User Manual

for S32K3 CRYPTO Driver

Document Number: UM34CRYPTOASR4.4 Rev0000R3.0.0 Rev. 1.0

Chapter 1

Revision History

Revision	Date	Author	Description
1.0	31.03.2023	NXP RTD Team	S32K3 Real-Time Drivers AUTOSAR 4.4 & R21-11 Version 3.0.0

Chapter 2

Introduction

This User Manual describes NXP Semiconductor AUTOSAR Crypto driver for S32K3 platform.

AUTOSAR CRYPTO driver configuration parameters and deviations from the specification are described in CRYPTO Driver chapter of this document. AUTOSAR CRYPTO driver requirements and APIs are described in the AUTOSAR CRYPTO driver software specification document.

2.1 Supported Derivatives

The software described in this document is intended to be used with the following microcontroller devices of NXP Semiconductors:

- s32k310_mqfp100
- s32k310_lqfp48
- s32k311_mqfp100 / MWCT2015S_mqfp100
- s32k311_lqfp48
- s32k312_mqfp100 / MWCT2016S_mqfp100
- s32k312_mqfp172 / MWCT2016S_mqfp172
- s32k314_mqfp172
- s32k314_mapbga257
- s32k322_mqfp100 / MWCT2D16S_mqfp100

- s32k322_mqfp172 / MWCT2D16S_mqfp172
- s32k324_mqfp172 / MWCT2D17S_mqfp172
- s32k324_mapbga257
- s32k341_mqfp100
- s32k341_mqfp172
- s32k342_mqfp100
- s32k342_mqfp172
- s32k344_mqfp172
- s32k344_mapbga257
- s32k394_mapbga289
- s32k396_mapbga289
- s32k358_mqfp172
- s32k358_mapbga289
- s32k328_mqfp172
- s32k328_mapbga289
- s32k338_mqfp172
- s32k338_mapbga289
- s32k348_mqfp172
- s32k348_mapbga289
- s32m274_lqfp64
- s32m276_lqfp64

All of the above microcontroller devices are collectively named as S32K3.

Note: MWCT part numbers contain NXP confidential IP for Qi Wireless Power.

2.2 Overview

AUTOSAR (AUTomotive Open System ARchitecture) is an industry partnership working to establish standards for software interfaces and software modules for automobile electronic control systems.

AUTOSAR:

- paves the way for innovative electronic systems that further improve performance, safety and environmental friendliness.
- is a strong global partnership that creates one common standard: "Cooperate on standards, compete on implementation".
- is a key enabling technology to manage the growing electrics/electronics complexity. It aims to be prepared for the upcoming technologies and to improve cost-efficiency without making any compromise with respect to quality.
- facilitates the exchange and update of software and hardware over the service life of the vehicle.

2.3 About This Manual

This Technical Reference employs the following typographical conventions:

- **Boldface** style: Used for important terms, notes and warnings.
- *Italic* style: Used for code snippets in the text. Note that C language modifiers such "const" or "volatile" are sometimes omitted to improve readability of the presented code.

Notes and warnings are shown as below:

Note

This is a note.

Warning

This is a warning

2.4 Acronyms and Definitions

Term	Definition
AES	Advanced Encryption Standard
API	Application Programming Interface
AUTOSAR	Automotive Open System Architecture
CAAM	Cryptographic Acceleration and Assurance Module
CMAC	Cipher-based Message Authentication Code
C/CPP	C and C++ Source Code
DET	Development Error Tracer
ECB	Electronic Code Book (refers to AES-ECB mode)
ECC	Elliptic Curve Cryptography
ECDSA	Elliptic Curve Digital Signature Algorithm
ECU	Electronic Control Unit
EdDSA	Edwards-curve Digital Signature Algorithm
FLS	Flash
GCM	Galois/Counter Mode (refers to AES-GCM mode)
GMAC	Galois Message Authentication Code
HSE	Hardware Security Engine
MAC	Message Authentication Code
MU	Messaging Unit
N/A	Not Applicable
NVM	Non-Volatile Memory
OID	Object Identifier an encoded identifier of a standardized object commonly used in public key certificates
RAM	Random Access Memory
RNG	Random number generator
ROM	Read-only Memory
RSA	A public-key cryptosystem named after the inventors Mr. Rivest, Mr. Shamir and Mr.Adleman
SHA	Secure Hash Algorithm
SHE	Secure Hardware Extension
TDES	Triple-DES operation

- The term "Application" is used for the software utilizing the Crypto Driver.

2.5 Reference List

#	Title	Version
1	Specification of Crypto Driver	AUTOSAR CP Release 4.↩ 4.0
2	S32K3xx Safety Manual	Rev. 3, Dec 2022
3	S32K39 and S32K37 Safety Manual	Rev. 1 Draft D, 16 Nov 2022
4	S32M276 Safety Manual	Rev. 1, Dec 2022

#	Title	Version
5	S32K3xx Reference Manual	Rev.6, Draft B, 01/2023
6	S32K39 and S32K37 Reference Manual	Rev. 2 Draft A, 11/2022
7	S32M27x Reference Manual	Rev.2, Draft A, — 02/2023
8	S32K3xx Data Sheet	Rev. 6, 11/2022
9	S32K396 Data Sheet	Rev. 1.1 — 08/2022
10	S32M2xx Data Sheet	Rev. 2 RC — 12/2022
11	S32K358_0P14E Mask Set Errata	Rev. 28, 9/2022
12	S32K396_0P40E Mask Set Errata	Rev. DEC2022, 12/2022
13	S32K311_0P98C Mask Set Errata	Rev. 6/March/2023, 3/2023
14	S32K312: Mask Set Errata for Mask 0P09C	Rev. 25/April/2022
15	S32K342: Mask Set Errata for Mask 0P97C	Rev. 10, 11/2022
16	S32K3x4: Mask Set Errata for Mask 0P55A/1P55A	Rev. 14/Oct/2022

Chapter 3

Driver

3.1 Requirements

Requirements for this driver are detailed in the AUTOSAR 4.4 Rev0000 CRYPTO Driver Software Specification document (See Table Reference List).

3.2 Driver Design Summary

The CRYPTO driver supports cryptographic primitives, key storage, key configuration and key management for cryptographic services by accessing the HSE core functionalities. The CRYPTO driver communicates with the HSE via descriptors whose addresses are passed via a messaging unit to the HSE. To issue any request to the HSE, the host must first fill out a descriptor associated with the request, then pass the descriptor address to the HSE via one of the MU channels. When the request has been fulfilled, the HSE will write the response in the same MU channel used by the host to trigger the request. The MU is a NXP IP which is present on this platform in 2 instances, each instance having a number of 4 channels.

The CRYPTO module provides the following major features for this release:

- Loading plain volatile AES keys (128, 192 and 256 bit) and NVM keys (authenticated and encrypted), SHE keys in plain and encrypted format

- Export symmetric keys AES, HMAC, SHE RAM and public ECC keys
- AES-ECB encrypt/decrypt one pass and streaming
- AES-GCM 128/192/256 encrypt/decrypt one pass and streaming
- AES-CBC encrypt/decrypt one pass and streaming
- AES-CTR encrypt/decrypt one pass and streaming
- AES-OFB encrypt/decrypt one pass and streaming
- AES-CFB encrypt/decrypt one pass and streaming
- RSA-PKCS1-V1_5 (1024, 2048, 3072, 4096) encrypt/decrypt
- RSAES-OAEP (1024, 2048, 3072, 4096) encrypt/decrypt
- AES-CMAC generate/verify one pass and streaming
- AES-FAST-CMAC fast one pass CMAC
- AES-GMAC generate/verify one pass and streaming
- AES-HMAC generate/verify one pass and streaming
- AES-XCBC-MAC generate/verify one pass and streaming
- Random number generation
- Generate public/private key pairs for RSA and ECC keys type (SECP256R1, SECP384R1, SECP521↩ R1, BRAINPOOLP256R1, BRAINPOOLP320R1, BRAINPOOLP384R1, BRAINPOOLP512R1, EC_25519, ED25519) or symmetric AES, HMAC keys
- Calculate Shared Secret ECC Diffie Hellman
- Hash services supporting the following hash functions: SHA1, SHA-224, SHA-256, SHA-384, SHA-512, SHA-512/224, SHA-512/256
- Format key application catalogs RAM and NVM
- Digital Signature Generation/Verification
  - RSASAA_PSS (1024, 2048, 3072, 4096) one-pass
  - RSASAA_PKCS1-v1_5 (1024, 2048, 3072, 4096) one-pass
  - ECDSA using ECC keys lying on Weierstrass curves
  - EdDSA using ECC keys lying on Twisted Edward curves (ED25519)
- SHE services supported via extensions:
  - SHE boot failure service
  - SHE boot ok service
  - SHE get status service
  - SHE get UID service
  - SHE debug challenge service
  - SHE debug authorization service
  - Miyaguchi-Preneel Compression
- Key Derivation services using the following schemes:
  - PBKDF2 with HMAC only
  - KDFX963 with HASH only
  - TLS12 with HMAC only

3.3 Hardware Resources

Communication between the CRYPTO driver and the HSE core is provided via the Messaging Unit module which enables two cores within the SoC to communicate and coordinate by passing messages (e.g. data, status and control) through the MU interface. The MU provides transmit and receive data registers for the communication between Core A and Core B.

3.4 Deviations from Requirements

The driver deviates from the AUTOSAR CRYPTO Driver software specification in some places. The table below identifies the AUTOSAR requirements that are not implemented or out of scope for the CRYPTO Driver.

Term	Definition
N/S	Out of scope
N/I	Not implemented
N/F	Not fully implemented

Below table identifies the AUTOSAR requirements that are not fully implemented, implemented differently or out of scope for the CRYPTO driver.

Requirement	Status	Description	Notes
SWS_Crypto_00121	N/S	If Crypto_ProcessJob() is called and the Job is in "ACTIVE" state, the Crypto_ProcessJob()shall check if the requested job matches the current job in the Crypto Driver Object and if yes, bypass it from queueing.	This requirement is Rejected because it is not clear. The Crypto_ProcessJob() response to upper layers is not defined when jobs are bypassed from queueing. Furthermore, the requirement introduces some performance penalties because the jobs with UPDATE or FINISH will have to wait until the queue is empty to be able to be processed, as the driver will be busy with the queued jobs.
SWS_Crypto_00141	N/S	If the random generator service is chosen and the corresponding entropy, the function shall return CRYPTO↩ _E_ENTROPY_EXHAUSTED. The function Crypto_Process↩ Job shall additionally report the runtime error CRYPTO_E_RE_↩ ENTROPY_EXHAUSTED.	This requirement is rejected because additional entropy is not needed to be provided by user as HSE generates entropy with TRNG.

Requirement	Status	Description	Notes
SWS_Crypto_00170	N/S	If no errors are detected by Crypto Driver, the service Crypto_CertificateParse() parses the certificate which is stored in the certificate data element and fills at least the key elements CRYPTO_KE_CERT_↩ SIGNEDDATA, CRYPTO_KE_↩ CERT_PARSEDPUBLICKEY and ...	This requirement is rejected because starting with the next ASR version (4.5) this functionality is moved from the Crypto driver to another dedicated module.
SWS_Crypto_00176	N/S	If the key element CRYPTO_KE_↩ CERTIFICATE_CURRENT_TIME is used during verification and the format of this timestamp does not match with the format of the timestamp of the certificate, the function Crypto_CertificateVerify shall report CRYPTO_E_PARAM_H ...	This requirement is rejected because starting with the next ASR version (4.5) this functionality is moved from the Crypto driver to another dedicated module.
SWS_Crypto_00177	N/S	If no errors are detected by Crypto Driver, the service Crypto_CertificateVerify() uses the key element CRYPTO_KE↩ _CERT_PARSEDPUBLICKEY of the key referenced by cryptoKeyId to do a signature verification. The data contained in the key element CRYPTO_ ...	This requirement is rejected because starting with the next ASR version (4.5) this functionality is moved from the Crypto driver to another dedicated module.
SWS_Crypto_00178	N/S	If certificate identified by verify↩ CryptoKeyId is verified successfully, the key identified by validateCrypto↩ KeyId shall be set to valid.	This requirement is rejected because starting with the next ASR version (4.5) this functionality is moved from the Crypto driver to another dedicated module.
SWS_Crypto_00184	N/S	Asymmetric key material with identification is specified in accordance to RFC5958 in ASN.1 format. The key material with the format specifier CRYPTO_KE_FORMAT↩ _BIN_IDENT_PRIVATEKEY_ PKCS8 needs to follow this format specification:OneAsymmetricKey ::= ...	This requirement is rejected because starting with the next ASR version (4.5) this functionality is moved from the Crypto driver to another dedicated module.
SWS_Crypto_00185	N/S	For CRYPTO_KE_FORMAT↩ _BIN_ RSA_PRIVATEKEY the parameter 'KeyMaterial OCTET STRING' for RSA private keys is defined according to RFC3447 and has the following contents:KeyMaterial ::= RSAPrivateKeyRSAPrivate↩ Key ::= SEQUENCE {version Version,modulus INT ...	This requirement is rejected because starting with the next ASR version (4.5) this functionality is moved from the Crypto driver to another dedicated module.

Requirement	Status	Description	Notes
SWS_Crypto_00186	N/S	The RSA public key in the format CRYPTO_KE_FORMAT_BIN _↩ RSA_PUBLICKEY is provided as follows:RSAPublicKey ::= BIT_↩ STRING {modulus INTEGER, – npublicExponent INTEGER, – e}The fields of type RSAPublicKey have the following meanings:modulus is the modu ...	This requirement is rejected because starting with the next ASR version (4.5) this functionality is moved from the Crypto driver to another dedicated module.
SWS_Crypto_00187	N/S	The RSA public key in the format CRYPTO_KE_FORMAT_BIN _IDENT_RSA_PUBLICKEY is provided as follows:Public↩ KeyInfo ::= SEQUENCE {Key↩ AlgorithmIdentifier ::= Algorithm↩ Identifier,publicKey ::= RSAPublic↩ Key}Explanation:Considering RFC5280, section 4.1, the ...	This requirement is rejected because starting with the next ASR version (4.5) this functionality is moved from the Crypto driver to another dedicated module.
SWS_Crypto_00188	N/S	The algorithm identifier for RSA keys shall have the value 1.2.840.113549.↩ 1.1.1. This corresponds to the ASN.1 coded OID value "2A 86 48 86 F7 0D 01 01 01". This OID shall be provided whenever an AlgorithmIdentifier for RSA is required. In other word ...	This requirement is rejected because starting with the next ASR version (4.5) this functionality is moved from the Crypto driver to another dedicated module.
SWS_Crypto_00199	N/S	If the Crypto Driver has a queue and if a synchronous job is issued and the priority is greater than the highest priority available in the queue, the Crypto Driver shall disable processing new jobs from the queue until the next call of the main funct ...	This requirement is Rejected because it is not clearly defined, introduces performance loss, complexity and interrupts the execution flow. An Autosar ticket was created to clarify this requirement, with the suggestion to use an asynchronous approach. To check all the details please see the Autosar ticket by using the id AR-108434.

3.5 Driver Limitations

- For this release, the functionality of the following APIs is not yet implemented:
  - Crypto_CertificateParse
  - Crypto_CertificateVerify
- Infix is not supported and only one CRYPTO driver instance can be configured. Therefore, the parameter CryptoInstanceId is always set to 0.
- Crypto_RandomSeed function will always return CRYPTO_E_NOT_SUPPORTED because of the fact that HSE firmware does not have support for such a functionality.
- Crypto_KeyElementSet is not supporting RSA keys

- Crypto_KeyElementSet is not supporting asymetric authenticated and encrypted keys
- Crypto_KeyElementGet is not supporting RSA keys
- Crypto_KeyElementGet is not supporting asymetric authenticated and encrypted keys
- RANDOMSEED service is not supported by Crypto_ProcessJob
- CERTIFICATEPARSE service is not supported by Crypto_ProcessJob
- CERTIFICATEVERIFY service is not supported by Crypto_ProcessJob
- Streaming mode related notes:
  - HASH:
    - job input pointer (inputPtr) or redirected input pointer (pCryptoElementArray) will be pass down to firmware for START operation mode while job input length (inputLength) or redirected input length (pu32CryptoElementActualSize) is not 0. Autosar does not require an input pointer for START operation mode but the input data can be processed at START by providing an input pointer and input length.
    - job input length (inputLength) or redirected input length (pu32CryptoElementActualSize) will be pass down to firmware for START operation mode, if no processing is desired at START operation mode the input length should be 0.
  - MACGENERATE:
    - HMAC: For START operation mode the job input pointer (inputPtr) or redirected input pointer (pCryptoElementArray) can not be NULL, an input pointer must be provided.
    - HMAC: For START operation mode the job input length (inputLength) or redirected input length (pu32CryptoElementActualSize) can not be 0 and an input pointer must be provided.
    - HMAC: For FINISH operation mode to skip the processing of the job input pointer (inputPtr) or redirected input pointer (pCryptoElementArray) the job input length (inputLength) or redirected input length (pu32CryptoElementActualSize) should be 0.
    - CMAC: job input pointer (inputPtr) will be pass down to firmware for START operation mode while job input length (inputLength) is not 0. Autosar does not require an input pointer for START operation mode but the input data can be processed at START by providing an input pointer and an input length.
    - CMAC: job input length (inputLength) will be pass down to firmware for START operation mode, if no processing is desired at START operation mode the input length should be 0.
    - CMAC: For FINISH operation mode the job input pointer (inputPtr) must be provided and the job input length (inputLength) should not be 0.
    - GMAC: job input pointer (inputPtr) will be pass down to firmware for START operation mode while job input length (inputLength) is not 0. Autosar does not require an input pointer for START operation mode but the input data can be processed at START by providing an input pointer and input length.
    - GMAC: job input length (inputLength) will be pass down to firmware for START operation mode, if no processing is desired at START operation mode the input length should be 0.
    - GMAC: For FINISH operation mode to skip the processing of the job input pointer (inputPtr) the job input length (inputLength) should be 0.
  - MACVERIFY:
    - HMAC: For START operation mode the job input pointer (inputPtr) or redirected input pointer (pCryptoElementArray) can not be NULL, an input pointer must be provided.
    - HMAC: For START operation mode the job input length (inputLength) or redirected input length (pu32CryptoElementActualSize) can not be 0 and an input pointer must be provided.

    * HMAC: For FINISH operation mode to skip the processing of the job input pointer (inputPtr) or redirected input pointer (pCryptoElementArray) the job input length (inputLength) or redirected input length (pu32CryptoElementActualSize) should be 0.
    * CMAC: job input pointer (inputPtr) will be pass down to firmware for START operation mode while job input length (inputLength) is not 0. Autosar does not require an input pointer for START operation mode but the input data can be processed at START by providing an input pointer and an input length.
    * CMAC: job input length (inputLength) will be pass down to firmware for START operation mode, if no processing is desired at START operation mode the input length should be 0.
    * CMAC: For FINISH operation mode the job input pointer (inputPtr) must be provided and the job input length (inputLength) should not be 0.
    * GMAC: job input pointer (inputPtr) will be pass down to firmware for START operation mode while job input length (inputLength) is not 0. Autosar does not require an input pointer for START operation mode but the input data can be processed at START by providing an input pointer and input length.
    * GMAC: job input length (inputLength) will be pass down to firmware for START operation mode, if no processing is desired at START operation mode the input length should be 0.
    * GMAC: For FINISH operation mode to skip the processing of the job input pointer (inputPtr) the job input length (inputLength) should be 0.
  - ENCRYPT:
    * job input pointer (inputPtr) or redirected input pointer (pCryptoElementArray) will be pass down to firmware for START operation mode while job input length (inputLength) or redirected input length (pu32CryptoElementActualSize) is not 0. Autosar does not require an input pointer for START operation mode but the input data can be processed at START by providing an input pointer and input length. When the input is processed the job output pointer (outputPtr) or redirected output pointer (pCryptoElementArray) and job output pointer length (outputLengthPtr) or redirected output pointer length (pu32CryptoElementActualSize) must be provieded.
    * job input length (inputLength) or redirected input length (pu32CryptoElementActualSize) will be pass down to firmware for START operation mode, if no processing is desired at START operation mode the input length should be 0.
    * For FINISH operation mode the job input pointer (inputPtr) or redirected input pointer (pCrypto↩ ElementArray) must be provided and the job input length (inputLength) or redirected input length (pu32CryptoElementActualSize) should not be 0.
  - DECRYPT:
    * job input pointer (inputPtr) or redirected input pointer (pCryptoElementArray) will be pass down to firmware for START operation mode while job input length (inputLength) or redirected input length (pu32CryptoElementActualSize) is not 0. Autosar does not require an input pointer for START operation mode but the input data can be processed at START by providing an input pointer and input length. When the input is processed the job output pointer (outputPtr) or redirected output pointer (pCryptoElementArray) and job output pointer length (outputLengthPtr) or redirected output pointer length (pu32CryptoElementActualSize) must be provieded.
    * job input length (inputLength) or redirected input length (pu32CryptoElementActualSize) will be pass down to firmware for START operation mode, if no processing is desired at START operation mode the input length should be 0.
    * For FINISH operation mode the job input pointer (inputPtr) or redirected input pointer (pCrypto↩ ElementArray) must be provided and the job input length (inputLength) or redirected input length (pu32CryptoElementActualSize) should not be 0.
  - AEADENCRYPT:
    * job input pointer (inputPtr) will be pass down to firmware for START operation mode while job input length (inputLength) is not 0. Autosar does not require an input pointer for START operation mode but the input data can be processed at START by providing an input pointer and input length. When

the input is processed the job output pointer (outputPtr) and job output pointer length (output←LengthPtr) must be provieded.

    * job input length (inputLength) will be pass down to firmware for START operation mode, if no processing is desired at START operation mode the input length should be 0.
    * job secondary input pointer (secondaryInputPtr), pointing to additional data, and secondary input length (secondaryInputLength) must be provided at START operation mode.
    * For FINISH operation mode to skip the processing of the job input pointer (inputPtr) the job input length (inputLength) should be 0.
  - AEADDECRYPT:
    * job input pointer (inputPtr) will be pass down to firmware for START operation mode while job input length (inputLength) is not 0. Autosar does not require an input pointer for START operation mode but the input data can be processed at START by providing an input pointer and input length. When the input is processed the job output pointer (outputPtr) and job output pointer length (output←LengthPtr) must be provieded.
    * job input length (inputLength) will be pass down to firmware for START operation mode, if no processing is desired at START operation mode the input length should be 0.
    * For FINISH operation mode to skip the processing of the job input pointer (inputPtr) the job input length (inputLength) should be 0.
    * job secondary input pointer (secondaryInputPtr), pointing to additional data, and secondary input length (secondaryInputLength) must be provided at START operation mode.
  - SIGNATUREGENERATE and SIGNATUREVERIFY:
    * job input pointer (inputPtr) will be pass down to firmware for START operation mode while job input length (inputLength) is not 0. Autosar does not require an input pointer for START operation mode but the input data can be processed at START by providing an input pointer and an input length.
    * job input length (inputLength) will be pass down to firmware for START operation mode, if no processing is desired at START operation mode the input length should be 0.
    * For FINISH operation mode to skip the processing of the job input pointer (inputPtr) the job input length (inputLength) should be 0.
  - RANDOMGENERATE is not supported in streaming.

- For Key Derivation PBKDF2, X963, TLS, only the following HASH algorithms are supported:
  - CRYPTO_ALGOFAM_SHA2_224
  - CRYPTO_ALGOFAM_SHA2_256
- Hse firmware supports HMAC with the following hash algorithms: CRYPTO_ALGOFAM_SHA1, CRYPTO_ALGOFAM_SHA2_224, CRYPTO_ALGOFAM_SHA2_256 , CRYPTO_ALGOFAM_SHA2←_384, CRYPTO_ALGOFAM_SHA2_512.
- Hse firmware does not support XCBCMAC therefore this service is not supported in Crypto driver either.
- Notes about compiling the driver code with gcc 9.2 with no optimizations, in order to ease the debug process:
  - The driver code is tested before being released and should be compiled when integrated in the application using the compiler options listed in the Integration Manual, chapter 'Build Options'. These are the only compiler options that guarantee the driver will behave as expected.
  - Still, during the integration process, the user might want to be able to easily debug the code, which can prove difficult because the switch for compiler optimizations is set to optimize the code for size (-Os).
  - User might be tempted to temporary disable the gcc optimizations using the -O0 switch instead of -Os. This is not recommended with gcc 9.2 because of a bug in the toolchain that would prevent the static not initialized variables being correctly placed in the memory sections they should be put in by means of Crypto_MemMap.h file. This might cause wrong behavior of Crypto driver due to the fact that some

internal variables are not correctly placed in non-cacheable memory areas and the requests sent from Crypto are not correctly seen by the HSE Firmware. Until a fix in gcc toolchain is made, the user should use the switch -Og (which stands for Optimizations for debuG) for debug purposes. The use of -Og switch allows user to easily debug the code while keeping it functioning correctly, as all the variables are put in the right memory sections.

