KEY384_BITS ((hseKeyBits_t)384U)
```

HSE_KEY512_BITS

```
#define HSE_KEY512_BITS ((hseKeyBits_t)512U)
```

HSE_KEY521_BITS

```
#define HSE_KEY521_BITS ((hseKeyBits_t)521U)
```

HSE_KEY638_BITS

```
#define HSE_KEY638_BITS ((hseKeyBits_t)638U)
```

HSE_KEY1024_BITS

```
#define HSE_KEY1024_BITS ((hseKeyBits_t)1024U)
```

HSE_KEY2048_BITS

```
#define HSE_KEY2048_BITS ((hseKeyBits_t)2048U)
```

HSE_KEY3072_BITS

```
#define HSE_KEY3072_BITS ((hseKeyBits_t)3072U)
```

HSE_KEY4096_BITS

```
#define HSE_KEY4096_BITS ((hseKeyBits_t)4096U)
```

HSE_KU_AES_BLOCK_MODE_ANY

```
#define HSE_KU_AES_BLOCK_MODE_ANY ((hseAesBlockModeMask_t)0U)
```

Any block mode below.

HSE_KU_AES_BLOCK_MODE_XTS

```
#define HSE_KU_AES_BLOCK_MODE_XTS ((hseAesBlockModeMask_t)(1U << 0U))
```

XTS mode (AES)

HSE_KU_AES_BLOCK_MODE_CTR

```
#define HSE_KU_AES_BLOCK_MODE_CTR ((hseAesBlockModeMask_t)(1U << HSE_CIPHER_BLOCK_
MODE_CTR))
```

CTR mode (AES)

HSE_KU_AES_BLOCK_MODE_CBC

```
#define HSE_KU_AES_BLOCK_MODE_CBC ((hseAesBlockModeMask_t)(1U << HSE_CIPHER_BLOCK_
MODE_CBC))
```

CBC mode (AES)

HSE_KU_AES_BLOCK_MODE_ECB

```
#define HSE_KU_AES_BLOCK_MODE_ECB ((hseAesBlockModeMask_t)(1U << HSE_CIPHER_BLOCK_
MODE_ECB))
```

ECB mode (AES)

HSE_KU_AES_BLOCK_MODE_CFB

```
#define HSE_KU_AES_BLOCK_MODE_CFB ((hseAesBlockModeMask_t)(1U << HSE_CIPHER_BLOCK_
MODE_CFB))
```

CFB mode (AES)

HSE_KU_AES_BLOCK_MODE_OFB

```
#define HSE_KU_AES_BLOCK_MODE_OFB ((hseAesBlockModeMask_t)(1U << HSE_CIPHER_BLOCK_MODE_OFB))
```

OFB mode (AES)

HSE_KU_AES_BLOCK_MODE_CCM

```
#define HSE_KU_AES_BLOCK_MODE_CCM ((hseAesBlockModeMask_t)(1U << 6U))
```

CCM mode (AES)

HSE_KU_AES_BLOCK_MODE_GCM

```
#define HSE_KU_AES_BLOCK_MODE_GCM ((hseAesBlockModeMask_t)(1U << 7U))
```

GCM mode (AES)

Typedef Documentation

hseKeyCatalogId_t

```
typedef uint8_t hseKeyCatalogId_t
```

HSE key catalog type.

A key catalog is a memory container that holds groups of keys. The catalog defines the type of storage (volatile / non-volatile) and the visibility to the application (host)

hseKeyGroupOwner_t

```
typedef uint8_t hseKeyGroupOwner_t
```

HSE Key Group owner.

hseKeyType_t

```
typedef uint8_t hseKeyType_t
```

HSE Key type. Specifies the Key type. It provides information about the interpretation of key data.

hseKeyFlags_t

```
typedef uint16_t hseKeyFlags_t
```

The key flags specifies the operations or restrictions that can be apply to a key.

hseSmrFlags_t

```
typedef uint32_t hseSmrFlags_t
```

The SMR flags.

A set of flags that define which secure memory region (SMR), shall be verified before the key can be used. For RAM keys, the SMR flags are forced to zero (not used).

hseEccCurveId_t

```
typedef uint8_t hseEccCurveId_t
```

The ECC curve IDs.

hseKeyBits_t

```
typedef uint16_t hseKeyBits_t
```

Some default key bits values.

The below values are only only a few possible values. Note that HSE supports key bit length different than those defined below (eg. TU Darmstadt curves 1 to 38).

hseAesBlockModeMask_t

```
typedef uint8_t hseAesBlockModeMask_t
```

Cipher modes flags for AES keys.

The values below are representing the cipher mode flags that an AES key can take.

5.2 HSE Key Management Utility Services

Data Structures

- struct hseLoadEccCurveSrv_t
- struct hseFormatKeyCatalogsSrv_t
- struct hseExtendKeyCatalogSrv_t
- struct hseEraseKeySrv_t
- struct hseGetKeyInfoSrv_t
- struct hseKeyVerifySrv_t

Macros

Type: (implicit C type)	
Name	**Value**
HSE_ERASE_NOT_USED	0U
HSE_ERASE_ALL_RAM_KEYS_ON_MU_IF	1U
HSE_ERASE_ALL_NVM_SYM_KEYS_ON_MU_IF	2U
HSE_ERASE_ALL_NVM_ASYM_KEYS_ON_MU_IF	3U
HSE_ERASE_ALL_NVM_KEYS_ON_MU_IF	4U
HSE_ERASE_KEYGROUP_ON_MU_IF	5U

Type: hseKeyVerAlgo_t	
Name	**Value**
HSE_KEY_VER_SHA256	HSE_HASH_ALGO_SHA2_256
HSE_KEY_VER_CMAC	HSE_MAC_ALGO_CMAC
HSE_KEY_VER_PUB_EXT_HASH	0xE5U

Typedefs

- typedef uint8_t hseEraseKeyOptions_t
- typedef uint8_t hseKeyVerAlgo_t

Data Structure Documentation

struct hseLoadEccCurveSrv_t

HSE Load ECC curve.

This service can be used to set the domain parameters for a Weierstrass ECC curve that is not supported by default. Twisted Edwards or Montgomery curve parameters cannot be loaded by this service.

Note

1. Loading a curve into the HSE modifies the SYS-IMAGE, making it necessary to publish it and store it in external flash on HSE_H/M devices.
2. The host needs super-user rights to update the NVM configuration, in order to use this service.

Data Fields

Type	Name	Description
hseEccCurveId_t	eccCurveId	INPUT: The ECC curve ID. Must be a user allocated curve ID (i.e. HSE_ECC_CURVEx).
uint8_t	reserved[3]	
hseKeyBits_t	pBitLen	INPUT: The bit length of the prime p.
hseKeyBits_t	nBitLen	INPUT: The bit length of the order n.
uint64_t	pA	INPUT: Elliptic curve parameter a. Must be represented as a big endian number, in the form of a byte array of length HSE_BITS_TO_BYTES (pBitLen), e.g. 256 bit curves need 32 byte arrays, 521 bit curves need 66 byte arrays.
uint64_t	pB	INPUT: Elliptic curve parameter b. Must be represented as a big endian number, in the form of a byte array of length HSE_BITS_TO_BYTES (pBitLen), e.g. 256 bit curves need 32 byte arrays, 521 bit curves need 66 byte arrays.
uint64_t	pP	INPUT: Elliptic curve prime p. Must be represented as a big endian number, in the form of a byte array of length HSE_BITS_TO_BYTES (pBitLen), e.g. 256 bit curves need 32 byte arrays, 521 bit curves need 66 byte arrays.
uint64_t	pN	INPUT: Elliptic curve order n. Must be represented as a big endian number, in the form of a byte array of length HSE_BITS_TO_BYTES (nBitLen), e.g. 256 bit curves need 32 byte arrays, 521 bit curves need 66 byte arrays.
uint64_t	pG	INPUT: Elliptic curve generator point. The x and y coordinates of the generator, represented as big endian numbers, each in the form of a byte array of length HSE_BITS_TO_BYTES (pBitLen), then concatenated. The HSE expects an array of size 2 ∗ HSE_BITS_TO_BYTES (pBitLen).

struct hseFormatKeyCatalogsSrv_t

HSE "Format Key Catalogs" service.

Used to configure the NVM or RAM key catalogs. The catalogs format should be define according to the total number of groups (HSE_TOTAL_NUM_OF_KEY_GROUPS). and the maximum available memory for NVM or RAM keys handled by the HSE Firmware (see HSE_MAX_NVM_STORE_SIZE and HSE_MAX_RAM_STORE_SIZE). If the catalog definition does not fit within the available memory, an error occurs and the key format fails. Each catalog should terminate with a zero filled entry.

The key catalogs (NVM and RAM) can only be formatted (or re-formatted) only if one of the following conditions is met:

- if the application has CUST_DEL SuperUser rights (see hseSysAuthorizationReqSrv_t).
- if HSE_STATUS_INSTALL_OK is cleared (there is no SYS-IMG installed). In this case, after formatting the key catalogs, the application will be granted with CUST and OEM SU rights (ANY).

Note

- Each catalog entry represent a key group of the same key type.
- Each group is identified by its index within the catalog.
- Each group has an owner (see hseKeyGroupOwner_t). NVM keys can be owned by CUST or OEM; RAM key owner is always HSE_KEY_OWNER_ANY .
- Note that a key group can contain keys that have keybitLen <= maxKeyBitLen. For example, the group of key type HSE_KEY_TYPE_AES of 256bits can contain AES128, AES192 and AES256 keys. If there are not enough slots for an AES128 key in an AES128 group, the key can be store in an AES256 slot.
- At least one group should be defined for each catalog (NVM or RAM).
- HSE_KEY_TYPE_SHARED_SECRET key group can only be used for RAM key catalog.
- HSE_KEY_TYPE_RSA_PAIR key group can only be used for NVM key catalog.
- A key group can belong to one or more MUs.
- Both NVM and RAM catalogs shall be set in the same manner.

Example of NVM key catalog configuration.

```
{
  {  HSE_MU0_MASK,  HSE_KEY_OWNER_CUST,  HSE_KEY_TYPE_AES,          20U,       HSE_
KEY128_BITS },
  {  HSE_MU0_MASK,  HSE_KEY_OWNER_CUST,  HSE_KEY_TYPE_ECC_PAIR,      2U,        HSE_
KEY256_BITS },
  {  HSE_MU1_MASK,  HSE_KEY_OWNER_OEM,   HSE_KEY_TYPE_AES,          20U,       HSE_
KEY256_BITS },
  {  HSE_MU1_MASK,  HSE_KEY_OWNER_OEM,   HSE_KEY_TYPE_HMAC,         10U,       HSE_
KEY512_BITS },
  {  HSE_MU1_MASK,  HSE_KEY_OWNER_OEM,   HSE_KEY_TYPE_ECC_PAIR,      2U,        HSE_
KEY256_BITS },
  {  HSE_MU1_MASK,  HSE_KEY_OWNER_OEM,   HSE_KEY_TYPE_ECC_PUB,       6U,        HSE_
KEY256_BITS },
  {  HSE_MU1_MASK,  HSE_KEY_OWNER_OEM,   HSE_KEY_TYPE_ECC_PUB_EXT,  10U,        HSE_
KEY256_BITS },
  {  0U,          0U,                0U,                       0U,        0U  }
}
```

SHE Key catalog configuration (see below configuration):

- NVM SHE keys shall be mapped on key group 0 in NVM key Catalog . Otherwise an error will be reported.
- In addition to the SHE keys KEY_1 to KEY_10 (key ID 0x4 to 0x0D), the HSE firmware allows the application to provision extra NVM SHE keys. These extended NVM SHE key groups must map to the key groups 1 to 4 in the NVM key catalogs, and shall contain 10 keys.
- Maximum 5 NVM SHE groups are allowed.

- RAM SHE key shall also be mapped on key group 0 in RAM key Catalog.
- The owner for SHE key group shall be set to HSE_KEY_OWNER_ANY .
- Any other non-SHE key group can be added after SHE key groups in NVM/RAM Key Catalogs.

NVM SHE Key Catalog Configuration:

- row0: MASTER_ECU_KEY, BOOT_MAC_KEY, KEY_1 to KEY_10
- row1: KEY_11 to KEY_20
- row2: KEY_21 to KEY_30
- row3: KEY_31 to KEY_40
- row4: KEY_41 to KEY_50

```
{
  { HSE_MU0_MASK, HSE_KEY_OWNER_ANY, HSE_KEY_TYPE_SHE, 12U , HSE_KEY128_BITS
},
  { HSE_MU0_MASK, HSE_KEY_OWNER_ANY, HSE_KEY_TYPE_SHE, 10U , HSE_KEY128_BITS
},
  { HSE_MU0_MASK, HSE_KEY_OWNER_ANY, HSE_KEY_TYPE_SHE, 10U , HSE_KEY128_BITS
},
  { HSE_MU0_MASK, HSE_KEY_OWNER_ANY, HSE_KEY_TYPE_SHE, 10U , HSE_KEY128_BITS
},
  { HSE_MU0_MASK, HSE_KEY_OWNER_ANY, HSE_KEY_TYPE_SHE, 10U , HSE_KEY128_BITS
},
  { 0U, 0U, 0U, 0U , 0U }
}
```

RAM SHE Key Catalog Configuration

```
{
  { HSE_MU0_MASK, HSE_KEY_OWNER_ANY, HSE_KEY_TYPE_SHE, 1U , HSE_KEY128_
    BITS },
  { 0U, 0U, 0U , 0U , 0U }
}
```

Data Fields

Type	Name	Description
uint64_t	pNvmKeyCatalogCfg	INPUT: Points to "NVM Key Catalog" table (table entries of type hseKeyGroupCfgEntry_t).
uint64_t	pRamKeyCatalogCfg	INPUT: Points to "RAM Key Catalog" table (table entries of type hseKeyGroupCfgEntry_t).

struct hseExtendKeyCatalogSrv_t

HSE "Extend Key Catalog" service.

Used to update the NVM or RAM key catalogs without reformatting and erasing the contents. The new key groups added via this services will be appended to the end of the groups already part of the targeted key catalog configuration. As a precondition, the key catalogs must be formatted before calling this service.

Data Fields

Type	Name	Description
hseKeyCatalogId_t	keyCatalogId	INPUT: The ID of the key catalog for which the group entry is appended. Can be only NVM or RAM.

Data Fields

Type	Name	Description
uint8_t	numOfKeyGroupEntries	INPUT: The number of key catalog entries that are appended to the targeted catalog.
uint8_t	reserved[2U]	Reserved bytes.
uint64_t	pKeyGroupEntries	INPUT: Points to the key catalog group entry/entries which are appended (see hseKeyGroupCfgEntry_t).

struct hseEraseKeySrv_t

HSE Erase key.

This service can be used to erase RAM or NVM keys. The erase service depends on HSE access right (see hseSysRights_t):

1. SuperUser rights (CUST or OEM):
   - NVM CUST keys can be erased only if the CUST SuperUser rights were granted (see hseSysAuthorizationReqSrv_t service)
   - NVM OEM keys can be erased only if the OEM SuperUser rights were granted (see hseSysAuthorizationReqSrv_t service)
   - RAM keys can be erased
2. User rights:
   - NVM keys can NOT be erased.
   - RAM keys can be erased.

Note

- The MU mask of the key group(s) must match the MU interface on which the erase request was sent.
- For NVM key erase, the MU interface on which the host was authorized as SuperUser must match the MU interface on which erase service request has been sent.
- SHE keys cannot be erased individually (as single slot or as single NVM group). When HSE_ERASE_ALL_NVM_SYM_KEYS_ON_MU_IF or HSE_ERASE_ALL_NVM_KEYS_ON_MU_IF options are used, the SHE keys would be erased only if system authorization was performed beforehand using MASTER_ECU key. Otherwise, the operation will be successful erasing other key types, but not SHE keys.

Data Fields

Type	Name	Description
hseKeyHandle_t	keyHandle	INPUT: The key handle. It is used if the erase option is HSE_ERASE_NOT_USED , specifying the one key to be erased or if the erase option is HSE_ERASE_KEYGROUP_ON_MU_IF , specifying the key catalog and group to be erased. Otherwise, it must be set to HSE_INVALID_KEY_HANDLE when used with the other erase options (HSE_ERASE_ALL_RAM_KEYS_ON_MU_IF , HSE_ERASE_ALL_NVM_SYM_KEYS_ON_MU_IF , HSE_ERASE_ALL_NVM_ASYM_KEYS_ON_MU_IF , HSE_ERASE_ALL_NVM_KEYS_ON_MU_IF). Note A single write-protected NVM key cannot be deleted. Write-protected NVM keys can be deleted when multiple keys are erased (using HSE_ERASE_ALL_NVM_SYM_KEYS_ON_MU_IF , HSE_ERASE_ALL_NVM_ASYM_KEYS_ON_MU_IF , HSE_ERASE_ALL_NVM_KEYS_ON_MU_IF or HSE_ERASE_KEYGROUP_ON_MU_IF options).
hseEraseKeyOptions_t	eraseKeyOptions	INPUT: The Erase key options (see hseEraseKeyOptions_t)
uint8_t	reserved[3]	

struct hseGetKeyInfoSrv_t

HSE Get Key Info service.

Return the key information (or properties) using the "key handle" as input parameter.

Data Fields

Type	Name	Description
hseKeyHandle_t	keyHandle	INPUT: The key handle.
uint64_t	pKeyInfo	OUTPUT: Address where to store hseKeyInfo_t (Specifies usage flags, restriction access, key bit length etc).

struct hseKeyVerifySrv_t

HSE Key Verify service.

This service is used to verify a CMAC or SHA256 over a key stored inside HSE. The CMAC and SHA256 tag are provided by the application.

Data Fields

Type	Name	Description
hseKeyHandle_t	keyHandle	INPUT: The key handle of the key that needs to be verified. The verification is performed on the following key formats/types: – symmetric key (HSE_KEY_TYPE_AES, HSE_KEY_TYPE_HMAC, HSE_KEY_TYPE_SHARED_SECRET, HSE_KEY_TYPE_SIPHASH): array of HSE_BITS_TO_BYTES(keyBitLength) size – HSE_KEY_TYPE_ECC_PUB: · Weierstrass curve keys: x-coordinate || y-coordinate (all in big endian); the length must be 2 ∗ HSE_BITS_TO_BYTES(keyBitLength) · Twisted Edwards curve keys: point Y (with the sign bit of X), in big endian; the length must be HSE_BITS_TO_BYTES(keyBitLength) · Montgomery curve keys: the x-coordinate, in big endian; the length must be HSE_BITS_TO_BYTES(keyBitLength) – HSE_KEY_TYPE_ECC_PAIR: · Weierstrass curve keys: x-coordinate || y-coordinate || scalar (all in big endian); the length must be 3 ∗ HSE_BITS_TO_BYTES(keyBitLength) · Twisted Edwards curve keys: point y (with the sign bit of X) || scalar, in big endian; the length must be 2 ∗ HSE_BITS_TO_BYTES(keyBitLength) · Montgomery curve keys: the X coordinate||scalar, in big endian; the length must be 2 ∗ HSE_BITS_TO_BYTES(keyBitLength) – HSE_KEY_TYPE_RSA_PUB / HSE_KEY_TYPE_DH_PUB : modulus || public exponent (all in big endian) – HSE_KEY_TYPE_RSA_PAIR / HSE_KEY_TYPE_DH_PAIR: modulus || public exponent || private exponent (all in big endian) – HSE_KEY_TYPE_ECC_PUB_EXT/ HSE_KEY_TYPE_RSA_PUB_EXT: verify the internal hash over the key container (e.g. certificate)

Data Fields

Type	Name	Description
hseKeyHandle_t	cmackeyHandle	INPUT: The key handle used for CMAC operation. For HSE_KEY_VER_SHA256 selected algorithms, this parameter is ignored.
hseKeyVerAlgo_t	keyVerAlgo	INPUT: Key verification algorithm. It can be HSE_KEY_VER_PUB_EXT_HASH, HSE_KEY_VER_CMAC, HSE_KEY_VER_SHA256 (see hseKeyVerAlgo_t) Note: If this parameter is set to HSE_KEY_VER_PUB_EXT_HASH, the key slot corresponding to keyHandle must be a PUB_EXT key slot.
uint8_t	tagLen	INPUT: The provided tag length. It can be: – a CMAC tag; the length must be between 8 - 16 bytes – a SHA256 hash; the length must be between 8 - 32 bytes Note: If keyVerAlgo == HSE_KEY_VER_PUB_EXT_HASH, this parameter is ignored (the internal hash is used instead).
uint8_t	reserved[2U]	Reserved bytes.
uint64_t	pTag	INPUT: Address where tag is stored (CMAC tag, SHA256 hash). Note: If keyVerAlgo == HSE_KEY_VER_PUB_EXT_HASH, this parameter is ignored (the internal hash is used instead).

Macro Definition Documentation

HSE_ERASE_NOT_USED

```
#define HSE_ERASE_NOT_USED (0U)
```

Erase key options not used.

HSE_ERASE_ALL_RAM_KEYS_ON_MU_IF

```
#define HSE_ERASE_ALL_RAM_KEYS_ON_MU_IF (1U)
```

Erase all RAM keys assigned to MU Interface on which the erase service is sent.

HSE_ERASE_ALL_NVM_SYM_KEYS_ON_MU_IF

```
#define HSE_ERASE_ALL_NVM_SYM_KEYS_ON_MU_IF (2U)
```

Erase all NVM symmetric keys assigned to MU Interface on which the erase service is sent (needs CUST/OEM SuperUser rights).

HSE_ERASE_ALL_NVM_ASYM_KEYS_ON_MU_IF

```
#define HSE_ERASE_ALL_NVM_ASYM_KEYS_ON_MU_IF (3U)
```

Erase all NVM asymmetric keys assigned to MU Interface on which the erase service is sent (needs CUST/OEM SuperUser rights).

HSE_ERASE_ALL_NVM_KEYS_ON_MU_IF

```
#define HSE_ERASE_ALL_NVM_KEYS_ON_MU_IF (4U)
```

Erase all NVM KEYS assigned to MU Interface on which the erase service is sent (needs CUST/OEM SuperUser rights).

HSE_ERASE_KEYGROUP_ON_MU_IF

```
#define HSE_ERASE_KEYGROUP_ON_MU_IF (5U)
```

Erase all keys assigned to the key group referenced in the key handle. The MU Interface on which the erase service is sent to must be part of the group mask. CUST/OEM SuperUser rights with KM privileges are needed to perform this operation. In case the key group as an owner (CUST/OEM) the SU rights must be provided for this owner.

HSE_KEY_VER_SHA256

```
#define HSE_KEY_VER_SHA256 ((hseKeyVerAlgo_t) HSE_HASH_ALGO_SHA2_256)
```

SHA256.

HSE_KEY_VER_CMAC

```
#define HSE_KEY_VER_CMAC ((hseKeyVerAlgo_t) HSE_MAC_ALGO_CMAC)
```

CMAC (AES)

HSE_KEY_VER_PUB_EXT_HASH

```
#define HSE_KEY_VER_PUB_EXT_HASH ((hseKeyVerAlgo_t)0xE5U)
```

Verify the internal hash of a PUB_EXT key (e.g. external stored certificate)

Typedef Documentation

hseEraseKeyOptions_t

```
typedef uint8_t hseEraseKeyOptions_t
```

Options to erase keys.

The erase key options are used only if the provided key handle is set to HSE_INVALID_KEY_HANDLE .

hseKeyVerAlgo_t

```
typedef uint8_t hseKeyVerAlgo_t
```

The algorithm used for key verification .

5.3 HSE Key Import/Export Services

Data Structures

- union hseKeyFormat_t
- struct hseImportKeySrv_t
- struct hseExportKeySrv_t
- struct hseImportKeySrv_t.cipher
- struct hseImportKeySrv_t.keyContainer
- struct hseExportKeySrv_t.cipher
- struct hseExportKeySrv_t.keyContainer

Macros

Type: hseEccKeyFormat_t

Name	Value
HSE_KEY_FORMAT_ECC_PUB_RAW	0U
HSE_KEY_FORMAT_ECC_PUB_UNCOMPRESSED	1U
HSE_KEY_FORMAT_ECC_PUB_COMPRESSED	2U

Typedefs

- typedef uint8_t hseEccKeyFormat_t

Data Structure Documentation

union hseKeyFormat_t

HSE key format.

Includes additional information about the format of the key. Currently only used for ECC keys.

Data Fields

Type	Name	Description
hseEccKeyFormat_t	eccKeyFormat	INPUT: ECC key format.
uint8_t	reserved[4]	

struct hseImportKeySrv_t

HSE Import Key Service.

This service can be used to import a key in an empty slot or to update an existing key.

1. Common key restrictions (which apply for both SuperUser and User rights):
   - Key flags (of key properties) are always applied.
   - The NVM provisioning keys can be installed/updated without authentication only having SuperUser rights; they can also be updated having User rights using the pre-installed provision keys.
   - The RAM provision keys can be imported only authenticated and can be used only to import RAM keys.
   - A key can be authenticated signing the key container (e.g. X.509 certificate or any container). The HOST shall provide a pointer to that key container, pointer(s) to key value(s) within the key container and pointer(s) to the tag/signature(s) (computed over the key container).

   - To import an encrypted/authenticated NVM key, the provided provision key(s) must have the same group owner as the imported NVM key.
   - To import an encrypted/authenticated NVM symmetric key using AEAD, the pointer to key info must be in the additional data
   - The key properties (keyInfo) along with the public key values are always imported in plain format.
2. Restrictions for SuperUser rights:
   - NVM keys:
     - In empty slots, an encrypted key can be imported only authenticated, and a plain key can be imported with/without authentication (public keys must be imported in plain).
     - In non-empty slots, NVM keys can be imported(overwritten) in plain/encrypted, only authenticated.
   - RAM keys:
     - An encrypted key can be imported only authenticated. A plain key can be imported with/without authentication. Exception: RAM provision keys can be imported only authenticated.
3. Restrictions for User rights:
   - NVM keys:
     - NVM secrets (symmetric keys and key pairs) can be imported only encrypted and authenticated. For key pair, private value must be encrypted and public value(s) unencrypted. NVM secrets imported from a signed key container MUST include the key properties (keyInfo) in the container (the provided key counter must be bigger than the previous one).
     - NVM public keys can be imported in plain, only authenticated. NVM public key imported from a signed key container can/cannot include the keyInfo in the container.
   - RAM keys:
     - An encrypted key can be imported only authenticated. A plain key can be imported with/without authentication.
     - key pairs can be imported only authenticated; private value encrypted and public value(s) unencrypted
     - public keys can be imported in plain, only authenticated.

Note

- The key catalogs must have been formatted prior to provisioning the keys.
- When AEAD is used to import a key, the container cannot be used.
- The key types *_PUB_EXT are stored in plain in the application NVM. For these key types, HSE stores only the key properties and the pointers to the public key values, as well as an authentication tag calculated over the key container: the authentication tag is verified by the HSE firmware whenever the related key is used by the host.
- For HSE_H/M devices, the SYS-IMAGE does not have to be written to application NVM after each key import operation; the SYS-IMAGE update process can be done at the end of the configuration process.

Data Fields

Type	Name	Description
hseKeyHandle_t	targetKeyHandle	INPUT: Specifies the slot where to add or updated a key. Note that the keyHandle identifies the key catalog, key group index and key slot index.
uint64_t	pKeyInfo	INPUT: Specifies usage flags, restriction access, key length in bits, etc for the key (see hseKeyInfo_t). Note – Only keys that are not write protected can be updated with this service. – NVM keys are secured against replay attacks by including a counter value stored within HSE. The anti-replay attack counter included in the key info header should be greater than the counter of the HSE key that will be updated (in case of key update). This mean that keyInfo MUST be included in the signed key container (when the Life Cycle is IN_FIELD). – For RAM keys the key counter is ignored (keyInfo may not be in the key container).

Data Fields

Type	Name	Description
uint64_t	pKey[3]	INPUT: Pointer to key values. A RSA private key should always be imported together with the public key. An ECC private key can be imported standalone if the public key (pKey[0]) is NULL and the public key length (keyLen[0]) is zero. The public key will be computed internally from the private key. – pKey[0]: · RSA public modulus n (big-endian). · ECC depends on the key format · Weierstrass curve keys: · raw format: X || Y, in big endian; keyLen[0] must be 2 ∗ HSE_BITS_TO_BYTES(keyBitLength) · uncompressed format: 0x04 || X || Y, in big endian; keyLen[0] must be 1 + 2 ∗ HSE_BITS_TO_BYTES(keyBitLength) · compressed format: 0x02 / x03 || X, in big endian; keyLen[0] must be 1 + HSE_BITS_TO_BYTES(keyBitLength) · Twisted Edwards curve keys: · raw format: point Y with the sign bit of X, in big endian; keyLen[0] must be HSE_BITS_TO_BYTES(keyBitLength) · Montgomery curve keys: · raw format: the X coordinate, in big endian; keyLen[0] must be HSE_BITS_TO_BYTES(keyBitLength) · Classic DH prime modulus p – pKey[1]: · RSA public exponent e (big-endian). · Classic DH public key – pKey[2]: · RSA private exponent d (big-endian). · ECC/ED25519 private scalar (big-endian). · The symmetric key (e.g AES, HMAC). · Classic DH private key
uint16_t	keyLen[3]	INPUT: The length in bytes for the above key values in the same order. Note that keyInfo.keyBitLen specifies the key length in bits.
uint8_t	reserved[2]	

Data Fields

Type	Name	Description
struct hseImportKeySrv_t.cipher	cipher	INPUT: Cipher parameters are used only if the cipherKeyHandle is not HSE_INVALID_KEY_HANDLE. Note – For AES-block cipher, if the keyBitLen is not multiple of AES block size (128bits), the key value have to be padded with zeros. – For RSAES NO PADDING, the keyBitLen of the imported key must be less than or equal to HSE_BITS_TO_BYTES(cipherKey_keyBitLen), and the key is considered a big-endian integer. – For RSAES-PKCS1-v1_5, the keyBitLen of the imported key shall not be greater than HSE_BITS_TO_BYTES(cipherKey_keyBitLen) -11 bytes. – For RSAES-OAEP, the keyBitLen of the imported key shall not be greater than HSE_BITS_TO_BYTES(cipherKey_keyBitLen) - 2 ∗ hashLen - 2 bytes.
struct hseImportKeySrv_t.keyContainer	keyContainer	INPUT: The keyContainer parameters should be used if the key comes in a signed key container: pointers to key values within the key container should be provided. The signature/tag is assumed to be done over the key container. Note – For NVM keys having User rights, the keyInfo MUST be included in the key container. – If the HOST is authorized (SU rights), the ∗_PUB_EXT key type can be imported from an unauthenticated key container (providing the key container without the signature).
hseKeyFormat_t	keyFormat	INPUT: Additional information about the format of the key. Key type specific.

struct hseExportKeySrv_t

HSE Export Key Service.

The key values and the key properties (optional) can be exported to the host via a key export service.

1. Common key restrictions (which apply for both SuperUser and User rights):
   - Key flags (of key properties) are always applied; this service can only be used if the key is exportable.
   - Provision/Authorization keys are NOT exportable (HSE_ KF_ ACCESS_ EXPORTABLE flag is ignored).
   - NVM keys can not be exported using RAM provision keys.
   - NVM/RAM symmetric keys can be exported only encrypted with/without authentication.
   - NVM/RAM public keys (from key pair or public key slots) can be exported in plain; keys may/may not be authenticated.
   - The private part of a key pair can NOT be exported (the private part is never disclosed to the host).
   - _PUB_EXT can NOT be exported.
   - To export an encrypted/authenticated NVM key, the provided provision key must have the same group owner as the exported NVM key (not applicable for RAM keys).
   - When AEAD is used to export a key, the container cannot be used.

Data Fields

Type	Name	Description
hseKeyHandle_t	targetKeyHandle	INPUT: The key handle to be exported. Note that the keyHandle identifies the key catalog, key group index and key slot index.
uint64_t	pKeyInfo	OUTPUT: Export the key information (see hseKeyInfo_t). Note – For symmetric keys exported in an authenticated key container, key information MUST be part of the key container; – For symmetric keys exported authenticated with AEAD, key information MUST be part of AAD (see hseAeadScheme_t); – For public keys this parameter is optional. It can be NULL.

Data Fields

Type	Name	Description
uint64_t	pKey[3]	OUTPUT: Addresses where to fill to key values. – pKey[0]: · RSA public modulus n. · ECC depends on the key format · Weierstrass curve keys: · raw format: X || Y, in big endian; the HSE will output 2 ∗ HSE_BITS_TO_BYTES(keyBitLength) bytes · uncompressed format: 0x04 || X || Y, in big endian; the HSE will output 1 + 2 ∗ HSE_BITS_TO_BYTES(keyBitLength) bytes · compressed format: 0x02 / x03 || X, in big endian; the HSE will output 1 + HSE_BITS_TO_BYTES(keyBitLength) bytes · Twisted Edwards curve keys: · raw format: point Y with the sign bit of X, in big endian; the HSE will output HSE_BITS_TO_BYTES(keyBitLength) bytes · Montgomery curve keys: · raw format: the X coordinate, in big endian; the HSE will output HSE_BITS_TO_BYTES(keyBitLength) bytes · Classic DH prime modulus p – pKey[1]: · RSA public exponent e. · Classic DH public key – pKey[2]: · The symmetric key (e.g AES, HMAC). · Classic DH private key
uint64_t	pKeyLen[3]	INPUT/OUTPUT: Addressed of uint16_t values of the length (in bytes) for the above buffers (INPUT). As output, it provides the lengths of the encrypted or unencrypted (only for public) keys. Note that the length in bits of the key is specified by hseKeyInfo_t.

Data Fields

Type	Name	Description
struct hseExportKeySrv_t.cipher	cipher	INPUT: Cipher parameters. Note – Only the private keys are encrypted and the encrypted value length is specified by the corresponding private key length (in bytes). – For AES-block cipher, if the keyBitLen of the exported is not multiple of AES block size (128bits), the key value will be padded with zeros. – For RSAES NO PADDING, the keyBitLen of the exported key must be less than or equal to HSE_BITS_TO_BYTES(cipherKey_keyBitLen), and the key is considered a big-endian integer. – For RSAES-PKCS1-v1_5, the keyBitLen of the exported key shall not be greater than HSE_BITS_TO_BYTES(cipherKey_keyBitLen) -11 bytes. – For RSAES-OAEP, the keyBitLen of the exported key shall not be greater than HSE_BITS_TO_BYTES(cipherKey_keyBitLen) - 2 ∗ hashLen - 2 bytes.
struct hseExportKeySrv_t.keyContainer	keyContainer	INPUT: The keyContainer parameters should be used when the key have to be exported in a key container that will be authenticated: pointers to where key values will be exported should be provided within the key container. Optionally, the pKeyInfo may point inside the key container. The signature/tag is done over the key container.
hseKeyFormat_t	keyFormat	INPUT: Additional information about the format of the key. Key type specific.

struct hseImportKeySrv_t.cipher

INPUT: Cipher parameters are used only if the cipherKeyHandle is not HSE_INVALID_KEY_HANDLE .

Note

– For AES-block cipher, if the keyBitLen is not multiple of AES block size (128bits), the key value have to be padded with zeros.

- For RSAES NO PADDING, the keyBitLen of the imported key must be less than or equal to HSE_BITS_TO_BYTES(cipherKey_keyBitLen), and the key is considered a big-endian integer.
- For RSAES-PKCS1-v1_5, the keyBitLen of the imported key shall not be greater than HSE_BITS_TO_BYTES(cipherKey_keyBitLen) -11 bytes.
- For RSAES-OAEP, the keyBitLen of the imported key shall not be greater than HSE_BITS_TO_BYTES(cipherKey_keyBitLen) - 2 ∗ hashLen - 2 bytes.

Data Fields

Type	Name	Description
hseKeyHandle_t	cipherKeyHandle	INPUT: Decryption key handle. The cipherKeyHandle can only be a provisioning key (HSE_KF_USAGE_KEY_PROVISION and HSE_KF_USAGE_DECRYPT flags are set). Note that the key handle identifies the cipher scheme below. In case of symmetric cipher scheme and authenticated encryption scheme(AEAD) the differentiation is made using the first byte of cipherScheme. Must be set to HSE_INVALID_KEY_HANDLE if not used.
hseCipherScheme_t	cipherScheme	Symmetric, asymmetric and AEAD cipher scheme. Note – Only the private keys are encrypted.

struct hseImportKeySrv_t.keyContainer

INPUT: The keyContainer parameters should be used if the key comes in a signed key container: pointers to key values within the key container should be provided. The signature/tag is assumed to be done over the key container.

Note

- For NVM keys having User rights, the keyInfo MUST be included in the key container.
- If the HOST is authorized (SU rights), the ∗_PUB_EXT key type can be imported from an unauthenticated key container (providing the key container without the signature).

Data Fields

Type	Name	Description
uint16_t	keyContainerLen	INPUT: The container length. Note The container includes only the signed block (without the signature).
uint8_t	reserved[2]	
uint64_t	pKeyContainer	INPUT: Address of the key container; includes the key value(s) and other information used to authenticate the key. (e.g. TBSCertificate for a X.509 certificate).
hseKeyHandle_t	authKeyHandle	INPUT: Authentication key handle (HSE_KF_USAGE_KEY_PROVISION and HSE_KF_USAGE_VERIFY flags are set). Must be set to HSE_INVALID_KEY_HANDLE if not used. An encrypted key can be imported only authenticated.
hseAuthScheme_t	authScheme	INPUT: Authentication scheme. Note that the key handle identifies the authentication scheme below. Note For the GMAC scheme, the minimum IV length is 12 bytes.
uint16_t	authLen[2]	INPUT: Byte length(s) of the authentication tag(s). Note – For MAC and RSA signature, only authLen[0] is used. – Both lengths are used for (R,S) (ECC or ED25519). – The MAC tag size must be minimum 16 bytes. – RSA signature size must be HSE_BITS_TO_BYTES(keyBitLength); – R or S size for ECDSA/EdDSA signature must be HSE_BITS_TO_BYTES(keyBitLength)
uint64_t	pAuth[2]	INPUT: Address(es) to authentication tag. Note – For MAC and RSA signature, only pAuth[0] is used. – Both pointers are used for (R,S) (ECC or ED25519).

struct hseExportKeySrv_t.cipher

INPUT: Cipher parameters.

Note

- Only the private keys are encrypted and the encrypted value length is specified by the corresponding private key length (in bytes).
- For AES-block cipher, if the keyBitLen of the exported is not multiple of AES block size (128bits), the key value will be padded with zeros.
- For RSAES NO PADDING, the keyBitLen of the exported key must be less than or equal to HSE_BITS_TO_BYTES(cipherKey_keyBitLen), and the key is considered a big-endian integer.
- For RSAES-PKCS1-v1_5, the keyBitLen of the exported key shall not be greater than HSE_BITS_TO_BYTES(cipherKey_keyBitLen) -11 bytes.
- For RSAES-OAEP, the keyBitLen of the exported key shall not be greater than HSE_BITS_TO_BYTES(cipherKey_keyBitLen) - 2 ∗ hashLen - 2 bytes.

Data Fields

Type	Name	Description
hseKeyHandle_t	cipherKeyHandle	INPUT: Encryption key handle. The cipherKeyHandle can only be a provisioning key (HSE_KF_USAGE_KEY_PROVISION and HSE_KF_USAGE_ENCRYPT flags are set). Note that the key handle will identifies the cipher scheme below. Must be set to HSE_INVALID_KEY_HANDLE if not used.
hseCipherScheme_t	cipherScheme	Symmetric, asymmetric and AEAD cipher scheme. Note – Only the private keys are encrypted.

struct hseExportKeySrv_t.keyContainer

INPUT: The keyContainer parameters should be used when the key have to be exported in a key container that will be authenticated: pointers to where key values will be exported should be provided within the key container. Optionally, the pKeyInfo may point inside the key container. The signature/tag is done over the key container.

Data Fields

Type	Name	Description
uint16_t	keyContainerLen	INPUT: The container length. Note The key container length is the size of the byte block to be signed (without the signature).
uint8_t	reserved[2]	
uint64_t	pKeyContainer	INPUT: Address of the key container; includes the key value(s) and other information used to authenticate the key. (e.g. TBSCertificate for a X.509 certificate).
hseKeyHandle_t	authKeyHandle	INPUT: Authentication key handle (HSE_KF_USAGE_KEY_PROVISION and HSE_KF_USAGE_SIGN flags are set). Note that the key handle identifies the authentication scheme below. Must be set to HSE_INVALID_KEY_HANDLE if not used.
hseAuthScheme_t	authScheme	INPUT: Authentication scheme. Note For the GMAC scheme, the minimum IV length is 12 bytes.
uint64_t	pAuthLen[2]	OUTPUT: Address(es) for the length(s) (uin16_t values) of the authentication tag. Note – For MAC and RSA signature, only pAuthLen[0] is used. – Both lengths are used for (R,S) (ECC or ED25519).
uint64_t	pAuth[2]	OUTPUT: Address of authentication tag. Note – For MAC and RSA signature, only pAuth[0] is used. – Both pointers are used for (R,S) (ECC or ED25519).

Macro Definition Documentation

HSE_KEY_FORMAT_ECC_PUB_RAW

```
#define HSE_KEY_FORMAT_ECC_PUB_RAW ((hseEccKeyFormat_t)0U)
```

Raw ECC public key: X || Y.

HSE_KEY_FORMAT_ECC_PUB_UNCOMPRESSED

```
#define HSE_KEY_FORMAT_ECC_PUB_UNCOMPRESSED ((hseEccKeyFormat_t)1U)
```

Standard ECC uncompressed public key: 0x04 || X || Y.

HSE_KEY_FORMAT_ECC_PUB_COMPRESSED

```
#define HSE_KEY_FORMAT_ECC_PUB_COMPRESSED ((hseEccKeyFormat_t)2U)
```

Standard ECC compressed public key: 0x02/0x03 || X.

Typedef Documentation

hseEccKeyFormat_t

```
typedef uint8_t hseEccKeyFormat_t
```

HSE ECC key format.

Additional info for Ecc key format for import and export For Weierstrass curve public keys:
- the raw format is the X coordinate concatenated with the Y coordinate (X || Y), in big endian
- the uncompressed format is a byte of 0x04, concatenated with the X coordinate and Y coordinates (0x04 || X || Y)
- the compressed format is a byte of 0x02 or 0x03, depending on the (lsb) of Y, concatenated with the X coordinate
  - (0x02 || X) if the lsb of Y is 0
  - (0x03 || X) if the lsb of Y is 1 For Twisted Edwards curve public keys:
- the raw format is the standard compressed format (point Y with the the sign bit of X), but in big endian For Montgomery curve public keys:
- the raw format is the X coordinate, in big endian

5.4 HSE Key Generate service

Data Structures

- struct hseKeyGenRsaScheme_t
- struct hseKeyGenEccScheme_t
- struct hseKeyGenTls12RsaPreMaster_t
- struct hseKeyGenerateSrv_t
- struct hseDHComputeSharedSecretSrv_t
- union hseKeyGenerateSrv_t.sch

Macros

Type: (implicit C type)	
Name	**Value**
HSE_KEY_GEN_SYM_RANDOM_KEY	1U
HSE_KEY_GEN_RSA_KEY_PAIR	2U
HSE_KEY_GEN_ECC_KEY_PAIR	3U
HSE_TLS12_RSA_PRE_MASTER_SECRET_GEN	5U

Typedefs

- typedef uint8_t hseKeyGenScheme_t

Data Structure Documentation

struct hseKeyGenRsaScheme_t

RSA key generate scheme.

It generates a RSA key pair. Note that the public modulus can be exported to HOST via this service or using the export key service.

Data Fields

Type	Name	Description
uint32_t	pubExpLength	INPUT: The length of public exponent "e". Should not be more than 16 bytes.
uint64_t	pPubExp	INPUT: The public exponent "e".
uint64_t	pModulus	OUTPUT: The public modulus n. It can be NULL (the modulus is not provided using this service). The size of this memory area must be at least the byte length of the public modulus.

struct hseKeyGenEccScheme_t

ECC Key Generate scheme.

It generates a ECC key pair.

Note

- the curve ID is specified by the keyInfo.specific.eccCurveId parameter.
- Note that the public key can be exported to HOST via this service or using the export key service.

Data Fields

Type	Name	Description
uint64_t	pPubKey	OUTPUT: Where to store the public key. If the public key is not needed at this point, pass a NULL pointer. The x and y coordinate of the public key will be passed concatenated one after another, as big-endian strings. The size of the buffer must be double the byte length of the prime n.

struct hseKeyGenTls12RsaPreMaster_t

Generate the pre-master secret for TLS 1.2 RSA key exchange.

It computes the pre-master secret for TLS 1.2 RSA key exchange as specified by rfc5246(TLS 1.2):
- The hseKeyGenerateSrv_t::targetKeyHandle must be a HSE_KEY_TYPE_SHARED_SECRET key slot.
- The hseKeyGenerateSrv_t::keyInfo must have the following key flags set: HSE_KF_USAGE_DERIVE, HSE_KF_ACCESS_EXPORTABLE.
- The rfc5246 specification is used:
  - keyInfo::keyBitLen must be 384bits (48bytes)
  - The pre-master secret is computed as ProtocolVersion (2bytes) concatenated with 46 byte random number. The ProtocolVersion = {3,3} for TLS 1.2.
- To encrypt the generated pre-master secret, the hseExportKeySrv_t service with (the proper RSA scheme) must be used. The encrypted pre-master secret is sent to the peer node.
- To decrypt an encrypted pre-master secret, the hseImportKeySrv_t service (with the proper RSA scheme) must be used. The destination key slot can be a HSE_KEY_TYPE_SHARED_SECRET (with HSE_KF_USAGE_DERIVE key flag set) that further can be used to derive the TLS 1.2 key_block.
- To generate the master secret the hseKdfTLS12PrfScheme_t service must be used.

Note

- This service can also be used to perform the RSA_PSK key exchange as specified by rfc4279. In the same manner as explained above, it can be used to generate the input needed for RSA encryption (see EncryptedPreMasterSecret). The EncryptedPreMasterSecret can be generated using the hseExportKeySrv_t service (on the client side), and imported using the the hseImportKeySrv_t service (on the server side). In this case, to generate the master secret the hseKdfTLS12PrfScheme_t service must be executed using the tlsPskUsage = HSE_TLS_KEY_EXCHANGE_RSA_PSK option.

Data Fields

Type	Name	Description
uint8_t	protocolVersion[2U]	INPUT: The TLS or DTLS version. E.g. for TLS1.2 must be {3, 3}; for DTLS1.2 must be { 254, 253 }. Note HSE does not check the provided values;it just concatenates the protocol version with 46 byte random number.
uint8_t	reserved[2U]	Reserved for future use.

struct hseKeyGenerateSrv_t

HSE Key generate service.

It can be used to generate a key pair (e.g. public and private RSA, ECC, classic DH) or a random symmetric key.

Note

- Key flags (of key properties) are always applied.
- The keys can be generated as follow:
  1. Restrictions for SuperUser rights:
     - NVM keys can only be generated in empty slots (an erase shall be performed in advance)
     - RAM keys can always be generated (RAM keys can be overwritten)
  2. Restrictions for User rights:
     - NVM keys can NOT be generated.
     - RAM keys can always be generated (RAM keys can be overwritten)

Data Fields

Type	Name	Description
hseKeyHandle_t	targetKeyHandle	INPUT: The target key handle (where to store the new key).

Data Fields

Type	Name	Description
hseKeyInfo_t	keyInfo	INPUT: Specifies usage flags, restriction access, key bit length etc for the key. Note – For random symmetric key, the key length in bits should be specified by keyBitLen. – For RSA, keyBitLen specifies the bit length of the public modulus which shall be generated. – For ECC, the keyInfo should specify the ECC curve ID and the length of the base point order. – For RSA TLS 1.2 pre-master secret, see the hseKeyGenTls12RsaPreMaster_t notes. – For classic DH, the keyBitLen specifies the bit length of the public modulus.
hseKeyGenScheme_t	keyGenScheme	INPUT: Specifies the key generation scheme (e.g random sym key, rsa key pair, ecc key pair, RSA TLS 1.2 pre-master secret, classic-DH key pair).
uint8_t	reserved[3]	
union hseKeyGenerateSrv_t.sch	sch	INPUT: The selected scheme parameters.

struct hseDHComputeSharedSecretSrv_t

DH Compute Shared Secret service.

Computes the Diffie-Hellman share secret for ECC or classic DH (e.g. the key exchange protocol). The share secret can only be computed in a shared secret slot, and can not be exported.

Data Fields

Type	Name	Description
hseKeyHandle_t	targetKeyHandle	INPUT: The target key handle (where to store the shared secret). It must specify a HSE_KEY_TYPE_SHARED_SECRET key slot.
hseKeyHandle_t	privKeyHandle	INPUT: The private key.

Data Fields

Type	Name	Description
hseKeyHandle_t	peerPubKeyHandle	INPUT: The peer public key. Must be previously imported into the HSE. Note that the peer public key can also be imported as a ∗_PUB_EXT key type (external public key stored on the application NVM)

union hseKeyGenerateSrv_t.sch

INPUT: The selected scheme parameters.

Data Fields

Type	Name	Description
hseNoScheme_t	symKey	INPUT: No scheme (parameter) is used for random symmetric key.
hseKeyGenRsaScheme_t	rsaKey	INPUT: The scheme used to generate a RSA key pair.
hseKeyGenEccScheme_t	eccKey	INPUT: The scheme used to generate a ECC key pair.
hseKeyGenTls12RsaPreMaster_t	rsaPreMaster	INPUT: The scheme used to generate the Rsa pre-master secret.

Macro Definition Documentation

HSE_KEY_GEN_SYM_RANDOM_KEY

```
#define HSE_KEY_GEN_SYM_RANDOM_KEY 1U
```

Generate a random symmetric key (e.g AES, HMAC).

HSE_KEY_GEN_RSA_KEY_PAIR

```
#define HSE_KEY_GEN_RSA_KEY_PAIR 2U
```

Generate a RSA key pair.

HSE_KEY_GEN_ECC_KEY_PAIR

```
#define HSE_KEY_GEN_ECC_KEY_PAIR 3U
```

Generate a ECC key pair.

HSE_TLS12_RSA_PRE_MASTER_SECRET_GEN

```
#define HSE_TLS12_RSA_PRE_MASTER_SECRET_GEN 5U
```

Generate the pre-master secret for TLS 1.2 RSA key exchange.

Typedef Documentation

hseKeyGenScheme_t