- Hse firmware does not support AES XTS therefore this service is not supported in Crypto driver either.

3.6 Driver usage and configuration tips

Crypto Driver Objects

A maximum of 2 Crypto Driver Objects can be configured in Tresos per core, one allowing access to symmetric cryptographic primitives and the second allowing access to asymmetric primitives. Each primitive can be executed with the help of Crypto_ProcessJob() API either in synchronous or asynchronous mode. If multiple Crypto_ProcessJob() in asynchronous mode are received, these are sequentially sent to the configured MU instance, each job on one MU channel, until no more API calls are received or until no more free MU channels are available. For the second case, the user can opt to enable the Crypto Driver Object's software queue by setting the CryptoQueueSize attribute of the CDO to a non zero value. In this case, the asynchronous job process requests that find all MU channels busy will be put in the software queue and processed later, when MU channels used by previous requests are becoming free. In order to exchange messages with the HSE, the Crypto driver uses one of the MU instances available on this platform. The 'MU Instance' Tresos attribute in 'CryptoDriverObject' container allows the user to choose which of the MU interfaces the Crypto driver will use for the Crypto Driver Object. If driver is used in single core configuration and there are 2 Crypto Driver Objects configured (one for Symmetric, one for Asymmetric primitives), the same MU instance should be used for both Crypto Driver Objects. If driver is used in multicore configuration, then every pair of Crypto Driver Objects configured on a core (one for Symmetric, one for Asymmetric primitives) should use the same MU instance. The MU instance used by the Crypto Driver Objects on one core should be distinct from and consecutive to the MU instance used by the Crypto Driver Objects on another core.

Polling vs Interrupt mode

Asynchronous jobs can be processed by the Crypto driver in 2 modes. In 'Polling' mode, the entity running on top of the Crypto driver should call periodically the API 'Crypto_MainFunction' in order to ensure that HSE responses are read when available from the MU and also to ensure that the jobs waiting in the CDO queue are being dequeued and sent to HSE through MU. In 'Interrupt' mode, there is no need to call the 'Crypto_MainFunction' API periodically. Both reading of responses from HSE and processing of the jobs waiting in the CDOs queues are done in interrupt context and is handled by the Crypto driver internally.

Crypto Keys

Key elements and keys have to be configured for all primitives supported in this release. Containers CryptoKey↩ Elements, CryptoKeyTypes and CryptoKeys should be activated or deactivated in Tresos in the same time. For a key it is mandatory to have a key type and configured key elements. The index of the different key elements from the different Crypto services are defined as in imported types table SWS_Csm_01022. A CryptoKeyElement having the CryptoKeyElementId set to 1,9 or 10 represents a key material and cannot be set by using the field Crypto↩ KeyElementInitValue. All the other CryptoKeyElementIds can be set either using CryptoKeyElementSet function or the Tresos field CryptoKeyElementInitValue. All the key elements that are key material are stored by HSE so UseHseKey field from Tresos should be enabled. All the other key elements that are different than key material are stored by Crypto Driver and Use HSE key field should be disabled. HSE key elements have 2 specific attributes that are used to identify a corresponding memory slot:

- HSE Key Catalog Group Ref is used in order to identify the Catalog of the key and the type of the catalog that could be RAM or NVM.
- HSE Key Slot will be used in order to set a specific slot from the selected catalog. The maximum index of slot should be according to the configured KeyCatalog.

For HSE key elements there are also other Tresos attributes that should be configured:

HSE Encrypted key Import This Tresos attribute should be enabled if the HSE key element will be further used in an encrypted key import operation

HSE Authenticated Key Import This Tresos attribute should be enabled if the HSE key element will be further used in an authenticated key import operation

HSE Key Export This Tresos attribute should be enabled if the HSE key element will be further used in an authenticated key export operation

HseKeyFlags HSE key elements should be configured with the correct HSE flag using the container from Tresos. For example if the key element will be used in an encryption operation HSE_KF_USAGE_ENCRYPT should be set.

HSE ECC Curve Id For HSE key elements of ECC type the ECC curve this attribute should be configured.This id will be used for import key operations.

HSE Key Counter For HSE key elements that are stored in NVM, the 28 bits counter should be configured. When updating a key value and attributes the new counter value must be greater than the current counter value. At counter saturation (0xFFFFFFF) the key counter cannot be updated anymore.

HSE SMR Flags If the SMR feature is enabled the application could restrict key usage depending on the SMR verification status. This restriction can be configured using SMR flags Tresos attribute, that is a map of bits that defines which secure memory region should be verified before the key can be used. Set to zero means not used.

A key has a state which is either 'valid' or 'invalid'. By default, all the keys are 'invalid and have to be set to valid by using the function Crypto_KeySetValid. If a key is in the state 'invalid', then the Crypto services which make use of that key will return CRYPTO_E_KEY_NOT_VALID value.

Loading a key

To load symmetric or asymmetric keys, the following sequence should be followed:

- The containers CryptoKeyElements, CryptoKeyTypes, CryptoKeys should be enabled.
- Depending of the desired memory location the containers NvmKeyCatalog or RamKeyCatalog should contain a slot for the imported key type having a length equal or larger than the key material.
- Crypto_KeyElementSet() API requires the following CryptoKeyElements to be referenced in CryptoKeyType of the CryptoKey container:
  - SHE keys in plain need a CryptoKeyElement(CryptoKeyElementId = 1) configured.
  - SHE keys encrypted need the following CryptoKeyElements configured: key material (CryptoKeyElementId = 1). Optionally, mac proof (CryptoKeyElementId = 2) and cipher proof (CryptoKeyElementId = 6) key elements can be configured.
  - Asymmetric (ECC) keys need a key material (CryptoKeyElementId = 1) CryptoKeyElement configured.
  - Symmetric keys need a key material (CryptoKeyElementId = 1) CryptoKeyElement configured.

- Key restrictions should be applied in key flags tab depending of the key purpose as they will apply to the key after a successful load.
- CryptoKeyElements extended fields:
  - 'Use HSE Key' enables HSE key usage and management.
  - 'HSE Key Catalog Group Ref' selects key group where the key is located in the NVM or RAM key catalog.
  - 'HSE Key Slot' selects the key slot inside the key group selected.
  - 'HSE Encrypted Key Import' enables the encrypted load of the HSE key. When enabled, all the parameters related to encrypted key import can be configured in the 'Encrypted Key Import' tab.
  - 'HSE Authenticated Key Import' enables the authenticated import of the HSE key. When enabled, all the parameters related to authenticated key import can be configured in the 'Authenticated Import' tab.
  - 'HSE Key Counter' used in updating a NVM key, the value must be greater than the current counter value.
  - 'HSE ECC Curve Id' the curve of the ECC key.
- The encrypted and authenticated key loading require knowledge of a provision key that is already present in HSE referenced in 'Encrypted Key Import' tab or 'Authenticated Import' tab along with cipher or authentication scheme that will be used in decryption or authentication of the loaded key.

Processing a primitive

To process a primitive (random number generation, MAC generation or verification, AES encrypt/decrypt, hash, signature generation or verification, AEAD decrypt/ encrypt, key generation, key derive, key exchange secret), the following sequence should be followed:

- If keys are needed, the containers CryptoKeyElements, CryptoKeyTypes, CryptoKeys should be enabled.
- Suppose AES GCM encryption is wanted, thus a key material and an initialization vector (IV) key element are required. In Tresos, two key elements have to be configured:
  - The key material itself, having CryptoKeyElementId 1, format CRYPTO_KE_FORMAT_BIN_OCTET, CryptoKeyElementInitValue should be left blank.
  - An initialization vector, having CryptoKeyElementId 5, format CRYPTO_KE_FORMAT_BIN↩ _OCTET. CryptoKeyElementInitValue should be set as the value wanted as IV (for instance↩ : 1a2f5326aaddccee297461ac).
- Inside the container CryptoKeyTypes, one CryptoKeyType should be configured containing two CryptoKey↩ ElementRef that should point to the above configured CryptoKeyElements.
- Inside the container CryptoKeys, one CryptoKey should be configured containing one CryptoKeyTypeRef that should point to the above configured CryptoKeyType and have a CryptoKeyId set(for instance: 1).
- In Tresos, a symmetric Crypto Driver Object should be configured, having a CryptoDriverObjectId set (for instance: 1) and also having the required primitive configured. For instance, in case AES GCM encryption is wanted, on the Crypto Primitives container the following should be set:
  - CryptoPrimitiveService set as CRYPTO_AEADENCRYPT
  - CryptoPrimitiveAlgorithmFamily set as CRYPTO_ALGOFAM_AES
  - CryptoPrimitiveAlgorithmMode set as CRYPTO_ALGOMODE_GCM
  - CryptoPrimitiveAlgorithmSecondaryFamily set as CRYPTO_ALGOFAM_NOT_SET

  This Crypto primitive ref should be linked to Crypto Driver Object with symmetric primitives(for instance: the Crypto Driver Object with CryptoDriverObjectId set to 1).

- Call the API function Crypto_KeyElementSet(1, 1, aes_key, 16), meaning a key material corresponding to a key with ID 1 and having the size 16 bytes is configured.
- Call the API function Crypto_KeySetValid(1) to enable a key with ID 1.
- Call the API function Crypto_ProcessJob(1, job) to process a job on Crypto Driver Object with ID 1, where the job should be defined as a Crypto_JobType structure.

- Suppose AES FAST_CMAC generation is wanted, thus a key material is required. In Tresos, one key element has to be configured: the key material itself, having CryptoKeyElementId 1, format CRYPTO_KE_↩ FORMAT_BIN_SHEKEYS, CryptoKeyElementInitValue should be left blank. Supposing the SHE RAM key will be used, the checkbox Use HSE Key should be enabled, the HSE Key Catalog Group Ref should be set to the first group in the RAM key catalog and the HSE Key Slot should be set to 0.
- Inside the container CryptoKeyTypes, one CryptoKeyType should be configured containing two CryptoKey↩ ElementRef that should point to the above configured CryptoKeyElement.
- Inside the container CryptoKeys, one CryptoKey should be configured containing one CryptoKeyTypeRef that should point to the above configured CryptoKeyType and have a CryptoKeyId set (for instance: 2).
- In Tresos, a symmetric Crypto Driver Object should be configured, having a CryptoDriverObjectId set (for instance: 1) and also having the required primitive configured. For instance, in case AES FAST_CMAC generation is wanted, on the Crypto Primitives container the following should be set:
  - CryptoPrimitiveAlgorithmFamily set as CRYPTO_ALGOFAM_AES
  - CryptoPrimitiveAlgorithmMode set as CRYPTO_ALGOMODE_CUSTOM_FAST_CMAC
  - CryptoPrimitiveAlgorithmSecondaryFamily set as CRYPTO_ALGOFAM_NOT_SET
  - CryptoPrimitiveService set as MAC_GENERATE

  This Crypto primitive ref should be linked to Crypto Driver Object with symmetric primitives(for instance: the Crypto Driver Object with CryptoDriverObjectId set to 1).
- Call the API function Crypto_KeyElementSet(2, 1, she_ram_key, 16), meaning a key material corresponding to a key with ID 1 and having the size 16 bytes is configured.
- Call the API function Crypto_KeySetValid(2) to enable a key with ID 2.
- Call the API function Crypto_ProcessJob(1, job) to process a job on Crypto Driver Object with ID 1, where the job should be defined as a Crypto_JobType structure.

Note: When configuring a FAST_CMAC CSM job to be sent to Crypto_ProcessJob() for processing, both input↩ Length and outputLengthPtr parameters should be provided in bits and not bytes.

Filling the CryptoPrimitives container with the entire list of primitives supported by the Crypto driver

At least one recommended configuration is available in the Crypto Tresos plugin, in the config_ext plugin sub-folder. The recommended configuration contains a list with all Crypto primitives that are supported by the driver. When used, it will automatically fill the CryptoPrimitives container with the services supported by the driver (HASH, ENCRYPT, DECRYPT, etc), each of these containers having inside the list of detailed supported combinations of primitives based on AlgorithmFamily, AlgorithmMode and SecondAlgorithmFamily. Before adding the Crypto plugin to a Tresos project, the user should select first in the 'Default Recommended Configuration' combo-box one of the recommended configurations available for the Crypto driver. When the button 'Add module configurations for selected modules' in the Tresos 'Module Configuration' window is pressed, if the Crypto driver is in the list of the selected plugins, it will be added to the Tresos project and its 'CryptoPrimitives' container will be automatically filled with the list of primitives supported.

There can be multiple recommended configurations available to select from when:

- the current platform has multiple derivatives and the list of primitives supported by the HSE Firmware differs between derivatives
- there are multiple HSE Firmware flavors for the current platform and derivative (eg. Standard, Premium) that the user can choose to run the Crypto driver on top of and the list of primitives supported by the HSE Firmware differs between flavors

If multiple recommended configurations are available to select from, please choose the one that contains in the description the derivative and/or the HSE Firmware flavor used by the application.

Key Management API functionality through Crypto_ProcessJob()

ASR 4.4 SWS requires that Crypto_ProcessJob() API is able to handle Key Management functionality. This means that same functionality achieved by calling the APIs Crypto_RandomSeed(), Crypto_KeyGenerate(), Crypto_KeyDerive(), Crypto_KeyExchangeCalcPubVal(), Crypto_KeyExchangeCalcSecret(), Crypto_KeySetValid(), Crypto_CertificateParse(), Crypto_CertificateVerify() should be available in Crypto_ProcessJob(), when the job is configured for key management services. In order to optimize driver's code size and execution time in case key management services functionality are not needed in Crypto_ProcessJob(), the entire functionality can be added/removed from the code by checking/unchecking the 'Enable Job Key Management Support' boolean control in the 'CryptoGeneral' tab of the plugin.

Crypto Timeout configuration

APIs that request a HSE service and are synchronous (eg. key management APIs, synchronous jobs, etc) are writing the request in the MU registers and after that remain in a loop, waiting for the HSE to respond. In case that for some reason HSE does not provide a response in a timely manner, the waiting loop should be exited after:

- it has been executed for a maximum allowed number of times or
- a number of microseconds have elapsed

'Crypto Timeout' attribute in the 'CryptoGeneral' tab of the Tresos plugin allow the user to configure the **default** value for any of the 2 cases above. If the 'Timeout Counter Type' attribute in the 'CryptoGeneral' tab of the Tresos plugin is configured to 'OSIF_COUNTER_DUMMY', the 'Crypto Timeout' attribute will contain the maximum number of times the waiting loop is allowed to be executed, before the driver reports a 'CRYPTO_E_RE_↩ OPERATION_TIMEOUT' runtime error. If the 'Timeout Counter Type' attribute in the 'CryptoGeneral' tab of the Tresos plugin is configured to 'OSIF_COUNTER_SYSTEM', the 'Crypto Timeout' attribute will contain the maximum number of microseconds the waiting loop is allowed to be executed, before the driver reports a 'CRYPTO↩ _E_RE_OPERATION_TIMEOUT' runtime error.

The value of the timeout can also be configured at runtime with the help of the Autosar extension function 'Crypto↩ _Exts_SetSynchronousRequestsTimeout'. The type of the timeout (number of loops vs microseconds) can only be set at configuration time, by choosing one of the OSIF_COUNTER_DUMMY or OSIF_COUNTER_SYSTEM values for the 'Timeout Counter Type' Tresos attribute. Thus, when calling the function 'Crypto_Exts_Set↩ SynchronousRequestsTimeout' at runtime, the 'u32Timeout' parameter value will be measured in either ticks or microseconds, depending on what value was chosen by the user at configuration time for the 'Timeout Counter Type' Tresos attribute.

Crypto driver persistent information

According with the Crypto ASR 4.4 SWS, some information like for example key validity or values of Crypto Key Elements marked as persistent should be stored by Crypto in a non volatile memory area. As the mechanism for implementing this support in the driver would be complex, Crypto driver relies on the upper layer to store or retrieve information to/from NVRAM, at driver's request. This is done through a NVM Blob handler and 2 blobs of information that should be kept persistent across resets. The Tresos attribute is optional and should be enabled if the feature is desired. When enabled, the Crypto driver will call the handler when it needs to notify the upper layer that the information in one of the NVRAM blob has been updated. The handler should be defined and implemented in the upper layer. Its name is configurable and should be set in the 'Update Nvram Blob Handler' filed in 'CryptoGeneral' tab of the Tresos plugin. The function must use the following prototype: Std_ReturnType (uint32 u32BlobId, uint32 u32BlobLength), the parameters purpose is as follows:

- 'u32BlobId' holds the identifier of the blob the Crypto driver is requesting the upper layer to save to NVRAM and can have one of the 2 possible following values:
  - CRYPTO_NVRAM_BLOB_0_ID
  - CRYPTO_NVRAM_BLOB_1_ID
- 'u32BlobLength' holds the length of the blob the Crypto driver is requesting the upper layer to update to NVRAM The two blobs should be defined in the upper layer as follows:
- Declare a variable: uint8 Crypto_au8NvramBlob0[CRYPTO_SIZEOF_NVRAM_BLOB_0];
- Declare a variable: uint8 Crypto_au8NvramBlob1[CRYPTO_SIZEOF_NVRAM_BLOB_1];

The first blob contains information about key validity flags, while the second contains information about lengths of key elements and actual values of the ones marked as persistent. To define the handler, add code in the body of the function that will save in non volatile memory the content of either Crypto_au8NvramBlob0[] or Crypto↩ _au8NvramBlob1[] arrays, depending on the value of the u32BlobId received parameter, CRYPTO_NVRAM_↩ BLOB_0_ID for the keyValid blob or CRYPTO_NVRAM_BLOB_1_ID for persistent Crypto Key Elements. The function should return E_OK if the Nvram save operation was successful and E_NOT_OK otherwise.

Redirection of Input Output of Crypto Jobs

This service is enabled by Tresos attribute Enable redirection support. The supported primitives for this feature are:

- CRYPTO_MACGENERATE - CRYPTO_ALGOMODE_HMAC
- CRYPTO_MACVERIFY - CRYPTO_ALGOMODE_HMAC
- CRYPTO_HASH
- CRYPTO_ENCRYPT - CRYPTO_ALGOFAM_AES
- CRYPTO_DECRYPT - CRYPTO_ALGOFAM_AES

Using this service input or output data of a job can be re-directed to a key element. This service is supported for key elements that are not key material, so the CryptoKeyElementId should be different than 1, 9 or 10. All the key elements should be statically configured at compile time and shall not be changed at runtime. For example if the primary input key and its element is wanted to be used instead of the input pointer, the user should configure the redirectionConfig field from the structure Crypto_JobRedirectionInfoType, by setting the least significant bit. The value of redirectionConfig is a bit coded value that is used to indicate, which of the input and output buffers are redirected. If the bit 1 is set instead, the secondaryInputBuffer is redirected to the secondary input key . A value of redirectionConfig of "00110001" indicates that the input should be taken from the inputKeyElement of inputKeyId and that the output buffer and secondary output buffer shall be redirected to the outputKeyElement of outputKeyId and secondaryOutputKeyElement of secondaryOutputKeyId.

Alternate Mapping of Crypto Job Key

This feature is enabled by Tresos attribute 'Enable Alternate Mapping of Crypto Job Key'.

When enabled, the Crypto driver will read the key related information of Csm jobs from an alternate location which is the cryptoKeyId member of the Crypto_JobType structure. The presence of the cryptoKeyId member in the Crypto_JobType structure is not requested by Autosar 4.4. Because of this reason, care must be taken to enable this boolean only if the CSM layer that is part of the same crypto stack with the current Crypto driver declares the cryptoKeyId as member of Crypto_JobType structure.

When disabled, the Crypto driver will read the key related information of Csm jobs from the cryIfKeyId member of Crypto_JobPrimitiveInfoType substructure of the Crypto_JobType structure, following the specification of Autosar 4.4 standard.

Processing MacVerify Jobs with SecondaryInputLength Using Bits

This feature is controlled by the boolean attribute with the label 'SecondaryInputLength For MacVerify Jobs Uses Bits' in the 'CryptoGeneral' container in the configuration tool of the Crypto plugin. The attribute above is used for configuring the measure unit the Crypto driver will use for processing the secondaryInputLength parameter of a received MacVerify job, when a Crypto_ProcessJob() API is called.

The need for the presence of this switch comes from an incoherence in the ASR 4.4 CSM spec. On one hand, the ASR 4.4 CSM spec declares in SWS_Csm_01009 that the secondaryInputLength is always measured in bytes. On the other hand, the same ASR 4.4 CSM spec states in SWS_Csm_01050 that the macLength parameter (which is mapped in the end on the secondaryInputLength member of the job that will be sent down to CryIf and then Crypto) contains 'the MAC length in BITS to be verified'.

The boolean attribute 'SecondaryInputLength For MacVerify Jobs Uses Bits' was added in order to support in the Crypto driver both cases when the secondaryInputLength parameter of a received MacVerify job is measured in bytes or bits.

When enabled, the Crypto driver will consider that the secondaryInputLength parameter of a received MacVerify job is measured in bits.

When disabled, the Crypto driver will consider that the secondaryInputLength parameter of a received MacVerify job is measured in bytes.

Processing FastCmac Jobs with inputLength and resultLength Using Bytes

This feature is controlled by the boolean attribute with the label 'InputLength And ResultLength For FastCmac Jobs Use Bytes' in the 'CryptoGeneral' container in the configuration tool of the Crypto plugin. The attribute above is used for configuring the measure unit the Crypto driver will use for processing the inputLength and resultLength parameters of a received FastCmac job, when a Crypto_ProcessJob() API is called.

The need for the presence of this switch comes from the fact that the FastCmac service is designed in HSE FW to work with lengths measured in bits and the initial implementation of the FastCmac support in Crypto driver was considering that being a custom CSM service, the inputLength and resultLength parameters in a FastCmac job will also be measured in bits. The ASR 4.4 CSM spec states in SWS_Csm_01009 that the inputLength is measured in bytes and in SWS_Csm_01011 that also resultLength is measured in bytes.

The boolean attribute 'InputLength And ResultLength For FastCmac Jobs Use Bytes' was added in order to support in the Crypto driver both cases when the inputLength and resultLength parameters of a received FastCmac job are measured in bytes or bits.

When enabled, the Crypto driver will consider that the inputLength and resultLength parameters of a received FastCmac job are measured in bytes.

When disabled, the Crypto driver will consider that the inputLength and resultLength parameters of a received FastCmac job are measured in bits.

ASR Extension services offered by 'Hse_Ip' interface

The Crypto driver code encapsulates one layer called 'Hse_Ip' which allows an upper entity to use directly all the services offered by HSE firmware. The Hse_Ip layer's services are available by including the header file 'Hse_Ip.h' and are listed below:

1. Hse_Ip_Init() Must be called prior to any other service request from Hse_Ip layer. It is responsible with initializing the internal variables of the layer and initializing the MU instance.
2. Hse_Ip_Deinit() It is responsible with deinitializing the internal variables of the layer.
3. Hse_Ip_GetFreeChannel() This function finds and locks the first available channel of the MU instance sent as parameter that the caller entity can use for sending a request to HSE FW.
4. Hse_Ip_ReleaseChannel() This releases the lock on an MU channel, making it available for other tasks.
5. Hse_Ip_ServiceRequest() This API is used to request a service to HSE. It has 4 parameters:
    - u8MuInstance: MU instance to be used for sending the request
    - u8MuChannel: Mu channel to be used for sending the request
    - pRequest: pointer to a request structure specifying if the request is synchronous or asynchronous, what is the value of the timeout in case it is synchronous or what is the name of the callback to be called when request completes, if it is asynchronous
    - pHseSrvDesc: pointer to a HSE request descriptor structure which should be filled by the caller. To avoid possible coherency issues when D-CACHE is enabled, the user shall ensure that the descriptor structure along with the buffers used as input or output parameters are allocated in the NON-CACHEABLE area (by means of MemMap)
6. Hse_Ip_MainFunction() Main function of the Hse_Ip layer. Should be called periodically by the upper layer, in order for the asynchronous requests in polling mode to have the chance to complete.
7. Hse_Ip_GetHseStatus() Function that allows the caller entity to get status information from HSE. Status information is retrieved as a map of 16 bits, each bit corresponding to the functional states of different parts of HSE firmware. The bits are defined in 'hse_interface.h' header file, under the type 'hseStatus_t' and for example they can have values like 'HSE_STATUS_BOOT_OK' or 'HSE_STATUS_RNG_INIT_OK'. The status information returned by HSE is identical no matter the MU instance used as parameter. The MU instance parameter is present only to allow the application the flexibility to read the information using one of the MU(s) that is/are assigned to the core the app runs on. This function is operational even when Hse_Ip layer is not yet initialized with a call to Hse_Ip_Init() API.
8. Hse_Ip_RegisterGenericCallback() This function allows the caller entity to register a function that will be called by HSE when some critical information needs to be reported, like for example the fact that some subsystems are down. In order for the function to be called, the following interrupt handler should be mounted in the vector table, on the position corresponding to the General Interrupt for the MU instance that is configured in the Tresos plugin: Mu_Ip_Mu0_OredGP_Isr or Mu_Ip_Mu1_OredGP_Isr.