```
typedef uint8_t hseKeyGenScheme_t
```

HSE Key Generate schemes.

5.5 HSE Key Derivation Service

Data Structures

- struct hseKdfSalt_t
- struct hseKdfExtractStepScheme_t
- struct hseKdfCommonParams_t
- struct hseKdfNxpGenericScheme_t
- struct hseKdfSP800_56COneStepScheme_t
- struct hseKdfSP800_108Scheme_t
- struct hseKdfSP800_56CTwoStepScheme_t
- struct hsePBKDF2Scheme_t
- struct hseHKDF_ExpandScheme_t
- struct hseKdfTLS12PrfScheme_t
- struct hseKdfIKEV2Scheme_t
- struct hseKeyDeriveSrv_t
- struct hseKeyDeriveCopyKeySrv_t
- union hseKdfExtractStepScheme_t.prfAlgo
- union hseKdfCommonParams_t.prfAlgo
- union hseKdfIKEV2Scheme_t.prfAlgo
- union hseKeyDeriveSrv_t.sch

Macros

Type: hseKdfSP800_108Mode_t

Name	Value
HSE_KDF_SP800_108_COUNTER	1U

Type: hseKdfPrf_t	
Name	**Value**
HSE_KDF_PRF_HASH	1U
HSE_KDF_PRF_HMAC	2U
HSE_KDF_PRF_CMAC	3U
HSE_KDF_PRF_XCBC_MAC	4U

Type: hseKdfHashAlgo_t	
Name	**Value**
HSE_KDF_SHA2_224	HSE_HASH_ALGO_SHA2_224
HSE_KDF_SHA2_256	HSE_HASH_ALGO_SHA2_256
HSE_KDF_SHA2_384	HSE_HASH_ALGO_SHA2_384
HSE_KDF_SHA2_512	HSE_HASH_ALGO_SHA2_512
HSE_KDF_SHA2_512_224	HSE_HASH_ALGO_SHA2_512_224
HSE_KDF_SHA2_512_256	HSE_HASH_ALGO_SHA2_512_256

Type: hseKdfAlgo_t	
Name	**Value**
HSE_KDF_ALGO_NXP_GENERIC	1U
HSE_KDF_ALGO_EXTRACT_STEP	2U
HSE_KDF_ALGO_SP800_56C_ONE_STEP	3U
HSE_KDF_ALGO_SP800_56C_TWO_STEP	4U
HSE_KDF_ALGO_SP800_108	5U
HSE_KDF_ALGO_PBKDF2HMAC	6U
HSE_KDF_ALGO_HKDF_EXPAND	7U
HSE_KDF_ALGO_ANS_X963	8U
HSE_KDF_ALGO_ISO18033_KDF1	9U
HSE_KDF_ALGO_ISO18033_KDF2	10U
HSE_KDF_ALGO_TLS12PRF	11U
HSE_KDF_ALGO_IKEV2	12U

Type: hseTlsPskUsage_t	
Name	**Value**
HSE_TLS_PSK_NOT_USED	0U
HSE_TLS_KEY_EXCHANGE_PSK	1U

HSE_TLS_KEY_EXCHANGE_ECDHE_PSK	2U
HSE_TLS_KEY_EXCHANGE_RSA_PSK	3U
HSE_TLS_KEY_EXCHANGE_DHE_PSK	4U

Type: hseKdfSP800_108CounterLen_t	
Name	**Value**
HSE_KDF_SP800_108_COUNTER_LEN_DEFAULT	0U
HSE_KDF_SP800_108_COUNTER_LEN_1	1U
HSE_KDF_SP800_108_COUNTER_LEN_2	2U

Type: hseIkev2Steps_t	
Name	**Value**
HSE_IKEV2_STEP_INIT_SA	1U
HSE_IKEV2_STEP_CHILD_SA	2U
HSE_IKEV2_STEP_REKEY_SA	3U

Typedefs

- typedef uint8_t hseKdfAlgo_t
- typedef uint8_t hseKdfHashAlgo_t
- typedef uint8_t hseKdfPrf_t
- typedef hseKdfHashAlgo_t hseHashPrfAlgo_t
- typedef hseKdfHashAlgo_t hseHmacPrfAlgo_t
- typedef uint8_t hseNoPrfAlgo_t
- typedef uint8_t hseKdfSP800_108Mode_t
- typedef uint8_t hseKdfSP800_108CounterLen_t
- typedef uint8_t hseIkev2Steps_t
- typedef uint8_t hseTlsPskUsage_t
- typedef hseKdfCommonParams_t hseKdfANSX963Scheme_t
- typedef hseKdfCommonParams_t hseKdfISO18033_KDF1Scheme_t
- typedef hseKdfCommonParams_t hseKdfISO18033_KDF2Scheme_t

Data Structure Documentation

struct hseKdfSalt_t

The KDF salt definition.

The salt is used as the MAC key during the execution of the randomness-extraction step (first step). The salt can be a secret (providing the key handle) or a non-secret (e.g. value computed from nonces exchanged as part of a key-establishment protocol).

Data Fields

Type	Name	Description
hseKeyHandle_t	saltKeyHandle	INPUT: The salt key handle (when the salt is provided as a secret). If (saltKeyHandle == HSE_INVALID_KEY_HANDLE), the salt shall be specified by saltLength and pSalt parameters. If the saltKeyHandle is valid, the salt length is the key size in bytes and should match the the input block size.
uint32_t	saltLength	INPUT: Length of the salt in bytes. Used only if saltKeyHandle == HSE_INVALID_KEY_HANDLE . The length of salt are determined by the PRF algorithm: – For HMAC-hash PRF, the saltLength should be equal with the input block size (e.g 64/128 bytes). If saltLength is shorter, it will be padded with zeros. The saltLength greater than input block size will be firstly hashed using HASH PRF and then use the resultant byte string. – CMAC requires keys that are N bits long (for N = 128, 192, or 256). In this case, the salt should be 16, 24, or 32 bytes, depending upon the AES variant. Note that the saltLength can also be zero. In this case, the salt is an all-zero byte array whose length is equal to input block size (for hash or CMAC).
uint64_t	pSalt	INPUT: The salt. Used only if saltKeyHandle == HSE_INVALID_KEY_HANDLE . If pSalt is not passed (pSalt is NULL), default_salt will be used (the default_salt is all-zero byte array of length determined by input block). Note If the HOST_ADDR is on 64 bits, the address must fall within the 32-bit address range.

struct hseKdfExtractStepScheme_t

KDF Extraction step.

The extraction step is a Pseudo-Random Function (PRF) that takes as inputs a shared secret (secretKeyHandle) and the salt which can be a secret (a key) or non-secret (a generated random number). From these inputs, the PRF generates a pseudo-random key (PRK). The PRK can be used for the Expansion phase. The size of the PRK is equal with the size of the PRF output.

The following PRFs can be performed:
– PRK = HMAC-hash(salt, secret);
– PRK = CMAC(salt, secret);

Data Fields

Type	Name	Description
hseKeyHandle_t	secretKeyHandle	INPUT: The shared secret to be used for the operation.
hseKeyHandle_t	targetKeyHandle	INPUT: The target key handle (where to store the new key). It should point to a HSE_KEY_TYPE_SHARED_SECRET slot. The application can use the generated PRK for the Expand phase (using the same key handle) or it can extract the key(s) (in different slots) using the hseKeyDeriveCopyKeySrv_t service. The size of the PRK is equal with the size of the PRF output (e.g. for hmac-sha256, the key bit length is 256 bits)
hseKdfPrf_t	kdfPrf	INPUT: Selected the PRF to be used. Supported options: HSE_KDF_PRF_HMAC , HSE_KDF_PRF_CMAC .
union hseKdfExtractStepScheme_t.prfAlgo	prfAlgo	INPUT: Selects the algorithm for the PRF.
uint8_t	reserved[2]	
hseKdfSalt_t	salt	INPUT: The salt which is used as key. The saltLength should be equal with the input block size (e.g 16/64/128 bytes). See hseKdfSalt_t comments.

struct hseKdfCommonParams_t

KDF Common parameters.

Common parameters for expansion step used for different KDFs (SP800_56CTwoStep, HKDF-Expand, prf+ from IKEV2 etc). The expansion inputs are the output from the extractor (pseudo-random key from hseKdfExtractStepScheme_t) and the public context information (pInfo).

Data Fields

Type	Name	Description
hseKeyHandle_t	srcKeyHandle	INPUT: The source key to be used for the operation. For the expansion step, the source key handle should be a pseudorandom key (PRK) or a shared secret. (usually, the output from the extraction step; see hseKdfExtractStepScheme_t).

Data Fields

Type	Name	Description
hseKeyHandle_t	targetKeyHandle	INPUT: The target key handle (where to store the new key).It should point to a HSE_KEY_TYPE_SHARED_SECRET slot. The user can extract the key(s) (in different slots) from the derived key material using the hseKeyDeriveCopyKeySrv_t service.
uint16_t	keyMatLen	INPUT: The key material length to be derived (it must be >= 16 bytes and <= slot size).
hseKdfPrf_t	kdfPrf	INPUT: The PRFs used for KDF. Supported options: HSE_KDF_PRF_HASH , HSE_KDF_PRF_HMAC , HSE_KDF_PRF_CMAC .
union hseKdfCommonParams_t.prfAlgo	prfAlgo	INPUT: Selects the algorithm for the PRF.
uint32_t	infoLength	INPUT: Length of the pInfo. It must be <= 256 bytes.
uint64_t	pInfo	INPUT: The Info.

struct hseKdfNxpGenericScheme_t

KDF NXP generic scheme.

Used for deriving a cryptographic key from a source key and seed as described below:

```
{
    K[0]= NULL;
    key_mat[0]= NULL;
    iter = key_mat_len/prfOut_size;
    if(0 != (key_mat_len%prfOut_size))
    {
       iter = iter+1;
    }
    for(i = 1; i <= iter;i++)
    {
         step1:  K[i] = Prf(srcKey, K[i-1] || seed)
         step2:  key_mat[i]= key_mat[i-1]  || K[i]
    }
    key_mat = truncate(key_mat_len, key_mat[iter]).
}
```

Note

- If the key_mat_len >= 32 bytes, the last 8 bytes from the key material can be exported to the HOST.
- For SHA PRF:
  - if srcKeyAfterSeed = FALSE, step1 is K[i] = SHA(srcKey || K[i-1] || seed)
  - if srcKeyAfterSeed = TRUE , step1 is K[i] = SHA(K[i-1] || seed || srcKey)

Data Fields

Type	Name	Description
hseKdfCommonParams_t	kdfCommon	INPUT: KDF common parameters. Only HASH PRF is supported. – hseKdfCommonParams_t::kdfPrf = HSE_KDF_PRF_HASH – hseKdfCommonParams_t::pInfo = Seed. – hseKdfCommonParams_t::infoLength = Seed length (must be <= 256 bytes). Zero means the Seed is not used.
bool_t	srcKeyAfterSeed	INPUT: Concatenate the source key after the seed.
uint8_t	reserved	
uint16_t	outputLength	INPUT: Output data length to be exported to the host. It should be <= 8 bytes and can be used only if hseKdfCommonParams_t::keyMatLen >= 32 bytes.
uint64_t	pOutput	OUTPUT: Export outputLength bytes to host (only if the hseKdfCommonParams_t::keyMatLen >= 32 bytes). It can be NULL.

struct hseKdfSP800_56COneStepScheme_t

SP800 56C One Step Key derivation.

Perform One step KDF specified in SP800-56C rev1.

Note

Length of the counter is always 32bits.

Data Fields

Type	Name	Description
hseKdfCommonParams_t	kdfCommon	INPUT: KDF common parameters. Only HASH and HMAC are supported. – kdfCommon::kdfPrf = HSE_KDF_PRF_HASH or HSE_KDF_PRF_HMAC . – kdfCommon::pInfo = Fixed Info specified according to SP800_56C OneStep.

Data Fields

Type	Name	Description
hseKdfSalt_t	salt	INPUT: The salt. The salt is used only if HMAC PRF is selected (it's used as key). The saltLength should be equal with the input block size (e.g 64/128 bytes). If saltLength is shorter, it will be padded with zeros; if saltLength is longer, it will be hashed.

struct hseKdfSP800_108Scheme_t

SP800 108 Key derivation.

The KDF(Counter mode) as defined by SP800-108.

Note

> The key material length ([L]_2) from SP800 108 is represented on 32 bits. The iteration counter ([i]_2) can have 8, 16 or 32 bits.

Data Fields

Type	Name	Description
hseKdfCommonParams_t	kdfCommon	INPUT: KDF common parameters. Only HMAC and CMAC are supported. – .kdfCommon.kdfPrf = HSE_KDF_PRF_HMAC or HSE_KDF_PRF_CMAC . – .kdfCommon.pInfo = the context-specific data according to SP800_108: "Label||0x00||Context||[L]_2". Note Source key should be a valid symmetric key of length that respects the constraints defined for kdf salt (see hseKdfSalt_t).
hseKdfSP800_108Mode_t	mode	INPUT: Selects the SP800_108 mode: Counter (e.g. Feedback, Pipeline not suppoted)
hseKdfSP800_108CounterLen_t	counterByteLength	INPUT: Selects the length in bytes of the counter ([i]_2). The length of the counter can be 1, 2 or 4 bytes. Note Any other value will be treated as the default value (4 bytes)
uint8_t	reserved[14]	

struct hseKdfSP800_56CTwoStepScheme_t

SP800 56C Two-step Key derivation.

Perform Two step KDF specified in SP800-56C.
SP800_56C Two Step includes SP800 108 parameters for Expansion Step, and additional the salt for Extraction Step.

Note

- OtherInput define by SP800 56C contains the salt, the key material length (L) and FixedInfo, which are provided as parameters by the service.
- Counter length ['r'] supported is 32 bits.

Data Fields

Type	Name	Description
hseKdfSP800_108Scheme_t	expand	INPUT: KDF common parameters. Only HMAC and CMAC are supported. – .expand.kdfCommon.kdfPrf = HSE_KDF_PRF_HMAC or HSE_KDF_PRF_CMAC . – .expand.kdfCommon.pInfo = FixedInfo which follows SP800-56C.
hseKdfSalt_t	salt	INPUT: The salt used for Extraction Step.

struct hsePBKDF2Scheme_t

Password Based Key Derivation Function 2.

Used for deriving a cryptographic key from a password

Data Fields

Type	Name	Description
hseKeyHandle_t	srcKeyHandle	INPUT: The source key to be used for the operation (shared secret).
hseKeyHandle_t	targetKeyHandle	INPUT: The target key handle (where to store the new key).It should point to a HSE_KEY_TYPE_SHARED_SECRET slot. The user can extract the key(s) (in different slots) from the derived key material using the hseKeyDeriveCopyKeySrv_t service.
uint16_t	keyMatLen	INPUT: The key material length to be derived (it must be <= slot size).

Data Fields

Type	Name	Description
hseHmacPrfAlgo_t	hmacHash	INPUT: The hash algorithm for HMAC PRF. Note HSE_M platforms do not support SHA2_384/512 hash algorithms for PBKDF2.
uint8_t	reserved	
uint32_t	iterations	INPUT: The number of iterations to be performed. Should take a value higher than 100.
uint32_t	saltLength	INPUT: Length of the salt. It must be < 8192 bytes.
uint64_t	pSalt	INPUT: A salt; 16 bytes or longer (randomly generated)

struct hseHKDF_ExpandScheme_t

HKDF-Expand KDF Function.

It is suitable for deriving keys of a fixed size used for other cryptographic operations.

Note

For TLS1.3, the HKDF-Extract function (first step) can be performed using hseKdfExtractStepScheme_t.

Data Fields

Type	Name	Description
hseKdfCommonParams_t	kdfCommon	INPUT: KDF common parameters. Only HMAC is supported. – .kdfCommon.kdfPrf = HSE_KDF_PRF_HMAC – .kdfCommon.pInfo = Application specific context. Can be NULL.
uint64_t	pIvOutput	OUTPUT: The TLS1.3 IV output. HSE exports the HKDF expansion output only if the kdfCommon.pInfo starts with the following concatenation: kdfCommon.keyMatLen(2 bytes big-endian) | "tls13 iv" (string of 8 bytes). The length of pIvOutput is the kdfCommon.keyMatLen. In this case kdfCommon.targetKeyHandle is not used.

struct hseKdfTLS12PrfScheme_t

TLS 1.2 PRF as specified by RFC 5246.

The PRF needed in TLS1.2 protocol to derive the master secret, the key block and the verify data.

Data Fields

Type	Name	Description
uint16_t	labelLength	INPUT: The label length in bytes (without '\0' termination). Only the following labels are valid in case of TLS 1.2 PRF. – master secret label - "master secret" – extended master secret - "extended master secret" (refer to rfc7627) – key expansion label - "key expansion" – client finished label - "client finished" – server finished label - "server finished" Note – The above arrays do not contain the string termination character. – The above label lengths are the only valid label lengths that must be provided by the Host Application (refer to RFC 5246).
uint8_t	reserved1[2U]	
uint64_t	pLabel	INPUT: The label of the TLS1.2 PRF operations. – If pLabel = "master secret" or "extended master secret", HSE computes the master secret; the hseKdfTLS12PrfScheme_t::keyMatLength must be 48 bytes. – If pLabel = "key expansion", HSE computes the key_block; the hseKdfTLS12PrfScheme_t::keyMatLength must be >= 32 bytes. HSE also outputs the client and server IVs (see pOutput). – if pLabel = "client finished" or "server finished", HSE computes the verify_data (see pOutput). Note – The pre-master shared secret (HSE_KEY_TYPE_SHARED_SECRET key slot) is deleted after master secret computation (see rfc5246).

Data Fields

Type	Name	Description
hseTlsPskUsage_t	tlsPskUsage	INPUT: Selects TLS-PSK algorithm usage. Used only for master secret computation (label = "master secret"). Ignored for other labels. Note – HSE_TLS_PSK_NOT_USED - pre-shared key not used – HSE_TLS_KEY_EXCHANGE_PSK - pre-master secret is computed as: If the PSK is N octets long, concatenate a uint16 with the value N, N zero octets, a second uint16 with the value N, and the PSK itself (refer to rfc4279) – HSE_TLS_KEY_EXCHANGE_ECDHE_PSK - pre-master secret is computed as: Let Z be the octet string of ECDH shared secret. The pre-master is the concatenation of a uint16 containing the length of Z (in octets), Z itself, a uint16 containing the length of the PSK (in octets), and the PSK itself (refer to rfc5489) – HSE_TLS_KEY_EXCHANGE_RSA_PSK - pre-master secret is computed as: concatenate a uint16 with the value 48, the 2-byte version number and the 46-byte random value, a uint16 containing the length of the PSK (in octets), and the PSK itself (the pre-master secret is thus 52 octets longer than the PSK); refer to rfc4279. – HSE_TLS_KEY_EXCHANGE_DHE_PSK - let Z be the value produced by classic DH computation. The pre-master secret is computed: concatenate a uint16 containing the length of Z (in octets), Z itself, a uint16 containing the length of the PSK (in octets), and the PSK itself.
uint8_t	reserved2[3U]	
hseKeyHandle_t	pskKeyHandle	INPUT: Pre-shared key handle. It can be any symmetric NVM key that has the HSE_KF_USAGE_DERIVE flag set. Used only for master secret computation and tlsPskUsage != HSE_TLS_PSK_NOT_USED .

Data Fields

Type	Name	Description
hseKeyHandle_t	srcKeyHandle	INPUT: The source key handle (it must point to a HSE_KEY_TYPE_SHARED_SECRET slot). – For label = "master secret": · if tlsPskUsage = HSE_TLS_PSK_NOT_USED , it must be the pre-master secret (e.g DH shared secret). · if tlsPskUsage = HSE_TLS_KEY_EXCHANGE_PSK , it is ignored (key handle is provided by pskKeyHandle). · if tlsPskUsage = HSE_TLS_KEY_EXCHANGE_ECDHE_PSK , it is the DH shared secret. · if tlsPskUsage = HSE_TLS_KEY_EXCHANGE_RSA_PSK , the shared secret slot contains: ProtocolVersion (2bytes) concatenated with 46 byte random number. – For key_block or verify_data, it must be the master secret.
hseHmacPrfAlgo_t	hmacHash	INPUT: The hash algorithm for HMAC PRF.
uint8_t	reserved3[1U]	
uint16_t	seedLength	INPUT: The seed length. It must be <= 256 bytes.
uint64_t	pSeed	INPUT: The seed for TLS 1.2 PRF. In TLS, this is usually a combination of user and random data. This is the concatenation of Server and Client Hello random data. – For master secret, it is concatenation of Server Random Data || Client Random Data. – For extended master secret, it is the session_hash (refer to rfc7627). – For Key Expansion, it is concatenation of Client Random Data || Server Random Data. Refer to RFC 5246 for more details.

Data Fields

Type	Name	Description
hseKeyHandle_t	targetKeyHandle	INPUT: The target key handle (where to store the new key). It shall point to a HSE_KEY_TYPE_SHARED_SECRET slot (this means HSE_KF_USAGE_DERIVE flag is set by default). It can be: – the derived master secret – the derived key_block. The user can extract the key(s) using the hseKeyDeriveCopyKeySrv_t service. The key_block is partitioned as follows: · client_write_MAC_key[] · server_write_MAC_key[] · client_write_key[] · server_write_key[] · client_write_IV[]; exported in pOutput below if pLabel = "key expansion" · server_write_IV[]; exported in pOutput below if pLabel = "key expansion" – not used for verify_data (pLabel = "client finished" or pLabel = "server finished")
uint16_t	keyMatLength	INPUT: The key material length (in bytes). – If pLabel = "master secret" or "extended master secret", the keyMatLength must be 48 bytes. – If pLabel = "key expansion" (key_block), the keyMatLength must be >= 32 bytes. It must be the total length for Client and Server keys without the IVs (only the MAC and encryption keys). – Not used for verify_data (if the pLabel = "client finished" or pLabel = "server finished")
uint16_t	outputLength	INPUT: The length for output data (pOutput) which can be: – For pLabel= "key expansion", the total length for client and server Initialization Vectors from key_block. Can be 0. If it is provided, it must be <= 32 bytes (2*block size). – For pLabel = "client finished" or "server finished", the verify_data length. Must be 12 bytes.

Data Fields

Type	Name	Description
uint64_t	pOutput	OUTPUT: The output data which can be: – For pLabel = "key expansion", concatenated client and server IVs of totalIvLength (client_write_IV[] || server_write_IV[]). Can be NULL. – For pLabel = "client finished" or "server finished", verify_data sent in the Finished message.

struct hseKdfIKEV2Scheme_t

IKEv2 KDF as specified by RFC 5996.

Two-step KDF that derives the needed keys in the Internet Key Exchange Protocol Version 2.
RFC5996 specifies the following Security Association (SA) steps:

- INIT_SA step computes:
  - SKEYSEED = prf(Ni | Nr, g^ir)
  - KEYMAT= prf+ (SKEYSEED, Ni | Nr | SPIi | SPIr)
- CHILD_SA step (it can use a new Shared Secret (g^ir) from the ephemeral DH of CREATE_CHILD_SA exchange).
  For new g^ir, it computes: KEYMAT = prf+(SK_d, g^ir (new) | Ni | Nr).
  Otherwise, it is computes: KEYMAT = prf+(SK_d, Ni | Nr).
- REKEY_SA step computes:
  - SKEYSEED = prf(SK_d (old), g^ir (new) | Ni | Nr)
  - KEYMAT= prf+ (SKEYSEED, Ni | Nr | SPIi | SPIr) from the new exchange

Note

- KEYMAT contains 7 keys: {SK_d | SK_ai | SK_ar | SK_ei | SK_er | SK_pi | SK_pr }. The host shall extract the keys in different slots using the hseKeyDeriveCopyKeySrv_t service.
- If the negotiated PRF is AES-XCBC-PRF-128 or AES-CMAC-PRF-128, only the first 64 bits of Ni and the first 64 bits of Nr are used in calculating SKEYSEED, but all the bits are used for input to the prf+ function.

Data Fields

Type	Name	Description
hseIkev2Steps_t	step	INPUT: The Security Association (SA) step: HSE_IKEV2_STEP_INIT_SA , HSE_IKEV2_STEP_CHILD_SA , HSE_IKEV2_STEP_REKEY_SA .
uint8_t	reserved[3]	

Data Fields

Type	Name	Description
hseKeyHandle_t	g_ir_keyHandle	INPUT: The key handle for g^ir from ephemeral DH. – For HSE_IKEV2_STEP_CHILD_SA , if no new g_ir is computed it shall be set to HSE_INVALID_KEY_HANDLE .
hseKeyHandle_t	sk_d_keyHandle	INPUT: The key handle of SK_d use within HSE_IKEV2_STEP_CHILD_SA or HSE_IKEV2_STEP_REKEY_SA steps; otherwise not used.
hseKeyHandle_t	targetKeyHandle	INPUT: The target key handle (where to store the new key). It shall point to a HSE_KEY_TYPE_SHARED_SECRET slot, and the user can extract the keys in different slots using the hseKeyDeriveCopyKeySrv_t service. The keys are partitioned in the slot as follows: {SK_d | SK_ai | SK_ar | SK_ei | SK_er | SK_pi | SK_pr }.
uint16_t	keyMatLen	INPUT: The key material length to be derived (it must be <= slot size). It should include the total length for all generated 7 keys. Note The lengths of SK_d, SK_pi, and SK_pr MUST be the preferred key length of the PRF agreed.
hseKdfPrf_t	kdfPrf	INPUT: The PRFs used for KDF. Supported options: HSE_KDF_PRF_HMAC , HSE_KDF_PRF_CMAC , HSE_KDF_PRF_XCBC_MAC .
union hseKdfIKEV2Scheme_t.prfAlgo	prfAlgo	INPUT: Selects the algorithm for the PRF.
uint32_t	inputLength	INPUT: Number of bytes from input to be processed. It must be between 16 and 528 bytes.
uint64_t	pInput	INPUT: Input data for each IKEv2 step: – HSE_IKEV2_STEP_INIT_SA step: concatenation of Ni | Nr | SPIi | SPIr – HSE_IKEV2_STEP_CHILD_SA step: concatenation of Ni | Nr – HSE_IKEV2_STEP_REKEY_SA step: concatenation of Ni | Nr | SPIi | SPIr from the new exchange.

Data Fields

Type	Name	Description
uint32_t	totalNonceLength	INPUT: Length of Ni | Nr concatenation. Used only for HSE_IKEV2_STEP_INIT_SA and HSE_IKEV2_STEP_REKEY_SA .

struct hseKeyDeriveSrv_t

HSE Key Derive service.

The key derive service (KDF) derives one or more secret keys from a secret value.

Note

- The key material can be derived only in HSE_KEY_TYPE_SHARED_SECRET slots (specified as targetKeyHandle), which can not be exported outside HSE.

Data Fields

Type	Name	Description
hseKdfAlgo_t	kdfAlgo	INPUT: The key derivation algorithm.
uint8_t	reserved[3]	
union hseKeyDeriveSrv_t.sch	sch	INPUT: The selected key derivation algorithm.

struct hseKeyDeriveCopyKeySrv_t

HSE Key Derive - Copy Key service.

This service can be used to extract keys (or a key) from the derived key material placed in a temporary shared secret slot (HSE_KEY_TYPE_SHARED_SECRET).

The key(s) can be copied in NVM/RAM slots as follow:

1. Restrictions for SuperUser rights:
   - keys can be copied in NVM key store from the derived key material only in empty slots (an erase shall be performed in advance if needed).
   - keys can be copied in RAM key store from the derived key material (RAM keys can be overwritten).
2. Restrictions for User rights:
   - keys can NOT be copied in NVM key store from the derived key material.
   - keys can be copied in RAM key store from the derived key material (RAM keys can be overwritten).

Data Fields

Type	Name	Description
hseKeyHandle_t	keyHandle	INPUT: The key handle to be used to extract a key value. The key handle should point to a HSE_KEY_TYPE_SHARED_SECRET key type.
uint16_t	startOffset	INPUT: Start offset from where to copy the key. The offset can be zero or a multiple of 4 bytes.
uint8_t	reserved[2]	
hseKeyHandle_t	targetKeyHandle	INPUT: The target key handle (where to store the new key).
hseKeyInfo_t	keyInfo	INPUT: Specifies usage flags, restriction access, key bit length etc for the key. Note that the length of the copied key is considered to be hseKeyInfo_t::keyBitLen. The minimum key length that can be copied is 16 bytes.

union hseKdfExtractStepScheme_t.prfAlgo

INPUT: Selects the algorithm for the PRF.

Data Fields

Type	Name	Description
hseHmacPrfAlgo_t	hmacHash	The hash algorithm used for HMAC.
hseNoPrfAlgo_t	cmac	Dummy byte.

union hseKdfCommonParams_t.prfAlgo

INPUT: Selects the algorithm for the PRF.

Data Fields

Type	Name	Description
hseHashPrfAlgo_t	hash	The KDF hash algorithm .
hseHmacPrfAlgo_t	hmacHash	The hash algorithm used for HMAC.
hseNoPrfAlgo_t	cmac	Dummy byte.

union hseKdfIKEV2Scheme_t.prfAlgo

INPUT: Selects the algorithm for the PRF.

Data Fields

Type	Name	Description
hseHmacPrfAlgo_t	hmacHash	The hash algorithm used for HMAC.
hseNoPrfAlgo_t	cmac	No PRF algorithm.
hseNoPrfAlgo_t	xCbcmac	No PRF algorithm.

union hseKeyDeriveSrv_t.sch

INPUT: The selected key derivation algorithm.

Data Fields

Type	Name	Description
hseKdfNxpGenericScheme_t	nxpGeneric	INPUT: NXP generic KDF scheme.
hseKdfExtractStepScheme_t	extractStep	Generic Extraction Step for Two-step KDFs.
hseKdfSP800_56COneStepScheme_t	SP800_56COneStep	INPUT: One-Step SP800_56C KDF scheme.
hseKdfSP800_56CTwoStepScheme_t	SP800_56CTwoStep	INPUT: Two-Step SP800_56C KDF scheme.
hseKdfSP800_108Scheme_t	SP800_108	INPUT: SP800 108 KDF scheme.
hsePBKDF2Scheme_t	PBKDF2	INPUT: PBKDF2 scheme.
hseHKDF_ExpandScheme_t	HKDF_Expand	INPUT: HKDF-Expand scheme.
hseKdfANSX963Scheme_t	ANS_X963	INPUT: ANS_X963 KDF scheme.
hseKdfISO18033_KDF1Scheme_t	ISO18033_KDF1	INPUT: ISO18033 KDF1 scheme.
hseKdfISO18033_KDF2Scheme_t	ISO18033_KDF2	INPUT: ISO18033 KDF2 scheme.
hseKdfTLS12PrfScheme_t	TLS12Prf	INPUT: TLS 1.2 PRF.
hseKdfIKEV2Scheme_t	IKEv2	INPUT: IKEv2 KDF scheme.

Macro Definition Documentation

HSE_KDF_ALGO_NXP_GENERIC

```
#define HSE_KDF_ALGO_NXP_GENERIC ((hseKdfAlgo_t)1U)
```

NXP Generic KDF.

HSE_KDF_ALGO_EXTRACT_STEP

```
#define HSE_KDF_ALGO_EXTRACT_STEP ((hseKdfAlgo_t)2U)
```

Generic Extraction Step for Two-step KDFs.

HSE_KDF_ALGO_SP800_56C_ONE_STEP

```
#define HSE_KDF_ALGO_SP800_56C_ONE_STEP ((hseKdfAlgo_t)3U)
```

One-step KDF as defined by SP800-56C rev1.

HSE_KDF_ALGO_SP800_56C_TWO_STEP

```
#define HSE_KDF_ALGO_SP800_56C_TWO_STEP ((hseKdfAlgo_t)4U)
```

Two-step KDF as defined by SP800-56C rev1.

HSE_KDF_ALGO_SP800_108

```
#define HSE_KDF_ALGO_SP800_108 ((hseKdfAlgo_t)5U)
```

KDF(Counter, Feedback, Pipeline) as defined by SP800-108.

HSE_KDF_ALGO_PBKDF2HMAC

```
#define HSE_KDF_ALGO_PBKDF2HMAC ((hseKdfAlgo_t)6U)
```

PBKDF2HMAC as defined by PKCS#5 v2.1 and RFC-8018.

HSE_KDF_ALGO_HKDF_EXPAND

```
#define HSE_KDF_ALGO_HKDF_EXPAND ((hseKdfAlgo_t)7U)
```

HKDF Expand KDFs as defined by RFC-5869.

HSE_KDF_ALGO_ANS_X963

```
#define HSE_KDF_ALGO_ANS_X963 ((hseKdfAlgo_t)8U)
```

KDF as defined by ANS X9.63.

HSE_KDF_ALGO_ISO18033_KDF1

```
#define HSE_KDF_ALGO_ISO18033_KDF1 ((hseKdfAlgo_t)9U)
```

KDF1 as defined by ISO18033.

HSE_KDF_ALGO_ISO18033_KDF2

```
#define HSE_KDF_ALGO_ISO18033_KDF2 ((hseKdfAlgo_t)10U)
```

KDF2 as defined by ISO18033.

HSE_KDF_ALGO_TLS12PRF

```
#define HSE_KDF_ALGO_TLS12PRF ((hseKdfAlgo_t)11U)
```

TLS 1.2 PRF as defined by RFC-5246.

HSE_KDF_ALGO_IKEV2

```
#define HSE_KDF_ALGO_IKEV2 ((hseKdfAlgo_t)12U)
```

KDF IKEv2 as defined by RFC-4306.

HSE_KDF_SHA2_224

```
#define HSE_KDF_SHA2_224 ((hseKdfHashAlgo_t) HSE_HASH_ALGO_SHA2_224)
```

HSE_KDF_SHA2_256

```
#define HSE_KDF_SHA2_256 ((hseKdfHashAlgo_t) HSE_HASH_ALGO_SHA2_256)
```

HSE_KDF_SHA2_384

```
#define HSE_KDF_SHA2_384 ((hseKdfHashAlgo_t) HSE_HASH_ALGO_SHA2_384)
```

HSE_KDF_SHA2_512

```
#define HSE_KDF_SHA2_512 ((hseKdfHashAlgo_t) HSE_HASH_ALGO_SHA2_512)
```

HSE_KDF_SHA2_512_224

```
#define HSE_KDF_SHA2_512_224 ((hseKdfHashAlgo_t) HSE_HASH_ALGO_SHA2_512_224)
```

HSE_KDF_SHA2_512_256

```
#define HSE_KDF_SHA2_512_256 ((hseKdfHashAlgo_t) HSE_HASH_ALGO_SHA2_512_256)
```

HSE_KDF_PRF_HASH

```
#define HSE_KDF_PRF_HASH ((hseKdfPrf_t)1U)
```

SHA2 families.

HSE_KDF_PRF_HMAC

```
#define HSE_KDF_PRF_HMAC ((hseKdfPrf_t)2U)
```

HMAC-SHA2 families.

HSE_KDF_PRF_CMAC

```
#define HSE_KDF_PRF_CMAC ((hseKdfPrf_t)3U)
```

CMAC.

HSE_KDF_PRF_XCBC_MAC

```
#define HSE_KDF_PRF_XCBC_MAC ((hseKdfPrf_t)4U)
```

XCBC_MAC (used only for IKEV2 KDF).

HSE_KDF_SP800_108_COUNTER

```
#define HSE_KDF_SP800_108_COUNTER ((hseKdfSP800_108Mode_t)1U)
```

SP800 108 Counter step.

HSE_KDF_SP800_108_COUNTER_LEN_DEFAULT

```
#define HSE_KDF_SP800_108_COUNTER_LEN_DEFAULT ((hseKdfSP800_108CounterLen_t)0U)
```

SP800 108 default counter length (4 bytes)

HSE_KDF_SP800_108_COUNTER_LEN_1

```
#define HSE_KDF_SP800_108_COUNTER_LEN_1 ((hseKdfSP800_108CounterLen_t)1U)
```

SP800 108 1 byte counter length.

HSE_KDF_SP800_108_COUNTER_LEN_2

```
#define HSE_KDF_SP800_108_COUNTER_LEN_2 ((hseKdfSP800_108CounterLen_t)2U)
```

SP800 108 2 bytes counter length.

HSE_IKEV2_STEP_INIT_SA

```
#define HSE_IKEV2_STEP_INIT_SA ((hseIkev2Steps_t)1U)
```

IKE_SA_INIT step - Initial Keying Material for the IKE SA.

HSE_IKEV2_STEP_CHILD_SA

```
#define HSE_IKEV2_STEP_CHILD_SA ((hseIkev2Steps_t)2U)
```

CHILD_SA step - Generating Keying Material for Child SAs.

HSE_IKEV2_STEP_REKEY_SA

```
#define HSE_IKEV2_STEP_REKEY_SA ((hseIkev2Steps_t)3U)
```

REKEY step - Rekeying IKE SAs Using a CREATE_CHILD_SA Exchange.

HSE_TLS_PSK_NOT_USED

```
#define HSE_TLS_PSK_NOT_USED ((hseTlsPskUsage_t)0U)
```

TLS PSK is not used.

HSE_TLS_KEY_EXCHANGE_PSK

```
#define HSE_TLS_KEY_EXCHANGE_PSK ((hseTlsPskUsage_t)1U)
```

Key Exchange PSK (refer to rfc4279)

HSE_TLS_KEY_EXCHANGE_ECDHE_PSK

```
#define HSE_TLS_KEY_EXCHANGE_ECDHE_PSK ((hseTlsPskUsage_t)2U)
```

Key Exchange ECDHE_PSK (refer to rfc5489)

HSE_TLS_KEY_EXCHANGE_RSA_PSK

```
#define HSE_TLS_KEY_EXCHANGE_RSA_PSK ((hseTlsPskUsage_t)3U)
```

Key Exchange RSA_PSK (refer to rfc4279)

HSE_TLS_KEY_EXCHANGE_DHE_PSK

```
#define HSE_TLS_KEY_EXCHANGE_DHE_PSK ((hseTlsPskUsage_t)4U)
```

Key Exchange DHE_PSK (refer to rfc4279)

Typedef Documentation

hseKdfAlgo_t

```
typedef uint8_t hseKdfAlgo_t
```

HSE Key derivation algorithms.

hseKdfHashAlgo_t

```
typedef uint8_t hseKdfHashAlgo_t
```

Hash algorithm available for KDF.

hseKdfPrf_t

```
typedef uint8_t hseKdfPrf_t
```

HSE KDF "Pseudo-Random Function" (PRF).

hseHashPrfAlgo_t

```
typedef hseKdfHashAlgo_t hseHashPrfAlgo_t
```

HSE PRF algorithm.

Algorithm for hash PRF (e.g SHA256)

hseHmacPrfAlgo_t

```
typedef hseKdfHashAlgo_t hseHmacPrfAlgo_t
```

Algorithm for hmac PRF (e.g SHA256)

hseNoPrfAlgo_t

```
typedef uint8_t hseNoPrfAlgo_t
```

No PRF algorithm.

hseKdfSP800_108Mode_t

```
typedef uint8_t hseKdfSP800_108Mode_t
```

SP800-108 KDF modes (only Counter mode supported).

hseKdfSP800_108CounterLen_t

```
typedef uint8_t hseKdfSP800_108CounterLen_t
```

SP800-108 KDF counter length (only 1, 2 and 4 bytes supported).

hseIkev2Steps_t

```
typedef uint8_t hseIkev2Steps_t
```

HSE IKEv2 exchange of messages steps.

hseTlsPskUsage_t

```
typedef uint8_t hseTlsPskUsage_t
```

TLS PSK usage.

hseKdfANSX963Scheme_t

```
typedef hseKdfCommonParams_t hseKdfANSX963Scheme_t
```

ANS X9.63 KDF as specified by SEC1-v2.

One-step KDF performed in the context of an ANS X9.63 key agreement scheme.
ANS X9.63 KDF supports:

- .kdfPrf = HSE_KDF_PRF_HASH (ANS X9.63 supports only hash PRF).
- .pInfo points to SharedInfo (optional, as defined by ANS X9.63).

hseKdfISO18033_KDF1Scheme_t

```
typedef hseKdfCommonParams_t hseKdfISO18033_KDF1Scheme_t
```

KDF1 as specified by ISO18033.

One-step KDF performed as specified by ISO18033.
ISO18033 KDF1 supports:

- .kdfPrf = HSE_KDF_PRF_HASH (ISO18033 supports only hash PRF).
- .pInfo = NULL.
- .infoLength = 0UL

hseKdfISO18033_KDF2Scheme_t

```
typedef hseKdfCommonParams_t hseKdfISO18033_KDF2Scheme_t
```

KDF2 as specified by ISO18033.

One-step KDF performed as specified by ISO18033.
ISO18033 KDF2 supports:

- .kdfPrf = HSE_KDF_PRF_HASH (ISO18033 supports only hash PRF).
- .pInfo = NULL.
- .infoLength = 0UL

6 Boot and Memory Verification Services

6.1 HSE Core Reset And Secure Memory Region (SMR) Services

Data Structures

- struct hseSmrDecrypt_t
- struct hseSmrEntry_t
- struct hseCrEntry_t
- struct hseSmrEntryInstallSrv_t
- struct hseSmrVerifySrv_t
- struct hseSmrEntryEraseSrv_t
- struct hseCrEntryInstallSrv_t
- struct hseCrOnDemandBootSrv_t
- struct hseCrEntryEraseSrv_t
- struct hseSmrEntryInstallSrv_t.cipher

Macros

Type: (implicit C type)	
Name	**Value**
HSE_SMR_DECRYPT_KEY_HANDLE_NOT_USED	0UL
HSE_SMR_VERSION_NOT_USED	0UL

Type: hseCrStartOption_t	
Name	**Value**
HSE_CR_AUTO_START	0x35A5U
HSE_CR_ON_DEMAND	0x5567U

Type: hseSmrVerificationOptions_t	
Name	**Value**
HSE_SMR_VERIFICATION_OPTION_NONE	0UL /** @brief Default verification of the SMR at run-time. *
HSE_SMR_VERIFICATION_OPTION_NO_LOAD	(3UL << 0U)) /** @brief SMR is verified from the external flash (using pSmrSrc address) even if pSmrDest is specified or if already loaded. Can be used only if SMR is in a memory mapped external flash (e.g. QSPI/LPDDR4 and not SD/eMMC). Additionally the SMR cannot be encrypted. *

HSE_SMR_VERIFICATION_OPTION_RELOAD	(3UL << 2U)) /** @brief SMR is loaded from the external flash and verified even if it is already loaded. Can be used only if SMR is in a memory mapped external flash (e.g. QSPI/LPDDR4 and not SD/eMMC). *

Type: hseSmrConfig_t	
Name	**Value**
HSE_SMR_CFG_FLAG_QSPI_FLASH	0x0U
HSE_SMR_CFG_FLAG_SD_FLASH	0x2U
HSE_SMR_CFG_FLAG_MMC_FLASH	0x3U
HSE_SMR_CFG_FLAG_INSTALL_AUTH	1U << 2U
HSE_SMR_CFG_FLAG_AUTH_AAD	1U << 3U

Type: hseCrSanction_t	
Name	**Value**
HSE_CR_SANCTION_DIS_INDIV_KEYS	0x7433U
HSE_CR_SANCTION_KEEP_CORE_IN_RESET	0x7455U
HSE_CR_SANCTION_RESET_SOC	0x8B17U
HSE_CR_SANCTION_DIS_ALL_KEYS	0x8B1EU

Typedefs

- typedef uint16_t hseCrSanction_t
- typedef uint16_t hseCrStartOption_t
- typedef uint8_t hseSmrConfig_t
- typedef uint16_t hseSmrVerificationOptions_t

Data Structure Documentation

struct hseSmrDecrypt_t

Defines the parameters to decrypt an encrypted SMR.

The parameters below are used in the SMR entry only with an encrypted SMR.

Note

The following algorithms can be used:

- If pGmacTag == NULL, the SMR must be encrypted using AES-CTR

– If pGmacTag != NULL, the SMR must be encrypted using AEAD-GCM with AAD = NULL (pGmacTag shall point to the GMAC Tag).

Data Fields

Type	Name	Description
hseKeyHandle_t	decryptKeyHandle	The key handle referencing the decryption key. – If decryptKeyHandle == HSE_SMR_DECRYPT_KEY_HANDLE_NOT_USED , the SMR is not encrypted; all the fields below are ignored. – If decryptKeyHandle != HSE_SMR_DECRYPT_KEY_HANDLE_NOT_USED , the decryptKeyHandle specifies the key used to decrypt the SMR. Note · The used algorithm is always AEAD-GCM, where GMAC and AAD are optional. · If the GMAC tag is provided (is not NULL), the same key is also used to verify the tag.
uint32_t	pGmacTag	The Tag used for GCM. If it set NULL, AES-CTR (instead of GCM) is used for decryption. – If pGmacTag == NULL, an internal hash is computed at installation over the encrypted SMR. This internal hash is used at verification phase. – If pGmacTag != NULL, the external stored GMAC tag (in flash) is used to verify the encrypted SMR. The length considered in this case is 16 bytes.
uint8_t	aadLength	Optional - the length in bytes of the Authenticated Additional Data (AAD). – If not used, the length can be zero; – If used, the length can be either 64 or 128 bytes. – If used, pGmacTag must also be provided.
uint8_t	reserved[3U]	Reserved - alignment.

Data Fields

Type	Name	Description
uint32_t	pAAD	Optional - the AAD used for AEAD. – Ignored if aadLength is zero; – If provided, the AAD is NOT stored by HSE internally; pAAD address must point to an external flash location that HSE will use during verification.

struct hseSmrEntry_t

Define the parameters of a Secure Memory Region (SMR) entry in a SMR table.

The SMR entry is installed and verified in two phases:

1. "Installation Phase" (using hseSmrEntryInstallSrv_t service).
   - The parameters related to SMR authetication and encryption, namely authScheme, authKeyHandle and if the SMR is encrypted, hseSmrDecrypt_t::decryptKeyHandle and hseSmrDecrypt_t::pGmacTag will be used by HSE at installation time from the hseSmrEntry_t structure referenced in the hseSmrEntryInstallSrv_t::pSmrEntry.
   - This phase happens at run-time and as a consequence any data provided to HSE must be memory-mapped (QSPI/LPDDR4/RAM). In case an SMR lying in SD/eMMC is installed, a copy of the data that is not stored by the HSE internally must be done available in RAM (e.g. SMR source, signature, AAD, GMAC tag, etc.). At installation time HSE will use the matching pointer fields from the hseSmrEntryInstallSrv_t structure to access the data.
2. "Verification Phase" that can be configured to be performed in two modes:
   - Verify with the Original/Installation Authentication TAG over the plaintext (HSE_SMR_CFG_FLAG_INSTALL_AUTH flag is set); the pInstAuthTag parameter must be provided and must point to original signature.
   - Verify using an internal computed hash (HSE_SMR_CFG_FLAG_INSTALL_AUTH flag is cleared); pInstAuthTag is not used in this case.
   - In the same manner, if the SMR is encrypted, HSE can use the provided hseSmrDecrypt_t::pGmacTag (original) or an internally computed hash to verify the encrypted SMR before decryption.

Data Fields

Type	Name	Description
uint32_t	pSmrSrc	Source address where the SMR needs to be loaded from. This address must be absolute address.

Data Fields

Type	Name	Description
uint64_t	pSmrDest	Destination address of SMR (where to copy the SMR after authentication). Note If the HOST_ADDR is on 64 bits: – Destination address + size of the SMR loaded from SD/MMC should be within 32-bit address range.
uint32_t	smrSize	The size in bytes of the SMR to be loaded/verified.
hseSmrConfig_t	configFlags	Configuration flags of SMR entry (see hseSmrConfig_t).
uint8_t	reserved0[3U]	Reserved for alignment.
uint32_t	checkPeriod	If checkPeriod != 0, HSE verify the SMR entry periodically (in background). Specifies the verification period in x100 milliseconds when HSE is running at maximum frequency. Otherwise, the period is multiplied by the factor max_freq/actual_freq (e.g. 100ms at 400MHz, 200ms at 200MHz, etc). Note – The value 0xFFFFFFFFUL invalid; the checkPeriod max value must be [MAX_UNSIGNED32_INT - 1]. – If the checkPeriod is non zero, the pSmrDest must be non zero and the configFlags must be zero. – The SMR periodic verification will start on next boot after PRE and POST boot verification. – If the periodic SMR verification is used, the HSE firmware always uses the internal hash for verification.
hseKeyHandle_t	authKeyHandle	The key handle used to check the authenticity of the plaintext SMR. Note – If the HSE_SMR_CFG_FLAG_INSTALL_AUTH flag is cleared, the authKeyHandle is used only in the Installation Phase. – The key flags must be configured as follow: HSE_KF_USAGE_VERIFY must be set, HSE_KF_USAGE_SIGN flag must NOT be set.

Data Fields

Type	Name	Description
hseAuthScheme_t	authScheme	The authentication scheme used to verify the SMR either during the Installation Phase or Verification phase. – If the HSE_SMR_CFG_FLAG_INSTALL_AUTH flag is set (see hseSmrConfig_t), the same authentication scheme (installation TAG) can be used to verify the authenticity of SMR during verification phase too; – Otherwise an internal authentication scheme is used. Note – The authKeyHandle must match the authentication scheme (e.g. a RSA key must be used for RSA signature). – Pure EDDSA scheme (eddsa.bHashEddsa != TRUE) is not supported for streaming installation. – Pure EDDSA scheme (eddsa.bHashEddsa != TRUE) is not supported with encrypted SMR. – EDDSA scheme Context (if used) can be maximum 16 bytes.
uint32_t	pInstAuthTag[2]	Optional - The location in external flash of the initial proof of authenticity over SMR. – If the HSE_SMR_CFG_FLAG_INSTALL_AUTH flag is set, it specifies the address(es) where the SMR original authentication TAG to be verified is located. – If the HSE_SMR_CFG_FLAG_INSTALL_AUTH flag is cleared, this field is not used (an internal authentication scheme is used). Note – The SMR authentication proof is always computed over the plain SMR. – For MAC and RSA signature, only pInstAuthTag[0] is used. – Both addresses are used for ECDSA and EDDSA signatures (specified by (r,s), with r at index 0, and s at index 1).
hseSmrDecrypt_t	smrDecrypt	Specifies the parameters for SMR decryption.

Data Fields

Type	Name	Description
uint32_t	versionOffset	Optional - The offset in SMR where the image version can be found. May be used to provide the SMR version which offers anti-rollback protection for the image against attacks during update. Note – Ignored if set to HSE_SMR_VERSION_NOT_USED (i.e. 0). – If used, it must be a valid offset within the SMR in the range [4, hseSmrEntry_t::smrSize - 4]. – Once used when installing an SMR, all subsequent updates of that SMR must have a version GREATER than the previous one. – During SMR update the version offset can be modified only having SU rights. The version value must still be GREATER than the previous one. – The version offset must be aligned to 4 bytes. – Not used for SHE based secure boot (must be set to HSE_SMR_VERSION_NOT_USED in this case).

struct hseCrEntry_t

Define the parameters of a Core Reset entry in CR table.

The CR table contains the configurations for each Application Core that HSE will use to perform the advanced secure boot.

Note

- SU right are needed to install/update a Core reset entry.
- Updating an existing CR entry is conditioned by having all preBoot and postBoot SMR(s) linked with the previous entry verified successfully (applicable only in OEM_PROD/IN_FIELD LCs).
- The core release strategy is defined by the HSE_CORE_RESET_RELEASE_ATTR_ID attribute ("ALL-AT-ONCE" or "ONE-BY-ONE")
- For devices with SD/eMMC support (e.g. HSE_H), the SMR having the source address in SD/eMMC can be used only if the following conditions are met:
  - The release core strategy is either set to "ALL-AT-ONCE" or "ONE-BY-ONE", the SMR in SD/eMMC is linked only to the first entry in the CR table (see hseAttrCoreResetRelease_t).
  - The startOption is HSE_CR_AUTO_START .
  - SMR type: either SMR is linked via preBootSmrMap or altPreBootSmrMap to the CR entry (i.e. will be loaded and verified in PRE-BOOT phase).

- SMR type: or SMR is linked via postBootSmrMap when preBootSmrMap & altPreBootSmrMap are zero (i.e. will be used for parallel secure boot - loaded in PRE-BOOT phase and verified POST-BOOT).

Data Fields

Type	Name	Description
hseAppCore_t	coreId	Identifies the core Id to be started (see hseAppCore_t for core mapping).
uint8_t	reserved0[1U]	
hseCrSanction_t	crSanction	The sanction applied if one of the SMR(s) linked to the CR entry failed the verification. Note – If at least one SMR from each PRE-BOOT bitfield (i.e. preBootSmrMap and altPreBootSmrMap) failed verification, the sanction will be applied prior to releasing the core from reset. – If on SMR specified by postBootSmrMap failed, the sanction will be applied after the core is released from reset. In this case, the HSE_CR_SANCTION_KEEP_CORE_IN_RESET option has no effect. – HSE_CR_SANCTION_DIS_INDIV_KEYS option has no effect on the behavior of the core itself, but will take effect on the key usage at run-time (see SMR flags from hseKeyInfo_t).
uint32_t	preBootSmrMap	The PRE-BOOT SMR(s) which need to be verified before releasing the core from pPassReset address. It's a 32 bits value, each bit specifies the particular SMR entry index from 0-31. HSE loads and verifies each PRE-BOOT SMR entry specified by this bitfield.
uint32_t	pPassReset	The primary address of the first instruction after a regular reset. The core starts the execution from this address if all preBootSmrMap SMR(s) have been successfully verified. Note – The pPassReset must be within a SMR specified by preBootSmrMap. – If preBootSmrMap == 0, pPassReset must be within a SMR specified by postBootSmrMap. In this case, the HSE will attempt a "parallel secure boot" for this core (see postBootSmrMap description below).

Data Fields

Type	Name	Description
uint32_t	altPreBootSmrMap	The ALT-PRE-BOOT SMR(s) which need to be verified before releasing the core from pAltReset address. It's a 32 bits value, each bit specifying the particular SMR entry index from 0-31. HSE verifies each SMR entry specified by this bitfield. The altPreBootSmrMap SMR(s) are verified ONLY if one of the SMR(s) specified by preBootSmrMap failed. Note – Once altPreBootSmrMap SMR(s) are loaded and the verification process is triggered, the preBootSmrMap SMR(s) will be considered overwritten/not loaded (see hseSmrVerifySrv_t). – If preBootSmrMap == 0, altPreBootSmrMap must be also 0 (cannot be used).
uint32_t	pAltReset	The alternative address of the first instruction after a regular reset. The core starts the execution if all altPreBootSmrMap SMR(s) have been successfully verified. Note – HSE will try to boot the core from the alternate address only if the preBootSmrMap SMR(s) verification failed. – The pAltReset must be within a SMR specified by altPreBootSmrMap. – If altPreBootSmrMap == 0, pAltReset field is ignored (can not used). – If the conditions to boot from pAltReset are not met (altPreBootSmrMap == 0, pAltReset == NULL or one of the altPreBootSmrMap SMR(s) fails) HSE will apply the sanctions as specified in crSanction field.

Data Fields

Type	Name	Description
uint32_t	postBootSmrMap	The POST-BOOT SMR(s) which need to be loaded after verifying the preBootSmrMap SMR(s) (if any). It's a 32 bits value, each bit specifying the particular SMR entry index from 0-31. HSE verifies each SMR entry specified by this bitfield. Note – If preBootSmrMap == 0 (no PRE-BOOT SMR is specified), the SMR(s) specified by postBootSmrMap will be loaded before the core is un-gated from pPassReset address. In this case, only the verification is done after the core is released from reset (POST-BOOT). This is referenced as "parallel secure boot". It can be used only if the POST-BOOT SMRs are in memory mapped locations (e.g. QuadSPI Flash, and not in SD/eMMC).
hseCrStartOption_t	startOption	Specifies if the Application Core is automaticaly released from reset or not.
uint8_t	reserved1[6U]	

struct hseSmrEntryInstallSrv_t

HSE Secure Memory Region Installation service (update or add new entry).

This service installs a SMR entry which needs to be verify during boot or runtime phase. The installation can be done in one-pass or streaming mode. The streaming mode is useful when the SMR content to be install is not entirely available in the system memory when the installation starts (OTA use case). The table below summarizes the fields needed to be provided for each access mode. Unused fields are ignored by the HSE. SMR(s) can be installed only in sequence, one at a time. This service does not use a stream ID as HSE uses internal contexts when processing in streaming mode.

Field \ Mode	One-pass	Start	Update	Finish
accessMode	*	*	*	*
entryIndex	*	*		
pSmrEntry	*	*		
pSmrData	*	*	*	*
smrDataLength	*	*	*	*
pAuthTag	*			*
authTagLength	*			*

Field \ Mode	One-pass	Start	Update	Finish
cipher.pIV	*	*		
cipher.pGmacTag	*			*

Note

- The provisioning of the original authentication tag shall be optional when LC == CUST_DEL. This allows to implement SHE use-case: autonomous bootstrap.
- In User mode, the SMR can be updated only changing the hseSmrEntry_t::pSmrSrc, hseSmrEntry_t::smrSize and hseSmrEntry_t::pInstAuthTag. Any other configuration fields (such as keyHandle, configFlags, verifMethod, etc.) of a SMR entry can only be updated if the host has SuperUser rights (for NVM Configuration).
- POST_BOOT and periodic SMR(s) source addresses cannot be in SD/MMC or external flash memory.
- The keys linked with a SMR entry (through smrFlags in hseKeyInfo_t) will become unavailable after successful installation of the SMR entry. The SMR must be verified (automatically at boot-time, periodically or via verify request at run-time) before the key can be used again.
- If a periodic SMR is updated during runtime using this service, the periodic checks for this SMR entry are disabled till the next reset.

(SHE boot):
The SMR #0 is the only SMR that can be associated to the SHE AES key BOOT_MAC_KEY as the SMR authentication key. In this case, the reference authentication tag is the CMAC value referred to as BOOT_MAC. The BOOT_MAC value can be initialized and updated via the SHE key update protocol.
In addition, when LC is set to CUST_DEL, BOOT_MAC can be automatically calculated as described below:

- On the first SMR #0 installation using BOOT_MAC_KEY, if BOOT_MAC is empty (i.e. not initialized) and if BOOT_MAC_KEY has been provisioned, the reference authentication tag is calculated by the HSE and saved in the BOOT_MAC slot. This specific installation process satisfies the SHE requirement referred to as "autonomous bootstrap configuration".
- When installing SMR #0 using the BOOT_MAC_KEY while the BOOT_MAC is already initialized, the BOOT_MAC value must be updated via the SHE key update protocol prior to issuing the SMR installation service.
- In all cases, the arrays pAuthTag and authTagLength are always discarded and should be set respectively to NULL and 0.
- If SMR #0 installation using the keyHandle for SHE(BOOT_MAC_KEY), HSE_SMR_CFG_FLAG_INSTALL_AUTH = 0 is not allowed.

Installing a NXP RFE SMRs entry:
For SAFXXXX, the protected NXP RFE images are installed configuring two image:

- one image (CODE) having the destination address only in RFE ITCM (2 SMRs, one as primary and one as back-up)
- and another image (configuration DATA) having the destination address RFE DTCM (2 SMRs, one as primary and one as back-up) All four SMR above are linked with the CR entry for RFE-M7 core (see the installation of NXP RFE CR entry). These images are encrypted and authenticated

by NXP (using the NXP ROM keys) and have specific handling on installation (refer to HSE FW reference manual)

To install a single image one must (see the example code below):

- Program the image(s) to the external flash to a chosen location, e.g. ExternalFlashAddr.
- Provide the encryption and authentication key handles of the ROM keys (#HSE_ROM_KEY_AES256_KEY2 and #HSE_ROM_KEY_RSA2048_PUB_KEY1).
- Provide the installation address of the image (can be the same as ExternalFlashAddr).
- Provide a SMR entryIndex for the installation Example of a single NXP RFE image installation (note that 4 SMRs must be installed):