ASR Extension services offered by 'Crypto_ASRExtension' interface

The Crypto driver code encapsulates one layer called 'Crypto_ASRExtension' which allows an upper entity to request the Crypto driver some extension services. The Crypto_ASRExtension layer's services are available by including the header file 'Crypto_ASRExtension.h' and are listed below:

- Crypto_Exts_FormatKeyCatalogs() Formats the HSE NVM and RAM keycatalogs, based on the configuration done in Tresos. Using this format service Host can configure the following:
  - A MU instance map which defines the MU that can be used to access the key group
  - The owner of the key group
  - The key type
  - The key number of key slots
  - The maximum key size in bits

  Before any keys can be used, HOST should trigger the key catalog formatting service. If the key type is set to SHE keys the entry must be configured at index 0 of the key catalog.

- Crypto_Exts_SetSynchronousRequestsTimeout() Sets the timeout for synchronous job requests. For more details, please see the paragraph **Crypto Timeout configuration** above
- Crypto_Exts_SHE_BootFailure() Applies sanctions if a failure was detected as per SHE specification.
- Crypto_Exts_SHE_BootOk() Marks successful boot verification as per SHE specification.
- Crypto_Exts_SHE_GetStatus() The function returns the contents of the status register as per SHE specification:
  - SECURE_BOOT is set if the secure booting is activated.
  - SECURE_BOOT is set if the secure booting is activated.
  - BOOT_FINISHED is set when the secure booting has been finished by calling either CMD_BOOT_↩ FAILURE or CMD_BOOT_OK or if secure boot failed in verifying BOOT_MAC.
  - BOOT_OK is set if the secure booting succeeded.
  - RND_INIT is set if the random number generator has been initialized.
  - EXT_DEBUGGER is set if host debug session is active.
  - INT_DEBUGGER is set when a debug session is active.
- Crypto_Exts_SHE_GetId() The function returns the identity (UID) and the value of the status register protected by a MAC over the concatenation of challenge, UID and status register.
- Crypto_Exts_SHE_DebugChal() The function returns a 128-bit random challenge that is used in conjunction with Crypto_Exts_SHE_DebugAuth().
- Crypto_Exts_SHE_DebugAuth() Performs authorization and erases all keys except SECRET_KEY and UID. The service will only work if no key is write-protected, has the WRITE_PROTECTED flag set.
- Crypto_Exts_MPCompression() One-way compression function used to derive a 128 bit output from a given message.

ASR Extension services offered through 'Crypto_KeyElementGet' API

The Crypto driver code encapsulates a functionality that allows an upper entity to request any of the services offered by the Hse Firmware by making a call to Crypto_KeyElementGet() API, providing a particular set of parameters. This functionality is by default disabled in the code and can be enabled by ticking the 'Enable Feeding Hse Descriptors Support' checkBox in the CryptoGeneral container in the configuration tool of the Crypto plugin. The following steps need be performed at configuration time:

- CryptoKeyElement container in the Crypto configuration tool shall contain a key element with the following properties:

  - CryptoKeyElementId set to 0xFEEDDE5C.
  - CryptoKeyElementSize set to 1.
  - 'Use Hse Key' set to false.
- CryptoKeyType container in the Crypto configuration tool shall contain an element referring only the key element described above.
- CryptoKey container in the Crypto configuration tool shall contain an element referring only the key type described above.

The following steps need be performed in code, in order to access the functionality:

- Declare a variable of type hseSrvDescriptor_t. Fill it with the parameters of the request to be feed directly to Hse Firmware. If data cache is enabled, make sure this variable is declared in a non-cacheable memory area.
- Declare a variable of type hseSrvResponse_t where the value of the Hse Firmware response to the request will be stored. Initialize it with something different than 0x0. Being a uint32 variable, this can be declared on the stack of the function doing the call to Crypto_KeyElementGet().
- Make the call to Crypto_KeyElementGet() API, providing the following parameters:
  - cryptoKeyId set to the identifier of the Crypto key configured at configuration time.
  - keyElementId set to the value 0xFEEDDE5C.
  - resultPtr set to the address of the Hse descriptor variable.
  - resultLengthPtr set to the address of the Hse response variable.

If the call is successful:

- The value returned by the function Crypto_KeyElementGet() will be set to E_OK.
- The value of the Hse response variable will be set to HSE_SRV_RSP_OK
- The Hse descriptor will contain the information requested.

Notes:

- The Crypto_KeyElementGet() API call will always perform a synchronous request to Hse Firmware to process the descriptor.
- The Crypto_KeyElementGet() API can be used to import keys in the firmware's key store but those keys cannot be used by AUTOSAR key management services as the Crypto driver does not have any information about them.

EdDSA context input

Since AUTOSAR requirements do not mention the context of EdDSA, user will need to fill the context by an extension input pointer:

- Signature generation: the context is filled at the address stored in secondaryInputPtr with secondaryInput↩ Length as its length.

- Signature verification: the context is filled at the address stored in tertiaryInputPtr with tertiaryInputLength as its length.

The max context size is 255 bytes for signature generation and signature verification.

KeyDerivation algorithm and hash algorithm input

The algorithm to be used is determined by the value of the first byte in the buffer of CRYPTO_KE_↩ KEYDERIVATION_ALGORITHM key element.

- For PBKDF2, this should be set to the value of CRYPTO_ALGOFAM_PBKDF2 which is 0x24.
- For KDFX963, this should be set to the value of CRYPTO_ALGOFAM_KDFX963 which is 0x25.

The hash algorithm to be used for key derivation is determined by the value of the first byte in the buffer of CRYPTO↩ _KE_KEYDERIVATION_ALGORITHM_SECONDARY key element. CRYPTO_KE_KEYDERIVATION_↩ ALGORITHM_SECONDARY defines an NXP specific key element id, having the value set to 100U. The values for the hash algorithm should be taken from Crypto_AlgorithmFamilyType enumeration.

The available hash algorithms for key derivation are defined under Crypto_AlgorithmFamilyType enumeration and listed below:

- CRYPTO_ALGOFAM_SHA2_224
- CRYPTO_ALGOFAM_SHA2_256
- CRYPTO_ALGOFAM_SHA2_384
- CRYPTO_ALGOFAM_SHA2_512
- CRYPTO_ALGOFAM_SHA2_512_224
- CRYPTO_ALGOFAM_SHA2_512_256

Iterations used in PBKDF2 Key Derivation

The number of iterations is given in the key element CRYPTO_KE_KEYDERIVATION_ITERATIONS. The Crypto driver will pass to the firmware a 32bit value containing the number of iterations taken from the key element(which is stored as an array of 8bit values), thus the key element must follow several rules:

- CryptoKeyElementSize must be equal or larger than 4
- If the value set in CryptoKeyElementInitValue will be used in the key derivation the string should be a 32bit value(8 nibbles)
- At run time the number of iterations can be updated using the Crypto_KeyElementSet() API having the size of the input (keyLength) set to 4

RSA OAEP Encryption/Decryption label usage

Job's SecondaryBuffer is used as storage of label information.

- secondaryInputLength stores the label length which must be less than 128.

- secondaryInputPtr points to the memory which stores the label data.

If label is not used, please assign label length as zero and label pointer as NULL.

RSA PSS used in authenticated key import/export

The salt length can be specified and may take the following values:

- between 0 and 62 if the key length is 128 bytes and SHA-512 is used as hash algorithm
- between 0 and hash length (output length of the chosen hash algorithm)

___Support for Crypto access of TCM regions The Crypto driver supports automatic backdoor access of DTCM and ITCM memory regions if the feature CryptoEnableTcmSupport is enabled. In this case, the driver will detect if the address it needs to access is within a TCM region, and add the core specific offset needed in order to allow the HSE to access the region.

Security Recommendations

CryptoDevErrorDetect is a boolean in the driver configuration that switches the development error detection and notification on or off, affecting multiple checks that the driver performs at run time including the API parameter checks required by Autosar specification, which are performed only when the CryptoDevErrorDetect is enabled. From the security point of view the recommendation is to enable CryptoDevErrorDetect in all product stages. This will affect the service execution time as the checks will be performed every time a Crypto driver service will be requested.

3.7 Runtime errors

The driver does not trigger any DEM runtime errors, but triggers the runtime DET errors listed in the table below:

Function	Error Code	Condition triggering the error
Crypto_ProcessJob(), Crypto_KeyExchangeCalcPubVal()	CRYPTO_E_RE_SMALL_↩ BUFFER	Buffer is too small for operation
Crypto_KeyElementGet()	CRYPTO_E_RE_KEY_NOT_↩ AVAILABLE	Requested key is not available
Crypto_KeyElementGet()	CRYPTO_E_RE_KEY_READ↩ _FAIL	Key cannot be read
Crypto_ProcessJob()	CRYPTO_E_RE_STREAM_↩ BUSY	No stream available to start a non singlecall job type
Crypto_Init(), Crypto_KeyCopy(), Crypto_KeyElementSet(), Crypto_KeyElementCopy(), Crypto_KeyElementCopyPartial(), Crypto_ProcessJob(), Crypto_KeySetValid()	CRYPTO_E_RE_NVRAM_↩ OPERATION_FAIL	Calling the upper layer services for reading/writing NVRAM information has failed
All Crypto APIs	CRYPTO_E_RE_↩ OPERATION_TIMEOUT	Timeout occurred while waiting for a response from HSE Firmware

3.8 Symbolic Names Disclaimer

All containers having symbolicNameValue set to TRUE in the AUTOSAR schema will generate defines like:

```
#define <Mip>Conf_<Container_ShortName>_<Container_ID>
```

For this reason it is forbidden to duplicate the names of such containers across the RTD configurations or to use names that may trigger other compile issues (e.g. match existing `#ifdefs` arguments).

Chapter 4

Tresos Configuration Plug-in

This chapter describes the Tresos configuration plug-in for the driver. All the parameters are described below.

- Module Crypto
    - Container CryptoDriverObjects
        - Container CryptoDriverObject
            - Parameter CryptoDriverObjectId
            - Parameter CryptoQueueSize
            - Parameter MuInstance
            - Parameter CryptoPrimitiveType
            - Reference CryptoPrimitiveRef
            - Reference CryptoDriverObjectEcucPartitionRef
    - Container CryptoGeneral
        - Parameter CryptoDevErrorDetect
        - Parameter CryptoVersionInfoApi
        - Parameter CryptoInstanceId
        - Parameter CryptoMainFunctionPeriod
        - Parameter CryptoMulticoreSupport
        - Parameter HseFwType
        - Parameter HseIpDevErrorDetect
        - Parameter CryptoTimeoutMethod
        - Parameter CryptoTimeoutDuration
        - Parameter CryptoJobKeyManagement
        - Parameter CryptoEnableRedirection
        - Parameter CryptoEnableFeedHseDesc
        - Parameter CryptoEnableTLS12KeyDeriveSupport
        - Parameter CryptoEnableUserModeSupport
        - Parameter CryptoAlternateJobKeyMapping
        - Parameter CryptoSecondaryInputLengthForMacVerifyJobsUsesBits
        - Parameter CryptoInputLengthAndResultLengthForFastCmacJobsUseBytes
        - Parameter CryptoEnableTCMSupport
        - Parameter CryptoAsyncJobProcessMethod
        - Parameter CryptoUpdateNvramBlobHandler

        * Reference CryptoEcucPartitionRef
    - Container CryptoKeyElements
        * Container CryptoKeyElement
            · Parameter CryptoKeyElementAllowPartialAccess
            · Parameter CryptoKeyElementFormat
            · Parameter CryptoKeyElementId
            · Parameter CryptoKeyElementInitValue
            · Parameter CryptoKeyElementPersist
            · Parameter CryptoKeyElementReadAccess
            · Parameter CryptoKeyElementSize
            · Parameter CryptoKeyElementWriteAccess
            · Parameter UseHseKey
            · Parameter HseKeySlot
            · Parameter EncryptedKeyImport
            · Parameter AuthenticatedKeyImport
            · Parameter EncryptedKeyExport
            · Parameter AuthenticatedKeyExport
            · Parameter HseKeyCounter
            · Parameter HseSMRFlags
            · Parameter HseEccCurveId
            · Parameter EncryptedImportHseKeySlot
            · Parameter EncryptedHseKeyBitLen
            · Parameter AuthenticatedImportHseKeySlot
            · Parameter EncryptedExportHseKeySlot
            · Parameter AuthenticatedExportPublicKey
            · Parameter AuthenticatedExportHseKeySlot
            · Reference CryptoKeyElementVirtualTargetRef
            · Reference HseKeyCatalogGroupRef
            · Reference EncryptedImportHseKeyCatalogGroupRef
            · Reference AuthenticatedImportHseKeyCatalogGroupRef
            · Reference EncryptedExportHseKeyCatalogGroupRef
            · Reference AuthenticatedExportHseKeyCatalogGroupRef
            · Container HseKeyFlags
            · Parameter HseKeyFlag
            · Container CipherScheme
            · Container AuthenticationScheme
            · Container KeyContainer
            · Parameter KeyContainerLength
            · Parameter pKeyContainer
            · Parameter AuthenticationTag0
            · Parameter AuthenticationTag1
            · Container ExportCipherScheme
            · Container ExportAuthenticationScheme
    - Container CryptoKeyTypes
        * Container CryptoKeyType
            · Reference CryptoKeyElementRef

- Container CryptoKeys
  - Container CryptoKey
    - Parameter CryptoKeyId
    - Reference CryptoKeyTypeRef
- Container CryptoPrimitives
  - Container CryptoPrimitive
    - Parameter CryptoPrimitiveAlgorithmFamily
    - Parameter CryptoPrimitiveAlgorithmMode
    - Parameter CryptoPrimitiveAlgorithmSecondaryFamily
    - Parameter CryptoPrimitiveService
- Container CommonPublishedInformation
  - Parameter ArReleaseMajorVersion
  - Parameter ArReleaseMinorVersion
  - Parameter ArReleaseRevisionVersion
  - Parameter ModuleId
  - Parameter SwMajorVersion
  - Parameter SwMinorVersion
  - Parameter SwPatchVersion
  - Parameter VendorApiInfix
  - Parameter VendorId
- Container NvmKeyCatalog
  - Parameter KeyType
  - Parameter NumOfKeySlots
  - Parameter MaxKeyBitLen
  - Parameter KeyOwner
  - Container MuMask
    - Parameter MU
- Container RamKeyCatalog
  - Parameter KeyType
  - Parameter NumOfKeySlots
  - Parameter MaxKeyBitLen
  - Parameter KeyOwner
  - Container MuMask
    - Parameter MU

4.1 Module Crypto

Configuration of the Crypto (CryptoDriver) module

Included containers:

- CryptoDriverObjects
- CryptoGeneral
- CryptoKeyElements
- CryptoKeyTypes
- CryptoKeys
- CryptoPrimitives
- CommonPublishedInformation
- NvmKeyCatalog
- RamKeyCatalog

Property	Value
type	ECUC-MODULE-DEF
lowerMultiplicity	0
upperMultiplicity	Infinite
postBuildVariantSupport	false
supportedConfigVariants	VARIANT-PRE-COMPILE

4.2 Container CryptoDriverObjects

Container for CRYPTO Objects, there can be maximum 2 Crypto Driver Objects configured: one for symmetric primitives and one for asymmetric primitives.

Included subcontainers:

- CryptoDriverObject

Property	Value
type	ECUC-PARAM-CONF-CONTAINER-DEF
lowerMultiplicity	1
upperMultiplicity	1
postBuildVariantMultiplicity	N/A
multiplicityConfigClasses	N/A

4.3 Container CryptoDriverObject

Configuration of a CryptoDriverObject

Included subcontainers:

- None

Property	Value
type	ECUC-PARAM-CONF-CONTAINER-DEF
lowerMultiplicity	1
upperMultiplicity	Infinite
postBuildVariantMultiplicity	false
multiplicityConfigClasses	VARIANT-PRE-COMPILE: PRE-COMPILE

4.4 Parameter CryptoDriverObjectId

Identifier of the Crypto Driver Object. The Crypto Driver Object offers different crypto primitives.

Property	Value
type	ECUC-INTEGER-PARAM-DEF
origin	AUTOSAR_ECUC
symbolicNameValue	true
lowerMultiplicity	1
upperMultiplicity	1
postBuildVariantMultiplicity	N/A
multiplicityConfigClasses	N/A
postBuildVariantValue	false
valueConfigClasses	VARIANT-PRE-COMPILE: PRE-COMPILE
defaultValue	0
max	4294967295
min	0

4.5 Parameter CryptoQueueSize

Size of the queue in the Crypto Driver. Defines the maximum number of jobs in the Crypto Driver Object queue. If it is set to 0, queueing is disabled in the Crypto Driver Object. Note: The node value will be used as the element number when declaring an array variable for the QUEUE feature. So the maximum value depends on the memory space of each platform.

Property	Value
type	ECUC-INTEGER-PARAM-DEF
origin	AUTOSAR_ECUC
symbolicNameValue	false
lowerMultiplicity	1
upperMultiplicity	1
postBuildVariantMultiplicity	N/A
multiplicityConfigClasses	N/A
postBuildVariantValue	false
valueConfigClasses	VARIANT-PRE-COMPILE: PRE-COMPILE
defaultValue	0
max	4294967295
min	0

4.6 Parameter MuInstance

Vendor specific: Selects one of the MU (Messaging Units) instances available on the platform to use for communication with HSE.

Property	Value
type	ECUC-ENUMERATION-PARAM-DEF
origin	NXP
symbolicNameValue	false
lowerMultiplicity	1
upperMultiplicity	1
postBuildVariantMultiplicity	N/A
multiplicityConfigClasses	N/A
postBuildVariantValue	false
valueConfigClasses	VARIANT-PRE-COMPILE: PRE-COMPILE
defaultValue	MU_0
literals	['MU_0', 'MU_1']

4.7 Parameter CryptoPrimitiveType

Vendor specific: Determines if the crypto algorithms (primitives) associated with the Crypto Driver Object are symmetric or asymmetric.

Property	Value
type	ECUC-ENUMERATION-PARAM-DEF
origin	NXP

Property	Value
symbolicNameValue	false
lowerMultiplicity	1
upperMultiplicity	1
postBuildVariantMultiplicity	N/A
multiplicityConfigClasses	N/A
postBuildVariantValue	false
valueConfigClasses	VARIANT-PRE-COMPILE: PRE-COMPILE
defaultValue	CRYPTO_SYMMETRIC_ALGORITHMS
literals	['CRYPTO_SYMMETRIC_ALGORITHMS', 'CRYPTO_ASYMMETRIC_↩ ALGORITHMS']

4.8 Reference CryptoPrimitiveRef

Refers to primitive in the CRYPTO.

Property	Value
type	ECUC-REFERENCE-DEF
origin	AUTOSAR_ECUC
lowerMultiplicity	1
upperMultiplicity	Infinite
postBuildVariantMultiplicity	false
multiplicityConfigClasses	VARIANT-PRE-COMPILE: PRE-COMPILE
postBuildVariantValue	false
valueConfigClasses	VARIANT-PRE-COMPILE: PRE-COMPILE
requiresSymbolicNameValue	False
destination	/AUTOSAR/EcucDefs/Crypto/CryptoPrimitives/CryptoPrimitive

4.9 Reference CryptoDriverObjectEcucPartitionRef

Maps the Crypto Driver Object to zero a multiple ECUC partitions. The ECUC partitions referenced are a subset of the ECUC partitions where the Crypto Driver Object is mapped to.

Property	Value
type	ECUC-REFERENCE-DEF
origin	AUTOSAR_ECUC
lowerMultiplicity	0
upperMultiplicity	Infinite
postBuildVariantMultiplicity	false
multiplicityConfigClasses	VARIANT-PRE-COMPILE: PRE-COMPILE

Property	Value
postBuildVariantValue	false
valueConfigClasses	VARIANT-PRE-COMPILE: PRE-COMPILE
requiresSymbolicNameValue	False
destination	/AUTOSAR/EcucDefs/EcuC/EcucPartitionCollection/EcucPartition

4.10 Container CryptoGeneral

Container for common configuration options

Included subcontainers:

- None

Property	Value
type	ECUC-PARAM-CONF-CONTAINER-DEF
lowerMultiplicity	1
upperMultiplicity	1
postBuildVariantMultiplicity	N/A
multiplicityConfigClasses	N/A

4.11 Parameter CryptoDevErrorDetect

Switches the development error detection and notification on or off.

Property	Value
type	ECUC-BOOLEAN-PARAM-DEF
origin	AUTOSAR_ECUC
symbolicNameValue	false
lowerMultiplicity	1
upperMultiplicity	1
postBuildVariantMultiplicity	N/A
multiplicityConfigClasses	N/A
postBuildVariantValue	false
valueConfigClasses	VARIANT-PRE-COMPILE: PRE-COMPILE
defaultValue	false

4.12 Parameter CryptoVersionInfoApi

Pre-processor switch to enable and disable availability of the API Crypto_GetVersionInfo().

Property	Value
type	ECUC-BOOLEAN-PARAM-DEF
origin	AUTOSAR_ECUC
symbolicNameValue	false
lowerMultiplicity	1
upperMultiplicity	1
postBuildVariantMultiplicity	N/A
multiplicityConfigClasses	N/A
postBuildVariantValue	false
valueConfigClasses	VARIANT-PRE-COMPILE: PRE-COMPILE
defaultValue	false

4.13 Parameter CryptoInstanceId

Instance ID of the Crypto driver. This ID is used to discern several crypto drivers in case more than one driver is used in the same ECU.

Property	Value
type	ECUC-INTEGER-PARAM-DEF
origin	AUTOSAR_ECUC
symbolicNameValue	false
lowerMultiplicity	1
upperMultiplicity	1
postBuildVariantMultiplicity	N/A
multiplicityConfigClasses	N/A
postBuildVariantValue	false
valueConfigClasses	VARIANT-PRE-COMPILE: PRE-COMPILE
defaultValue	0
max	255
min	0

4.14 Parameter CryptoMainFunctionPeriod

Specifies the period of main function Crypto_MainFunction in seconds.

Property	Value
type	ECUC-FLOAT-PARAM-DEF

Property	Value
origin	AUTOSAR_ECUC
symbolicNameValue	false
lowerMultiplicity	0
upperMultiplicity	1
postBuildVariantMultiplicity	false
multiplicityConfigClasses	VARIANT-PRE-COMPILE: PRE-COMPILE
postBuildVariantValue	false
valueConfigClasses	VARIANT-PRE-COMPILE: PRE-COMPILE
defaultValue	0.2
max	9.9999999E7
min	0.0

4.15 Parameter CryptoMulticoreSupport

Vendor specific: Enables/Disables Multicore Support.

Property	Value
type	ECUC-BOOLEAN-PARAM-DEF
origin	NXP
symbolicNameValue	false
lowerMultiplicity	1
upperMultiplicity	1
postBuildVariantMultiplicity	N/A
multiplicityConfigClasses	N/A
postBuildVariantValue	false
valueConfigClasses	VARIANT-PRE-COMPILE: PRE-COMPILE
defaultValue	false

4.16 Parameter HseFwType

Vendor specific: Selects the version of the HSE Firmware the Crypto driver will run on.

Property	Value
type	ECUC-ENUMERATION-PARAM-DEF
origin	NXP
symbolicNameValue	false
lowerMultiplicity	1
upperMultiplicity	1
postBuildVariantMultiplicity	N/A

Property	Value
multiplicityConfigClasses	N/A
postBuildVariantValue	false
valueConfigClasses	VARIANT-PRE-COMPILE: PRE-COMPILE
defaultValue	STANDARD
literals	['STANDARD']

4.17 Parameter HseIpDevErrorDetect

Vendor specific: Switches the Hse Ip layer development error detection on or off.

Property	Value
type	ECUC-BOOLEAN-PARAM-DEF
origin	NXP
symbolicNameValue	false
lowerMultiplicity	1
upperMultiplicity	1
postBuildVariantMultiplicity	N/A
multiplicityConfigClasses	N/A
postBuildVariantValue	false
valueConfigClasses	VARIANT-PRE-COMPILE: PRE-COMPILE
defaultValue	false

4.18 Parameter CryptoTimeoutMethod

Vendor specific: Counter type used in timeout detection for HSE service request.

Based on selected counter type the timeout value will be interpreted as follows:

OSIF_COUNTER_DUMMY - Ticks.

OSIF_COUNTER_SYSTEM - Microseconds.

OSIF_COUNTER_CUSTOM - Defined by user implementation of timing services

Note: If OSIF_COUNTER_SYSTEM or OSIF_COUNTER_CUSTOM are selected make sure the corresponding timer is enabled in OsIf General configuration.

Property	Value
type	ECUC-ENUMERATION-PARAM-DEF
origin	NXP
symbolicNameValue	false

Property	Value
lowerMultiplicity	1
upperMultiplicity	1
postBuildVariantMultiplicity	N/A
multiplicityConfigClasses	N/A
postBuildVariantValue	false
valueConfigClasses	VARIANT-PRE-COMPILE: PRE-COMPILE
defaultValue	OSIF_COUNTER_DUMMY
literals	['OSIF_COUNTER_DUMMY', 'OSIF_COUNTER_SYSTEM', 'OSIF_↩ COUNTER_CUSTOM']

4.19 Parameter CryptoTimeoutDuration

Vendor specific: Timeout duration defines the waiting period for HSE to respond to a synchronous request initiated by Crypto driver.

Based on selected counter type (Timeout Counter Type) the measuring unit will be determined as shown below:

OSIF_COUNTER_DUMMY - Crypto Timeout is interpreted as ticks.

OSIF_COUNTER_SYSTEM - Crypto Timeout is interpreted as microseconds.

OSIF_COUNTER_CUSTOM - Crypto Timeout is interpreted as defined by user implementation of timing services

Property	Value
type	ECUC-INTEGER-PARAM-DEF
origin	NXP
symbolicNameValue	false
lowerMultiplicity	1
upperMultiplicity	1
postBuildVariantMultiplicity	N/A
multiplicityConfigClasses	N/A
postBuildVariantValue	false
valueConfigClasses	VARIANT-PRE-COMPILE: PRE-COMPILE
defaultValue	1000000000
max	4294967295
min	1

4.20 Parameter CryptoJobKeyManagement

Vendor specific: Switch for enabling/disabling the support in Crypto driver for the Crypto_ProcessJob() service to process key management related primitives.

The key management services that can be processed by Cypto_ProcessJob() when this switch is enabled are:

RandomSeed

KeyGenerate

KeyDerive

KeyExchangeCalcPubVal

KeyExchangeCalcSecret

CertificateParse

CertificateVerify

KeySetValid

Property	Value
type	ECUC-BOOLEAN-PARAM-DEF
origin	NXP
symbolicNameValue	false
lowerMultiplicity	1
upperMultiplicity	1
postBuildVariantMultiplicity	N/A
multiplicityConfigClasses	N/A
postBuildVariantValue	false
valueConfigClasses	VARIANT-PRE-COMPILE: PRE-COMPILE
defaultValue	false

4.21 Parameter CryptoEnableRedirection

Vendor specific: The input and/or output data of a job can be re-directed to a key element.

Property	Value
type	ECUC-BOOLEAN-PARAM-DEF
origin	NXP
symbolicNameValue	False
lowerMultiplicity	1
upperMultiplicity	1
postBuildVariantMultiplicity	N/A
multiplicityConfigClasses	N/A
postBuildVariantValue	false
valueConfigClasses	VARIANT-PRE-COMPILE: PRE-COMPILE
defaultValue	false

4.22 Parameter CryptoEnableFeedHseDesc

Vendor specific: Switch for enabling/disabling the support in Crypto driver for feeding the Hse Firmware with descriptors requesting Hse services using the Crypto_KeyElementGet() API defined by Autosar Crypto spec.

When enabled, the layer on top of Crypto driver can request any of the services supported by Hse Firmware, if the following conditions are met:

CryptoKeyElement container in the Crypto configuration tool contains a key element with the following properties:

CryptoKeyElementId set to 0xFEEDDE5C

CryptoKeyElementSize set to 1

Use Hse Key set to false

CryptoKeyType container in the Crypto configuration tool contains an element referring only the key element described one step above

CryptoKey container in the Crypto configuration tool contains an element referring only the key type described one step above

The Crypto_KeyElementGet() API is called with the following parameters:

cryptoKeyId parameter set to the id of the Crypto Key configured above

keyElementId parameter set to the value 0xFEEDDE5C

resultPtr set to the address of a Hse descriptor filled with all the information needed for the request

resultLengthPtr set to the address of a uint32 variable which will contain the value of the status response received from Hse after the call

When Crypto driver receives a Crypto_KeyElementGet() API call with the parameters for feeding the Hse Firwmare with a Hse descriptor, it will perform a synchronous request to Hse Firmware to process the descriptor.

Property	Value
type	ECUC-BOOLEAN-PARAM-DEF
origin	NXP
symbolicNameValue	False
lowerMultiplicity	1
upperMultiplicity	1
postBuildVariantMultiplicity	N/A
multiplicityConfigClasses	N/A
postBuildVariantValue	false
valueConfigClasses	VARIANT-PRE-COMPILE: PRE-COMPILE
defaultValue	false

4.23 Parameter CryptoEnableTLS12KeyDeriveSupport

Vendor specific: Enables/Disables support in driver for TLS12 key derivation when using the Crypto_KeyDerive() API.

Property	Value
type	ECUC-BOOLEAN-PARAM-DEF
origin	NXP
symbolicNameValue	false
lowerMultiplicity	1
upperMultiplicity	1
postBuildVariantMultiplicity	N/A
multiplicityConfigClasses	N/A
postBuildVariantValue	false
valueConfigClasses	VARIANT-PRE-COMPILE: PRE-COMPILE
defaultValue	false

4.24 Parameter CryptoEnableUserModeSupport

Vendor specific: When this parameter is enabled, the Crypto module will adapt to run from User Mode, with the following measures:

Using 'call trusted function' stubs for all internal function calls that access registers requiring supervisor mode.

for more information, please see chapter User Mode Support in IM

Property	Value
type	ECUC-BOOLEAN-PARAM-DEF
origin	NXP
symbolicNameValue	false
lowerMultiplicity	1
upperMultiplicity	1
postBuildVariantMultiplicity	N/A
multiplicityConfigClasses	N/A
postBuildVariantValue	false
valueConfigClasses	VARIANT-PRE-COMPILE: PRE-COMPILE
defaultValue	false

4.25 Parameter CryptoAlternateJobKeyMapping

Vendor specific: Switch for enabling/disabling the support in Crypto driver for reading the key related information of Csm jobs from an alternate location.

When enabled, the Crypto driver will read the key related information of Csm jobs from an alternate location which is the cryptoKeyId member of the Crypto_JobType structure. The presence of the cryptoKeyId member in the Crypto_JobType structure is not reqeusted by Autosar 4.4. Because of this reason, care must be taken to enable this boolean only if the CSM layer that is part of the same crypto stack with the current Crypto driver declares the cryptoKeyId as member of Crypto_JobType structure.

When disabled, the Crypto driver will read the key related information of Csm jobs from the cryIfKeyId member of Crypto_JobPrimitiveInfoType substructure of the Crypto_JobType structure, following the specification of Autosar 4.4 standard.

Property	Value
type	ECUC-BOOLEAN-PARAM-DEF
origin	NXP
symbolicNameValue	False
lowerMultiplicity	1
upperMultiplicity	1
postBuildVariantMultiplicity	N/A
multiplicityConfigClasses	N/A
postBuildVariantValue	false
valueConfigClasses	VARIANT-PRE-COMPILE: PRE-COMPILE
defaultValue	false

4.26 Parameter CryptoSecondaryInputLengthForMacVerifyJobsUsesBits

Vendor specific: Switch for configuring the measure unit the Crypto driver will use for processing the secondaryInputLength parameter of a received MacVerify job, when a Crypto_ProcessJob() API is called.

The need for the presence of this switch comes from an incoherence in the ASR 4.4 CSM spec. On one hand, the ASR 4.4 CSM spec declares in SWS_Csm_01009 that the secondaryInputLength is always measured in bytes. On the other hand, the same ASR 4.4 CSM spec states in SWS_Csm_01050 that the macLength parameter (which is mapped in the end on the secondaryInputLength member of the job that will be sent down to CryIf and then Crypto) contains 'the MAC length in BITS to be verified'.

This switch was added in order to support in the Crypto driver both cases when the secondaryInputLength parameter of a received MacVerify job is measured in bytes or bits.

When enabled, the Crypto driver will consider that the secondaryInputLength parameter of a received MacVerify job is measured in bits.

When disabled, the Crypto driver will consider that the secondaryInputLength parameter of a received MacVerify job is measured in bytes.

Property	Value
type	ECUC-BOOLEAN-PARAM-DEF
origin	NXP
symbolicNameValue	False

Property	Value
lowerMultiplicity	1
upperMultiplicity	1
postBuildVariantMultiplicity	N/A
multiplicityConfigClasses	N/A
postBuildVariantValue	false
valueConfigClasses	VARIANT-PRE-COMPILE: PRE-COMPILE
defaultValue	false

4.27 Parameter CryptoInputLengthAndResultLengthForFastCmacJobsUseBytes

Vendor specific: Switch for configuring the measure unit the Crypto driver will use for processing the inputLength and resultLength parameters of a received FastCmac job, when a Crypto_ProcessJob() API is called.

The need for the presence of this switch comes from the fact that the FastCmac service is designed in HSE FW to work with lengths measured in bits and the initial implementation of the FastCmac support in Crypto driver was considering that being a custom CSM service, the inputLength and resultLength parameters in a FastCmac job will also be measured in bits. The ASR 4.4 CSM spec states in SWS_Csm_01009 that the inputLength is measured in bytes and in SWS_Csm_01011 that also resultLength is measured in bytes.

This switch was added in order to support in the Crypto driver both cases when the inputLength and resultLength parameters of a received FastCmac job are measured in bytes or bits.

When enabled, the Crypto driver will consider that the inputLength and resultLength parameters of a received FastCmac job are measured in bytes.

When disabled, the Crypto driver will consider that the inputLength and resultLength parameters of a received FastCmac job are measured in bits.

Property	Value
type	ECUC-BOOLEAN-PARAM-DEF
origin	NXP
symbolicNameValue	False
lowerMultiplicity	1
upperMultiplicity	1
postBuildVariantMultiplicity	N/A
multiplicityConfigClasses	N/A
postBuildVariantValue	false
valueConfigClasses	VARIANT-PRE-COMPILE: PRE-COMPILE
defaultValue	false

4.28 Parameter CryptoEnableTCMSupport

Vendor specific: Switch for enabling/disabling the support in Crypto driver to allow the HSE to access memory locations in DTCM and ITCM.

Property	Value
type	ECUC-BOOLEAN-PARAM-DEF
origin	NXP
symbolicNameValue	False
lowerMultiplicity	1
upperMultiplicity	1
postBuildVariantMultiplicity	N/A
multiplicityConfigClasses	N/A
postBuildVariantValue	false
valueConfigClasses	VARIANT-PRE-COMPILE: PRE-COMPILE
defaultValue	false

4.29 Parameter CryptoAsyncJobProcessMethod

Vendor specific: Selects one of the process methods for asynchronous jobs.

Property	Value
type	ECUC-ENUMERATION-PARAM-DEF
origin	NXP
symbolicNameValue	False
lowerMultiplicity	1
upperMultiplicity	1
postBuildVariantMultiplicity	N/A
multiplicityConfigClasses	N/A
postBuildVariantValue	false
valueConfigClasses	VARIANT-PRE-COMPILE: PRE-COMPILE
defaultValue	POLLING
literals	['INTERRUPT', 'POLLING']

4.30 Parameter CryptoUpdateNvramBlobHandler

Vendor specific: Crypto driver works with 2 blobs of information that should be kept persistent across resets. One blob contains information about key validity flags, while the second contains information about lengths of key elements and actual values of the ones marked as persistent. There are 2 cases for handling this information:

The blobs are stored inside Crypto driver.

The blobs are stored in the upper layer.

1. In order to use this option, do not enable the optional attribute 'Update Nvram Blob Handler'.

Given the fact that Crypto driver has no support for working with non volatile memory, in this case the information in the blobs will not be persistent across resets.

2. In order to use this option, please enable the optional attribute 'Update Nvram Blob Handler' and set it's value to a valid C function name.

When using this option, the upper layer will have to:

Declare a variable: uint8 Crypto_au8NvramBlob0[CRYPTO_SIZEOF_NVRAM_BLOB_0];

Declare a variable: uint8 Crypto_au8NvramBlob1[CRYPTO_SIZEOF_NVRAM_BLOB_1];

Implement in the code the body of a function having:

The name given in the attribute 'Update Nvram Blob Handler'.

The following prototype: Std_ReturnType <Function name>(uint32 u32BlobId, uint32 u32BlobLength)

Add code in the body of the function above that will save in non volatile memory the content of either Crypto_au8NvramBlob0[] or Crypto_au8NvramBlob1[] arrays, depending on the value of the u32BlobId received parameter, CRYPTO_NVRAM_BLOB_0_ID for the keyValid blob or CRYPTO_NVRAM_BLOB_1_ID for persistent Crypto Key Elements. The function should return E_OK if the Nvram save operation was successful and E_NOT_OK otherwise.

Property	Value
type	ECUC-FUNCTION-NAME-DEF
origin	NXP
symbolicNameValue	false
lowerMultiplicity	0
upperMultiplicity	1
postBuildVariantMultiplicity	false
multiplicityConfigClasses	VARIANT-PRE-COMPILE: PRE-COMPILE
postBuildVariantValue	false
valueConfigClasses	VARIANT-PRE-COMPILE: PRE-COMPILE
defaultValue	Crypto_UpdateNvramBlob

4.31 Reference CryptoEcucPartitionRef

Maps the Crypto driver to zero a multiple ECUC partitions to make the modules API available in this partition.

Property	Value
type	ECUC-REFERENCE-DEF
origin	AUTOSAR_ECUC

Property	Value
lowerMultiplicity	0
upperMultiplicity	Infinite
postBuildVariantMultiplicity	false
multiplicityConfigClasses	VARIANT-PRE-COMPILE: PRE-COMPILE
postBuildVariantValue	false
valueConfigClasses	VARIANT-PRE-COMPILE: PRE-COMPILE
requiresSymbolicNameValue	False
destination	/AUTOSAR/EcucDefs/EcuC/EcucPartitionCollection/EcucPartition

4.32 Container CryptoKeyElements

Container for Crypto key elements

Included subcontainers:

- CryptoKeyElement

Property	Value
type	ECUC-PARAM-CONF-CONTAINER-DEF
lowerMultiplicity	0
upperMultiplicity	1
postBuildVariantMultiplicity	false
multiplicityConfigClasses	VARIANT-PRE-COMPILE: PRE-COMPILE

4.33 Container CryptoKeyElement

Configuration of a CryptoKeyElement

Included subcontainers:

- HseKeyFlags
- CipherScheme
- AuthenticationScheme
- KeyContainer
- ExportCipherScheme
- ExportAuthenticationScheme

Property	Value
type	ECUC-PARAM-CONF-CONTAINER-DEF
lowerMultiplicity	1
upperMultiplicity	Infinite
postBuildVariantMultiplicity	false
multiplicityConfigClasses	VARIANT-PRE-COMPILE: PRE-COMPILE

4.34 Parameter CryptoKeyElementAllowPartialAccess

Enable or disable writing and reading the key element with data smaller than the size of the element.

Property	Value
type	ECUC-BOOLEAN-PARAM-DEF
origin	AUTOSAR_ECUC
symbolicNameValue	false
lowerMultiplicity	1
upperMultiplicity	1
postBuildVariantMultiplicity	N/A
multiplicityConfigClasses	N/A
postBuildVariantValue	false
valueConfigClasses	VARIANT-PRE-COMPILE: PRE-COMPILE
defaultValue	false

4.35 Parameter CryptoKeyElementFormat

Defines the format for the key element. This is the format used to provide or extract the key data from the driver.

Property	Value
type	ECUC-ENUMERATION-PARAM-DEF
origin	AUTOSAR_ECUC
symbolicNameValue	false
lowerMultiplicity	1
upperMultiplicity	1
postBuildVariantMultiplicity	N/A
multiplicityConfigClasses	N/A
postBuildVariantValue	false
valueConfigClasses	VARIANT-PRE-COMPILE: PRE-COMPILE
defaultValue	CRYPTO_KE_FORMAT_BIN_OCTET

Property	Value
literals	['CRYPTO_KE_FORMAT_BIN_IDENT_PRIVATEKEY_PKCS8', 'CRYPTO_KE_FORMAT_BIN_IDENT_PUBLICKEY', 'CRYPTO_↩ KE_FORMAT_BIN_OCTET', 'CRYPTO_KE_FORMAT_BIN_RSA↩ _PRIVATEKEY', 'CRYPTO_KE_FORMAT_BIN_RSA_PUBLICKEY', 'CRYPTO_KE_FORMAT_BIN_SHEKEYS']

4.36 Parameter CryptoKeyElementId

Vendor specific: Identifier of the CRYPTO key element.

CryptoKeyElementId set to CRYPTO_KE_EXPORT_AUTHENTICATION with value equal to 99 requires the following setup:

Empty Initialization Value

This element is not persistent

Read Acces options:CRYPTO_RA_ALLOWED/CRYPTO_RA_DENIED

Key Element Size must be equal or bigger than the size of the tag or signature, for ECDSA and EDDSA a size bigger than the sum of the signatures sizes must be specfied

Element size can be bigger or smaller than the output generated by the authentication

No need for HSE key

Property	Value
type	ECUC-INTEGER-PARAM-DEF
origin	AUTOSAR_ECUC
symbolicNameValue	true
lowerMultiplicity	1
upperMultiplicity	1
postBuildVariantMultiplicity	N/A
multiplicityConfigClasses	N/A
postBuildVariantValue	false
valueConfigClasses	VARIANT-PRE-COMPILE: PRE-COMPILE
defaultValue	1
max	4294967295
min	0

4.37 Parameter CryptoKeyElementInitValue

Value which will be used to fill the element during initialization. This node is a hexadecimal string. Please use an even number of 0-9 a-f A-F characters, without spaces. If this field is configured, it should have a number of bytes smaller or equal to CryptoKeyElementSize field.

Property	Value
type	ECUC-STRING-PARAM-DEF
origin	AUTOSAR_ECUC
symbolicNameValue	false
lowerMultiplicity	1
upperMultiplicity	1
postBuildVariantMultiplicity	N/A
multiplicityConfigClasses	N/A
postBuildVariantValue	false
valueConfigClasses	VARIANT-PRE-COMPILE: PRE-COMPILE
defaultValue	

4.38 Parameter CryptoKeyElementPersist

Enables or disables the storage of the key element value in the non-volatile memory. This functionality behaves like described below:

If the checkbox 'Use HSE key' is checked, the value in the checkbox 'CryptoKeyElementPersist' is ignored and:

If the HSE key is part of a NVM key catalog group, the key element will be persistent, stored inside HSE.

If the HSE key is part of a RAM key catalog group, the key element will be non-persistent.

If the checkbox 'Use HSE key' is not checked, the value in the checkbox 'CryptoKeyElementPersist' is considered and:

If the checkbox 'CryptoKeyElementPersist' is checked, the key element will be persistent, stored in a Crypto driver blob.

If the checkbox 'CryptoKeyElementPersist' is not checked, the key element will be non-persistent, stored in an internal Crypto driver RAM buffer.

Property	Value
type	ECUC-BOOLEAN-PARAM-DEF
origin	AUTOSAR_ECUC
symbolicNameValue	false
lowerMultiplicity	1
upperMultiplicity	1
postBuildVariantMultiplicity	N/A
multiplicityConfigClasses	N/A
postBuildVariantValue	false
valueConfigClasses	VARIANT-PRE-COMPILE: PRE-COMPILE
defaultValue	false

4.39 Parameter CryptoKeyElementReadAccess

Define the reading access rights of the key element.

Property	Value
type	ECUC-ENUMERATION-PARAM-DEF
origin	AUTOSAR_ECUC
symbolicNameValue	false
lowerMultiplicity	1
upperMultiplicity	1
postBuildVariantMultiplicity	N/A
multiplicityConfigClasses	N/A
postBuildVariantValue	false
valueConfigClasses	VARIANT-PRE-COMPILE: PRE-COMPILE
defaultValue	CRYPTO_RA_ALLOWED
literals	['CRYPTO_RA_ALLOWED', 'CRYPTO_RA_DENIED', 'CRYPTO_RA_↩ ENCRYPTED', 'CRYPTO_RA_INTERNAL_COPY']

4.40 Parameter CryptoKeyElementSize

Maximum size of the Crypto Key Element value, in bytes. Will be used by Crypto driver to reserve internal memory for those Crypto Key Elements that do not use a HSE key.

Property	Value
type	ECUC-INTEGER-PARAM-DEF
origin	AUTOSAR_ECUC
symbolicNameValue	false
lowerMultiplicity	1
upperMultiplicity	1
postBuildVariantMultiplicity	N/A
multiplicityConfigClasses	N/A
postBuildVariantValue	false
valueConfigClasses	VARIANT-PRE-COMPILE: PRE-COMPILE
defaultValue	16
max	4294967295
min	1

4.41 Parameter CryptoKeyElementWriteAccess

Defines the writing access rights of the key element

Property	Value
type	ECUC-ENUMERATION-PARAM-DEF
origin	AUTOSAR_ECUC
symbolicNameValue	false
lowerMultiplicity	1
upperMultiplicity	1
postBuildVariantMultiplicity	N/A
multiplicityConfigClasses	N/A
postBuildVariantValue	false
valueConfigClasses	VARIANT-PRE-COMPILE: PRE-COMPILE
defaultValue	CRYPTO_WA_ALLOWED
literals	['CRYPTO_WA_ALLOWED', 'CRYPTO_WA_DENIED', 'CRYPTO_WA_↩ ENCRYPTED', 'CRYPTO_WA_INTERNAL_COPY']

4.42 Parameter UseHseKey

Vendor specific: Enables or disables the usage of a HSE key.

Property	Value
type	ECUC-BOOLEAN-PARAM-DEF
origin	NXP
symbolicNameValue	false
lowerMultiplicity	1
upperMultiplicity	1
postBuildVariantMultiplicity	N/A
multiplicityConfigClasses	N/A
postBuildVariantValue	false
valueConfigClasses	VARIANT-PRE-COMPILE: PRE-COMPILE
defaultValue	true

4.43 Parameter HseKeySlot

Vendor specific: Slot of the key inside the key group selected above in the 'HSE Key Catalog Group Ref'.

Property	Value
type	ECUC-INTEGER-PARAM-DEF
origin	NXP
symbolicNameValue	False
lowerMultiplicity	1
upperMultiplicity	1

Property	Value
postBuildVariantMultiplicity	N/A
multiplicityConfigClasses	N/A
postBuildVariantValue	false
valueConfigClasses	VARIANT-PRE-COMPILE: PRE-COMPILE
defaultValue	0
max	255
min	0

4.44 Parameter EncryptedKeyImport

Vendor specific: Enables or disables the encrypted import of the HSE key. When enabled, all the parameters related to encrypted key import can be configured in the 'Encrypted Key Import' tab.

Property	Value
type	ECUC-BOOLEAN-PARAM-DEF
origin	NXP
symbolicNameValue	false
lowerMultiplicity	1
upperMultiplicity	1
postBuildVariantMultiplicity	N/A
multiplicityConfigClasses	N/A
postBuildVariantValue	false
valueConfigClasses	VARIANT-PRE-COMPILE: PRE-COMPILE
defaultValue	false

4.45 Parameter AuthenticatedKeyImport

Vendor specific: Enables or disables the authenticated import of the HSE key. When enabled, all the parameters related to authenticated key import can be configured in the 'Authenticated Import' tab.

Property	Value
type	ECUC-BOOLEAN-PARAM-DEF
origin	NXP
symbolicNameValue	false
lowerMultiplicity	1
upperMultiplicity	1
postBuildVariantMultiplicity	N/A
multiplicityConfigClasses	N/A
postBuildVariantValue	false
valueConfigClasses	VARIANT-PRE-COMPILE: PRE-COMPILE
defaultValue	false

4.46 Parameter EncryptedKeyExport

Vendor specific: Enables or disables encrypted export of the HSE key.

When enabled, two types of keys can be exported, symmetric keys and public keys from key pair or public key slots.

Symmetric keys require that export is made encrypted, all the parameters related to this operation can be configured in the 'Encrypted Key Export' tab.

Public keys can be exported in plain and optionally can be authenticated.

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).