```
smrEntry.pSmrSrc                              = ExternalFlashAddr;
smrEntry.authKeyHandle                        = HSE_ROM_KEY_RSA2048_PUB_KEY1;
smrEntry.smrDecrypt.decryptKeyHandle          = HSE_ROM_KEY_AES256_KEY2;
hseDescriptor.srvId                           = HSE_SRV_ID_SMR_ENTRY_INSTALL;
hseDescriptor.smrEntryInstallReq.accessMode = HSE_ACCESS_MODE_ONE_PASS;
hseDescriptor.smrEntryInstallReq.entryIndex = Ind;
hseDescriptor.smrEntryInstallReq.pSmrEntry  = HSE_PTR_TO_HOST_ADDR(&smrEntry);
hseDescriptor.smrEntryInstallReq.pSmrData   = InstallationAddr;
response = SendDescToHse(&hseDescriptor);
```

Constraints and additional notes:

- Only HSE_ACCESS_MODE_ONE_PASS access mode can be used.
- All parameters not specified in the above example are ignored.

Data Fields

Type	Name	Description
hseAccessMode_t	accessMode	INPUT: Specifies the access mode: ONE-PASS, START, UPDATE, FINISH. Note – Streaming is not supported for Pure EDDSA scheme (eddsa.bHashEddsa != TRUE). STREAMING USAGE: Used in all steps.
uint8_t	entryIndex	INPUT: Identifies the index of SMR entry (in the SMR table) which has to be installed/updated. Refer to HSE_NUM_OF_SMR_ENTRIES STREAMING USAGE: Used in START.
uint8_t	reserved[2U]	
uint64_t	pSmrEntry	INPUT: Address of SMR entry structure containing the configuration properties to be installed (refer to hseSmrEntry_t).

Data Fields

Type	Name	Description
uint64_t	pSmrData	INPUT: The address where SMR data to be installed is located. STREAMING USAGE: Used in all steps, but ignored if smrDataLength is zero. Note – If SMR#0 is used for SHE-boot and the BOOT_MAC slot is empty then the BOOT_MAC is be calculated by HSE FW at the time of SMR installation. – For HSE_H/M devices, if the SMR is flashed in SD/eMMC, the application need to copy SMR data in System RAM (and pSmrData must point to that System RAM address)
uint32_t	smrDataLength	INPUT: The length of the SMR data. In case of streaming mode, the total size of SMR is computed by summing the length of SMR chunks provided during Update/Finish STREAMING USAGE: Used in all steps. – START: Must be a multiple of 64/128 bytes, or zero. Cannot be zero for HMAC. – UPDATE: Must be a multiple of 64/128 bytes. Cannot be zero. Refrain from issuing the service request, instead of passing zero. – FINISH: Can be any value (For CMAC & XCBC-MAC, zero length is invalid). Note – Depending on the algorithm used, the length must be: · Multiple of 64 bytes: · CMAC, GMAC, XCBC-MAC; · HMAC, RSA, ECDSA with underlying hash: SHA1, SHA2_224, SHA2_256; · Multiple of 128 bytes: · HMAC, RSA, ECDSA with underlying hash: SHA2_384, SHA2_512, SHA2_512_224, SHA2_512_256; – Miyaguchi-Preneel not supported as hash algorithm; – HMAC: SHA3 not supported as hash algorithm. – Pure EDDSA scheme (eddsa.bHashEddsa != TRUE): not supported in streaming mode.

Data Fields

Type	Name	Description
uint64_t	pAuthTag[2]	INPUT: The address where SMR Original authentication tag to be verify is located. Note – The SMR authentication proof is always computed over the plain SMR. – For MAC and RSA signature, only pAuthTag[0] is used. – Both pointers are used for ECDSA and EDDSA signatures (specified as (r,s), with r at index 0, and s at index 1). – ignored if SMR#0 is SHE-boot. STREAMING USAGE: Used in FINISH.
uint16_t	authTagLength[2]	INPUT: The length of the SMR authentication proof (tag/signature). Note – For MAC and RSA signature, only authTagLength[0] is used. – Both pointers are used for ECDSA and EDDSA signatures (specified the length of (r,s), with r at index 0, and s at index 1). – Ignored if SMR#0 is used for SHE-boot. – The MAC tag size must be minimum 16 bytes. – RSA signature size must be HSE_BYTES_TO_BITS(keyBitLength); – R or S size for ECDSA/EDDSA signature must be HSE_BYTES_TO_BITS(keyBitLength) STREAMING USAGE: Used in FINISH.

Data Fields

Type	Name	Description
struct hseSmrEntryInstallSrv_t.cipher	cipher	INPUT: Optional - Cipher parameters used for installing encrypted SMR(s). Note – These parameters are use only if hseSmrDecrypt_t::decryptKeyHandle != HSE_SMR_DECRYPT_KEY_HANDLE_NOT_USED (see hseSmrDecrypt_t). – The pointers that are specified in this structure shall be provided from a memory-mapped location (QSPI/LPDDR4/RAM). – In case an SMR lying in SD/eMMC external flash is installed, a copy of GMAC tag (if used) shall be done in RAM and provided via the fields below. The pointers provided via hseSmrEntryInstallSrv_t::pSmrEntry shall point to the location in external flash that will be used by HSE at boot-time.

struct hseSmrVerifySrv_t

HSE Secure Memory Region verification service.

This service starts the on-demand verification of a secure memory region by specifying the index in the SMR table.

Data Fields

Type	Name	Description
uint8_t	entryIndex	INPUT: Specifies the entry in the SMR table to be verified (max HSE_NUM_OF_SMR_ENTRIES). This service loads and verifies on-demand an SMR entry (in SRAM). Note (HSE_H/M) – The SMR(s) used in CORE RESET table can be verified on-demand only if they were loaded before in SRAM or the BOOT_SEQ = 0. Otherwise, an error will be reported (NOT ALLOWED). – The SMR(s) that are not part of the CORE RESET table configuration can be loaded and verified at run time. Note that on the second call of this service, the HSE will only performed the verification in SRAM. Using this service, the SMR(s) can not be loaded and verified from SD/MMC memory.
uint8_t	reserved	RFU. Set to 0 for compatibility with future updates.
hseSmrVerificationOptions_t	options	INPUT: Options for customizing the on-demand SMR verification (see hseSmrVerificationOptions_t). Values not defined or not applicable are ignored.

struct hseSmrEntryEraseSrv_t

SMR entry erase service.

This service erases one SMR entry from the internal HSE memory.

Note

- SuperUser (SU) access rights with privileges over HSE_SYS_AUTH_NVM_CONFIG data are required to perform this service.

Data Fields

Type	Name	Description
uint8_t	smrEntryInd	INPUT: Identifies the index in the SMR table for the entry to be erased.
uint8_t	reserved[3]	

struct hseCrEntryInstallSrv_t

Core Reset entry install (update or add new entry)

This service updates an existing or add a new entry in the Core Reset table.

Note

- SMR entries that are linked with the installed CR entry (via preBoot/altPreBoot/postBoot SMR maps) must be installed in HSE prior to the CR installation.
- SuperUser rights (for NVM Configuration) are needed to perform this service.
- Updating an existing CR entry is conditioned by having all preBoot and postBoot SMR(s) linked with the previous entry verified successfully (applicable only in OEM_PROD/IN_FIELD LCs).

Installing a NXP RFE Core Reset entry:
For SAFXXXX, the HSE FW provides the functionality of installing the protected NXP RFE images (e.g. RFE - CORE1 on SAF85XX platform). This CR entry are linked with the NXP RFE images (4 SMRs, two for primary and two for back-up) and have a specific handling on installation (refer to HSE FW reference manual). To install such an entry one must (refer to the example code below):

- Install the corresponding NXP SMR images (see SMR installation NXP RFE images)
- Link the NXP RFE images (4 SMRs) to the CR entry to be installed.
- Provide the crEntryIndex index (CR_IND_RFE). Example of RFE CR installation for NXP RFE im:

```
crEntry.coreId = HSE_APP_CORE1;
crEntry.preBootSmrMap = ((1UL « SMR_IND_RFE_ITCM_PRIMARY) | (1UL « SMR_IND_RFE_DTCM_PRIMARY));
crEntry.altPreBootSmrMap = ((1UL « SMR_IND_RFE_ITCM_BACKUP) | (1UL « SMR_IND_RFE_DTCM_BACKUP));
desc.srvId = HSE_SRV_ID_CORE_RESET_ENTRY_INSTALL;
desc.crEntryInstallReq.crEntryIndex = CR_IND_RFE;
desc.crEntryInstallReq.pCrEntry = HSE_PTR_TO_HOST_ADDR(&crEntry);
response = SendHseDescriptor(&desc);
```

Constraints and additional notes:

- The referenced NXP SMR must be installed prior to CR entry installation.
- All parameters not specified in the above example are ignored.

Data Fields

Type	Name	Description
uint8_t	crEntryIndex	INPUT: Identifies the index in the Core Reset table which has to be added/updated Refer to HSE_NUM_OF_CORE_RESET_ENTRIES .
uint8_t	reserved[3]	
uint64_t	pCrEntry	INPUT: Address of Core Reset entry structure (refer to hseCrEntry_t).

struct hseCrOnDemandBootSrv_t

On-demand boot of a Core Reset entry.

This service triggers the loading, verification and reset release of a core that is not automatically started (at boot time).

Note

- This service can be called only once and only for the Core Reset entries that have the startOption option set to HSE_CR_ON_DEMAND .
- Using this service, the SMR(s) can not be loaded and verified from SD/MMC memory.

Data Fields

Type	Name	Description
uint8_t	crEntryIndex	INPUT: Identifies the index in the Core Reset table which has to be released from reset after loading and verification. Refer to HSE_NUM_OF_CORE_RESET_ENTRIES .
uint8_t	reserved[3]	

struct hseCrEntryEraseSrv_t

Core Reset entry erase service.

This service erases one Core Reset entry from the internal HSE memory.

Note

- SuperUser (SU) access rights with privileges over HSE_SYS_AUTH_NVM_CONFIG data are required to perform this service.

Data Fields

Type	Name	Description
uint8_t	crEntryInd	INPUT: Identifies the index in the Core Reset table for the entry to be erased.
uint8_t	reserved[3]	

struct hseSmrEntryInstallSrv_t.cipher

INPUT: Optional - Cipher parameters used for installing encrypted SMR(s).

Note

- These parameters are use only if hseSmrDecrypt_t::decryptKeyHandle != HSE_SMR_DECRYPT_KEY_HANDLE_NOT_USED (see hseSmrDecrypt_t).

- The pointers that are specified in this structure shall be provided from a memory-mapped location (QSPI/LPDDR4/RAM).
- In case an SMR lying in SD/eMMC external flash is installed, a copy of GMAC tag (if used) shall be done in RAM and provided via the fields below.
  The pointers provided via hseSmrEntryInstallSrv_t::pSmrEntry shall point to the location in external flash that will be used by HSE at boot-time.

Data Fields

Type	Name	Description
uint64_t	pIV	INPUT: Initialization Vector/Nonce. The length of the IV is 16 bytes. Will be stored by HSE internally. STREAMING USAGE: Used in START.
uint64_t	pGmacTag	INPUT: Optional - tag used for AEAD. The length considered for the GMAC tag is 16 bytes (if used - see hseSmrDecrypt_t). Note – Used only if hseSmrDecrypt_t::pGmacTag != NULL. – Must point to the same data as hseSmrDecrypt_t::pGmacTag, however the memory location may differ (QSPI/LPDDR4/RAM vs QSPI/LPDDR4/SD/eMMC). STREAMING USAGE: Used in FINISH.
uint64_t	pAAD	INPUT: Optional - the AAD used for AEAD. The length considered for the AAD is specified via pSmrEntry->smrDecrypt.aadLength (see hseSmrDecrypt_t). Note – Used only if length is not zero. – Must point to the same data as pSmrEntry->smrDecrypt.pAAD, however the memory location may differ (QSPI/LPDDR4/RAM vs QSPI/LPDDR4/SD/eMMC). STREAMING USAGE: Used in START.

Macro Definition Documentation

HSE_SMR_DECRYPT_KEY_HANDLE_NOT_USED

```
#define HSE_SMR_DECRYPT_KEY_HANDLE_NOT_USED (0UL)
```

Decryption of SMR is not used.

HSE_SMR_VERSION_NOT_USED

```
#define HSE_SMR_VERSION_NOT_USED (0UL)
```

SMR version is not used (value to ignore hseSmrEntryInstallSrv_t::versionOffset field).

HSE_CR_SANCTION_DIS_INDIV_KEYS

```
#define HSE_CR_SANCTION_DIS_INDIV_KEYS ((hseCrSanction_t)0x7433U)
```

Disable individual keys; if at least one SMR entry specified by the key smrFlags (see hseKeyInfo_t) is not verified, the key can not be used.

HSE_CR_SANCTION_KEEP_CORE_IN_RESET

```
#define HSE_CR_SANCTION_KEEP_CORE_IN_RESET ((hseCrSanction_t)0x7455U)
```

The HSE keeps in reset the core (if the verification of at least one SMR entry fails)

HSE_CR_SANCTION_RESET_SOC

```
#define HSE_CR_SANCTION_RESET_SOC ((hseCrSanction_t)0x8B17U)
```

The HSE reset the SoC.

HSE_CR_SANCTION_DIS_ALL_KEYS

```
#define HSE_CR_SANCTION_DIS_ALL_KEYS ((hseCrSanction_t)0x8B1EU)
```

Disable all keys.

HSE_CR_AUTO_START

```
#define HSE_CR_AUTO_START ((hseCrStartOption_t)0x35A5U)
```

The Core is released from reset automatically at startup (if the coresponding SMR(s) are loaded and verified).

HSE_CR_ON_DEMAND

```
#define HSE_CR_ON_DEMAND ((hseCrStartOption_t)0x5567U)
```

The Core is not released from reset automatically; this can be triggered by another Application Core using hseCrOnDemandBootSrv_t service.

HSE_SMR_CFG_FLAG_QSPI_FLASH

```
#define HSE_SMR_CFG_FLAG_QSPI_FLASH ((hseSmrConfig_t)0x0U)
```

Identifies the Interface (where the SMR needs to be copied from)

HSE_SMR_CFG_FLAG_SD_FLASH

```
#define HSE_SMR_CFG_FLAG_SD_FLASH ((hseSmrConfig_t)0x2U)
```

Identifies the Interface (where the SMR needs to be copied from)

HSE_SMR_CFG_FLAG_MMC_FLASH

```
#define HSE_SMR_CFG_FLAG_MMC_FLASH ((hseSmrConfig_t)0x3U)
```

Identifies the Interface (where the SMR needs to be copied from)

HSE_SMR_CFG_FLAG_INSTALL_AUTH

```
#define HSE_SMR_CFG_FLAG_INSTALL_AUTH ((hseSmrConfig_t)(1U << 2U))
```

If it is set, the authentication scheme and tag provided during installation phase (installation TAG) are used also during the verification phase. If it is cleared, during installation HSE will compute and store an internal hash digest (SHA2-256) During verification phase, HSE will use this internal digest.

Note

- If the HSE_SMR_CFG_FLAG_INSTALL_AUTH flag is cleared and SHE-boot is used (SMR #0 with BOOT_MAC_KEY), HSE FW will return HSE_SRV_RSP_NOT_ALLOWED on SMR#0 installation request.

HSE_SMR_CFG_FLAG_AUTH_AAD

```
#define HSE_SMR_CFG_FLAG_AUTH_AAD ((hseSmrConfig_t)(1U << 3U))
```

If this bit is set, the authentication is computed over [AAD || Plain] image.

Note

- The SMR has to be configured with AEAD-GCM decryption (i.e. AAD and GMAC tag are provided as part of decryption parameters).

HSE_SMR_VERIFICATION_OPTION_NONE

```
#define HSE_SMR_VERIFICATION_OPTION_NONE ((hseSmrVerificationOptions_t)0UL) /**
@brief Default verification of the SMR at run-time. */
```

HSE_SMR_VERIFICATION_OPTION_NO_LOAD

```
#define HSE_SMR_VERIFICATION_OPTION_NO_LOAD ((hseSmrVerificationOptions_t)(3UL <<
0U)) /** @brief SMR is verified from the external flash (using pSmrSrc address) even
if pSmrDest is specified or if already loaded. Can be used only if SMR is in a
memory mapped external flash (e.g. QSPI/LPDDR4 and not SD/eMMC). Additionally the
SMR cannot be encrypted. */
```

HSE_SMR_VERIFICATION_OPTION_RELOAD

```
#define HSE_SMR_VERIFICATION_OPTION_RELOAD ((hseSmrVerificationOptions_t)(3UL <<
2U)) /** @brief SMR is loaded from the external flash and verified even if it is
already loaded. Can be used only if SMR is in a memory mapped external flash (e.g.
QSPI/LPDDR4 and not SD/eMMC). */
```

Typedef Documentation

hseCrSanction_t

```
typedef uint16_t hseCrSanction_t
```

CORE sanctions to be applied if the verification of at least one SMR entry fails on both Primary and Backup SMR maps as defined in CR entry (hseCrEntry_t::preBootSmrMap and hseCrEntry_t::altPreBootSmrMap)

hseCrStartOption_t

```
typedef uint16_t hseCrStartOption_t
```

The start option for a Core Reset Entry.

hseSmrConfig_t

```
typedef uint8_t hseSmrConfig_t
```

Specifies the boot interface (where the SMR needs to be copied from).

Note

- For HSE_H/M, the SMR source memory can be:
  - QSPI Flash
  - SD card
  - MMC
  - LPDDR4 Flash (used only for S32ZE devices)
  - for different SMR(s), any combination of the above memory interfaces, except MMC and SD (e.g. QSPI/LPDDR4 Flash and SD, QSPI/LPDDR4 Flash and MMC).
- For HSE_B, the source memory flags (QSPI/SD/MMC/LPDDR4) are not used.

hseSmrVerificationOptions_t