Property	Value
type	ECUC-BOOLEAN-PARAM-DEF
origin	NXP
symbolicNameValue	false
lowerMultiplicity	1
upperMultiplicity	1
postBuildVariantMultiplicity	N/A
multiplicityConfigClasses	N/A
postBuildVariantValue	false
valueConfigClasses	VARIANT-PRE-COMPILE: PRE-COMPILE
defaultValue	false

4.47 Parameter AuthenticatedKeyExport

Vendor specific: Enables or disables authenticated export of the HSE key.

When enabled, two types of keys can be exported authenticated, symmetric keys and public keys from key pair or public key slots.

Symmetric keys state that export is optionally authenticated, all the parameters related to this operation can be configured in the 'Authenticated Key Export' tab.

Public keys can be exported in plain and optionally can be authenticated.

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).

Property	Value
type	ECUC-BOOLEAN-PARAM-DEF
origin	NXP
symbolicNameValue	false

Property	Value
lowerMultiplicity	1
upperMultiplicity	1
postBuildVariantMultiplicity	N/A
multiplicityConfigClasses	N/A
postBuildVariantValue	false
valueConfigClasses	VARIANT-PRE-COMPILE: PRE-COMPILE
defaultValue	false

4.48 Parameter HseKeyCounter

Vendor specific: 28 bits counter used to prevent the rollback attacks on key. When updating a key value and attributes the new counter value must be greater than the current counter value. At counter saturation(0xFFFFFFF)the key cannot be updated anymore.

Property	Value
type	ECUC-INTEGER-PARAM-DEF
origin	NXP
symbolicNameValue	false
lowerMultiplicity	1
upperMultiplicity	1
postBuildVariantMultiplicity	N/A
multiplicityConfigClasses	N/A
postBuildVariantValue	false
valueConfigClasses	VARIANT-PRE-COMPILE: PRE-COMPILE
defaultValue	0
max	268435455
min	0

4.49 Parameter HseSMRFlags

Vendor specific: A map of bits 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.

Property	Value
type	ECUC-INTEGER-PARAM-DEF
origin	NXP
symbolicNameValue	false
lowerMultiplicity	1
upperMultiplicity	1

Property	Value
postBuildVariantMultiplicity	N/A
multiplicityConfigClasses	N/A
postBuildVariantValue	false
valueConfigClasses	VARIANT-PRE-COMPILE: PRE-COMPILE
defaultValue	0
max	4294967295
min	0

4.50 Parameter HseEccCurveId

The curve on which the ECC key is set. This parameter is configurable only when the key element is linked to a catalog group of type HSE_KEY_TYPE_ECC_PAIR or HSE_KEY_TYPE_ECC_PUB.

Property	Value
type	ECUC-ENUMERATION-PARAM-DEF
origin	NXP
symbolicNameValue	false
lowerMultiplicity	1
upperMultiplicity	1
postBuildVariantMultiplicity	N/A
multiplicityConfigClasses	N/A
postBuildVariantValue	false
valueConfigClasses	VARIANT-PRE-COMPILE: PRE-COMPILE
defaultValue	ECC_CURVE_NONE
literals	['ECC_CURVE_NONE', 'ECC_SEC_SECP256R1', 'ECC_SEC_SECP384↩ R1', 'ECC_SEC_SECP521R1', 'ECC_BRAINPOOL_BRAINPOOLP256↩ R1', 'ECC_BRAINPOOL_BRAINPOOLP320R1', 'ECC_BRAINPOOL_↩ BRAINPOOLP384R1', 'ECC_BRAINPOOL_BRAINPOOLP512R1', 'ECC_↩ USER_CURVE1', 'ECC_25519_ED25519', 'ECC_25519_CURVE25519']

4.51 Parameter EncryptedImportHseKeySlot

Vendor specific: Slot of the key inside the key group selected above in the 'Encryption HSE Key Catalog Group Ref'.

Property	Value
type	ECUC-INTEGER-PARAM-DEF
origin	NXP
symbolicNameValue	False
lowerMultiplicity	1
upperMultiplicity	1

Property	Value
postBuildVariantMultiplicity	N/A
multiplicityConfigClasses	N/A
postBuildVariantValue	false
valueConfigClasses	VARIANT-PRE-COMPILE: PRE-COMPILE
defaultValue	0
max	255
min	0

4.52 Parameter EncryptedHseKeyBitLen

Vendor specific: The bit length of the encrypted key.

Property	Value
type	ECUC-INTEGER-PARAM-DEF
origin	NXP
symbolicNameValue	false
lowerMultiplicity	1
upperMultiplicity	1
postBuildVariantMultiplicity	N/A
multiplicityConfigClasses	N/A
postBuildVariantValue	false
valueConfigClasses	VARIANT-PRE-COMPILE: PRE-COMPILE
defaultValue	128
max	4294967295
min	1

4.53 Parameter AuthenticatedImportHseKeySlot

Vendor specific: Slot of the key inside the key group selected above in the ’Authentication HSE Key Catalog Group Ref’.

Property	Value
type	ECUC-INTEGER-PARAM-DEF
origin	NXP
symbolicNameValue	False
lowerMultiplicity	1
upperMultiplicity	1
postBuildVariantMultiplicity	N/A
multiplicityConfigClasses	N/A

Property	Value
postBuildVariantValue	false
valueConfigClasses	VARIANT-PRE-COMPILE: PRE-COMPILE
defaultValue	0
max	255
min	0

4.54 Parameter EncryptedExportHseKeySlot

Vendor specific: Slot of the key inside the key group selected above in the 'Encryption HSE Key Catalog Group Ref'.

Property	Value
type	ECUC-INTEGER-PARAM-DEF
origin	NXP
symbolicNameValue	False
lowerMultiplicity	1
upperMultiplicity	1
postBuildVariantMultiplicity	N/A
multiplicityConfigClasses	N/A
postBuildVariantValue	false
valueConfigClasses	VARIANT-PRE-COMPILE: PRE-COMPILE
defaultValue	0
max	255
min	0

4.55 Parameter AuthenticatedExportPublicKey

Vendor specific: Enables or disables authencticated export of the public HSE key.

Public keys can be exported in plain and optionally can be authenticated.

Property	Value
type	ECUC-BOOLEAN-PARAM-DEF
origin	NXP
symbolicNameValue	false
lowerMultiplicity	1
upperMultiplicity	1
postBuildVariantMultiplicity	N/A
multiplicityConfigClasses	N/A
postBuildVariantValue	false
valueConfigClasses	VARIANT-PRE-COMPILE: PRE-COMPILE
defaultValue	false

4.56 Parameter AuthenticatedExportHseKeySlot

Vendor specific: Slot of the key inside the key group selected above in the 'Authentication HSE Key Catalog Group Ref'.

Property	Value
type	ECUC-INTEGER-PARAM-DEF
origin	NXP
symbolicNameValue	False
lowerMultiplicity	1
upperMultiplicity	1
postBuildVariantMultiplicity	N/A
multiplicityConfigClasses	N/A
postBuildVariantValue	false
valueConfigClasses	VARIANT-PRE-COMPILE: PRE-COMPILE
defaultValue	0
max	255
min	0

4.57 Reference CryptoKeyElementVirtualTargetRef

Refers to a key element which will contain the actual data. If the Reference is configured, the key element will be a virtual key element. Functionality not implemented in the current release.

Property	Value
type	ECUC-REFERENCE-DEF
origin	AUTOSAR_ECUC
lowerMultiplicity	0
upperMultiplicity	1
postBuildVariantMultiplicity	false
multiplicityConfigClasses	VARIANT-PRE-COMPILE: PRE-COMPILE
postBuildVariantValue	false
valueConfigClasses	VARIANT-PRE-COMPILE: PRE-COMPILE
requiresSymbolicNameValue	False
destination	/AUTOSAR/EcucDefs/Crypto/CryptoKeyElements/CryptoKeyElement

4.58 Reference HseKeyCatalogGroupRef

Vendor specific: The 'HSE Key Catalog Group Ref' identifies the key group where the key is located in the NVM or RAM key catalog.

Property	Value
type	ECUC-CHOICE-REFERENCE-DEF
origin	NXP
lowerMultiplicity	1
upperMultiplicity	1
postBuildVariantMultiplicity	N/A
multiplicityConfigClasses	N/A
postBuildVariantValue	false
valueConfigClasses	VARIANT-PRE-COMPILE: PRE-COMPILE
requiresSymbolicNameValue	False
destinations	['/TS_T40D34M30I0R0/Crypto/NvmKeyCatalog', '/TS_T40D34M30I0R0/↩ Crypto/RamKeyCatalog']

4.59 Reference EncryptedImportHseKeyCatalogGroupRef

Vendor specific: The 'Encryption HSE Key Catalog Group Ref' identifies the key group where the key is located in the NVM or RAM key catalog.

Property	Value
type	ECUC-CHOICE-REFERENCE-DEF
origin	NXP
lowerMultiplicity	1
upperMultiplicity	1
postBuildVariantMultiplicity	N/A
multiplicityConfigClasses	N/A
postBuildVariantValue	false
valueConfigClasses	VARIANT-PRE-COMPILE: PRE-COMPILE
requiresSymbolicNameValue	False
destinations	['/TS_T40D34M30I0R0/Crypto/NvmKeyCatalog', '/TS_T40D34M30I0R0/↩ Crypto/RamKeyCatalog']

4.60 Reference AuthenticatedImportHseKeyCatalogGroupRef

Vendor specific: The 'Authentication HSE Key Catalog Group Ref' identifies the key group where the key is located in the NVM or RAM key catalog.

Property	Value
type	ECUC-CHOICE-REFERENCE-DEF
origin	NXP
lowerMultiplicity	1

Property	Value
upperMultiplicity	1
postBuildVariantMultiplicity	N/A
multiplicityConfigClasses	N/A
postBuildVariantValue	false
valueConfigClasses	VARIANT-PRE-COMPILE: PRE-COMPILE
requiresSymbolicNameValue	False
destinations	['/TS_T40D34M30I0R0/Crypto/NvmKeyCatalog', '/TS_T40D34M30I0R0/↩ Crypto/RamKeyCatalog']

4.61 Reference EncryptedExportHseKeyCatalogGroupRef

Vendor specific: The 'Encryption HSE Key Catalog Group Ref' identifies the key group where the key is located in the NVM or RAM key catalog.

Property	Value
type	ECUC-CHOICE-REFERENCE-DEF
origin	NXP
lowerMultiplicity	1
upperMultiplicity	1
postBuildVariantMultiplicity	N/A
multiplicityConfigClasses	N/A
postBuildVariantValue	false
valueConfigClasses	VARIANT-PRE-COMPILE: PRE-COMPILE
requiresSymbolicNameValue	False
destinations	['/TS_T40D34M30I0R0/Crypto/NvmKeyCatalog', '/TS_T40D34M30I0R0/↩ Crypto/RamKeyCatalog']

4.62 Reference AuthenticatedExportHseKeyCatalogGroupRef

Vendor specific: The 'Authentication HSE Key Catalog Group Ref' identifies the key group where the key is located in the NVM or RAM key catalog.

Property	Value
type	ECUC-CHOICE-REFERENCE-DEF
origin	NXP
lowerMultiplicity	1
upperMultiplicity	1
postBuildVariantMultiplicity	N/A
multiplicityConfigClasses	N/A

Property	Value
postBuildVariantValue	false
valueConfigClasses	VARIANT-PRE-COMPILE: PRE-COMPILE
requiresSymbolicNameValue	False
destinations	['/TS_T40D34M30I0R0/Crypto/NvmKeyCatalog', '/TS_T40D34M30I0R0/↩ Crypto/RamKeyCatalog']

4.63 Container HseKeyFlags

Vendor specific: Configuration of HSE key flags

Included subcontainers:

- None

Property	Value
type	ECUC-PARAM-CONF-CONTAINER-DEF
lowerMultiplicity	0
upperMultiplicity	Infinite
postBuildVariantMultiplicity	false
multiplicityConfigClasses	VARIANT-PRE-COMPILE: PRE-COMPILE

4.64 Parameter HseKeyFlag

Vendor specific: The key flag specifies the operations or restrictions that can be applied to a key.

Property	Value
type	ECUC-ENUMERATION-PARAM-DEF
origin	NXP
symbolicNameValue	False
lowerMultiplicity	1
upperMultiplicity	1
postBuildVariantMultiplicity	N/A
multiplicityConfigClasses	N/A
postBuildVariantValue	false
valueConfigClasses	VARIANT-PRE-COMPILE: PRE-COMPILE
defaultValue	USAGE_ENCRYPT

Property	Value
literals	['USAGE_ENCRYPT', 'USAGE_DECRYPT', 'USAGE_SIGN', 'USAGE↩_VERIFY', 'USAGE_EXCHANGE', 'USAGE_DERIVE', 'USAGE_KEY↩_PROVISION', 'USAGE_AUTHORIZATION', 'ACCESS_WRITE_PROT', 'ACCESS_DEBUG_PROT', 'ACCESS_EXPORTABLE', 'USAGE_XTS_↩TWEAK', 'USAGE_OTFAD_DECRYPT']

4.65 Container CipherScheme

Vendor specific: Symmetric or asymmetric cipher scheme.

Included choices:

- SymmetricCipher
- AeadCipherScheme
- RSACipherScheme

Property	Value
type	ECUC-CHOICE-CONTAINER-DEF
lowerMultiplicity	1
upperMultiplicity	1
postBuildVariantMultiplicity	N/A
multiplicityConfigClasses	N/A

4.66 Container AuthenticationScheme

Vendor specific: Authentication Scheme.

Included choices:

- MacScheme
- SignatureScheme

Property	Value
type	ECUC-CHOICE-CONTAINER-DEF
lowerMultiplicity	1
upperMultiplicity	1
postBuildVariantMultiplicity	N/A
multiplicityConfigClasses	N/A

4.67 Container KeyContainer

Vendor specific: The Key Container parameters should be used if the key comes in a signed key container.

Included subcontainers:

- None

Property	Value
type	ECUC-PARAM-CONF-CONTAINER-DEF
lowerMultiplicity	1
upperMultiplicity	1
postBuildVariantMultiplicity	N/A
multiplicityConfigClasses	N/A

4.68 Parameter KeyContainerLength

Vendor specific: The container length in bytes.

Property	Value
type	ECUC-INTEGER-PARAM-DEF
origin	NXP
symbolicNameValue	False
lowerMultiplicity	1
upperMultiplicity	1
postBuildVariantMultiplicity	N/A
multiplicityConfigClasses	N/A
postBuildVariantValue	false
valueConfigClasses	VARIANT-PRE-COMPILE: PRE-COMPILE
defaultValue	0
max	65535
min	0

4.69 Parameter pKeyContainer

Vendor specific: Address of the key container, includes the key value(s) and other information used to authenticate the key.

Property	Value
type	ECUC-INTEGER-PARAM-DEF

Property	Value
origin	NXP
symbolicNameValue	False
lowerMultiplicity	1
upperMultiplicity	1
postBuildVariantMultiplicity	N/A
multiplicityConfigClasses	N/A
postBuildVariantValue	false
valueConfigClasses	VARIANT-PRE-COMPILE: PRE-COMPILE
defaultValue	0
max	4294967295
min	0

4.70 Parameter AuthenticationTag0

Authentication tag containing MAC, RSA signature, first part of the signature (r) for ECDSA or EDDSA with the second part (s) of the signature placed in AuthenticationTag1. This node is a hexadecimal string. Please use an even number of characters in range [0-9 a-f A-F], without spaces.

Property	Value
type	ECUC-STRING-PARAM-DEF
origin	NXP
symbolicNameValue	false
lowerMultiplicity	1
upperMultiplicity	1
postBuildVariantMultiplicity	N/A
multiplicityConfigClasses	N/A
postBuildVariantValue	false
valueConfigClasses	VARIANT-PRE-COMPILE: PRE-COMPILE
defaultValue	

4.71 Parameter AuthenticationTag1

Authentication tag containing the second part of the signature (s) for ECDSA or EDDSA. This node is a hexadecimal string. Please use an even number of characters in range [0-9 a-f A-F], without spaces.

Property	Value
type	ECUC-STRING-PARAM-DEF
origin	NXP
symbolicNameValue	false

Property	Value
lowerMultiplicity	1
upperMultiplicity	1
postBuildVariantMultiplicity	N/A
multiplicityConfigClasses	N/A
postBuildVariantValue	false
valueConfigClasses	VARIANT-PRE-COMPILE: PRE-COMPILE
defaultValue	

4.72 Container ExportCipherScheme

Vendor specific: Symmetric or asymmetric cipher scheme.

Included choices:

- ExportSymmetricCipher
- ExportAeadCipherScheme
- ExportRSACipherScheme

Property	Value
type	ECUC-CHOICE-CONTAINER-DEF
lowerMultiplicity	1
upperMultiplicity	1
postBuildVariantMultiplicity	N/A
multiplicityConfigClasses	N/A

4.73 Container ExportAuthenticationScheme

Vendor specific: Authentication Scheme.

Included choices:

- ExportMacScheme
- ExportSignatureScheme

Property	Value
type	ECUC-CHOICE-CONTAINER-DEF
lowerMultiplicity	1

Property	Value
upperMultiplicity	1
postBuildVariantMultiplicity	N/A
multiplicityConfigClasses	N/A

4.74 Container CryptoKeyTypes

Container for CRYPTO key types

Included subcontainers:

- CryptoKeyType

Property	Value
type	ECUC-PARAM-CONF-CONTAINER-DEF
lowerMultiplicity	0
upperMultiplicity	1
postBuildVariantMultiplicity	false
multiplicityConfigClasses	VARIANT-PRE-COMPILE: PRE-COMPILE

4.75 Container CryptoKeyType

Configuration of a CryptoKeyType

Included subcontainers:

- None

Property	Value
type	ECUC-PARAM-CONF-CONTAINER-DEF
lowerMultiplicity	1
upperMultiplicity	Infinite
postBuildVariantMultiplicity	false
multiplicityConfigClasses	VARIANT-PRE-COMPILE: PRE-COMPILE

4.76 Reference CryptoKeyElementRef

Refers to a Crypto Key Element, which holds the data of the Crypto Key Element.

Property	Value
type	ECUC-REFERENCE-DEF
origin	AUTOSAR_ECUC
lowerMultiplicity	1
upperMultiplicity	Infinite
postBuildVariantMultiplicity	false
multiplicityConfigClasses	VARIANT-PRE-COMPILE: PRE-COMPILE
postBuildVariantValue	false
valueConfigClasses	VARIANT-PRE-COMPILE: PRE-COMPILE
requiresSymbolicNameValue	False
destination	/AUTOSAR/EcucDefs/Crypto/CryptoKeyElements/CryptoKeyElement

4.77 Container CryptoKeys

Container for CRYPTO keys

Included subcontainers:

- CryptoKey

Property	Value
type	ECUC-PARAM-CONF-CONTAINER-DEF
lowerMultiplicity	0
upperMultiplicity	1
postBuildVariantMultiplicity	false
multiplicityConfigClasses	VARIANT-PRE-COMPILE: PRE-COMPILE

4.78 Container CryptoKey

Configuration of a CryptoKey

Included subcontainers:

- None

Property	Value
type	ECUC-PARAM-CONF-CONTAINER-DEF
lowerMultiplicity	1
upperMultiplicity	Infinite
postBuildVariantMultiplicity	false
multiplicityConfigClasses	VARIANT-PRE-COMPILE: PRE-COMPILE

4.79 Parameter CryptoKeyId

Identifier of the Crypto Driver key.

Property	Value
type	ECUC-INTEGER-PARAM-DEF
origin	AUTOSAR_ECUC
symbolicNameValue	true
lowerMultiplicity	1
upperMultiplicity	1
postBuildVariantMultiplicity	N/A
multiplicityConfigClasses	N/A
postBuildVariantValue	false
valueConfigClasses	VARIANT-PRE-COMPILE: PRE-COMPILE
defaultValue	0
max	4294967295
min	0

4.80 Reference CryptoKeyTypeRef

Refers to a pointer in the CRYPTO to a CryptoKeyType. The CryptoKeyType provides the information about which key elements are contained in a CryptoKey.

Property	Value
type	ECUC-REFERENCE-DEF
origin	AUTOSAR_ECUC
lowerMultiplicity	1
upperMultiplicity	1
postBuildVariantMultiplicity	N/A
multiplicityConfigClasses	N/A
postBuildVariantValue	false
valueConfigClasses	VARIANT-PRE-COMPILE: PRE-COMPILE
requiresSymbolicNameValue	False
destination	/AUTOSAR/EcucDefs/Crypto/CryptoKeyTypes/CryptoKeyType

4.81 Container CryptoPrimitives

Container for CRYPTO primitives

Included subcontainers:

- CryptoPrimitive

Property	Value
type	ECUC-PARAM-CONF-CONTAINER-DEF
lowerMultiplicity	0
upperMultiplicity	Infinite
postBuildVariantMultiplicity	false
multiplicityConfigClasses	VARIANT-PRE-COMPILE: PRE-COMPILE

4.82 Container CryptoPrimitive

Configuration of a CryptoPrimitive

Included subcontainers:

- None

Property	Value
type	ECUC-PARAM-CONF-CONTAINER-DEF
lowerMultiplicity	0
upperMultiplicity	Infinite
postBuildVariantMultiplicity	false
multiplicityConfigClasses	VARIANT-PRE-COMPILE: PRE-COMPILE

4.83 Parameter CryptoPrimitiveAlgorithmFamily

Determines the algorithm family used for the crypto service

Property	Value
type	ECUC-ENUMERATION-PARAM-DEF
origin	AUTOSAR_ECUC
symbolicNameValue	false
lowerMultiplicity	1

Property	Value
upperMultiplicity	1
postBuildVariantMultiplicity	N/A
multiplicityConfigClasses	N/A
postBuildVariantValue	false
valueConfigClasses	VARIANT-PRE-COMPILE: PRE-COMPILE
defaultValue	CRYPTO_ALGOFAM_RNG
literals	['CRYPTO_ALGOFAM_AES', 'CRYPTO_ALGOFAM_BRAINPOOL', 'CRYPTO_ALGOFAM_ECCNIST', 'CRYPTO_ALGOFAM_NOT_SET', 'CRYPTO_ALGOFAM_RNG', 'CRYPTO_ALGOFAM_RSA', 'CRYPTO↩ _ALGOFAM_SHA1', 'CRYPTO_ALGOFAM_SHA2_224', 'CRYPTO_↩ ALGOFAM_SHA2_256', 'CRYPTO_ALGOFAM_SHA2_384', 'CRYPTO↩ _ALGOFAM_SHA2_512', 'CRYPTO_ALGOFAM_SHA2_512_224', 'CRYPTO_ALGOFAM_SHA2_512_256', 'CRYPTO_ALGOFAM_ED25519']

4.84 Parameter CryptoPrimitiveAlgorithmMode

Determines the algorithm mode used for the crypto service

Property	Value
type	ECUC-ENUMERATION-PARAM-DEF
origin	AUTOSAR_ECUC
symbolicNameValue	false
lowerMultiplicity	1
upperMultiplicity	1
postBuildVariantMultiplicity	N/A
multiplicityConfigClasses	N/A
postBuildVariantValue	false
valueConfigClasses	VARIANT-PRE-COMPILE: PRE-COMPILE
defaultValue	CRYPTO_ALGOMODE_CUSTOM_DRBG_PR
literals	['CRYPTO_ALGOMODE_CBC', 'CRYPTO_ALGOMODE_CFB', 'CRYPTO_ALGOMODE_CMAC', 'CRYPTO_ALGOMODE_CUSTOM_↩ FAST_CMAC', 'CRYPTO_ALGOMODE_CTR', 'CRYPTO_ALGOMODE_↩ CTRDRBG', 'CRYPTO_ALGOMODE_CUSTOM_DRBG_PR', 'CRYPTO↩ _ALGOMODE_CUSTOM_DRBG_RS', 'CRYPTO_ALGOMODE_ECB', 'CRYPTO_ALGOMODE_GCM', 'CRYPTO_ALGOMODE_GMAC', 'CRYPTO_ALGOMODE_HMAC', 'CRYPTO_ALGOMODE_NOT_SET', 'CRYPTO_ALGOMODE_OFB', 'CRYPTO_ALGOMODE_RSAES_↩ OAEP', 'CRYPTO_ALGOMODE_RSAES_PKCS1_v1_5', 'CRYPTO↩ _ALGOMODE_RSASSA_PKCS1_v1_5', 'CRYPTO_ALGOMODE_↩ RSASSA_PSS']

4.85 Parameter CryptoPrimitiveAlgorithmSecondaryFamily

Determines the algorithm secondary family used for the crypto service

Property	Value
type	ECUC-ENUMERATION-PARAM-DEF
origin	AUTOSAR_ECUC
symbolicNameValue	false
lowerMultiplicity	1
upperMultiplicity	1
postBuildVariantMultiplicity	N/A
multiplicityConfigClasses	N/A
postBuildVariantValue	false
valueConfigClasses	VARIANT-PRE-COMPILE: PRE-COMPILE
defaultValue	CRYPTO_ALGOFAM_NOT_SET
literals	['CRYPTO_ALGOFAM_AES', 'CRYPTO_ALGOFAM_BRAINPOOL', 'CRYPTO_ALGOFAM_ECCNIST', 'CRYPTO_ALGOFAM_NOT_SET', 'CRYPTO_ALGOFAM_RNG', 'CRYPTO_ALGOFAM_RSA', 'CRYPTO↩ _ALGOFAM_SHA1', 'CRYPTO_ALGOFAM_SHA2_224', 'CRYPTO_↩ ALGOFAM_SHA2_256', 'CRYPTO_ALGOFAM_SHA2_384', 'CRYPTO↩ _ALGOFAM_SHA2_512', 'CRYPTO_ALGOFAM_SHA2_512_224', 'CRYPTO_ALGOFAM_SHA2_512_256', 'CRYPTO_ALGOFAM_ED25519']

4.86 Parameter CryptoPrimitiveService

Determines the crypto service used for defining the capabilities

Property	Value
type	ECUC-ENUMERATION-PARAM-DEF
origin	AUTOSAR_ECUC
symbolicNameValue	false
lowerMultiplicity	1
upperMultiplicity	1
postBuildVariantMultiplicity	N/A
multiplicityConfigClasses	N/A
postBuildVariantValue	false
valueConfigClasses	VARIANT-PRE-COMPILE: PRE-COMPILE
defaultValue	RANDOM
literals	['HASH', 'MAC_GENERATE', 'MAC_VERIFY', 'ENCRYPT', 'DECRYPT', 'AEAD_ENCRYPT', 'AEAD_DECRYPT', 'SIGNATURE_GENERATE', 'SIGNATURE_VERIFY', 'RANDOM']

4.87 Container CommonPublishedInformation

Common container, aggregated by all modules. It contains published information about vendor and versions.

Included subcontainers:

- None

Property	Value
type	ECUC-PARAM-CONF-CONTAINER-DEF
lowerMultiplicity	1
upperMultiplicity	1
postBuildVariantMultiplicity	N/A
multiplicityConfigClasses	N/A

4.88 Parameter ArReleaseMajorVersion

Vendor specific: Major version number of AUTOSAR specification on which the appropriate implementation is based on.

Property	Value
type	ECUC-INTEGER-PARAM-DEF
origin	NXP
symbolicNameValue	false
lowerMultiplicity	1
upperMultiplicity	1
postBuildVariantMultiplicity	N/A
multiplicityConfigClasses	N/A
postBuildVariantValue	false
valueConfigClasses	VARIANT-PRE-COMPILE: PUBLISHED-INFORMATION
defaultValue	4
max	4
min	4

4.89 Parameter ArReleaseMinorVersion

Vendor specific: Minor version number of AUTOSAR specification on which the appropriate implementation is based on.

Property	Value
type	ECUC-INTEGER-PARAM-DEF
origin	NXP
symbolicNameValue	false
lowerMultiplicity	1
upperMultiplicity	1
postBuildVariantMultiplicity	N/A
multiplicityConfigClasses	N/A
postBuildVariantValue	false
valueConfigClasses	VARIANT-PRE-COMPILE: PUBLISHED-INFORMATION
defaultValue	4
max	4
min	4

4.90 Parameter ArReleaseRevisionVersion

Vendor specific: Patch version number of AUTOSAR specification on which the appropriate implementation is based on.

Property	Value
type	ECUC-INTEGER-PARAM-DEF
origin	NXP
symbolicNameValue	false
lowerMultiplicity	1
upperMultiplicity	1
postBuildVariantMultiplicity	N/A
multiplicityConfigClasses	N/A
postBuildVariantValue	false
valueConfigClasses	VARIANT-PRE-COMPILE: PUBLISHED-INFORMATION
defaultValue	0
max	0
min	0

4.91 Parameter ModuleId

Vendor specific: Module ID of this module.

Property	Value
type	ECUC-INTEGER-PARAM-DEF
origin	NXP

Property	Value
symbolicNameValue	false
lowerMultiplicity	1
upperMultiplicity	1
postBuildVariantMultiplicity	N/A
multiplicityConfigClasses	N/A
postBuildVariantValue	false
valueConfigClasses	VARIANT-PRE-COMPILE: PUBLISHED-INFORMATION
defaultValue	114
max	114
min	114

4.92 Parameter SwMajorVersion

Major version number of the vendor specific implementation of the module. The numbering is vendor specific.

Property	Value
type	ECUC-INTEGER-PARAM-DEF
origin	NXP
symbolicNameValue	false
lowerMultiplicity	1
upperMultiplicity	1
postBuildVariantMultiplicity	N/A
multiplicityConfigClasses	N/A
postBuildVariantValue	false
valueConfigClasses	VARIANT-PRE-COMPILE: PUBLISHED-INFORMATION
defaultValue	3
max	3
min	3

4.93 Parameter SwMinorVersion

Minor version number of the vendor specific implementation of the module. The numbering is vendor specific.

Property	Value
type	ECUC-INTEGER-PARAM-DEF
origin	NXP
symbolicNameValue	false
lowerMultiplicity	1
upperMultiplicity	1

Property	Value
postBuildVariantMultiplicity	N/A
multiplicityConfigClasses	N/A
postBuildVariantValue	false
valueConfigClasses	VARIANT-PRE-COMPILE: PUBLISHED-INFORMATION
defaultValue	0
max	0
min	0

4.94 Parameter SwPatchVersion

Patch level version number of the vendor specific implementation of the module. The numbering is vendor specific.

Property	Value
type	ECUC-INTEGER-PARAM-DEF
origin	NXP
symbolicNameValue	false
lowerMultiplicity	1
upperMultiplicity	1
postBuildVariantMultiplicity	N/A
multiplicityConfigClasses	N/A
postBuildVariantValue	false
valueConfigClasses	VARIANT-PRE-COMPILE: PUBLISHED-INFORMATION
defaultValue	0
max	0
min	0

4.95 Parameter VendorApiInfix

In driver modules which can be instantiated several times on a single ECU, BSW00347 requires that the name of APIs is extended by the VendorId and a vendor specific name.

This parameter is used to specify the vendor specific name. In total, the implementation specific name is generated as follows:

<ModuleName>_>VendorId>_<VendorApiInfix>.

E.g. assuming that the VendorId of the implementor is 123 and the implementer chose a VendorApiInfix of "v11r456" a api name Can_Write defined in the SWS will translate to Can_123_v11r456Write.

This parameter is mandatory for all modules with upper multiplicity > 1. It shall not be used for modules with upper multiplicity =1.

Property	Value
type	ECUC-STRING-PARAM-DEF
origin	NXP
symbolicNameValue	false
lowerMultiplicity	0
upperMultiplicity	1
postBuildVariantMultiplicity	false
multiplicityConfigClasses	VARIANT-PRE-COMPILE: PUBLISHED-INFORMATION
postBuildVariantValue	false
valueConfigClasses	VARIANT-PRE-COMPILE: PUBLISHED-INFORMATION
defaultValue	

4.96 Parameter VendorId

Vendor ID of the dedicated implementation of this module according to the AUTOSAR vendor list.

Property	Value
type	ECUC-INTEGER-PARAM-DEF
origin	NXP
symbolicNameValue	false
lowerMultiplicity	1
upperMultiplicity	1
postBuildVariantMultiplicity	N/A
multiplicityConfigClasses	N/A
postBuildVariantValue	false
valueConfigClasses	VARIANT-PRE-COMPILE: PUBLISHED-INFORMATION
defaultValue	43
max	43
min	43

4.97 Container NvmKeyCatalog

Vendor specific: Configuration of NVM Keys Catalog.

SHE:

&nbsp;&nbsp;- NVM SHE keys shall be mapped on key group 0 in NVM key Catalog.

&nbsp;&nbsp;- In addition to the SHE keys KEY_1 to KEY_10 (key ID 0x4 to 0x0D), the HSE firmware allows extended NVM SHE key groups maped to the key groups 1 to 4 in the NVM key catalogs, and shall contain 10 keys.

&nbsp;&nbsp;- Maximum 5 NVM SHE groups are allowed.

&nbsp;&nbsp;- The owner for SHE key group shall be set to HSE_KEY_OWNER_ANY.

&nbsp;&nbsp;- Any other non-SHE key group can be added after SHE key groups.

Included subcontainers:

- MuMask

Property	Value
type	ECUC-PARAM-CONF-CONTAINER-DEF
lowerMultiplicity	1
upperMultiplicity	256
postBuildVariantMultiplicity	false
multiplicityConfigClasses	VARIANT-PRE-COMPILE: PRE-COMPILE

4.98 Parameter KeyType

Vendor specific: Specifies the key type. It provides information about the interpretation of key data

Property	Value
type	ECUC-ENUMERATION-PARAM-DEF
origin	NXP
symbolicNameValue	False
lowerMultiplicity	1
upperMultiplicity	1
postBuildVariantMultiplicity	N/A
multiplicityConfigClasses	N/A
postBuildVariantValue	false
valueConfigClasses	VARIANT-PRE-COMPILE: PRE-COMPILE
defaultValue	AES
literals	['SHE', 'AES', 'HMAC', 'ECC_PAIR', 'ECC_PUB', 'ECC_PUB_EXT', 'RSA↩ _PAIR', 'RSA_PUB', 'RSA_PUB_EXT']

4.99 Parameter NumOfKeySlots

Vendor specific: The number of key slots in the current key group.

Property	Value
type	ECUC-INTEGER-PARAM-DEF
origin	NXP

Property	Value
symbolicNameValue	False
lowerMultiplicity	1
upperMultiplicity	1
postBuildVariantMultiplicity	N/A
multiplicityConfigClasses	N/A
postBuildVariantValue	false
valueConfigClasses	VARIANT-PRE-COMPILE: PRE-COMPILE
defaultValue	10
max	256
min	1

4.100 Parameter MaxKeyBitLen

Vendor specific: The maximum length of the key (in bits). All stored keys can have keyBitLen lower or equal to MaxKeyBitLen

Property	Value
type	ECUC-INTEGER-PARAM-DEF
origin	NXP
symbolicNameValue	False
lowerMultiplicity	1
upperMultiplicity	1
postBuildVariantMultiplicity	N/A
multiplicityConfigClasses	N/A
postBuildVariantValue	false
valueConfigClasses	VARIANT-PRE-COMPILE: PRE-COMPILE
defaultValue	128
max	65535
min	1

4.101 Parameter KeyOwner

Vendor specific: Specifies the key group owner.

Property	Value
type	ECUC-ENUMERATION-PARAM-DEF
origin	NXP
symbolicNameValue	False
lowerMultiplicity	1

Property	Value
upperMultiplicity	1
postBuildVariantMultiplicity	N/A
multiplicityConfigClasses	N/A
postBuildVariantValue	false
valueConfigClasses	VARIANT-PRE-COMPILE: PRE-COMPILE
defaultValue	OWNER_CUST
literals	['OWNER_ANY', 'OWNER_CUST', 'OWNER_OEM']

4.102 Container MuMask

Vendor specific: Specifies the MU instance(s) where the keys in the key group are allowed to be used on. The keys in a key group can be allowed to be used on one or more MUs.

Included subcontainers:

- None

Property	Value
type	ECUC-PARAM-CONF-CONTAINER-DEF
lowerMultiplicity	1
upperMultiplicity	Infinite
postBuildVariantMultiplicity	false
multiplicityConfigClasses	VARIANT-PRE-COMPILE: PRE-COMPILE

4.103 Parameter MU

Vendor specific: Specifies one MU instance where the keys in the key group are allowed to be used on. The keys in a key group can be allowed to be used on one or more MUs.

Property	Value
type	ECUC-ENUMERATION-PARAM-DEF
origin	NXP
symbolicNameValue	False
lowerMultiplicity	1
upperMultiplicity	1
postBuildVariantMultiplicity	N/A
multiplicityConfigClasses	N/A
postBuildVariantValue	false

Property	Value
valueConfigClasses	VARIANT-PRE-COMPILE: PRE-COMPILE
defaultValue	MU_0
literals	['MU_0', 'MU_1']

4.104 Container RamKeyCatalog

Vendor specific: Configuration of RAM Keys Catalog.

SHE:

&nbsp;&nbsp;- RAM SHE key shall be mapped on key group 0 in RAM key Catalog.

&nbsp;&nbsp;- The owner for SHE key group shall be set to HSE_KEY_OWNER_ANY.

&nbsp;&nbsp;- Any other non-SHE key group can be added after SHE key groups.

Included subcontainers:

- MuMask

Property	Value
type	ECUC-PARAM-CONF-CONTAINER-DEF
lowerMultiplicity	1
upperMultiplicity	256
postBuildVariantMultiplicity	false
multiplicityConfigClasses	VARIANT-PRE-COMPILE: PRE-COMPILE

4.105 Parameter KeyType

Vendor specific: Specifies the key type. It provides information about the interpretation of key data

Property	Value
type	ECUC-ENUMERATION-PARAM-DEF
origin	NXP
symbolicNameValue	False
lowerMultiplicity	1
upperMultiplicity	1
postBuildVariantMultiplicity	N/A
multiplicityConfigClasses	N/A

Property	Value
postBuildVariantValue	false
valueConfigClasses	VARIANT-PRE-COMPILE: PRE-COMPILE
defaultValue	AES
literals	['SHE', 'AES', 'HMAC', 'SHARED_SECRET', 'ECC_PAIR', 'ECC_PUB', 'ECC_PUB_EXT', 'RSA_PUB', 'RSA_PUB_EXT']

4.106 Parameter NumOfKeySlots

Vendor specific: The number of key slots in the current key group.

Property	Value
type	ECUC-INTEGER-PARAM-DEF
origin	NXP
symbolicNameValue	False
lowerMultiplicity	1
upperMultiplicity	1
postBuildVariantMultiplicity	N/A
multiplicityConfigClasses	N/A
postBuildVariantValue	false
valueConfigClasses	VARIANT-PRE-COMPILE: PRE-COMPILE
defaultValue	10
max	256
min	1

4.107 Parameter MaxKeyBitLen

Vendor specific: The maximum length of the key (in bits). All stored keys can have keyBitLen lower or equal to MaxKeyBitLen

Property	Value
type	ECUC-INTEGER-PARAM-DEF
origin	NXP
symbolicNameValue	False
lowerMultiplicity	1
upperMultiplicity	1
postBuildVariantMultiplicity	N/A
multiplicityConfigClasses	N/A
postBuildVariantValue	false
valueConfigClasses	VARIANT-PRE-COMPILE: PRE-COMPILE

Property	Value
defaultValue	128
max	65535
min	1

4.108 Parameter KeyOwner

Vendor specific: Specifies the key group owner.

Property	Value
type	ECUC-ENUMERATION-PARAM-DEF
origin	NXP
symbolicNameValue	False
lowerMultiplicity	1
upperMultiplicity	1
postBuildVariantMultiplicity	N/A
multiplicityConfigClasses	N/A
postBuildVariantValue	false
valueConfigClasses	VARIANT-PRE-COMPILE: PRE-COMPILE
defaultValue	OWNER_ANY
literals	['OWNER_ANY', 'OWNER_CUST', 'OWNER_OEM']

4.109 Container MuMask

Vendor specific: Specifies the MU instance(s) where the keys in the key group are allowed to be used on. The keys in a key group can be allowed to be used on one or more MUs.

Included subcontainers:

- None

Property	Value
type	ECUC-PARAM-CONF-CONTAINER-DEF
lowerMultiplicity	1
upperMultiplicity	Infinite
postBuildVariantMultiplicity	false
multiplicityConfigClasses	VARIANT-PRE-COMPILE: PRE-COMPILE

4.110 Parameter MU

Vendor specific: Specifies one MU instance where the keys in the key group are allowed to be used on. The keys in a key group can be allowed to be used on one or more MUs.

Property	Value
type	ECUC-ENUMERATION-PARAM-DEF
origin	NXP
symbolicNameValue	False
lowerMultiplicity	1
upperMultiplicity	1
postBuildVariantMultiplicity	N/A
multiplicityConfigClasses	N/A
postBuildVariantValue	false
valueConfigClasses	VARIANT-PRE-COMPILE: PRE-COMPILE
defaultValue	MU_0
literals	['MU_0', 'MU_1']

This chapter describes the Tresos configuration plug-in for the CRYPTO Driver. The most of the parameters are described below.

Chapter 5

Module Index

5.1 Software Specification

Here is a list of all modules:

Chapter 6

Module Documentation

6.1 CRYPTO_ASR

6.1.1 Detailed Description

Macros

- #define CRYPTO_E_UNINIT

  API request called before initialization of Crypto Driver.
- #define CRYPTO_E_INIT_FAILED

  Initiation of Crypto Driver failed.
- #define CRYPTO_E_PARAM_POINTER

  API request called with invalid parameter (Nullpointer).
- #define CRYPTO_E_PARAM_HANDLE

  API request called with invalid parameter (out of range).
- #define CRYPTO_E_PARAM_VALUE

  API request called with invalid parameter (invalid value).
- #define CRYPTO_E_PARAM_CONFIG

  The Crypto module is not properly configured (Extension of Development Errors).
- #define CRYPTO_E_NOT_SUPPORTED

  The service request failed because it is not supported by the driver (Extension of Development Errors).
- #define CRYPTO_E_INVALID_PARAM

  The service request failed because at least one parameter is invalid (Extension of Development Errors).
- #define CRYPTO_E_RE_SMALL_BUFFER

  Runtime error codes (passed to DET).
- #define CRYPTO_E_RE_KEY_NOT_AVAILABLE

  Requested key is not available.
- #define CRYPTO_E_RE_KEY_READ_FAIL

  Key cannot be read.
- #define CRYPTO_E_RE_ENTROPY_EXHAUSTED

  Entropy is too low.
- #define CRYPTO_E_RE_OPERATION_TIMEOUT

The service request failed because timeout occurred (Extension of Runtime Errors).
- #define CRYPTO_E_RE_STREAM_BUSY
  The service request failed because there was no stream available for the job (Extension of Runtime Errors).
- #define CRYPTO_E_RE_NVRAM_OPERATION_FAIL
  The service request failed because the application defined function repoted an error (Extension of Runtime Errors).
- #define CRYPTO_INIT_ID
  AUTOSAR API's service IDs.
- #define CRYPTO_GETVERSIONINFO_ID
  API service ID for Crypto_GetVersionInfo function.
- #define CRYPTO_PROCESSJOB_ID
  API service ID for Crypto_ProcessJob function.
- #define CRYPTO_CANCELJOB_ID
  API service ID for Crypto_CancelJob function.
- #define CRYPTO_KEYSETVALID_ID
  API service ID for Crypto_KeySetValid function.
- #define CRYPTO_KEYELEMENTSET_ID
  API service ID for Crypto_KeyElementSet function.
- #define CRYPTO_KEYELEMENTCOPY_ID
  API service ID for Crypto_KeyElementCopy function.
- #define CRYPTO_KEYCOPY_ID
  API service ID for Crypto_KeyCopy function.
- #define CRYPTO_KEYELEMENTCOPYPARTIAL_ID
  API service ID for Crypto_KeyElementCopyPartial function.
- #define CRYPTO_KEYELEMENTIDSGET_ID
  API service ID for Crypto_KeyElementIdsGet function.
- #define CRYPTO_CERTIFICATEPARSE_ID
  API service ID for Crypto_CertificateParse function.
- #define CRYPTO_CERTIFICATEVERIFY_ID
  API service ID for Crypto_CertificateVerify function.
- #define CRYPTO_KEYDERIVE_ID
  API service ID for Crypto_KeyDerive function.
- #define CRYPTO_KEYEXCHANGECALCSECRET_ID
  API service ID for Crypto_KeyExchangeCalcSecret function.
- #define CRYPTO_KEYGENERATE_ID
  API service ID for Crypto_KeyGenerate function.
- #define CRYPTO_RANDOMSEED_ID
  API service ID for Crypto_RandomSeed function.
- #define CRYPTO_KEYELEMENTGET_ID
  API service ID for Crypto_KeyElementGet function.
- #define CRYPTO_KEYEXCHANGECALCPUBVAL_ID
  API service ID for Crypto_KeyExchangeCalcPubVal function.
- #define CYRPTO_KE_KEYEXCHANGE_SHAREDVALUE
  Redefine the fixed key element name to the one used by the driver.

Types Reference

- typedef void Crypto_ConfigType
  Configuration data structure of Crypto module.

Function Reference

- void Crypto_Init (const Crypto_ConfigType ∗configPtr)

  Initializes the Crypto Driver.
- void Crypto_GetVersionInfo (Std_VersionInfoType ∗versioninfo)

  Returns the version information of this module.
- Std_ReturnType Crypto_ProcessJob (uint32 objectId, Crypto_JobType ∗job)

  Performs the crypto primitive that is configured in the job parameter.
- Std_ReturnType Crypto_CancelJob (uint32 objectId, Crypto_JobInfoType ∗job)

  This interface removes the provided job from the queue and cancels the processing of the job if possible.
- Std_ReturnType Crypto_KeyElementSet (uint32 cryptoKeyId, uint32 keyElementId, const uint8 ∗keyPtr, uint32 keyLength)

  Sets the given key element bytes to the key identified by cryptoKeyId.
- Std_ReturnType Crypto_KeySetValid (uint32 cryptoKeyId)

  Sets the key state of the key identified by cryptoKeyId to valid.
- Std_ReturnType Crypto_KeyElementGet (uint32 cryptoKeyId, uint32 keyElementId, uint8 ∗resultPtr, uint32 ∗resultLengthPtr)

  This interface shall be used to get a key element of the key identified by the cryptoKeyId and store the key element in the memory location pointed by the result pointer.
- Std_ReturnType Crypto_KeyElementCopy (uint32 cryptoKeyId, uint32 keyElementId, uint32 targetCrypto↩ KeyId, uint32 targetKeyElementId)

  Copies a key element to another key element in the same Crypto driver.
- Std_ReturnType Crypto_KeyElementCopyPartial (uint32 cryptoKeyId, uint32 keyElementId, uint32 key↩ ElementSourceOffset, uint32 keyElementTargetOffset, uint32 keyElementCopyLength, uint32 targetCrypto↩ KeyId, uint32 targetKeyElementId)

  Copies a key element to another key element in the same Crypto driver.
- Std_ReturnType Crypto_KeyCopy (uint32 cryptoKeyId, uint32 targetCryptoKeyId)

  Copies a key with all its elements to another key in the same crypto driver.
- Std_ReturnType Crypto_KeyElementIdsGet (uint32 cryptoKeyId, uint32 ∗keyElementIdsPtr, uint32 ∗key↩ ElementIdsLengthPtr)

  Used to retrieve information which key elements are available in a given key.
- Std_ReturnType Crypto_RandomSeed (uint32 cryptoKeyId, const uint8 ∗seedPtr, uint32 seedLength)

  This function generates the internal seed state using the provided entropy source. Furthermore, this function can be used to update the seed state with new entropy.
- Std_ReturnType Crypto_KeyGenerate (uint32 cryptoKeyId)

  Generates new key material and stores it in the key identified by cryptoKeyId.
- Std_ReturnType Crypto_KeyDerive (uint32 cryptoKeyId, uint32 targetCryptoKeyId)

  Derives a new key by using the key elements in the given key identified by the cryptoKeyId.
- Std_ReturnType Crypto_KeyExchangeCalcPubVal (uint32 cryptoKeyId, uint8 ∗publicValuePtr, uint32 ∗publicValueLengthPtr)

  Calculates the public value for the key exchange and stores the public key in the memory location pointed by the public value pointer.
- Std_ReturnType Crypto_KeyExchangeCalcSecret (uint32 cryptoKeyId, const uint8 ∗partnerPublicValuePtr, uint32 partnerPublicValueLength)

  Calculates the shared secret key.
- Std_ReturnType Crypto_CertificateParse (uint32 cryptoKeyId)

  Parses the certificate data stored in the key element CRYPTO_KE_CERT_DATA and fills the key elements CRYPTO_KE_CERT_SIGNEDDATA, CRYPTO_KE_CERT_PARSEDPUBLICKEY and CRYPTO_KE_↩ CERT_SIGNATURE.

- Std_ReturnType Crypto_CertificateVerify (uint32 cryptoKeyId, uint32 verifyCryptoKeyId, Crypto_Verify↩ ResultType ∗verifyPtr)

  Verifies the certificate stored in the key referenced by cryptoValidateKeyId with the certificate stored in the key referenced by cryptoKeyId.