```
typedef uint16_t hseSmrVerificationOptions_t
```

Options for customizing SMR run-time verification.

7 SHE Specification

7.1 HSE SHE Specification Services

Data Structures

- struct hseSheLoadKeySrv_t
- struct hseSheLoadPlainKeySrv_t
- struct hseSheExportRamKeySrv_t
- struct hseSheGetIdSrv_t

Data Structure Documentation

struct hseSheLoadKeySrv_t

SHE load key service.

Load a SHE key into the HSE according to the SHE memory update protocol.

Note

> The SHE keys can be used for any supported AES operations (e.g. AES with all block modes, AEAD etc.) given the proper flags are set. One exception is BOOT_MAC_KEY, which can only be used with CMAC verify operation.

Data Fields

Type	Name	Description
hseKeyGroupIdx_t	sheGroupIndex	Group Index for the SHE NVM catalog, ranging from 0 to 4. This parameter also decides the KDF input constants "CENC" & "CMAC" to be used in memory update protocol operation. – For (1 <= keyID <= 3), this parameter is ignored and taken as zero to decide "CENC" and "CMAC". – For (keyID = 14) and (4 <= authID <= 13), this parameter is used to select auth-user-key (authID) group and to decide "CENC" & "CMAC". – For (keyID = 14) and (authID = 0), this parameter is ignored & taken as zero to decide "CENC" & "CMAC". – For (4 <= keyID <= 13) and (4 <= authID <= 13), given that keyID = authID, this parameter is used to select both user-key (keyID) group & auth-user-key (authID) group and to decide "CENC" & "CMAC". – For (4 <= keyID <= 13) and (authID = 1), this parameter is used to select user-key (keyID) group and to decide "CENC" & "CMAC".
uint8_t	reserved[3]	
uint64_t	pM1	INPUT: Pointer to M1.
uint64_t	pM2	INPUT: Pointer to M2.
uint64_t	pM3	INPUT: Pointer to M3.
uint64_t	pM4	OUTPUT: Pointer to M4.
uint64_t	pM5	OUTPUT: Pointer to M5.

struct hseSheLoadPlainKeySrv_t

SHE load plain key service.

Load a SHE RAM key from plain text

Data Fields

Type	Name	Description
uint64_t	pKey	INPUT: Pointer to the unencrypted key.

struct hseSheExportRamKeySrv_t

SHE export RAM key service.

Export a SHE RAM key in the format used for re-loading with SHE Load key. This export can happen only if RAM key was loaded using SHE RAM plain key service.

Data Fields

Type	Name	Description
uint64_t	pM1	OUTPUT: Pointer to M1.
uint64_t	pM2	OUTPUT: Pointer to M2.
uint64_t	pM3	OUTPUT: Pointer to M3.
uint64_t	pM4	OUTPUT: Pointer to M4.
uint64_t	pM5	OUTPUT: Pointer to M5.

struct hseSheGetIdSrv_t

SHE get ID service.

Returns the Identity (UID) and the value of the status register protected by a MAC over a challenge and the data. If MASTER_ECU_KEY is empty, the returned MAC has to be set to zero.

Note

> For HSE_H/M, the UID is read by HSE from system fuses. The application must provide read-only access (xRDC restriction) to HSE to read the UID.

Data Fields

Type	Name	Description
uint64_t	pChallenge	INPUT: Pointer to 128-bit Challenge.
uint64_t	pId	OUTPUT: Pointer to 120-bit UID.
uint64_t	pSreg	OUTPUT: Pointer to 8-bit Status Register (SREG). Refer to HSE Status for status related information (boot, debug, etc.)
uint64_t	pMac	OUTPUT: Pointer to 128-bit CMAC(CHALLENGE | ID | SREG) using MASTER_ECU_KEY as key.

8 Monotonic Counters Services

8.1 HSE Monotonic Counters

Data Structures

- struct hseIncrementCounterSrv_t
- struct hseReadCounterSrv_t
- struct hsePublishLoadCntTblSrv_t
- struct hseConfigSecCounterSrv_t

Macros

Type: (implicit C type)	
Name	**Value**
HSE_NVM_CONTAINER_CNT_TBL_SIZE	(HSE_NUM_OF_MONOTONIC_COUNTERS ∗ sizeof(uint64_t)) + 48U

Type: hseNvmCntTblAction_t	
Name	**Value**
HSE_NVM_CNT_TBL_ACTION_PUBLISH	0x01U
HSE_NVM_CNT_TBL_ACTION_LOAD	0x02U

Typedefs

- typedef uint8_t hseNvmCntTblAction_t

Note

For HSE_B (devices with internal flash), the first service request after reset that depends on Monotonic Counters, i.e., hseConfigSecCounterSrv_t, hseReadCounterSrv_t, hseIncrementCounterSrv_t and hseCmacWithCounterSrv_t will take more time because of Monotonic Counters Initialization. Further requests will take usual time.

Data Structure Documentation

struct hseIncrementCounterSrv_t

Increment a monotonic counter service with a specific value.

– For HSE_H/M, the counters are volatile. Host application has to publish/load the monotonic counter table using hsePublishLoadCntTblSrv_t service.

– For HSE_B, the host application shall use the hseConfigSecCounterSrv_t service to initialize and configure the secure counters.

- If the counter is saturated, an error is reported.

Data Fields

Type	Name	Description
uint32_t	counterIndex	INPUT: The counter Index.
uint32_t	value	INPUT: The value to be added.

struct hseReadCounterSrv_t

Read a monotonic counter service.

Data Fields

Type	Name	Description
uint32_t	counterIndex	INPUT: The counter Index.
uint64_t	pCounterVal	OUTPUT: The address where the counter value is returned (a uint64_t value).

struct hsePublishLoadCntTblSrv_t

Publish or load the monotonic counter table.

This is supported only for HSE_H/M devices and should be used to publish/load the monotonic counter table in NVM

Data Fields

Type	Name	Description
hseNvmCntTblAction_t	action	INPUT: Publish/load the NVM container for the Monotonic Counter table.
uint8_t	reserved[3]	
uint64_t	pNvmContainerCntTbl	OUTPUT: The address of the NVM container for the Monotonic Counter table. The size of the NVM container is HSE_NVM_CONTAINER_CNT_TBL_SIZE .

struct hseConfigSecCounterSrv_t

Initialize and configure a secure counter.

HSE supports 16 X 64 bits secure counters, each counter having associated a CounterIndex from 0 to 15. By default, all the counters are disabled.
The secure counter (SC) consists of 2 separate bitfields: Rollover Protection (RP) + Volatile Counter (VC).

Monotonic Counters Services

HSE stores the secure counter in data flash each time the Rollover Protection (RP) is updated.
The purpose of this service is to enable the secure counter and configure the Rollover Protection bitfield size. The objective is to reduce the rate at which NVM programming operations occur.
If the secure counter is already configured and this service is called, HSE re-configures the counter with the new Rollover Protection (RP) and reset it to 0.

Note

- SuperUser rights are needed to configure/enable the monotonic counters.
- For HSE_B (devices with internal flash)
  - WARNING: The HSE erases a flash sector after 511 Rollover Protection updates in data flash. The number of data flash erases is limited to 100.000. The application shall configure each secure counter depending on the SC update rate and the number of enabled counters.
  - The secure counter configuration is stored in data flash each time hseConfigSecCounterSrv_t is called.
  - If RPBitSize = 64bits, the HSE stores the SC in flash each time is updated.
- For HSE_H/M (flashless devices)
  - The RPBitSize is configured for all the enabled secure counters. If the RP of a counter is updated, a warning event is trigger called HSE_WA_PUBLISH_COUNTER_TBL through MUB_GSR register. The application shall clear the warning bit (W1C) and use the hsePublishLoadCntTblSrv_t service to publish and store the counter table in the external flash. Note that the counter table must be loaded at initialization time by the application (anti-rollback protection is not supported).

Example: Let's consider the RPBitSize = 40bits and SC = 0x0000000000000001.
This means Rollover Protection (40bits) + Volatile Counter (24bits).
The secure counter (SC) will be stored in flash if the incremental value is >= 0xFFFFFF. Otherwise, the counter will be updated but not stored.
SC = 0x0000000000000001+0xFFFFFF = 0x0000000001000000 (RP was changed)

Data Fields

Type	Name	Description
uint32_t	counterIndex	INPUT: - For HSE_B, specifies the counter Index. – For HSE_H/M, specifies the number of counters to be enabled (max 16). E.g. if it is set to 5, the counters with the index from 0 to 4 are enabled.
uint8_t	RPBitSize	INPUT: The Rollover Protection bit size (refer to service comments). It shall be >= 32 bits and <= 64 bits.
uint8_t	reserved[3]	

Macro Definition Documentation

HSE_NVM_CONTAINER_CNT_TBL_SIZE

```
#define HSE_NVM_CONTAINER_CNT_TBL_SIZE (( HSE_NUM_OF_MONOTONIC_COUNTERS*
sizeof(uint64_t)) + 48U)
```

The size of the NVM container for the Monotonic Counter table (in bytes).

HSE_NVM_CNT_TBL_ACTION_PUBLISH

```
#define HSE_NVM_CNT_TBL_ACTION_PUBLISH ((hseNvmCntTblAction_t)0x01U)
```

Publish the Nvm Container for the Monotonic Counter table.

HSE_NVM_CNT_TBL_ACTION_LOAD

```
#define HSE_NVM_CNT_TBL_ACTION_LOAD ((hseNvmCntTblAction_t)0x02U)
```

Load the Nvm Container for the Monotonic Counter table.

Typedef Documentation

hseNvmCntTblAction_t

```
typedef uint8_t hseNvmCntTblAction_t
```

Publish or load the NVM container for the Monotonic Counter table.

9 Random Number Generator Services

9.1 HSE Random Number Generator services

Data Structures

- struct hseGetRandomNumSrv_t

Macros

Type: hseRngClass_t	
Name	**Value**
HSE_RNG_CLASS_DRG3	0U
HSE_RNG_CLASS_DRG4	1U
HSE_RNG_CLASS_PTG3	2U

Typedefs

- typedef uint8_t hseRngClass_t

Data Structure Documentation

struct hseGetRandomNumSrv_t

Get random number service.

Note

- When the HSE_STATUS_RNG_INIT_OK status flag is cleared (failed), the call of any service that requires a random number (e.g. Get Random Number, ECDSA signature generate etc.) triggers a RNG re-initialization before requesting the random number. If the RNG re-initialization is executed successfully, the HSE_STATUS_RNG_INIT_OK status flag is set to 1.
- If the HSE_SRV_RSP_RNG_INIT_IN_PROGRESS status is received (RNG initialization is in progress), the application must try when the HSE_STATUS_RNG_INIT_OK status is set in the FSR register.

Data Fields

Type	Name	Description
hseRngClass_t	rngClass	INPUT: The RNG class.
uint8_t	reserved[3]	
uint32_t	randomNumLength	INPUT: Length of the generated random number in bytes. The maximum value for one request is 512 bytes.

Data Fields

Type	Name	Description
uint64_t	pRandomNum	OUTPUT: The address where the random number will be stored.

Macro Definition Documentation

HSE_RNG_CLASS_DRG3

```
#define HSE_RNG_CLASS_DRG3 ((hseRngClass_t)0U)
```

DRG.3 class uses the RNG engine with prediction resistance disabled. This is the most efficient class in terms of performance.

HSE_RNG_CLASS_DRG4

```
#define HSE_RNG_CLASS_DRG4 ((hseRngClass_t)1U)
```

DRG.4 (AIS-20/SP800-90A) class uses the RNG engine with prediction resistance enabled. Using the prediction resistance will impact the performance, as every call to Get Random invokes reseed internally.

HSE_RNG_CLASS_PTG3

```
#define HSE_RNG_CLASS_PTG3 ((hseRngClass_t)2U)
```

PTG.3 (AIS 31/SP800-90B) class uses the RNG engine with prediction resistance enabled and will reseed for each 16 bytes of data. This is the most costly class in terms of performance.

Typedef Documentation

hseRngClass_t

```
typedef uint8_t hseRngClass_t
```

HSE RNG classes.

Note

> Additional entropy (personalization string) is not needed to be provide by user. The entropy generated by the TRNG already ensures this with high probability.

10 Network Protocol Acceleration Services

11 Common Types and Definitions

11.1 HSE Common Types

Data Structures

- struct hseSrvMetaData_t
- struct hseRsaOAEPScheme_t
- struct hseEcdsaScheme_t
- struct hseEddsaSignScheme_t
- struct hseRsaPssSignScheme_t
- struct hseRsaPkcs1v15Scheme_t
- struct hseSignScheme_t
- struct hseSymCipherScheme_t
- struct hseAeadScheme_t
- struct hseRsaCipherScheme_t
- union hseCipherScheme_t
- struct hseCmacScheme_t
- struct hseHmacScheme_t
- struct hseGmacScheme_t
- struct hseMacScheme_t
- union hseAuthScheme_t
- struct hseScatterList_t
- union hseSignScheme_t.sch
- union hseRsaCipherScheme_t.sch
- union hseMacScheme_t.sch

Macros

Type: (implicit C type)	
Name	**Value**
HSE_MAX_DESCR_SIZE	256U
HSE_ALL_MU_MASK	HSE_MU0_MASK | HSE_MU1_MASK | HSE_MU2_MASK | HSE_MU3_MASK
HSE_SGT_OPTION_INPUT_OUTPUT_MASK	HSE_SGT_OPTION_INPUT | HSE_SGT_OPTION_OUTPUT
HSE_SGT_FINAL_CHUNK_BIT_MASK	0x40000000UL

Type: hseSGTOption_t	
Name	**Value**
HSE_SGT_OPTION_NONE	0U
HSE_SGT_OPTION_INPUT	1U << 0U
HSE_SGT_OPTION_OUTPUT	1U << 1U

Type: hseMacAlgo_t

Name	Value
HSE_MAC_ALGO_CMAC	0x11U
HSE_MAC_ALGO_GMAC	0x12U
HSE_MAC_ALGO_XCBC_MAC	0x13U
HSE_MAC_ALGO_HMAC	0x20U

Type: hseCipherBlockMode_t	
Name	**Value**
HSE_CIPHER_BLOCK_MODE_NULL	0U
HSE_CIPHER_BLOCK_MODE_CTR	1U
HSE_CIPHER_BLOCK_MODE_CBC	2U
HSE_CIPHER_BLOCK_MODE_ECB	3U
HSE_CIPHER_BLOCK_MODE_CFB	4U
HSE_CIPHER_BLOCK_MODE_OFB	5U

Type: hseSignSchemeEnum_t	
Name	**Value**
HSE_SIGN_ECDSA	0x80U
HSE_SIGN_EDDSA	0x81U
HSE_SIGN_RSASSA_PKCS1_V15	0x93U
HSE_SIGN_RSASSA_PSS	0x94U

Type: hseAccessMode_t	
Name	**Value**
HSE_ACCESS_MODE_ONE_PASS	0U
HSE_ACCESS_MODE_START	1U
HSE_ACCESS_MODE_UPDATE	2U
HSE_ACCESS_MODE_FINISH	3U

Type: hseMuMask_t	
Name	**Value**
HSE_MU0_MASK	1U << 0U
HSE_MU1_MASK	1U << 1U
HSE_MU2_MASK	1U << 2U
HSE_MU3_MASK	1U << 3U

Type: hseAppCore_t

Name	Value
HSE_APP_CORE0	0U
HSE_APP_CORE1	1U
HSE_APP_CORE2	2U
HSE_APP_CORE3	3U
HSE_APP_CORE4	4U
HSE_APP_CORE5	5U
HSE_APP_CORE6	6U
HSE_APP_CORE7	7U
HSE_APP_CORE8	8U
HSE_APP_CORE9	9U
HSE_APP_CORE10	10U
HSE_APP_CORE11	11U
HSE_APP_CORE12	12U
HSE_APP_CORE13	13U
HSE_APP_CORE14	14U
HSE_APP_CORE15	15U

Type: hseAuthDir_t	
Name	**Value**
HSE_AUTH_DIR_VERIFY	0U
HSE_AUTH_DIR_GENERATE	1U

Type: hseCipherAlgo_t	
Name	**Value**
HSE_CIPHER_ALGO_NULL	0x00U
HSE_CIPHER_ALGO_AES	0x10U

Type: hseRsaAlgo_t	
Name	**Value**
HSE_RSA_ALGO_NO_PADDING	0x90U
HSE_RSA_ALGO_RSAES_OAEP	0x91U
HSE_RSA_ALGO_RSAES_PKCS1_V15	0x92U

Type: hseCipherDir_t	
Name	**Value**
HSE_CIPHER_DIR_DECRYPT	0U
HSE_CIPHER_DIR_ENCRYPT	1U

Type: hseHashAlgo_t	
Name	**Value**
HSE_HASH_ALGO_NULL	0U
HSE_HASH_RESERVED1	1U
HSE_HASH_ALGO_SHA_1	2U
HSE_HASH_ALGO_SHA2_224	3U
HSE_HASH_ALGO_SHA2_256	4U
HSE_HASH_ALGO_SHA2_384	5U
HSE_HASH_ALGO_SHA2_512	6U
HSE_HASH_ALGO_SHA2_512_224	7U
HSE_HASH_ALGO_SHA2_512_256	8U
HSE_HASH_ALGO_SHA3_224	9U
HSE_HASH_ALGO_SHA3_256	10U
HSE_HASH_ALGO_SHA3_384	11U
HSE_HASH_ALGO_SHA3_512	12U
HSE_HASH_ALGO_MP	13U

Type: hseAuthCipherMode_t	
Name	**Value**
HSE_AUTH_CIPHER_MODE_CCM	0x11U
HSE_AUTH_CIPHER_MODE_GCM	0x12U

Typedefs

- typedef uint8_t hseMuMask_t
- typedef uint8_t hseSGTOption_t
- typedef uint8_t hseAccessMode_t
- typedef uint8_t hseHashAlgo_t
- typedef uint8_t hseCipherAlgo_t
- typedef uint8_t hseCipherBlockMode_t
- typedef uint8_t hseCipherDir_t
- typedef uint8_t hseAuthCipherMode_t
- typedef uint8_t hseAuthDir_t
- typedef uint8_t hseMacAlgo_t
- typedef uint8_t hseSignSchemeEnum_t
- typedef uint8_t hseRsaAlgo_t
- typedef uint8_t hseAppCore_t
- typedef uint32_t hseSrvId_t
- typedef uint8_t hseStreamId_t
- typedef uint32_t hseKeyHandle_t
- typedef uint8_t hseKeyGroupIdx_t
- typedef uint8_t hseKeySlotIdx_t
- typedef uint32_t hseNoScheme_t

Data Structure Documentation

struct hseSrvMetaData_t

HSE service metadata.

Each service has a metadata (e.g. priority)

Data Fields

Type	Name	Description
uint8_t	reserved[4]	For future use.

struct hseRsaOAEPScheme_t

RSAES OAEP Scheme.

Includes parameters needed for RSAES OAEP encryption/ decryption.

Data Fields

Type	Name	Description
hseHashAlgo_t	hashAlgo	INPUT: The Hash algorithm for RSA OAEP padding.
uint8_t	reserved[3]	
uint32_t	labelLength	INPUT: Optional OAEP label length (it can be 0). Must be less than 128.
uint64_t	pLabel	INPUT: Optional OAEP label (it can be NULL if label length is 0). Must be less than 128 bytes long.

struct hseEcdsaScheme_t

ECDSA signature scheme.

Includes parameters needed for ECDSA signature generate/verify.

Data Fields

Type	Name	Description
hseHashAlgo_t	hashAlgo	INPUT: The hash algorithm used to hash the input before applying the ECDSA operation. Must not be HSE_HASH_ALGO_NULL .
uint8_t	reserved[3]	

struct hseEddsaSignScheme_t

EDDSA signature scheme.

Includes parameters needed for EDDSA signature generate/verify.

EdDSA signature mode vs. EdDDA sign scheme parameters:

EdDSA signature mode	EdDsa sign scheme parameters
PureEdDSA	bHashEddsa == FALSE && contextLength/pContext == 0
Context	bHashEddsa == FALSE && contextLength/pContext != 0
HashEdDSA	bHashEddsa == TRUE

Data Fields

Type	Name	Description
bool_t	bHashEddsa	INPUT: Whether to pre-hash the input, and perform a HashEddsa signature.
uint8_t	contextLength	INPUT: The length of the EDDSA context. Length of zero means no context.
uint8_t	reserved[2]	
uint64_t	pContext	INPUT: The EDDSA context. Ignored if contextLength is zero. Must remain unchanged until the signing operation is finished (especially in streaming), or the signature will be incorrect.

struct hseRsaPssSignScheme_t

RSASSA_PSS signature scheme.

Includes parameters needed for RSASSA_PSS signature generate/verify.

Data Fields

Type	Name	Description
hseHashAlgo_t	hashAlgo	INPUT: The hash algorithm used to hash the input before applying the RSA operation. Must not be HSE_HASH_ALGO_NULL .
uint8_t	reserved[3]	

Data Fields

Type	Name	Description
uint32_t	saltLength	INPUT: The length of the salt in bytes. It must fulfill one of the following conditions: – 0 <= saltLength <= 62 if the key length is 128 bytes and SHA-512 is used as hash algorithm; – 0 <= saltLength <= hashLength otherwise, where hashLength denotes the output length of the chosen hash algorithm.

struct hseRsaPkcs1v15Scheme_t

RSASSA_PKCS1_V15 signature scheme.

Includes parameters needed for RSASSA_PKCS1_V15 signature generate/verify.

Data Fields

Type	Name	Description
hseHashAlgo_t	hashAlgo	INPUT: The hash algorithm Must not be HSE_HASH_ALGO_NULL.
uint8_t	reserved[3]	

struct hseSignScheme_t

The HSE signature scheme.

Includes parameters needed for signature generate/verify.

Data Fields

Type	Name	Description
hseSignSchemeEnum_t	signSch	INPUT: Signature scheme.
uint8_t	reserved[3]	
union hseSignScheme_t.sch	sch	INPUT: Additional information for selected Signature scheme.

struct hseSymCipherScheme_t

HSE symmetric cipher scheme.

Includes parameters needed for a symmetric cipher.

Data Fields

Type	Name	Description
hseCipherAlgo_t	cipherAlgo	INPUT: Select an symmetric cipher.
hseCipherBlockMode_t	cipherBlockMode	INPUT: Specifies the cipher block mode.
uint8_t	reserved[2]	
uint32_t	ivLength	INPUT: Initialization Vector length(at least 16 bytes).
uint64_t	pIV	INPUT: Initialization Vector/Nonce.

struct hseAeadScheme_t

Data Fields

Type	Name	Description
hseAuthCipherMode_t	authCipherMode	INPUT: Specifies the authenticated cipher mode.
uint8_t	reserved[1]	
uint16_t	tagLength	INPUT: Specifies the tag length.
uint64_t	pTag	INPUT: Tag pointer.
uint32_t	ivLength	INPUT: Initialization Vector length(at least 12 bytes).
uint64_t	pIV	INPUT: Initialization Vector/Nonce.
uint32_t	aadLength	INPUT: The length of Additional Data (in bytes). Can be zero.
uint64_t	pAAD	INPUT: The AAD Header data. Ignored if aadLength is zero.

struct hseRsaCipherScheme_t

RSA cipher scheme.

Performs the RSA encryption/decryption).

Data Fields

Type	Name	Description
hseRsaAlgo_t	rsaAlgo	INPUT: RSA algorithm.
uint8_t	reserved[3]	

Data Fields

Type	Name	Description
union hseRsaCipherScheme_t.sch	sch	INPUT: Scheme for selected RSA algorithm.

union hseCipherScheme_t

HSE Cipher scheme.

Includes parameters needed for symmetric cipher/RSA encryption and decryption.

Data Fields

Type	Name	Description
hseSymCipherScheme_t	symCipher	INPUT: Symmetric cipher scheme.
hseAeadScheme_t	aeadCipher	INPUT: Authenticated encryption scheme (AEAD-GCM/CCM).
hseRsaCipherScheme_t	rsaCipher	INPUT: RSA cipher scheme.

struct hseCmacScheme_t

CMAC scheme.

Includes parameters needed for CMAC tag generation/verification.

Data Fields

Type	Name	Description
hseCipherAlgo_t	cipherAlgo	INPUT: Select a cipher algorithm for CMAC.
uint8_t	reserved[3]	

struct hseHmacScheme_t

HMAC scheme.

Includes parameters needed for HMAC tag generation/verification.

Data Fields

Type	Name	Description
hseHashAlgo_t	hashAlgo	INPUT: Specifies the hash algorithm for HMAC. SHA3 and Miyaguchi-Preneel are not supported for HMAC.
uint8_t	reserved[3]	

struct hseGmacScheme_t

GMAC scheme (AES only).

Includes parameters needed for GMAC tag generation/verification.

Data Fields

Type	Name	Description
uint32_t	ivLength	INPUT: Initialization Vector length. Zero is not allowed. Recommended 12 bytes or greater.
uint64_t	pIV	INPUT: Initialization Vector/Nonce.

struct hseMacScheme_t

HSE MAC scheme.

Includes parameters needed for MAC computation.

Data Fields

Type	Name	Description
hseMacAlgo_t	macAlgo	INPUT: Select an MAC algorithm.
uint8_t	reserved[3]	
union hseMacScheme_t.sch	sch	INPUT: The scheme (or parameters) for the selected mac algorithm.

union hseAuthScheme_t

HSE authentication scheme.

Includes parameters needed for authentication.

Data Fields

Type	Name	Description
hseMacScheme_t	macScheme	INPUT: MAC scheme.
hseSignScheme_t	sigScheme	INPUT: Signature scheme.

struct hseScatterList_t

HSE Scatter List .

The input and output data can be provided as a scatter list. A scatter list is used when the input/output is not a continuous buffer (the buffer is spread across multiple memory locations). The input and output pointers are specified as a list of entries as below.

Data Fields

Type	Name	Description
uint32_t	length	The length of the chunk. Maximum size must be less than 2^30. The final chunk from scatter list must have bit30 set to 1 (e.g. length = chunk_len | HSE_SGT_FINAL_CHUNK_BIT_MASK)
uint64_t	pPtr	Pointer to the chunk.

union hseSignScheme_t.sch

INPUT: Additional information for selected Signature scheme.

Data Fields

Type	Name	Description
hseEcdsaScheme_t	ecdsa	INPUT: ECDSA signature scheme.
hseEddsaSignScheme_t	eddsa	INPUT: EDDSA signature scheme.
hseRsaPssSignScheme_t	rsaPss	INPUT: RSA PSS signature scheme.
hseRsaPkcs1v15Scheme_t	rsaPkcs1v15	INPUT: RSASSA_PKCS1_V15 signature scheme.

union hseRsaCipherScheme_t.sch

INPUT: Scheme for selected RSA algorithm.

Data Fields

Type	Name	Description
hseRsaOAEPScheme_t	rsaOAEP	INPUT: RSA-OAEP scheme.
hseNoScheme_t	rsaPkcs1v15	INPUT: No scheme for RSA-PKCS1V15.

union hseMacScheme_t.sch

INPUT: The scheme (or parameters) for the selected mac algorithm.

Data Fields

Type	Name	Description
hseCmacScheme_t	cmac	INPUT: CMAC scheme (AES).
hseHmacScheme_t	hmac	INPUT: HMAC scheme.
hseGmacScheme_t	gmac	INPUT: GMAC scheme. Supports only AES.
hseNoScheme_t	xCbcmac	INPUT: No scheme parameters; supports only AES128.

Macro Definition Documentation

HSE_MAX_DESCR_SIZE

```
#define HSE_MAX_DESCR_SIZE (256U)
```

Absolute maximum HSE service descriptor size. This is determined by the HSE-HOST shared memory size, the number of MUs and the number of channels per MU.

HSE_MU0_MASK

```
#define HSE_MU0_MASK ((hseMuMask_t)1U << 0U)
```

MU Instance 0.

HSE_MU1_MASK

```
#define HSE_MU1_MASK ((hseMuMask_t)1U << 1U)
```

MU Instance 1.

HSE_MU2_MASK

```
#define HSE_MU2_MASK ((hseMuMask_t)1U << 2U)
```

MU Instance 2.

HSE_MU3_MASK

```
#define HSE_MU3_MASK ((hseMuMask_t)1U << 3U)
```

MU Instance 3.

HSE_ALL_MU_MASK

```
#define HSE_ALL_MU_MASK ( HSE_MU0_MASK| HSE_MU1_MASK| HSE_MU2_MASK| HSE_MU3_MASK)
```

Mask for all MU Instances.

HSE_SGT_OPTION_NONE

```
#define HSE_SGT_OPTION_NONE ((hseSGTOption_t)0U)
```

Scatter list is not used.

HSE_SGT_OPTION_INPUT

```
#define HSE_SGT_OPTION_INPUT ((hseSGTOption_t)1U << 0U)
```

Input pointer is provided a scatter list.

HSE_SGT_OPTION_OUTPUT

```
#define HSE_SGT_OPTION_OUTPUT ((hseSGTOption_t)1U << 1U)
```

Output pointer is provided a scatter list.

HSE_SGT_OPTION_INPUT_OUTPUT_MASK

```
#define HSE_SGT_OPTION_INPUT_OUTPUT_MASK ( HSE_SGT_OPTION_INPUT| HSE_SGT_OPTION_
OUTPUT )
```

Mask for input/output scatter-gatther option.

HSE_SGT_FINAL_CHUNK_BIT_MASK

```
#define HSE_SGT_FINAL_CHUNK_BIT_MASK (0x40000000UL)
```

Scatter-gather Final chunk BIT. This bit is set in the "length" field of the chunk (see hseScatterList_t).

HSE_ACCESS_MODE_ONE_PASS

```
#define HSE_ACCESS_MODE_ONE_PASS ((hseAccessMode_t)0U)
```

ONE-PASS access mode.

HSE_ACCESS_MODE_START

```
#define HSE_ACCESS_MODE_START ((hseAccessMode_t)1U)
```

START access mode

HSE_ACCESS_MODE_UPDATE

```
#define HSE_ACCESS_MODE_UPDATE ((hseAccessMode_t)2U)
```

UPDATE access mode

HSE_ACCESS_MODE_FINISH

```
#define HSE_ACCESS_MODE_FINISH ((hseAccessMode_t)3U)
```

FINISH access mode

HSE_HASH_ALGO_NULL