6.1.2 Macro Definition Documentation

6.1.2.1 CRYPTO_E_UNINIT

```
#define CRYPTO_E_UNINIT
```

API request called before initialization of Crypto Driver.

Definition at line 147 of file Crypto.h.

6.1.2.2 CRYPTO_E_INIT_FAILED

```
#define CRYPTO_E_INIT_FAILED
```

Initiation of Crypto Driver failed.

Definition at line 152 of file Crypto.h.

6.1.2.3 CRYPTO_E_PARAM_POINTER

```
#define CRYPTO_E_PARAM_POINTER
```

API request called with invalid parameter (Nullpointer).

Definition at line 157 of file Crypto.h.

6.1.2.4 CRYPTO_E_PARAM_HANDLE

```
#define CRYPTO_E_PARAM_HANDLE
```

API request called with invalid parameter (out of range).

Definition at line 162 of file Crypto.h.

6.1.2.5 CRYPTO_E_PARAM_VALUE

```
#define CRYPTO_E_PARAM_VALUE
```

API request called with invalid parameter (invalid value).

Definition at line 167 of file Crypto.h.

6.1.2.6 CRYPTO_E_PARAM_CONFIG

```
#define CRYPTO_E_PARAM_CONFIG
```

The Crypto module is not properly configured (Extension of Development Errors).

Definition at line 173 of file Crypto.h.

6.1.2.7 CRYPTO_E_NOT_SUPPORTED

```
#define CRYPTO_E_NOT_SUPPORTED
```

The service request failed because it is not supported by the driver (Extension of Development Errors).

Definition at line 179 of file Crypto.h.

6.1.2.8 CRYPTO_E_INVALID_PARAM

```
#define CRYPTO_E_INVALID_PARAM
```

The service request failed because at least one parameter is invalid (Extension of Development Errors).

Definition at line 184 of file Crypto.h.

6.1.2.9 CRYPTO_E_RE_SMALL_BUFFER

```
#define CRYPTO_E_RE_SMALL_BUFFER
```

Runtime error codes (passed to DET).

Buffer is too small for operation.

Definition at line 195 of file Crypto.h.

6.1.2.10 CRYPTO_E_RE_KEY_NOT_AVAILABLE

```
#define CRYPTO_E_RE_KEY_NOT_AVAILABLE
```

Requested key is not available.

Definition at line 201 of file Crypto.h.

6.1.2.11 CRYPTO_E_RE_KEY_READ_FAIL

```
#define CRYPTO_E_RE_KEY_READ_FAIL
```

Key cannot be read.

Definition at line 206 of file Crypto.h.

6.1.2.12 CRYPTO_E_RE_ENTROPY_EXHAUSTED

```
#define CRYPTO_E_RE_ENTROPY_EXHAUSTED
```

Entropy is too low.

Definition at line 212 of file Crypto.h.

6.1.2.13 CRYPTO_E_RE_OPERATION_TIMEOUT

```
#define CRYPTO_E_RE_OPERATION_TIMEOUT
```

The service request failed because timeout occurred (Extension of Runtime Errors).

Definition at line 217 of file Crypto.h.

6.1.2.14 CRYPTO_E_RE_STREAM_BUSY

```
#define CRYPTO_E_RE_STREAM_BUSY
```

The service request failed because there was no stream available for the job (Extension of Runtime Errors).

Definition at line 222 of file Crypto.h.

6.1.2.15 CRYPTO_E_RE_NVRAM_OPERATION_FAIL

```
#define CRYPTO_E_RE_NVRAM_OPERATION_FAIL
```

The service request failed because the application defined function repoted an error (Extension of Runtime Errors).

Definition at line 228 of file Crypto.h.

6.1.2.16 CRYPTO_INIT_ID

```
#define CRYPTO_INIT_ID
```

AUTOSAR API's service IDs.

API service ID for Crypto_Init function.

Definition at line 238 of file Crypto.h.

6.1.2.17 CRYPTO_GETVERSIONINFO_ID

```
#define CRYPTO_GETVERSIONINFO_ID
```

API service ID for Crypto_GetVersionInfo function.

Definition at line 245 of file Crypto.h.

6.1.2.18 CRYPTO_PROCESSJOB_ID

```
#define CRYPTO_PROCESSJOB_ID
```

API service ID for Crypto_ProcessJob function.

Definition at line 251 of file Crypto.h.

6.1.2.19 CRYPTO_CANCELJOB_ID

```
#define CRYPTO_CANCELJOB_ID
```

API service ID for Crypto_CancelJob function.

Definition at line 256 of file Crypto.h.

6.1.2.20 CRYPTO_KEYSETVALID_ID

```
#define CRYPTO_KEYSETVALID_ID
```

API service ID for Crypto_KeySetValid function.

Definition at line 262 of file Crypto.h.

6.1.2.21 CRYPTO_KEYELEMENTSET_ID

```
#define CRYPTO_KEYELEMENTSET_ID
```

API service ID for Crypto_KeyElementSet function.

Definition at line 267 of file Crypto.h.

6.1.2.22 CRYPTO_KEYELEMENTCOPY_ID

```
#define CRYPTO_KEYELEMENTCOPY_ID
```

API service ID for Crypto_KeyElementCopy function.

Definition at line 272 of file Crypto.h.

6.1.2.23 CRYPTO_KEYCOPY_ID

```
#define CRYPTO_KEYCOPY_ID
```

API service ID for Crypto_KeyCopy function.

Definition at line 277 of file Crypto.h.

6.1.2.24 CRYPTO_KEYELEMENTCOPYPARTIAL_ID

```
#define CRYPTO_KEYELEMENTCOPYPARTIAL_ID
```

API service ID for Crypto_KeyElementCopyPartial function.

Definition at line 282 of file Crypto.h.

6.1.2.25 CRYPTO_KEYELEMENTIDSGET_ID

```
#define CRYPTO_KEYELEMENTIDSGET_ID
```

API service ID for Crypto_KeyElementIdsGet function.

Definition at line 288 of file Crypto.h.

6.1.2.26 CRYPTO_CERTIFICATEPARSE_ID

```
#define CRYPTO_CERTIFICATEPARSE_ID
```

API service ID for Crypto_CertificateParse function.

Definition at line 293 of file Crypto.h.

6.1.2.27 CRYPTO_CERTIFICATEVERIFY_ID

```
#define CRYPTO_CERTIFICATEVERIFY_ID
```

API service ID for Crypto_CertificateVerify function.

Definition at line 298 of file Crypto.h.

6.1.2.28 CRYPTO_KEYDERIVE_ID

```
#define CRYPTO_KEYDERIVE_ID
```

API service ID for Crypto_KeyDerive function.

Definition at line 304 of file Crypto.h.

6.1.2.29 CRYPTO_KEYEXCHANGECALCSECRET_ID

```
#define CRYPTO_KEYEXCHANGECALCSECRET_ID
```

API service ID for Crypto_KeyExchangeCalcSecret function.

Definition at line 309 of file Crypto.h.

6.1.2.30 CRYPTO_KEYGENERATE_ID

```
#define CRYPTO_KEYGENERATE_ID
```

API service ID for Crypto_KeyGenerate function.

Definition at line 314 of file Crypto.h.

6.1.2.31 CRYPTO_RANDOMSEED_ID

```
#define CRYPTO_RANDOMSEED_ID
```

API service ID for Crypto_RandomSeed function.

Definition at line 319 of file Crypto.h.

6.1.2.32 CRYPTO_KEYELEMENTGET_ID

```
#define CRYPTO_KEYELEMENTGET_ID
```

API service ID for Crypto_KeyElementGet function.

Definition at line 324 of file Crypto.h.

6.1.2.33 CRYPTO_KEYEXCHANGECALCPUBVAL_ID

```
#define CRYPTO_KEYEXCHANGECALCPUBVAL_ID
```

API service ID for Crypto_KeyExchangeCalcPubVal function.

Definition at line 328 of file Crypto.h.

6.1.2.34 CYRPTO_KE_KEYEXCHANGE_SHAREDVALUE

```
#define CYRPTO_KE_KEYEXCHANGE_SHAREDVALUE
```

Redefine the fixed key element name to the one used by the driver.

Definition at line 336 of file Crypto.h.

6.1.3 Types Reference

6.1.3.1 Crypto_ConfigType

```
typedef void Crypto_ConfigType
```

Configuration data structure of Crypto module.

Definition at line 351 of file Crypto.h.

6.1.4 Function Reference

6.1.4.1 Crypto_Init()

```
void Crypto_Init (
            const Crypto_ConfigType * configPtr )
```

Initializes the Crypto Driver.

Initializes the internal variables of the driver, initializes the MU communication layer.

Parameters

in	*configPtr*	Holds the pointer to the configuration data structure of CryIf module

Returns

void

Precondition

The HSE firmware is installed and running.

6.1.4.2 Crypto_GetVersionInfo()

```
void Crypto_GetVersionInfo (
            Std_VersionInfoType * versioninfo )
```

Returns the version information of this module.

Writes the version information attributes of this module in the location pointed by versioninfo parameter.

Parameters

in,out	*versioninfo*	Pointer where to store the version information of this module

Returns

void

Precondition

None.

6.1.4.3 Crypto_ProcessJob()

```
Std_ReturnType Crypto_ProcessJob (
            uint32 objectId,
            Crypto_JobType * job )
```

Performs the crypto primitive that is configured in the job parameter.

Performs the crypto primitive, that is configured in the job parameter.

Parameters

in	*object←Id*	Holds the identifier of the Crypto Driver Object
in,out	*job*	Pointer to the configuration of the job. Contains structures with job and primitive relevant information but also pointer to result buffers.

Returns

Result of the operation

Return values

E_OK	Request successful
E_NOT_OK	Request failed
CRYPTO_E_BUSY	Request failed, Crypro Driver Object is Busy
CRYPTO_E_KEY_NOT_VALID	Request failed, the key is not valid
CRYPTO_E_KEY_SIZE_MISMATCH	Request failed, a key element has the wrong size
CRYPTO_E_QUEUE_FULL	Request failed, the queue is full
CRYPTO_E_ENTROPY_EXHAUSTION	Request failed, the entropy is exhausted
CRYPTO_E_SMALL_BUFFER	The provided buffer is too small to store the result
CRYPTO_E_JOB_CANCELED	The service request failed because the synchronous Job has been canceled

Precondition

Crypto driver is initialized on the current partition.

6.1.4.4 Crypto_CancelJob()

```
Std_ReturnType Crypto_CancelJob (
            uint32 objectId,
            Crypto_JobInfoType * job )
```

This interface removes the provided job from the queue and cancels the processing of the job if possible.

This interface removes the provided job from the queue and cancels the processing of the job if possible.

Parameters

in	*object↩ Id*	Holds the identifier of the Crypto Driver Object.
in,out	*job*	Pointer to the configuration of the job. Contains structures with job and primitive relevant information.

Returns

Result of the operation

Return values

E_OK	Request successful, job has been removed
E_NOT_OK	Request failed, job couldn't be removed

Precondition

Crypto driver is initialized on the current partition.

6.1.4.5 Crypto_KeyElementSet()

```
Std_ReturnType Crypto_KeyElementSet (
            uint32 cryptoKeyId,
            uint32 keyElementId,
            const uint8 * keyPtr,
            uint32 keyLength )
```

Sets the given key element bytes to the key identified by cryptoKeyId.

Sets the given key element bytes to the key identified by cryptoKeyId.

Parameters

in	*cryptoKeyId*	Holds the identifier of the key whose key element shall be set
in	*key↩ ElementId*	Holds the identifier of the key element which shall be set
in	*keyPtr*	Holds the pointer to the key data which shall be set as key element
in	*keyLength*	Contains the length of the key element in bytes

Returns

Result of the operation

Return values

E_OK	Request successful
E_NOT_OK	Request failed
CRYPTO_E_BUSY	Request failed, Crypto Driver Object is busy
CRYPTO_E_KEY_WRITE_FAIL	Request failed because write access was denied
CRYPTO_E_KEY_NOT_AVAILABLE	Request failed because the key is not available
CRYPTO_E_KEY_SIZE_MISMATCH	Request failed, key element size does not match size of provided data

Precondition

Crypto driver is initialized on the current partition.

6.1.4.6 Crypto_KeySetValid()

```
Std_ReturnType Crypto_KeySetValid (
            uint32 cryptoKeyId )
```

Sets the key state of the key identified by cryptoKeyId to valid.

Sets the key state of the key identified by cryptoKeyId to valid.

Parameters

in	*crypto↩ KeyId*	Holds the identifier of the key which shall be set to valid

Returns

Result of the operation

Return values

E_OK	Request successful
E_NOT_OK	Request failed
CRYPTO_E_BUSY	Request failed, Crypto Driver Object is busy

Precondition

Crypto driver is initialized on the current partition.

6.1.4.7 Crypto_KeyElementGet()

```
Std_ReturnType Crypto_KeyElementGet (
            uint32 cryptoKeyId,
            uint32 keyElementId,
            uint8 * resultPtr,
            uint32 * resultLengthPtr )
```

This interface shall be used to get a key element of the key identified by the cryptoKeyId and store the key element in the memory location pointed by the result pointer.

This interface shall be used to get a key element of the key identified by the cryptoKeyId and store the key element in the memory location pointed by the result pointer. Note: If the actual key element is directly mapped to flash memory, there could be a bigger delay when calling this function (synchronous operation).

Parameters

in	*cryptoKeyId*	Holds the identifier of the key whose key element shall be returned
in	*keyElementId*	Holds the identifier of the key element which shall be returned
out	*resultPtr*	Holds the pointer of the buffer for the returned key element
in,out	*resultLengthPtr*	Holds a pointer to a memory location in which the length information is stored. On calling this function this parameter shall contain the size of the buffer provided by resultPtr. If the key element is configured to allow partial access, this parameter contains the amount of data which should be read from the key element. The size may not be equal to the size of the provided buffer anymore. When the request has finished, the amount of data that has been stored shall be stored. If the key identified by the cryptoKeyId is exported authenticated this parameter shall have the size of the exported key because the tag or signature will be generated over this length.

Returns

Result of the operation

Return values

E_OK	Request successful
E_NOT_OK	Request failed
CRYPTO_E_BUSY	Request failed, Crypto Driver Object is busy
CRYPTO_E_KEY_NOT_AVAILABLE	Request failed, the requested key element is not available
CRYPTO_E_KEY_READ_FAIL	Request failed because read access was denied
CRYPTO_E_SMALL_BUFFER	The provided buffer is too small to store the result
CRYPTO_E_KEY_EMPTY	Request failed, source key element is uninitialized

Precondition

Crypto driver is initialized on the current partition.

6.1.4.8 Crypto_KeyElementCopy()

```
Std_ReturnType Crypto_KeyElementCopy (
            uint32 cryptoKeyId,
            uint32 keyElementId,
            uint32 targetCryptoKeyId,
            uint32 targetKeyElementId )
```

Copies a key element to another key element in the same Crypto driver.

Copies a key element to another key element in the same Crypto driver. Note: If the actual key element is directly mapped to flash memory, there could be a bigger delay when calling this function (synchronous operation)

Parameters

in	*cryptoKeyId*	Holds the identifier of the key whose key element shall be the source element
in	*keyElementId*	Holds the identifier of the key element which shall be the source for the copy operation
in	*targetCryptoKeyId*	Holds the identifier of the key whose key element shall be the destination element
in	*targetKey↩ ElementId*	Holds the identifier of the key element which shall be the destination for the copy operation

Returns

Result of the operation

Return values

E_OK	Request successful

Return values

E_NOT_OK	Request failed
CRYPTO_E_BUSY	Request failed, Crypto Driver Object is busy
CRYPTO_E_KEY_NOT_AVAILABLE	Request failed, the requested key element is not available
CRYPTO_E_KEY_READ_FAIL	Request failed, not allowed to extract key element
CRYPTO_E_KEY_WRITE_FAIL	Request failed, not allowed to write key element
CRYPTO_E_KEY_SIZE_MISMATCH	Request failed, key element sizes are not compatible
CRYPTO_E_KEY_EMPTY	Request failed, source key element is uninitialized

Precondition

Crypto driver is initialized on the current partition.

6.1.4.9 Crypto_KeyElementCopyPartial()

```
Std_ReturnType Crypto_KeyElementCopyPartial (
            uint32 cryptoKeyId,
            uint32 keyElementId,
            uint32 keyElementSourceOffset,
            uint32 keyElementTargetOffset,
            uint32 keyElementCopyLength,
            uint32 targetCryptoKeyId,
            uint32 targetKeyElementId )
```

Copies a key element to another key element in the same Crypto driver.

Copies a key element to another key element in the same crypto driver. The keyElementSourceOffset and key↩ ElementCopyLength allows to copy just a part of the source key element into the destination. The offset of the target key is also specified with this function.

Parameters

in	*cryptoKeyId*	Holds the identifier of the key whose key element shall be the source element
in	*keyElementId*	Holds the identifier of the key element which shall be the source for the copy operation
in	*keyElementSourceOffset*	This is the offset of the of the source key element indicating the start index of the copy operation.
in	*keyElementTargetOffset*	This is the offset of the of the target key element indicating the start index of the copy operation.
in	*keyElementCopyLength*	Specifies the number of bytes that shall be copied
in	*targetCryptoKeyId*	Holds the identifier of the key whose key element shall be the destination element.
in	*targetKeyElementId*	Holds the identifier of the key element which shall be the destination for the copy operation.

Returns

Result of the operation

Return values

E_OK	Request successful
E_NOT_OK	Request failed
CRYPTO_E_BUSY	Request failed, Crypto Driver Object is busy
CRYPTO_E_KEY_NOT_AVAILABLE	Request failed, the requested key element is not available
CRYPTO_E_KEY_READ_FAIL	Request failed, not allowed to extract key element
CRYPTO_E_KEY_WRITE_FAIL	Request failed, not allowed to write key element
CRYPTO_E_KEY_SIZE_MISMATCH	Request failed, key element sizes are not compatible
CRYPTO_E_KEY_EMPTY	Request failed, source key element is uninitialized

Precondition

Crypto driver is initialized on the current partition.

6.1.4.10 Crypto_KeyCopy()

```
Std_ReturnType Crypto_KeyCopy (
            uint32 cryptoKeyId,
            uint32 targetCryptoKeyId )
```

Copies a key with all its elements to another key in the same crypto driver.

Copies a key with all its elements to another key in the same crypto driver. Note: If the actual key element is directly mapped to flash memory, there could be a bigger delay when calling this function (synchronous operation)

Parameters

in	*cryptoKeyId*	Holds the identifier of the key whose key element shall be the source element
in	*targetCrypto↩ KeyId*	Holds the identifier of the key whose key element shall be the destination element

Returns

Result of the operation

Return values

E_OK	Request successful
E_NOT_OK	Request failed

Return values

CRYPTO_E_BUSY	Request failed, Crypto Driver Object is busy
CRYPTO_E_KEY_NOT_AVAILABLE	Request failed, the requested key element is not available
CRYPTO_E_KEY_READ_FAIL	Request failed, not allowed to extract key element
CRYPTO_E_KEY_WRITE_FAIL	Request failed, not allowed to write key element
CRYPTO_E_KEY_SIZE_MISMATCH	Request failed, key element sizes are not compatible
CRYPTO_E_KEY_EMPTY	Request failed, source key element is uninitialized

Precondition

Crypto driver is initialized on the current partition.

6.1.4.11 Crypto_KeyElementIdsGet()

```
Std_ReturnType Crypto_KeyElementIdsGet (
            uint32 cryptoKeyId,
            uint32 * keyElementIdsPtr,
            uint32 * keyElementIdsLengthPtr )
```

Used to retrieve information which key elements are available in a given key.

Used to retrieve information which key elements are available in a given key.

Parameters

`in`	*cryptoKeyId*	Holds the identifier of the key whose available element ids shall be exported
`out`	*keyElementIdsPtr*	Contains the pointer to the array where the ids of the key elements shall be stored
`in,out`	*keyElementIdsLengthPtr*	Holds a pointer to the memory location in which the number of key elements in the given key is stored. On calling this function, this parameter shall contain the size of the buffer provided by keyElementIdsPtr. When the request has finished, the actual number of key elements shall be stored.

Returns

Result of the operation

Return values

E_OK	Request successful
E_NOT_OK	Request failed

Return values

CRYPTO_E_BUSY	Request failed, Crypto Driver Object is busy
CRYPTO_E_SMALL_BUFFER	The provided buffer is too small to store the result

Precondition

Crypto driver is initialized on the current partition.

6.1.4.12 Crypto_RandomSeed()

```
Std_ReturnType Crypto_RandomSeed (
            uint32 cryptoKeyId,
            const uint8 * seedPtr,
            uint32 seedLength )
```

This function generates the internal seed state using the provided entropy source. Furthermore, this function can be used to update the seed state with new entropy.

This function generates the internal seed state using the provided entropy source. Furthermore, this function can be used to update the seed state with new entropy.

Parameters

in	*crypto↩ KeyId*	Holds the identifier of the key for which a new seed shall be generated
in	*seedPtr*	Holds a pointer to the memory location which contains the data to feed the entropy
in	*seedLength*	Contains the length of the entropy in bytes

Returns

Result of the operation

Return values

E_OK	Request successful
E_NOT_OK	Request failed

Precondition

Crypto driver is initialized on the current partition.

6.1.4.13 Crypto_KeyGenerate()

```
Std_ReturnType Crypto_KeyGenerate (
            uint32 cryptoKeyId )
```

Generates new key material and stores it in the key identified by cryptoKeyId.

Generates new key material and stores it in the key identified by cryptoKeyId.

Parameters

in	*crypto↩ KeyId*	Holds the identifier of the key which is to be updated with the generated value

Returns

Result of the operation

Return values

E_OK	Request successful
E_NOT_OK	Request failed
CRYPTO_E_BUSY	Request failed, Crypto Driver Object is busy
CRYPTO_E_KEY_EMPTY	Request failed, source key element is uninitialized

Precondition

Crypto driver is initialized on the current partition.

6.1.4.14 Crypto_KeyDerive()

```
Std_ReturnType Crypto_KeyDerive (
            uint32 cryptoKeyId,
            uint32 targetCryptoKeyId )
```

Derives a new key by using the key elements in the given key identified by the cryptoKeyId.

Derives a new key by using the key elements in the given key identified by the cryptoKeyId. The given key contains the key elements for the password, salt. The derived key is stored in the key element with the id 1 of the key identified by targetCryptoKeyId. The number of iterations is given in the key element CRYPTO_KE_KEYDERIVATION↩ _ITERATIONS.

Parameters

in	*cryptoKeyId*	Holds the identifier of the key which is used for key derivation
in	*targetCrypto↩ KeyId*	Holds the identifier of the key which is used to store the derived key

Returns

Result of the operation

Return values

E_OK	Request successful
E_NOT_OK	Request failed
CRYPTO_E_BUSY	Request failed, Crypto Driver Object is busy
CRYPTO_E_KEY_EMPTY	Request failed, source key element is uninitialized

Precondition

Crypto driver is initialized on the current partition.

6.1.4.15 Crypto_KeyExchangeCalcPubVal()

```
Std_ReturnType Crypto_KeyExchangeCalcPubVal (
            uint32 cryptoKeyId,
            uint8 * publicValuePtr,
            uint32 * publicValueLengthPtr )
```

Calculates the public value for the key exchange and stores the public key in the memory location pointed by the public value pointer.

Calculates the public value for the key exchange and stores the public key in the memory location pointed by the public value pointer.

Parameters

in	*cryptoKeyId*	Holds the identifier of the key which shall be used for the key exchange protocol
out	*publicValuePtr*	Contains the pointer to the data where the public value shall be stored
in,out	*publicValueLengthPtr*	Holds a pointer to the memory location in which the public value length information is stored. On calling this function, this parameter shall contain the size of the buffer provided by publicValuePtr. When the request has finished, the actual length of the returned value shall be stored.

Returns

Result of the operation

Return values

E_OK	Request successful

Return values

E_NOT_OK	Request failed
CRYPTO_E_BUSY	Request failed, Crypto Driver Object is busy
CRYPTO_E_SMALL_BUFFER	The provided buffer is too small to store the result
CRYPTO_E_KEY_EMPTY	Request failed, source key element is uninitialized

Precondition

Crypto driver is initialized on the current partition.

6.1.4.16 Crypto_KeyExchangeCalcSecret()

```
Std_ReturnType Crypto_KeyExchangeCalcSecret (
            uint32 cryptoKeyId,
            const uint8 * partnerPublicValuePtr,
            uint32 partnerPublicValueLength )
```

Calculates the shared secret key.

Calculates the shared secret key for the key exchange with the key material of the key identified by the cryptoKeyId and the partner public key. The shared secret key is stored as a key element in the same key.