```
#define HSE_HASH_ALGO_NULL ((hseHashAlgo_t)0U)
```

None.

HSE_HASH_RESERVED1

```
#define HSE_HASH_RESERVED1 ((hseHashAlgo_t)1U)
```

Reserved (MD5 obsolete)

HSE_HASH_ALGO_SHA_1

```
#define HSE_HASH_ALGO_SHA_1 ((hseHashAlgo_t)2U)
```

SHA1 hash.

HSE_HASH_ALGO_SHA2_224

```
#define HSE_HASH_ALGO_SHA2_224 ((hseHashAlgo_t)3U)
```

SHA2_224 hash.

HSE_HASH_ALGO_SHA2_256

```
#define HSE_HASH_ALGO_SHA2_256 ((hseHashAlgo_t)4U)
```

SHA2_256 hash.

HSE_HASH_ALGO_SHA2_384

```
#define HSE_HASH_ALGO_SHA2_384 ((hseHashAlgo_t)5U)
```

SHA2_384 hash.

HSE_HASH_ALGO_SHA2_512

```
#define HSE_HASH_ALGO_SHA2_512 ((hseHashAlgo_t)6U)
```

SHA2_512 hash.

HSE_HASH_ALGO_SHA2_512_224

```
#define HSE_HASH_ALGO_SHA2_512_224 ((hseHashAlgo_t)7U)
```

SHA2_512_224 hash.

HSE_HASH_ALGO_SHA2_512_256

```
#define HSE_HASH_ALGO_SHA2_512_256 ((hseHashAlgo_t)8U)
```

SHA2_512_256 hash.

HSE_HASH_ALGO_SHA3_224

```
#define HSE_HASH_ALGO_SHA3_224 ((hseHashAlgo_t)9U)
```

SHA3_224 hash.

HSE_HASH_ALGO_SHA3_256

```
#define HSE_HASH_ALGO_SHA3_256 ((hseHashAlgo_t)10U)
```

SHA3_256 hash.

HSE_HASH_ALGO_SHA3_384

```
#define HSE_HASH_ALGO_SHA3_384 ((hseHashAlgo_t)11U)
```

SHA3_384 hash.

HSE_HASH_ALGO_SHA3_512

```
#define HSE_HASH_ALGO_SHA3_512 ((hseHashAlgo_t)12U)
```

SHA3_512 hash.

HSE_HASH_ALGO_MP

```
#define HSE_HASH_ALGO_MP ((hseHashAlgo_t)13U)
```

Miyaguchi-Preneel compression using AES-ECB with 128-bit key size (SHE spec support).

HSE_CIPHER_ALGO_NULL

```
#define HSE_CIPHER_ALGO_NULL ((hseCipherAlgo_t)0x00U)
```

NULL cipher.

HSE_CIPHER_ALGO_AES

```
#define HSE_CIPHER_ALGO_AES ((hseCipherAlgo_t)0x10U)
```

AES cipher.

HSE_CIPHER_BLOCK_MODE_NULL

```
#define HSE_CIPHER_BLOCK_MODE_NULL ((hseCipherBlockMode_t)0U)
```

NULL cipher.

HSE_CIPHER_BLOCK_MODE_CTR

```
#define HSE_CIPHER_BLOCK_MODE_CTR ((hseCipherBlockMode_t)1U)
```

CTR mode (AES)

HSE_CIPHER_BLOCK_MODE_CBC

```
#define HSE_CIPHER_BLOCK_MODE_CBC ((hseCipherBlockMode_t)2U)
```

CBC mode (AES)

HSE_CIPHER_BLOCK_MODE_ECB

```
#define HSE_CIPHER_BLOCK_MODE_ECB ((hseCipherBlockMode_t)3U)
```

ECB mode (AES)

HSE_CIPHER_BLOCK_MODE_CFB

```
#define HSE_CIPHER_BLOCK_MODE_CFB ((hseCipherBlockMode_t)4U)
```

CFB mode (AES)

HSE_CIPHER_BLOCK_MODE_OFB

```
#define HSE_CIPHER_BLOCK_MODE_OFB ((hseCipherBlockMode_t)5U)
```

OFB mode (AES)

HSE_CIPHER_DIR_DECRYPT

```
#define HSE_CIPHER_DIR_DECRYPT ((hseCipherDir_t)0U)
```

Decrypt.

HSE_CIPHER_DIR_ENCRYPT

```
#define HSE_CIPHER_DIR_ENCRYPT ((hseCipherDir_t)1U)
```

Encrypt.

HSE_AUTH_CIPHER_MODE_CCM

```
#define HSE_AUTH_CIPHER_MODE_CCM ((hseAuthCipherMode_t)0x11U)
```

CCM mode.

HSE_AUTH_CIPHER_MODE_GCM

```
#define HSE_AUTH_CIPHER_MODE_GCM ((hseAuthCipherMode_t)0x12U)
```

GCM mode.

HSE_AUTH_DIR_VERIFY

```
#define HSE_AUTH_DIR_VERIFY ((hseAuthDir_t)0U)
```

Verify authentication tag.

HSE_AUTH_DIR_GENERATE

```
#define HSE_AUTH_DIR_GENERATE ((hseAuthDir_t)1U)
```

Generate authentication tag.

HSE_MAC_ALGO_CMAC

```
#define HSE_MAC_ALGO_CMAC ((hseMacAlgo_t)0x11U)
```

CMAC (AES)

HSE_MAC_ALGO_GMAC

```
#define HSE_MAC_ALGO_GMAC ((hseMacAlgo_t)0x12U)
```

GMAC (AES)

HSE_MAC_ALGO_XCBC_MAC

```
#define HSE_MAC_ALGO_XCBC_MAC ((hseMacAlgo_t)0x13U)
```

XCBC MAC (AES128)

HSE_MAC_ALGO_HMAC

```
#define HSE_MAC_ALGO_HMAC ((hseMacAlgo_t)0x20U)
```

HMAC.

HSE_SIGN_ECDSA

```
#define HSE_SIGN_ECDSA ((hseSignSchemeEnum_t)0x80U)
```

ECDSA signature scheme.

HSE_SIGN_EDDSA

```
#define HSE_SIGN_EDDSA ((hseSignSchemeEnum_t)0x81U)
```

EdDSA signature scheme.

HSE_SIGN_RSASSA_PKCS1_V15

```
#define HSE_SIGN_RSASSA_PKCS1_V15 ((hseSignSchemeEnum_t)0x93U)
```

RSASSA_PKCS1_V15 signature scheme.

HSE_SIGN_RSASSA_PSS

```
#define HSE_SIGN_RSASSA_PSS ((hseSignSchemeEnum_t)0x94U)
```

RSASSA_PSS signature scheme.

HSE_RSA_ALGO_NO_PADDING

```
#define HSE_RSA_ALGO_NO_PADDING ((hseRsaAlgo_t)0x90U)
```

The input will be treated as an unsigned integer and perform a modular exponentiation of the input

HSE_RSA_ALGO_RSAES_OAEP

```
#define HSE_RSA_ALGO_RSAES_OAEP ((hseRsaAlgo_t)0x91U)
```

RSAES OAEP cipher.

HSE_RSA_ALGO_RSAES_PKCS1_V15

```
#define HSE_RSA_ALGO_RSAES_PKCS1_V15 ((hseRsaAlgo_t)0x92U)
```

ECDSA RSAES_PKCS1_V15 cipher.

HSE_APP_CORE0

```
#define HSE_APP_CORE0 ((hseAppCore_t)0U)
```

Core0.

HSE_APP_CORE1

```
#define HSE_APP_CORE1 ((hseAppCore_t)1U)
```

Core1.

HSE_APP_CORE2

```
#define HSE_APP_CORE2 ((hseAppCore_t)2U)
```

Core2.

HSE_APP_CORE3

```
#define HSE_APP_CORE3 ((hseAppCore_t)3U)
```

Core3.

HSE_APP_CORE4

```
#define HSE_APP_CORE4 ((hseAppCore_t)4U)
```

Core4.

HSE_APP_CORE5

```
#define HSE_APP_CORE5 ((hseAppCore_t)5U)
```

Core5.

HSE_APP_CORE6

```
#define HSE_APP_CORE6 ((hseAppCore_t)6U)
```

Core6.

HSE_APP_CORE7

```
#define HSE_APP_CORE7 ((hseAppCore_t)7U)
```

Core7.

HSE_APP_CORE8

```
#define HSE_APP_CORE8 ((hseAppCore_t)8U)
```

Core8.

HSE_APP_CORE9

```
#define HSE_APP_CORE9 ((hseAppCore_t)9U)
```

Core9.

HSE_APP_CORE10

```
#define HSE_APP_CORE10 ((hseAppCore_t)10U)
```

Core10.

HSE_APP_CORE11

```
#define HSE_APP_CORE11 ((hseAppCore_t)11U)
```

Core11.

HSE_APP_CORE12

```
#define HSE_APP_CORE12 ((hseAppCore_t)12U)
```

Core12.

HSE_APP_CORE13

```
#define HSE_APP_CORE13 ((hseAppCore_t)13U)
```

Core13.

HSE_APP_CORE14

```
#define HSE_APP_CORE14 ((hseAppCore_t)14U)
```

Core14.

HSE_APP_CORE15

```
#define HSE_APP_CORE15 ((hseAppCore_t)15U)
```

Core15.

Typedef Documentation

hseMuMask_t

```
typedef uint8_t hseMuMask_t
```

HSE Message Unite (MU) masks.

hseSGTOption_t

```
typedef uint8_t hseSGTOption_t
```

HSE Scatter-Gather Option .

Specifies if the input or output data is provided a scatter list (see hseScatterList_t).

Note

> The remaining bit are ignored when SGT option is used.

hseAccessMode_t

```
typedef uint8_t hseAccessMode_t
```

HSE access modes.

hseHashAlgo_t

```
typedef uint8_t hseHashAlgo_t
```

HASH algorithm types.

hseCipherAlgo_t

```
typedef uint8_t hseCipherAlgo_t
```

Symmetric Cipher Algorithms .

hseCipherBlockMode_t

```
typedef uint8_t hseCipherBlockMode_t
```

Symmetric Cipher Block Modes.

hseCipherDir_t

```
typedef uint8_t hseCipherDir_t
```

HSE cipher direction: encryption/decryption.

hseAuthCipherMode_t

```
typedef uint8_t hseAuthCipherMode_t
```

HSE Authenticated cipher/encryption mode (only AES supported).

hseAuthDir_t

```
typedef uint8_t hseAuthDir_t
```

HSE authentication direction: generate/verify.

hseMacAlgo_t

```
typedef uint8_t hseMacAlgo_t
```

HSE MAC algorithm.

hseSignSchemeEnum_t

```
typedef uint8_t hseSignSchemeEnum_t
```

Signature scheme enumeration.

hseRsaAlgo_t

```
typedef uint8_t hseRsaAlgo_t
```

RSA algorithm types.

hseAppCore_t

```
typedef uint8_t hseAppCore_t
```

The application core IDs (that can be started). Only the IDs for the table below must be provided for a specific platform; otherwise an error will be reported.

Core assignment table:

CoreID	S32G2XX	S32R45	S32R41	SAF85XX	S32G3XX	S32ZE
0	M7_0	M7_0	M7_0	M7_0	M7_0	M33 (SMU)
1	M7_1	M7_1	M7_1	M7_1(RFE)	M7_1	LLCE_0(CE M33_0)∗
2	M7_2	M7_2	A53_0	A53_0	M7_2	LLCE_1(CE M33_1)∗
3	A53_0	A53_0	BBE32EP DSP	BBE32EP DSP	M7_3	CEVA_SPF2∗
4	A53_1	A53_1			A53_0	R52_0
5	A53_2	A53_2			A53_1	R52_1
6	A53_3	A53_3			A53_2	R52_2
7	LLCE_0∗				A53_3	R52_3
8	LLCE_1∗				A53_4	R52_4
9	LLCE_2∗				A53_5	R52_5
10	LLCE_3∗				A53_6	R52_6
11					A53_7	R52_7
12					LLCE_0∗	
13					LLCE_1∗	
14					LLCE_2∗	
15					LLCE_3∗	

Note

: The cores marked with "*" are currently not supported to be loaded by the HSE FW

hseSrvId_t

```
typedef uint32_t hseSrvId_t
```

HSE Service IDs.

hseStreamId_t

```
typedef uint8_t hseStreamId_t
```

Stream ID type.

The stream ID identifies the stream to be used in streaming operations.

hseKeyHandle_t

```
typedef uint32_t hseKeyHandle_t
```

Key Handle type.

The keyHandle identifies the key catalog(byte2), group index in catalog(byte1) and key slot index (byte0)

hseKeyGroupIdx_t

```
typedef uint8_t hseKeyGroupIdx_t
```

HSE key group index.

A group represents a set of keys of the same type. Each group is identified by its index within the catalog where it is declared

hseKeySlotIdx_t

```
typedef uint8_t hseKeySlotIdx_t
```

HSE key slot index.

A key slot represent a memory container for a single key. A group contains several key slots as defined during the key configuration

hseNoScheme_t

```
typedef uint32_t hseNoScheme_t
```

No scheme (or parameters) are defined.

11.2 HSE Defines

Macros

Type: (implicit C type)	
Name	**Value**
HSE_SRV_VER_0	0x00000000UL
HSE_SRV_VER_1	0x01000000UL
NUM_OF_ELEMS (x)	sizeof(x)/sizeof((x)[0])
SIZE_OF_STRING (string)	(sizeof(string) - 1U)
HSE_BITS_TO_BYTES (bitLen)	((((bitLen) + 7UL) >> 3UL))
HSE_BITS_TO_BYTES_UINT16 (bitLen)	(uint16_t) HSE_BITS_TO_BYTES (bitLen)
HSE_BYTES_TO_BITS (byteLen)	((byteLen) << 3UL)
HOST_ADDR	uint64_t
NULL_HOST_ADDR	(HOST_ADDR)0UL
HSE_PTR_TO_HOST_ADDR (ptr)	(HOST_ADDR)(uintptr_t)(ptr)

Type: hseDigestLen_t	
Name	**Value**
HSE_SHA1_DIGEST_LEN	20U
HSE_SHA224_DIGEST_LEN	28U
HSE_SHA256_DIGEST_LEN	32U
HSE_SHA384_DIGEST_LEN	48U
HSE_SHA512_DIGEST_LEN	64U
HSE_MAX_DIGEST_LEN	64U

Type: hseAlgoCapIdx_t	
Name	**Value**
HSE_CAP_IDX_RANDOM	0U
HSE_CAP_IDX_SHE	1U
HSE_CAP_IDX_AES	2U
HSE_CAP_IDX_XTS_AES	3U
HSE_CAP_IDX_AEAD_GCM	4U
HSE_CAP_IDX_AEAD_CCM	5U
HSE_CAP_IDX_RESERVED1	6U /∗ Reserved MD5 obsolete∗

HSE_CAP_IDX_SHA1	7U
HSE_CAP_IDX_SHA2	8U
HSE_CAP_IDX_SHA3	9U
HSE_CAP_IDX_MP	10U
HSE_CAP_IDX_CMAC	11U
HSE_CAP_IDX_HMAC	12U
HSE_CAP_IDX_GMAC	13U
HSE_CAP_IDX_XCBC_MAC	14U
HSE_CAP_IDX_RSAES_NO_PADDING	15U
HSE_CAP_IDX_RSAES_OAEP	16U
HSE_CAP_IDX_RSAES_PKCS1_V15	17U
HSE_CAP_IDX_RSASSA_PSS	18U
HSE_CAP_IDX_RSASSA_PKCS1_V15	19U
HSE_CAP_IDX_ECDH	20U
HSE_CAP_IDX_ECDSA	21U
HSE_CAP_IDX_EDDSA	22U
HSE_CAP_IDX_MONTDH	23U
HSE_CAP_IDX_CLASSIC_DH	24U
HSE_CAP_IDX_KDF_SP800_56C	25U
HSE_CAP_IDX_KDF_SP800_108	26U
HSE_CAP_IDX_KDF_ANS_X963	27U
HSE_CAP_IDX_KDF_ISO18033_KDF1	28U
HSE_CAP_IDX_KDF_ISO18033_KDF2	29U
HSE_CAP_IDX_PBKDF2	30U
HSE_CAP_IDX_KDF_TLS12_PRF	31U
HSE_CAP_IDX_HKDF	32U
HSE_CAP_IDX_KDF_IKEV2	33U

Type: hseBlockLen_t	
Name	**Value**
HSE_AES_BLOCK_LEN	16U

Typedefs

- typedef uint8_t hseDigestLen_t
- typedef uint8_t hseBlockLen_t
- typedef uint8_t hseAlgoCapIdx_t

Macro Definition Documentation

HSE_SRV_VER_0

```
#define HSE_SRV_VER_0 (0x00000000UL)
```

HSE Service versions.

HSE_SRV_VER_1

```
#define HSE_SRV_VER_1 (0x01000000UL)
```

NUM_OF_ELEMS

```
#define NUM_OF_ELEMS( x ) (sizeof(x)/sizeof((x)[0]))
```

Compute the number of elements of an array.

SIZE_OF_STRING

```
#define SIZE_OF_STRING( string ) (sizeof(string) - 1U)
```

Compute the size of a string initialized with quotation marks.

HSE_BITS_TO_BYTES

```
#define HSE_BITS_TO_BYTES( bitLen ) ((((bitLen) + 7UL) >> 3UL))
```

Translate bits to bytes.

HSE_BITS_TO_BYTES_UINT16

```
#define HSE_BITS_TO_BYTES_UINT16( bitLen ) ((uint16_t) HSE_BITS_TO_BYTES(bitLen))
```

Translate bits to bytes (uint16_t)

HSE_BYTES_TO_BITS

```
#define HSE_BYTES_TO_BITS( byteLen ) ((byteLen) << 3UL)
```

Translate bytes to bits.

HOST_ADDR

```
#define HOST_ADDR uint64_t
```

Host address size.

NULL_HOST_ADDR

```
#define NULL_HOST_ADDR (( HOST_ADDR)0UL)
```

NULL host address.

HSE_PTR_TO_HOST_ADDR

```
#define HSE_PTR_TO_HOST_ADDR( ptr ) (( HOST_ADDR)(uintptr_t)(ptr))
```

Pointer to Host address

HSE_SHA1_DIGEST_LEN

```
#define HSE_SHA1_DIGEST_LEN ((hseDigestLen_t)20U)
```

SHA1 digest length in bytes.

HSE_SHA224_DIGEST_LEN

```
#define HSE_SHA224_DIGEST_LEN ((hseDigestLen_t)28U)
```

SHA224 digest length in bytes.

HSE_SHA256_DIGEST_LEN

```
#define HSE_SHA256_DIGEST_LEN ((hseDigestLen_t)32U)
```

SHA256 digest length in bytes.

HSE_SHA384_DIGEST_LEN

```
#define HSE_SHA384_DIGEST_LEN ((hseDigestLen_t)48U)
```

SHA384 digest length in bytes.

HSE_SHA512_DIGEST_LEN

```
#define HSE_SHA512_DIGEST_LEN ((hseDigestLen_t)64U)
```

SHA512 digest length in bytes.

HSE_MAX_DIGEST_LEN

```
#define HSE_MAX_DIGEST_LEN ((hseDigestLen_t)64U)
```

Max digest buffer in bytes.

HSE_AES_BLOCK_LEN

```
#define HSE_AES_BLOCK_LEN ((hseBlockLen_t)16U)
```

AES block length in bytes

HSE_CAP_IDX_RANDOM

```
#define HSE_CAP_IDX_RANDOM ((hseAlgoCapIdx_t)0U)
```

HSE_CAP_IDX_SHE

```
#define HSE_CAP_IDX_SHE ((hseAlgoCapIdx_t)1U)
```

HSE_CAP_IDX_AES

```
#define HSE_CAP_IDX_AES ((hseAlgoCapIdx_t)2U)
```

HSE_CAP_IDX_XTS_AES

```
#define HSE_CAP_IDX_XTS_AES ((hseAlgoCapIdx_t)3U)
```

HSE_CAP_IDX_AEAD_GCM

```
#define HSE_CAP_IDX_AEAD_GCM ((hseAlgoCapIdx_t)4U)
```

HSE_CAP_IDX_AEAD_CCM

```
#define HSE_CAP_IDX_AEAD_CCM ((hseAlgoCapIdx_t)5U)
```

HSE_CAP_IDX_RESERVED1

```
#define HSE_CAP_IDX_RESERVED1 ((hseAlgoCapIdx_t)6U) /* Reserved (MD5 obsolete)*/
```

HSE_CAP_IDX_SHA1

```
#define HSE_CAP_IDX_SHA1 ((hseAlgoCapIdx_t)7U)
```

HSE_CAP_IDX_SHA2

```
#define HSE_CAP_IDX_SHA2 ((hseAlgoCapIdx_t)8U)
```

HSE_CAP_IDX_SHA3

```
#define HSE_CAP_IDX_SHA3 ((hseAlgoCapIdx_t)9U)
```

HSE_CAP_IDX_MP

```
#define HSE_CAP_IDX_MP ((hseAlgoCapIdx_t)10U)
```

HSE_CAP_IDX_CMAC

```
#define HSE_CAP_IDX_CMAC ((hseAlgoCapIdx_t)11U)
```

HSE_CAP_IDX_HMAC

```
#define HSE_CAP_IDX_HMAC ((hseAlgoCapIdx_t)12U)
```

HSE_CAP_IDX_GMAC

```
#define HSE_CAP_IDX_GMAC ((hseAlgoCapIdx_t)13U)
```

HSE_CAP_IDX_XCBC_MAC

```
#define HSE_CAP_IDX_XCBC_MAC ((hseAlgoCapIdx_t)14U)
```

HSE_CAP_IDX_RSAES_NO_PADDING

```
#define HSE_CAP_IDX_RSAES_NO_PADDING ((hseAlgoCapIdx_t)15U)
```

HSE_CAP_IDX_RSAES_OAEP

```
#define HSE_CAP_IDX_RSAES_OAEP ((hseAlgoCapIdx_t)16U)
```

HSE_CAP_IDX_RSAES_PKCS1_V15

```
#define HSE_CAP_IDX_RSAES_PKCS1_V15 ((hseAlgoCapIdx_t)17U)
```

HSE_CAP_IDX_RSASSA_PSS

```
#define HSE_CAP_IDX_RSASSA_PSS ((hseAlgoCapIdx_t)18U)
```

HSE_CAP_IDX_RSASSA_PKCS1_V15

```
#define HSE_CAP_IDX_RSASSA_PKCS1_V15 ((hseAlgoCapIdx_t)19U)
```

HSE_CAP_IDX_ECDH

```
#define HSE_CAP_IDX_ECDH ((hseAlgoCapIdx_t)20U)
```

HSE_CAP_IDX_ECDSA

```
#define HSE_CAP_IDX_ECDSA ((hseAlgoCapIdx_t)21U)
```

HSE_CAP_IDX_EDDSA

```
#define HSE_CAP_IDX_EDDSA ((hseAlgoCapIdx_t)22U)
```

HSE_CAP_IDX_MONTDH

```
#define HSE_CAP_IDX_MONTDH ((hseAlgoCapIdx_t)23U)
```

HSE_CAP_IDX_CLASSIC_DH

```
#define HSE_CAP_IDX_CLASSIC_DH ((hseAlgoCapIdx_t)24U)
```

HSE_CAP_IDX_KDF_SP800_56C

```
#define HSE_CAP_IDX_KDF_SP800_56C ((hseAlgoCapIdx_t)25U)
```

HSE_CAP_IDX_KDF_SP800_108

```
#define HSE_CAP_IDX_KDF_SP800_108 ((hseAlgoCapIdx_t)26U)
```

HSE_CAP_IDX_KDF_ANS_X963

```
#define HSE_CAP_IDX_KDF_ANS_X963 ((hseAlgoCapIdx_t)27U)
```

HSE_CAP_IDX_KDF_ISO18033_KDF1

```
#define HSE_CAP_IDX_KDF_ISO18033_KDF1 ((hseAlgoCapIdx_t)28U)
```

HSE_CAP_IDX_KDF_ISO18033_KDF2

```
#define HSE_CAP_IDX_KDF_ISO18033_KDF2 ((hseAlgoCapIdx_t)29U)
```

HSE_CAP_IDX_PBKDF2

```
#define HSE_CAP_IDX_PBKDF2 ((hseAlgoCapIdx_t)30U)
```

HSE_CAP_IDX_KDF_TLS12_PRF

```
#define HSE_CAP_IDX_KDF_TLS12_PRF ((hseAlgoCapIdx_t)31U)
```

HSE_CAP_IDX_HKDF

```
#define HSE_CAP_IDX_HKDF ((hseAlgoCapIdx_t)32U)
```

HSE_CAP_IDX_KDF_IKEV2

```
#define HSE_CAP_IDX_KDF_IKEV2 ((hseAlgoCapIdx_t)33U)
```

Typedef Documentation

hseDigestLen_t

```
typedef uint8_t hseDigestLen_t
```

hseBlockLen_t

```
typedef uint8_t hseBlockLen_t
```

hseAlgoCapIdx_t

```
typedef uint8_t hseAlgoCapIdx_t
```

The capabilities indices for each enabled algorithm.

12 Features Implementation

12.1 HSE Platform

Macros

Type: (implicit C type)	
Name	**Value**
HSE_H	-
HSE_SPT_64BIT_ADDR	-
HSE_SPT_PHYSICAL_TAMPER_CONFIG	-
HSE_SPT_TEMP_SENS_VIO_CONFIG	-
HSE_SPT_CMU	-
HSE_NUM_OF_OTFAD_INSTANCES	1U
HSE_SPT_INDIRECT_SYSIMG_ACCESS	-
HSE_SPT_OTP_BOOT_SEQ_ATTR	-

Macro Definition Documentation

HSE_H

```
#define HSE_H
```

HSE_SPT_64BIT_ADDR

```
#define HSE_SPT_64BIT_ADDR
```

HSE supports 64-bit address.

HSE_SPT_PHYSICAL_TAMPER_CONFIG

```
#define HSE_SPT_PHYSICAL_TAMPER_CONFIG
```

HSE supports the physical tamper configuration.

HSE_SPT_TEMP_SENS_VIO_CONFIG

```
#define HSE_SPT_TEMP_SENS_VIO_CONFIG
```

HSE supports the temperature sensor configuration.

HSE_SPT_CMU

```
#define HSE_SPT_CMU
```

Support for CMU tamper (clock integrity verification)

HSE_NUM_OF_OTFAD_INSTANCES

```
#define HSE_NUM_OF_OTFAD_INSTANCES (1U)
```

The supported number of OTFAD Instances.

HSE_SPT_INDIRECT_SYSIMG_ACCESS

```
#define HSE_SPT_INDIRECT_SYSIMG_ACCESS
```

Support Indirect SYS-IMG access (refer to HSE FW Reference Manual)

HSE_SPT_OTP_BOOT_SEQ_ATTR

```
#define HSE_SPT_OTP_BOOT_SEQ_ATTR
```

Enable support for HSE_OTP_BOOT_SEQ_ATTR_ID attribute.

12.2 HSE High Features

Macros

Type: (implicit C type)	
Name	**Value**
HSE_SPT_FLASHLESS_DEV	-
HSE_SPT_RANDOM	-
HSE_SPT_SHE	-
HSE_SPT_AES	-
HSE_SPT_XTS_AES	-
HSE_SPT_CIPHER_BLOCK_MODE_CFB	-

HSE_SPT_CIPHER_BLOCK_MODE_CTR	-
HSE_SPT_CIPHER_BLOCK_MODE_ECB	-
HSE_SPT_CIPHER_BLOCK_MODE_OFB	-
HSE_SPT_AEAD_GCM	-
HSE_SPT_AEAD_CCM	-
HSE_SPT_AUTHENC	-
HSE_SPT_CRC32	-
HSE_SPT_HASH	-
HSE_SPT_SHA1	-
HSE_SPT_SHA2_224	-
HSE_SPT_SHA2_256	-
HSE_SPT_SHA2_384	-
HSE_SPT_SHA2_512	-
HSE_SPT_SHA2_512_224	-
HSE_SPT_SHA2_512_256	-
HSE_SPT_MIYAGUCHI_PRENEEL	-
HSE_SPT_MAC	-
HSE_SPT_FAST_CMAC	-
HSE_SPT_CMAC	-
HSE_SPT_HMAC	-
HSE_SPT_GMAC	-
HSE_SPT_XCBC_MAC	-
HSE_SPT_CMAC_WITH_COUNTER	-
HSE_SPT_SIPHASH	-
HSE_SPT_RSA	-
HSE_SPT_RSAES_NO_PADDING	-
HSE_SPT_RSAES_OAEP	-
HSE_SPT_RSAES_PKCS1_V15	-
HSE_SPT_RSASSA_PSS	-
HSE_SPT_RSASSA_PKCS1_V15	-
HSE_SPT_ECC	-
HSE_SPT_ECDH	-
HSE_SPT_ECDSA	-
HSE_SPT_EDDSA	-
HSE_SPT_MONTDH	-
HSE_SPT_ECC_USER_CURVES	-
HSE_SPT_EC_SEC_SECP256R1	-
HSE_SPT_EC_BRAINPOOL_BRAINPOOLP256R1	-
HSE_SPT_EC_25519_ED25519	-
HSE_SPT_EC_25519_CURVE25519	-
HSE_SPT_KEY_GEN	-

HSE_SPT_SYM_RND_KEY_GEN	-
HSE_SPT_ECC_KEY_PAIR_GEN	-
HSE_SPT_RSA_KEY_PAIR_GEN	-
HSE_SPT_TLS12_RSA_PRE_MASTER_SECRET_GEN	-
HSE_SPT_KEY_DERIVE	-
HSE_SPT_KDF_NXP_GENERIC	-
HSE_SPT_KDF_SP800_56C_ONESTEP	-
HSE_SPT_KDF_SP800_56C_TWOSTEP	-
HSE_SPT_KDF_SP800_108	-
HSE_SPT_KDF_ANS_X963	-
HSE_SPT_KDF_ISO18033_KDF1	-
HSE_SPT_KDF_ISO18033_KDF2	-
HSE_SPT_PBKDF2	-
HSE_SPT_KDF_TLS12_PRF	-
HSE_SPT_HKDF	-
HSE_SPT_KDF_IKEV2	-
HSE_SPT_NXP_ROM_KEYS	-
HSE_SPT_NXP_ROM_ECC_PUB_KEYS	-
HSE_SPT_FORMAT_KEY_CATALOGS	-
HSE_SPT_EXTEND_KEY_CATALOG	-
HSE_SPT_GET_KEY_INFO	-
HSE_SPT_KEY_VERIFY	-
HSE_SPT_IMPORT_KEY	-
HSE_SPT_EXPORT_KEY	-
HSE_SPT_KEY_MGMT_POLICIES	-
HSE_MAX_RAM_KEYS	20U
HSE_MAX_NVM_SYM_KEYS	40U
HSE_MAX_NVM_ASYM_KEYS	12U
HSE_SPT_MONOTONIC_COUNTERS	-
HSE_NUM_OF_MONOTONIC_COUNTERS	16U
HSE_SPT_BOOTDATASIGN	-
HSE_SPT_BSB	-
HSE_SPT_SMR_CR	-
HSE_NUM_OF_SMR_ENTRIES	8U
HSE_NUM_OF_CORE_RESET_ENTRIES	4U
HSE_SPT_SMR_DECRYPT	-
HSE_SD_MMC_BOOT	-
HSE_SPT_OTFAD	-
HSE_NUM_OF_OTFAD_ENTRIES	4U
HSE_SPT_STREAM_CTX_IMPORT_EXPORT	-

HSE_SPT_MU_CONFIG	-
HSE_SPT_CUST_SEC_POLICY	-
HSE_SPT_OEM_SEC_POLICY	-
HSE_SPT_SELF_TEST	-
HSE_SPT_MEM_REGION_PROTECT	-
HSE_MAX_NUM_OF_MEM_REGIONS	12U
HSE_SPT_OTA_FIRMWARE_UPDATE	-
HSE_SPT_OTA_FIRMWARE_SIZE	-
HSE_SPT_SGT_OPTION	-
HSE_MAX_NUM_OF_SGT_ENTRIES	16U
HSE_SPT_RESET_SOC_ON_TAMPER_ATTR	-
HSE_SPT_APP_SPECIFIC_DATA_ATTR	-
HSE_NUM_OF_MU_INSTANCES	4U
HSE_NUM_OF_CHANNELS_PER_MU	16U
HSE_STREAM_COUNT	4U
HSE_NUM_OF_USER_ECC_CURVES	3U
HSE_TOTAL_NUM_OF_KEY_GROUPS	64U
HSE_MAX_NVM_STORE_SIZE	31848U
HSE_MAX_RAM_STORE_SIZE	16384U
HSE_AES_KEY_BITS_LENS	{128U, 192U, 256U}
HSE_MAX_SHARED_SECRET_BITS_LEN	4096U
HSE_MIN_HMAC_KEY_BITS_LEN	128U
HSE_MAX_HMAC_KEY_BITS_LEN	512U
HSE_MIN_ECC_KEY_BITS_LEN	192U
HSE_MAX_ECC_KEY_BITS_LEN	256U
HSE_MIN_RSA_KEY_BITS_LEN	1024U
HSE_MAX_RSA_KEY_BITS_LEN	2048U
HSE_MAX_RSA_PUB_EXP_SIZE	16U
HSE_DEFAULT_MIN_FAST_CMAC_TAG_BITLEN	32U
HSE_SIPHASH_KEY_BIT_LENS	128U
HSE_SPT_SIGN	-
HSE_SPT_AEAD	-
HSE_SPT_COMPUTE_DH	-
HSE_SPT_SHA2	-

Macro Definition Documentation

HSE_SPT_FLASHLESS_DEV

```
#define HSE_SPT_FLASHLESS_DEV
```

The device is flashless (external flash).

HSE_SPT_RANDOM

```
#define HSE_SPT_RANDOM
```

Support for Random Number Generation.

HSE_SPT_SHE

```
#define HSE_SPT_SHE
```

Support for SHE specification.

Note

AES and CMAC features must be enabled.

HSE_SPT_AES

```
#define HSE_SPT_AES
```

Support for AES_(128, 192, 256)_(ECB, CBC, CFB, OFB, CTR). AES-CBC is supported on all platforms by default.

HSE_SPT_XTS_AES

```
#define HSE_SPT_XTS_AES
```

Support for XTS AES.

HSE_SPT_CIPHER_BLOCK_MODE_CFB

```
#define HSE_SPT_CIPHER_BLOCK_MODE_CFB
```

AES-CFB cipher mode supported.

HSE_SPT_CIPHER_BLOCK_MODE_CTR

```
#define HSE_SPT_CIPHER_BLOCK_MODE_CTR
```

AES-CTR cipher mode supported.

HSE_SPT_CIPHER_BLOCK_MODE_ECB

```
#define HSE_SPT_CIPHER_BLOCK_MODE_ECB
```

AES-ECB cipher mode supported.

HSE_SPT_CIPHER_BLOCK_MODE_OFB

```
#define HSE_SPT_CIPHER_BLOCK_MODE_OFB
```

AES-OFB cipher mode supported.

HSE_SPT_AEAD_GCM

```
#define HSE_SPT_AEAD_GCM
```

Support for AEAD AES GCM as defined in FIPS PUB 197, NIST SP 800-38D, RFC-5288 and RFC-4106.

HSE_SPT_AEAD_CCM

```
#define HSE_SPT_AEAD_CCM
```

Support for AEAD AES CCM as defined in FIPS PUB 197, NIST SP 800-38C, RFC-6655 and RFC-4309.

HSE_SPT_AUTHENC

```
#define HSE_SPT_AUTHENC
```

Support for Dual Purpose Crypto Service (Authenticated encryption)

HSE_SPT_CRC32

```
#define HSE_SPT_CRC32
```

Support CRC computation.

HSE_SPT_HASH

```
#define HSE_SPT_HASH
```

Hash support.

HSE_SPT_SHA1

```
#define HSE_SPT_SHA1
```

Support for SHA-1 as defined in FIPS PUB 180-4.

HSE_SPT_SHA2_224

```
#define HSE_SPT_SHA2_224
```

Support for SHA2_224 in FIPS PUB 180-4.

HSE_SPT_SHA2_256

```
#define HSE_SPT_SHA2_256
```

Support for SHA2_256 in FIPS PUB 180-4.

HSE_SPT_SHA2_384

```
#define HSE_SPT_SHA2_384
```

Support for SHA2_384 in FIPS PUB 180-4.

HSE_SPT_SHA2_512

```
#define HSE_SPT_SHA2_512
```

Support for SHA2_512 in FIPS PUB 180-4.

HSE_SPT_SHA2_512_224

```
#define HSE_SPT_SHA2_512_224
```

Support for SHA2_512_224 in FIPS PUB 180-4.

HSE_SPT_SHA2_512_256

```
#define HSE_SPT_SHA2_512_256
```

Support for SHA2_512_256 in FIPS PUB 180-4.

HSE_SPT_MIYAGUCHI_PRENEEL

```
#define HSE_SPT_MIYAGUCHI_PRENEEL
```

Miyaguchi-Preneel compression function (SHE spec support)

HSE_SPT_MAC

```
#define HSE_SPT_MAC
```

MAC support.

HSE_SPT_FAST_CMAC

```
#define HSE_SPT_FAST_CMAC
```

Support for AES fast CMAC (optimized)

HSE_SPT_CMAC

```
#define HSE_SPT_CMAC
```

Support for AES CMAC as defined in NIST SP 800-38B.

HSE_SPT_HMAC

```
#define HSE_SPT_HMAC
```

Support for HMAC_SHA1 and HMAC_SHA2 as defined in FIPS PUB 198-1 and SP 800-107.

HSE_SPT_GMAC

```
#define HSE_SPT_GMAC
```

Support for AES GMAC as defined in NIST SP 800-38D.

HSE_SPT_XCBC_MAC

```
#define HSE_SPT_XCBC_MAC
```

Support for AES XCBC_MAC_96 as defined in RFC-3566.

HSE_SPT_CMAC_WITH_COUNTER

```
#define HSE_SPT_CMAC_WITH_COUNTER
```

Support for CMAC with counter.

HSE_SPT_SIPHASH

```
#define HSE_SPT_SIPHASH
```

Support for SipHash.

HSE_SPT_RSA

```
#define HSE_SPT_RSA
```

RSA support.

HSE_SPT_RSAES_NO_PADDING

```
#define HSE_SPT_RSAES_NO_PADDING
```

RSA modular exponentiation operations(RSAEP and RSADP).

HSE_SPT_RSAES_OAEP

```
#define HSE_SPT_RSAES_OAEP
```

Support for RSAES_OAEP as defined by RFC-8017.

HSE_SPT_RSAES_PKCS1_V15

```
#define HSE_SPT_RSAES_PKCS1_V15
```

Support for RSAES_PKCS1_V15 as defined by PKCS#1 v2.2.

HSE_SPT_RSASSA_PSS

```
#define HSE_SPT_RSASSA_PSS
```

Support for RSASSA_PSS as defined by FIPS 186-4.

HSE_SPT_RSASSA_PKCS1_V15

```
#define HSE_SPT_RSASSA_PKCS1_V15
```

Support RSASSA_PKCS1_V15 as defined by PKCS#1 v2.2.

HSE_SPT_ECC

```
#define HSE_SPT_ECC
```

Support for ECC.

HSE_SPT_ECDH

```
#define HSE_SPT_ECDH
```

ECDH support.

HSE_SPT_ECDSA

```
#define HSE_SPT_ECDSA
```

ECDSA support.

HSE_SPT_EDDSA

```
#define HSE_SPT_EDDSA
```

Twisted Edwards EdDSA (e.g. ED25519, ED448) support.

HSE_SPT_MONTDH

```
#define HSE_SPT_MONTDH
```

Montgomery DH (e.g X25519 curve) support.

HSE_SPT_ECC_USER_CURVES

```
#define HSE_SPT_ECC_USER_CURVES
```

Support to set ECC curve (not supported by default)

HSE_SPT_EC_SEC_SECP256R1

```
#define HSE_SPT_EC_SEC_SECP256R1
```

Support Ecc p256v1.

HSE_SPT_EC_BRAINPOOL_BRAINPOOLP256R1

```
#define HSE_SPT_EC_BRAINPOOL_BRAINPOOLP256R1
```

Support Ecc BrainPool p256r1.

HSE_SPT_EC_25519_ED25519

```
#define HSE_SPT_EC_25519_ED25519
```

Twisted Edwards ED25519 curve support (used with EdDSA)

HSE_SPT_EC_25519_CURVE25519

```
#define HSE_SPT_EC_25519_CURVE25519
```

Montgomery X25519 curve support (used with MONTDH)

HSE_SPT_KEY_GEN

```
#define HSE_SPT_KEY_GEN
```

Key Generate support.

HSE_SPT_SYM_RND_KEY_GEN

```
#define HSE_SPT_SYM_RND_KEY_GEN
```

Support for symmetric random key generation.

HSE_SPT_ECC_KEY_PAIR_GEN

```
#define HSE_SPT_ECC_KEY_PAIR_GEN
```

Support for ECC key-pair generation.

HSE_SPT_RSA_KEY_PAIR_GEN

```
#define HSE_SPT_RSA_KEY_PAIR_GEN
```

Support for RSA key-pair generation.

HSE_SPT_TLS12_RSA_PRE_MASTER_SECRET_GEN

```
#define HSE_SPT_TLS12_RSA_PRE_MASTER_SECRET_GEN
```

Support for RSA key exchange.

HSE_SPT_KEY_DERIVE

```
#define HSE_SPT_KEY_DERIVE
```

KDF support.

HSE_SPT_KDF_NXP_GENERIC

```
#define HSE_SPT_KDF_NXP_GENERIC
```

NXP Generic KDF.

HSE_SPT_KDF_SP800_56C_ONESTEP

```
#define HSE_SPT_KDF_SP800_56C_ONESTEP
```

Support for KDF One-step as defined by SP800-56C rev1.

HSE_SPT_KDF_SP800_56C_TWOSTEP

```
#define HSE_SPT_KDF_SP800_56C_TWOSTEP
```

Support for KDF Two-step as defined by SP800-56C rev1.

HSE_SPT_KDF_SP800_108

```
#define HSE_SPT_KDF_SP800_108
```

Support for KDF(Counter, Feedback, Pipeline) as defined by SP800-108.

HSE_SPT_KDF_ANS_X963

```
#define HSE_SPT_KDF_ANS_X963
```

Support for KDF as defined by ANS X9.63.

HSE_SPT_KDF_ISO18033_KDF1

```
#define HSE_SPT_KDF_ISO18033_KDF1
```

Support for KDF1 as defined by ISO18033.

HSE_SPT_KDF_ISO18033_KDF2

```
#define HSE_SPT_KDF_ISO18033_KDF2
```

Support for KDF2 as defined by ISO18033.

HSE_SPT_PBKDF2

```
#define HSE_SPT_PBKDF2
```

Support for PBKDF2 as defined as defined by PKCS#5 v2.1 and RFC-8018.

HSE_SPT_KDF_TLS12_PRF

```
#define HSE_SPT_KDF_TLS12_PRF
```

KDF Support for TLS 1.2 as defined by RFC-5246.

HSE_SPT_HKDF

```
#define HSE_SPT_HKDF
```

Support for HMAC-based Extract-and-Expand KDF as defined by RFC-5869.

HSE_SPT_KDF_IKEV2

```
#define HSE_SPT_KDF_IKEV2
```

KDF Support for IKEv2 as defined by RFC-4306.

HSE_SPT_NXP_ROM_KEYS

```
#define HSE_SPT_NXP_ROM_KEYS
```

Support NXP ROM keys.

HSE_SPT_NXP_ROM_ECC_PUB_KEYS

```
#define HSE_SPT_NXP_ROM_ECC_PUB_KEYS
```

Support NXP ECC ROM public keys.

HSE_SPT_FORMAT_KEY_CATALOGS

```
#define HSE_SPT_FORMAT_KEY_CATALOGS
```

Support Format Key Catalogs service.

HSE_SPT_EXTEND_KEY_CATALOG

```
#define HSE_SPT_EXTEND_KEY_CATALOG
```

Support Format Key Catalogs Extend service.

HSE_SPT_GET_KEY_INFO

```
#define HSE_SPT_GET_KEY_INFO
```

Support Get Key Info Service.

HSE_SPT_KEY_VERIFY

```
#define HSE_SPT_KEY_VERIFY
```

Support Key Verify Service.

HSE_SPT_IMPORT_KEY

```
#define HSE_SPT_IMPORT_KEY
```

Support Import Key Service.

HSE_SPT_EXPORT_KEY

```
#define HSE_SPT_EXPORT_KEY
```

Support Export Key Service.

HSE_SPT_KEY_MGMT_POLICIES

```
#define HSE_SPT_KEY_MGMT_POLICIES
```

Support Key Management configurable policies.

HSE_MAX_RAM_KEYS

```
#define HSE_MAX_RAM_KEYS (20U)
```

Maximum number of keys in RAM keystore.

HSE_MAX_NVM_SYM_KEYS

```
#define HSE_MAX_NVM_SYM_KEYS (40U)
```

Maximum number of symmetric keys in NVM store.

HSE_MAX_NVM_ASYM_KEYS

```
#define HSE_MAX_NVM_ASYM_KEYS (12U)
```

Maximum number of asymmetric keys in NVM store.

HSE_SPT_MONOTONIC_COUNTERS

```
#define HSE_SPT_MONOTONIC_COUNTERS
```

Monotonic Counter support.

HSE_NUM_OF_MONOTONIC_COUNTERS

```
#define HSE_NUM_OF_MONOTONIC_COUNTERS (16U)
```

The supported number of monotonic counters.

HSE_SPT_BOOTDATASIGN

```
#define HSE_SPT_BOOTDATASIGN
```

Boot Data Sign Support.

HSE_SPT_BSB

```
#define HSE_SPT_BSB
```

Basic Secure Booting(BSB) Support.

HSE_SPT_SMR_CR

```
#define HSE_SPT_SMR_CR
```

Advance Secure Booting(ASB) Secure memory regions verification (SMR) & Core Reset(CR) Table Support.

HSE_NUM_OF_SMR_ENTRIES

```
#define HSE_NUM_OF_SMR_ENTRIES (8U)
```

The supported number of SMR entries.

HSE_NUM_OF_CORE_RESET_ENTRIES

```
#define HSE_NUM_OF_CORE_RESET_ENTRIES (4U)
```

The supported number of CORE RESET entries.

HSE_SPT_SMR_DECRYPT

```
#define HSE_SPT_SMR_DECRYPT
```

Support encrypted SMRs.

HSE_SD_MMC_BOOT

```
#define HSE_SD_MMC_BOOT
```

Enable support of SD & MMC card.

HSE_SPT_OTFAD

```
#define HSE_SPT_OTFAD
```

On-The-Fly AES Decryption (OTFAD) support.

HSE_NUM_OF_OTFAD_ENTRIES

```
#define HSE_NUM_OF_OTFAD_ENTRIES (4U)
```

The supported number of OTFAD entries per instance.

HSE_SPT_STREAM_CTX_IMPORT_EXPORT

```
#define HSE_SPT_STREAM_CTX_IMPORT_EXPORT
```

Support Import/Export of streaming context for symmetric operations.

HSE_SPT_MU_CONFIG

```
#define HSE_SPT_MU_CONFIG
```

Support MU configuration and XRDC for SHARED memory configuration.

HSE_SPT_CUST_SEC_POLICY

```
#define HSE_SPT_CUST_SEC_POLICY
```

Support of Customer Security Policy.

HSE_SPT_OEM_SEC_POLICY

```
#define HSE_SPT_OEM_SEC_POLICY
```

Support of Oem Security Policy.

HSE_SPT_SELF_TEST

```
#define HSE_SPT_SELF_TEST
```

Support self test.

HSE_SPT_MEM_REGION_PROTECT

```
#define HSE_SPT_MEM_REGION_PROTECT
```

Support memory region protection.

HSE_MAX_NUM_OF_MEM_REGIONS

```
#define HSE_MAX_NUM_OF_MEM_REGIONS (12U)
```

Maximum number of memory regions configurable through HSE_SPT_MEM_REGION_PROTECT service.

HSE_SPT_OTA_FIRMWARE_UPDATE

```
#define HSE_SPT_OTA_FIRMWARE_UPDATE
```

Support OTA Firmware Update.

HSE_SPT_OTA_FIRMWARE_SIZE

```
#define HSE_SPT_OTA_FIRMWARE_SIZE
```

Support OTA Firmware Update Size.

HSE_SPT_SGT_OPTION

```
#define HSE_SPT_SGT_OPTION
```

Enable support for Scatter Gatter Table.

HSE_MAX_NUM_OF_SGT_ENTRIES

```
#define HSE_MAX_NUM_OF_SGT_ENTRIES (16U)
```

Maximum number for SGT entries.

HSE_SPT_RESET_SOC_ON_TAMPER_ATTR

```
#define HSE_SPT_RESET_SOC_ON_TAMPER_ATTR
```

Support "Reset Soc On Tamper" attribute.

HSE_SPT_APP_SPECIFIC_DATA_ATTR

```
#define HSE_SPT_APP_SPECIFIC_DATA_ATTR
```

Enable support for "Application Specific Data" attribute.

HSE_NUM_OF_MU_INSTANCES

```
#define HSE_NUM_OF_MU_INSTANCES (4U)
```

The maxim number of MU interfaces.

HSE_NUM_OF_CHANNELS_PER_MU

```
#define HSE_NUM_OF_CHANNELS_PER_MU (16U)
```

The maxim number of channels per MU interface.

HSE_STREAM_COUNT

```
#define HSE_STREAM_COUNT (4U)
```

HSE stream count per MU interface.

HSE_NUM_OF_USER_ECC_CURVES

```
#define HSE_NUM_OF_USER_ECC_CURVES (3U)
```

The number of ECC curves the user can load into the HSE.

HSE_TOTAL_NUM_OF_KEY_GROUPS

```
#define HSE_TOTAL_NUM_OF_KEY_GROUPS (64U)
```

The total number of catalog configuration entries for both NVM and RAM catalogs.

HSE_MAX_NVM_STORE_SIZE

```
#define HSE_MAX_NVM_STORE_SIZE (31848U)
```

NVM key store size (in bytes)

HSE_MAX_RAM_STORE_SIZE

```
#define HSE_MAX_RAM_STORE_SIZE (16384U)
```

RAM key store size (in bytes)

HSE_AES_KEY_BITS_LENS

```
#define HSE_AES_KEY_BITS_LENS {128U, 192U, 256U}
```

AES key bit length (set to zero to disable a AES key size)

HSE_MAX_SHARED_SECRET_BITS_LEN

```
#define HSE_MAX_SHARED_SECRET_BITS_LEN (4096U)
```

Max shared secret bit length.

HSE_MIN_HMAC_KEY_BITS_LEN

```
#define HSE_MIN_HMAC_KEY_BITS_LEN (128U)
```

Min HMAC key bit length.

HSE_MAX_HMAC_KEY_BITS_LEN

```
#define HSE_MAX_HMAC_KEY_BITS_LEN (512U)
```

Max HMAC key bit length.

HSE_MIN_ECC_KEY_BITS_LEN

```
#define HSE_MIN_ECC_KEY_BITS_LEN (192U)
```

Min ECC key bit length.

HSE_MAX_ECC_KEY_BITS_LEN

```
#define HSE_MAX_ECC_KEY_BITS_LEN (256U)
```

Max ECC key bit length.

HSE_MIN_RSA_KEY_BITS_LEN

```
#define HSE_MIN_RSA_KEY_BITS_LEN (1024U)
```

Min RSA key bit length.

HSE_MAX_RSA_KEY_BITS_LEN

```
#define HSE_MAX_RSA_KEY_BITS_LEN (2048U)
```

Max RSA key bit length.

HSE_MAX_RSA_PUB_EXP_SIZE

```
#define HSE_MAX_RSA_PUB_EXP_SIZE (16U)
```

Max RSA public exponent size (in bytes)

HSE_DEFAULT_MIN_FAST_CMAC_TAG_BITLEN

```
#define HSE_DEFAULT_MIN_FAST_CMAC_TAG_BITLEN (32U)
```

FAST CMAC default min bit length.

HSE_SIPHASH_KEY_BIT_LENS

```
#define HSE_SIPHASH_KEY_BIT_LENS (128U)
```

SipHash key bit length.

HSE_SPT_SIGN

```
#define HSE_SPT_SIGN
```

HSE_SPT_AEAD

```
#define HSE_SPT_AEAD
```

HSE_SPT_COMPUTE_DH

```
#define HSE_SPT_COMPUTE_DH
```

HSE_SPT_SHA2

```
#define HSE_SPT_SHA2
```

How to Reach Us:

Home Page:
nxp.com

Web Support:
nxp.com/support