Parameters

in	*cryptoKeyId*	Holds the identifier of the key which shall be used for the key exchange protocol
in	*partnerPublicValuePtr*	Holds the pointer to the memory location which contains the partner's public value
in	*partnerPublicValueLength*	Contains the length of the partner's public value in bytes. On calling this function, this parameter shall contain the size of the buffer provided by publicValuePtr. When the request has finished, the actual length of the returned value shall be stored.

Returns

Result of the operation

Return values

E_OK	Request successful
E_NOT_OK	Request failed
CRYPTO_E_BUSY	Request failed, Crypto Driver Object is busy
CRYPTO_E_SMALL_BUFFER	The provided buffer is too small to store the result
CRYPTO_E_KEY_EMPTY	Request failed, source key element is uninitialized

Precondition

Crypto driver is initialized on the current partition.

6.1.4.17 Crypto_CertificateParse()

```
Std_ReturnType Crypto_CertificateParse (
            uint32 cryptoKeyId )
```

Parses the certificate data stored in the key element CRYPTO_KE_CERT_DATA and fills the key elements CRYPTO_KE_CERT_SIGNEDDATA, CRYPTO_KE_CERT_PARSEDPUBLICKEY and CRYPTO_KE_↩ CERT_SIGNATURE.

Parses the certificate data stored in the key element CRYPTO_KE_CERT_DATA and fills the key elements CRYPTO_KE_CERT_SIGNEDDATA, CRYPTO_KE_CERT_PARSEDPUBLICKEY and CRYPTO_KE_↩ CERT_SIGNATURE.

Parameters

in	*crypto↩ KeyId*	Holds the identifier of the key which shall be parsed

Returns

Result of the operation

Return values

E_OK	Request successful
E_NOT_OK	Request failed
CRYPTO_E_BUSY	Request failed, Crypto Driver Object is busy

Precondition

Crypto driver is initialized on the current partition.

6.1.4.18 Crypto_CertificateVerify()

```
Std_ReturnType Crypto_CertificateVerify (
            uint32 cryptoKeyId,
            uint32 verifyCryptoKeyId,
            Crypto_VerifyResultType * verifyPtr )
```

Verifies the certificate stored in the key referenced by cryptoValidateKeyId with the certificate stored in the key referenced by cryptoKeyId.

Verifies the certificate stored in the key referenced by cryptoValidateKeyId with the certificate stored in the key referenced by cryptoKeyId.

Parameters

in	*cryptoKeyId*	Holds the identifier of the key which shall be used to validate the certificate
in	*verifyCrypto←KeyId*	Holds the identifier of the key contain
out	*verifyPtr*	Holds a pointer to the memory location which will contain the result of the certificate verification

Returns

Result of the operation

Return values

E_OK	Request successful
E_NOT_OK	Request failed
CRYPTO_E_BUSY	Request failed, Crypto Driver Object is busy

Precondition

Crypto driver is initialized on the current partition.

6.2 CRYPTO_ASR_EXTENSIONS

6.2.1 Detailed Description

Macros

- #define CRYPTO_EXTS_FORMATKEYCATALOGS_ID

  API service ID for Crypto_Exts_FormatKeyCatalogs function.
- #define CRYPTO_EXTS_SETSYNCREQUESTSTIMEOUT_ID

  API service ID for Crypto_Exts_SetSynchronousRequestsTimeout function.
- #define CRYPTO_EXTS_SHE_BOOTFAILURE_ID

  API service ID for Crypto_Exts_She_BootFailure function.
- #define CRYPTO_EXTS_SHE_BOOTOK_ID

  API service ID for Crypto_Exts_She_BootOk function.
- #define CRYPTO_EXTS_SHE_GETSTATUS_ID

  API service ID for Crypto_Exts_She_GetStatus function.

- #define CRYPTO_EXTS_SHE_GETID_ID

  API service ID for Crypto_Exts_She_GetId function.
- #define CRYPTO_EXTS_SHE_DEBUGCHAL_ID

  API service ID for Crypto_Exts_She_DebugChal function.
- #define CRYPTO_EXTS_SHE_DEBUGAUTH_ID

  API service ID for Crypto_Exts_She_DebugAuth function.
- #define CRYPTO_EXTS_SHE_MPCOMPRESSION_ID

  API service ID for Crypto_Exts_She_MPCompression function.
- #define CRYPTO_ALGOMODE_CUSTOM_SIPHASH_2_4_128

  Defines for Crypto ASR extension functionality.

Function Reference

- Std_ReturnType Crypto_Exts_FormatKeyCatalogs (void)

  Format Key Catalogs service.
- Std_ReturnType Crypto_Exts_SetSynchronousRequestsTimeout (uint32 u32Timeout)

  Sets the timeout for synchronous job requests.
- Std_ReturnType Crypto_Exts_SHE_BootFailure (void)

  SHE boot failure service.
- Std_ReturnType Crypto_Exts_SHE_BootOk (void)

  SHE boot ok service.
- Std_ReturnType Crypto_Exts_SHE_GetStatus (uint8 ∗pStatus)

  SHE get status service.
- Std_ReturnType Crypto_Exts_SHE_GetId (const uint8 ∗pChallenge, const uint8 ∗pId, const uint8 ∗pSreg, const uint8 ∗pMac)

  SHE get id service.
- Std_ReturnType Crypto_Exts_SHE_DebugChal (const uint8 ∗pChallenge)

  SHE debug challenge service.
- Std_ReturnType Crypto_Exts_SHE_DebugAuth (const uint8 ∗pAuthorization)

  SHE debug authorization service.
- Std_ReturnType Crypto_Exts_MPCompression (const uint8 ∗pInput, uint32 u32InputLen, const uint8 ∗p↩ Result, const uint32 ∗pResultLen)

  Miyaguchi-Preneel Compression.

6.2.2 Macro Definition Documentation

6.2.2.1 CRYPTO_EXTS_FORMATKEYCATALOGS_ID

```
#define CRYPTO_EXTS_FORMATKEYCATALOGS_ID
```

API service ID for Crypto_Exts_FormatKeyCatalogs function.

Definition at line 93 of file Crypto_ASRExtension.h.

6.2.2.2 CRYPTO_EXTS_SETSYNCREQUESTSTIMEOUT_ID

```
#define CRYPTO_EXTS_SETSYNCREQUESTSTIMEOUT_ID
```

API service ID for Crypto_Exts_SetSynchronousRequestsTimeout function.

Definition at line 98 of file Crypto_ASRExtension.h.

6.2.2.3 CRYPTO_EXTS_SHE_BOOTFAILURE_ID

```
#define CRYPTO_EXTS_SHE_BOOTFAILURE_ID
```

API service ID for Crypto_Exts_She_BootFailure function.

Definition at line 103 of file Crypto_ASRExtension.h.

6.2.2.4 CRYPTO_EXTS_SHE_BOOTOK_ID

```
#define CRYPTO_EXTS_SHE_BOOTOK_ID
```

API service ID for Crypto_Exts_She_BootOk function.

Definition at line 108 of file Crypto_ASRExtension.h.

6.2.2.5 CRYPTO_EXTS_SHE_GETSTATUS_ID

```
#define CRYPTO_EXTS_SHE_GETSTATUS_ID
```

API service ID for Crypto_Exts_She_GetStatus function.

Definition at line 113 of file Crypto_ASRExtension.h.

6.2.2.6 CRYPTO_EXTS_SHE_GETID_ID

```
#define CRYPTO_EXTS_SHE_GETID_ID
```

API service ID for Crypto_Exts_She_GetId function.

Definition at line 118 of file Crypto_ASRExtension.h.

6.2.2.7 CRYPTO_EXTS_SHE_DEBUGCHAL_ID

```
#define CRYPTO_EXTS_SHE_DEBUGCHAL_ID
```

API service ID for Crypto_Exts_She_DebugChal function.

Definition at line 123 of file Crypto_ASRExtension.h.

6.2.2.8 CRYPTO_EXTS_SHE_DEBUGAUTH_ID

```
#define CRYPTO_EXTS_SHE_DEBUGAUTH_ID
```

API service ID for Crypto_Exts_She_DebugAuth function.

Definition at line 128 of file Crypto_ASRExtension.h.

6.2.2.9 CRYPTO_EXTS_SHE_MPCOMPRESSION_ID

```
#define CRYPTO_EXTS_SHE_MPCOMPRESSION_ID
```

API service ID for Crypto_Exts_She_MPCompression function.

Definition at line 133 of file Crypto_ASRExtension.h.

6.2.2.10 CRYPTO_ALGOMODE_CUSTOM_SIPHASH_2_4_128

```
#define CRYPTO_ALGOMODE_CUSTOM_SIPHASH_2_4_128
```

Defines for Crypto ASR extension functionality.

Definition at line 178 of file Crypto_ASRExtension.h.

6.2.3 Function Reference

6.2.3.1 Crypto_Exts_FormatKeyCatalogs()

```
Std_ReturnType Crypto_Exts_FormatKeyCatalogs (
            void  )
```

Format Key Catalogs service.

Used to configure the NVM and RAM key catalogs.

Parameters

in	*none*	
in,out	*none*	
out	*none*	

Returns

Std_ReturnType E_OK: The operation was executed successfully. E_NOT_OK: The operation could not be executed successfully.

Precondition

Crypto driver is initialized on the current partition.

6.2.3.2 Crypto_Exts_SetSynchronousRequestsTimeout()

```
Std_ReturnType Crypto_Exts_SetSynchronousRequestsTimeout (
            uint32 u32Timeout )
```

Sets the timeout for synchronous job requests.

Sets the timeout for synchronous job requests

Parameters

in	*u32Timeout*	- Timeout value, based on the configured 'Timeout Counter Type' the value is interpreted as ticks, microseconds or user defined unit.

Returns

void

Precondition

Crypto driver is initialized on the current partition.

6.2.3.3 Crypto_Exts_SHE_BootFailure()

```
Std_ReturnType Crypto_Exts_SHE_BootFailure (
            void  )
```

SHE boot failure service.

Used to impose sanctions during invalid boot.

Parameters

in	*none*	

Returns

Std_ReturnType E_OK: The operation was executed successfully. E_NOT_OK: The operation could not be executed successfully.

Precondition

Crypto driver is initialized on the current partition.

6.2.3.4 Crypto_Exts_SHE_BootOk()

```
Std_ReturnType Crypto_Exts_SHE_BootOk (
             void  )
```

SHE boot ok service.

Used to mark successful boot verification.

Parameters

in	*none*	

Returns

Std_ReturnType E_OK: The operation was executed successfully. E_NOT_OK: The operation could not be executed successfully.

Precondition

Crypto driver is initialized on the current partition.

6.2.3.5 Crypto_Exts_SHE_GetStatus()

```
Std_ReturnType Crypto_Exts_SHE_GetStatus (
             uint8 * pStatus )
```

SHE get status service.

Used to return the contents of the status register.

Parameters

out	*pStatus*	- Pointer to uint8 location where the function will write the SHE status

Returns

Std_ReturnType E_OK: The operation was executed successfully. E_NOT_OK: The operation could not be executed successfully.

Precondition

Crypto driver is initialized on the current partition.

6.2.3.6 Crypto_Exts_SHE_GetId()

```
Std_ReturnType Crypto_Exts_SHE_GetId (
            const uint8 * pChallenge,
            const uint8 * pId,
            const uint8 * pSreg,
            const uint8 * pMac )
```

SHE get id service.

Used return the identity and the value of the status register protected by a MAC over a challenge and the data.

Parameters

in	*pChallenge*	- Pointer to a 128-bit buffer where from the challenge will be taken
out	*pId*	- Pointer to a 128-bit buffer where UID will be stored
out	*pSreg*	- Pointer to a 8-bit buffer where status register will be stored
out	*pMac*	- Pointer to a 128-bit buffer where MAC key will be stored

Returns

Std_ReturnType E_OK: The operation was executed successfully. E_NOT_OK: The operation could not be executed successfully.

Precondition

Crypto driver is initialized on the current partition.

6.2.3.7 Crypto_Exts_SHE_DebugChal()

```
Std_ReturnType Crypto_Exts_SHE_DebugChal (
            const uint8 * pChallenge )
```

SHE debug challenge service.

Used to generate a 128-bit random challenge output value that is used in conjunction with the DEBUG_AUTH command.

Parameters

out	*pChallenge*	- Pointer to uint8 location where the output challenge will be stored

Returns

Std_ReturnType E_OK: The operation was executed successfully. E_NOT_OK: The operation could not be executed successfully.

Precondition

Crypto driver is initialized on the current partition.

6.2.3.8 Crypto_Exts_SHE_DebugAuth()

```
Std_ReturnType Crypto_Exts_SHE_DebugAuth (
            const uint8 * pAuthorization )
```

SHE debug authorization service.

Erases all user keys.

Parameters

in	*pAuthorization*	- Pointer to uint8 location storing authorization value

Returns

Std_ReturnType E_OK: The operation was executed successfully. E_NOT_OK: The operation could not be executed successfully.

Precondition

Crypto driver is initialized on the current partition.

6.2.3.9 Crypto_Exts_MPCompression()

```
Std_ReturnType Crypto_Exts_MPCompression (
            const uint8 * pInput,
            uint32 u32InputLen,
            const uint8 * pResult,
            const uint32 * pResultLen )
```

Miyaguchi-Preneel Compression.

One-way compression function used to derive a 128 bit output from a given message

Parameters

in	*pInputKey*	Message start address
in	*u32InputKeyLen*	Message length (bytes) address
out	*pResult*	Output address
	[in.out]	pResultLen Message length (bytes) for output buffer

Returns

Std_ReturnType E_OK: The operation was executed successfully. E_NOT_OK: The operation could not be executed successfully.

Precondition

Crypto driver is initialized on the current partition.

6.3 HSE_IP

6.3.1 Detailed Description

Data Structures

- struct Hse_Ip_ReqType

  Structure defining how a request sent from the upper layer to Hse_Ip should look like. More...
- struct Hse_Ip_MuStateType

  Structure defining the internal state machine of the Hse_Ip layer for a given MU instance. More...

Macros

- #define HSE_IP_INVALID_MU_CHANNEL_U8

  Macro returned when no MU channel is available.
- #define HSE_IP_SRV_RSP_NO_RESPONSE

  Extension to the list of responses returned by HSE.

Types Reference

- typedef void(∗ Hse_Ip_pfResponseCallbackType) (uint8 u8MuInstance, uint8 u8MuChannel, hseSrv↩ Response_t HseResponse, void ∗pCallbackParam)

  Type defining HseSrv callback type for notifications that will be sent to the caller when a response is received from HSE, as a result of an asynchronous request.
- typedef void(∗ Hse_Ip_pfGenericPurposeCallbackType) (uint8 u8MuInstance, uint32 u32HseNotifEvents)

  Type defining HseSrv callback type for notifications that will be sent to the caller when HSE triggers an interrupt signaling certain intern events.

Enum Reference

- enum Hse_Ip_StatusType

  Enum defining the possible return type values for the HSE IP API.
- enum Hse_Ip_ReqTypeType

  Enum defining the possible types of service requests that can be sent to HSE.

Function Reference

- Hse_Ip_StatusType Hse_Ip_Init (uint8 u8MuInstance, Hse_Ip_MuStateType ∗pHseIpMuState)

  Initializes the HSE Host driver.
- Hse_Ip_StatusType Hse_Ip_Deinit (uint8 u8MuInstance)

  Deinitializes the HSE Host driver.
- uint8 Hse_Ip_GetFreeChannel (uint8 u8MuInstance)

  Retrieves the next free channel to be used by the application.
- void Hse_Ip_ReleaseChannel (uint8 u8MuInstance, uint8 u8MuChannel)

  Releases a channel previously obtained.
- hseSrvResponse_t Hse_Ip_ServiceRequest (uint8 u8MuInstance, uint8 u8MuChannel, Hse_Ip_ReqType ∗p↩ Request, hseSrvDescriptor_t ∗pHseSrvDesc)

  Sends a service request to HSE.
- void Hse_Ip_MainFunction (uint8 u8MuInstance)

  Function that should be called cyclically to process the requests sent using asynchronous poll method .
- hseStatus_t Hse_Ip_GetHseStatus (uint8 u8MuInstance)

  Returns the HSE firmware status.
- void Hse_Ip_RegisterGenericCallback (uint8 u8MuInstance, uint32 u32NotifEventsMask, Hse_Ip_pfGenericPurposeCallba pfCallback)

  Registers a general purpose callback.
- void Hse_Ip_RxIrqHandler (uint8 u8MuInstance)

  Rx interrupt handler.
- void Hse_Ip_GeneralPurposeIrqHandler (uint8 u8MuInstance)

  General Purpose interrupt handler.

6.3.2 Data Structure Documentation

6.3.2.1 struct Hse_Ip_ReqType

Structure defining how a request sent from the upper layer to Hse_Ip should look like.

Definition at line 153 of file Hse_Ip.h.

Data Fields

- Hse_Ip_ReqTypeType eReqType
- Hse_Ip_pfResponseCallbackType pfCallback
- void * pCallbackParam
- uint32 u32Timeout

6.3.2.1.1 Field Documentation

6.3.2.1.1.1 eReqType `Hse_Ip_ReqTypeType eReqType`

Selects the request type (SYNC/ASYNC)

Definition at line 155 of file Hse_Ip.h.

6.3.2.1.1.2 pfCallback `Hse_Ip_pfResponseCallbackType pfCallback`

The callback for asynchronous request

Definition at line 156 of file Hse_Ip.h.

6.3.2.1.1.3 pCallbackParam `void* pCallbackParam`

Parameter used to call the asynchronous callback(can be NULL)

Definition at line 157 of file Hse_Ip.h.

6.3.2.1.1.4 u32Timeout `uint32 u32Timeout`

Timeout for the synchronous requests (in us or ticks depending on selected counter)

Definition at line 158 of file Hse_Ip.h.

6.3.2.2 struct Hse_Ip_MuStateType

Structure defining the internal state machine of the Hse_Ip layer for a given MU instance.

Definition at line 165 of file Hse_Ip.h.

Data Fields

- Hse_Ip_ReqType ∗ apChannelRequest [HSE_NUM_OF_CHANNELS_PER_MU]
- volatile boolean abChannelAllocated [HSE_NUM_OF_CHANNELS_PER_MU]
- Hse_Ip_pfGenericPurposeCallbackType pfGenericPurposeCallback

6.3.2.2.1 Field Documentation

6.3.2.2.1.1 apChannelRequest `Hse_Ip_ReqType* apChannelRequest[HSE_NUM_OF_CHANNELS_PER_MU]`

Reference to channel request

Definition at line 167 of file Hse_Ip.h.

6.3.2.2.1.2 abChannelAllocated `volatile boolean abChannelAllocated[HSE_NUM_OF_CHANNELS_PER_MU]`

Channel allocated flag

Definition at line 168 of file Hse_Ip.h.

6.3.2.2.1.3 pfGenericPurposeCallback `Hse_Ip_pfGenericPurposeCallbackType pfGenericPurposeCallback`

General purpose callback

Definition at line 169 of file Hse_Ip.h.

6.3.3 Macro Definition Documentation

6.3.3.1 HSE_IP_INVALID_MU_CHANNEL_U8

```
#define HSE_IP_INVALID_MU_CHANNEL_U8
```

Macro returned when no MU channel is available.

Definition at line 107 of file Hse_Ip.h.

6.3.3.2 HSE_IP_SRV_RSP_NO_RESPONSE

```
#define HSE_IP_SRV_RSP_NO_RESPONSE
```

Extension to the list of responses returned by HSE.

Definition at line 110 of file Hse_Ip.h.

6.3.4 Types Reference

6.3.4.1 Hse_Ip_pfResponseCallbackType

```
typedef void(* Hse_Ip_pfResponseCallbackType) (uint8 u8MuInstance, uint8 u8MuChannel, hseSrvResponse_↩
t HseResponse, void *pCallbackParam)
```

Type defining HseSrv callback type for notifications that will be sent to the caller when a response is received from HSE, as a result of an asynchronous request.

Definition at line 117 of file Hse_Ip.h.

6.3.4.2 Hse_Ip_pfGenericPurposeCallbackType

```
typedef void(* Hse_Ip_pfGenericPurposeCallbackType) (uint8 u8MuInstance, uint32 u32HseNotifEvents)
```

Type defining HseSrv callback type for notifications that will be sent to the caller when HSE triggers an interrupt signaling certain intern events.

Definition at line 123 of file Hse_Ip.h.

6.3.5 Enum Reference

6.3.5.1 Hse_Ip_StatusType

```
enum Hse_Ip_StatusType
```

Enum defining the possible return type values for the HSE IP API.

Enumerator

HSE_IP_STATUS_SUCCESS	Operation success status
HSE_IP_STATUS_ERROR	Operation error status

Definition at line 128 of file Hse_Ip.h.

6.3.5.2 Hse_Ip_ReqTypeType

```
enum Hse_Ip_ReqTypeType
```

Enum defining the possible types of service requests that can be sent to HSE.

Enumerator

HSE_IP_REQTYPE_SYNC	Synchronous - the service request function does not return until the HSE completes the request, or the timeout expires
HSE_IP_REQTYPE_ASYNC_IRQ	Asynchronous using interrupts - the service request function returns right after sending the request to HSE; an interrupt is triggered when HSE completes the request (application can be notified through the channel callback)
HSE_IP_REQTYPE_ASYNC_POLL	Asynchronous polling - the service request function returns right after sending the request to HSE; application must poll the driver by calling Hse_Ip_MainFunction

Definition at line 138 of file Hse_Ip.h.

6.3.6 Function Reference

6.3.6.1 Hse_Ip_Init()

```
Hse_Ip_StatusType Hse_Ip_Init (
            uint8 u8MuInstance,
            Hse_Ip_MuStateType * pHseIpMuState )
```

Initializes the HSE Host driver.

This function initializes the HSE host driver over an MU instance. It initializes the state structure with default values.

Parameters

in	*u8MuInstance*	MU Instance number
in	*pHseIpMuState*	Pointer to the state structure which will be used for holding the internal state of the driver.

Returns

An error code or HSE_IP_STATUS_SUCCESS

6.3.6.2 Hse_Ip_Deinit()

```
Hse_Ip_StatusType Hse_Ip_Deinit (
            uint8 u8MuInstance )
```

Deinitializes the HSE Host driver.

This function clears the reference to the previous state structure.

Parameters

in	*u8MuInstance*	MU Instance number

Returns

An error code or HSE_IP_STATUS_SUCCESS

6.3.6.3 Hse_Ip_GetFreeChannel()

```
uint8 Hse_Ip_GetFreeChannel (
            uint8 u8MuInstance )
```

Retrieves the next free channel to be used by the application.

This function finds the next available channel and locks it for the use of the current task. If all channels are allocated, the function returns HSE_IP_INVALID_MU_CHANNEL_U8.

Parameters

in	*u8MuInstance*	MU Instance number

Returns

HSE channel number or HSE_IP_INVALID_MU_CHANNEL_U8

6.3.6.4 Hse_Ip_ReleaseChannel()

```
void Hse_Ip_ReleaseChannel (
            uint8 u8MuInstance,
            uint8 u8MuChannel )
```

Releases a channel previously obtained.

This releases the lock on an MU channel, making it available for other tasks.

Parameters

in	*u8MuInstance*	MU Instance number
in	*u8MuChannel*	MU channel to be released

Returns

void

6.3.6.5 Hse_Ip_ServiceRequest()

```
hseSrvResponse_t Hse_Ip_ServiceRequest (
            uint8 u8MuInstance,
            uint8 u8MuChannel,
            Hse_Ip_ReqType * pRequest,
            hseSrvDescriptor_t * pHseSrvDesc )
```

Sends a service request to HSE.

This function sends a service request to HSE on the specified channel. If the request type is synchronous, this function will not return until either the request has been services, or the timeout expires. If the request type is asynchronous, the function returns right after launching the service request to HSE. The application then either needs to poll the result of the request (calling Hse_Ip_MainFunction()) or wait to be notified by the interrupt when the service is done.

Parameters

in	*u8MuInstance*	MU Instance number
in	*u8MuChannel*	MU channel number
in	*pRequest*	Structure that describes the request parameters: type (sync/interrupts/polling), callback, timeout
in	*pHseSrvDesc*	Requested service descriptor

Returns

HSE service response

6.3.6.6 Hse_Ip_MainFunction()

```
void Hse_Ip_MainFunction (
            uint8 u8MuInstance )
```

Function that should be called cyclically to process the requests sent using asynchronous poll method .

After an asynchronous poll request is sent using Hse_Ip_ServiceRequest() service, the layer on top of the Hse_Ip should call periodically the Hse_Ip_MainFunction() in order to retrieve message processing status from HSE and when a response is received, call the callback sent at request time.

Parameters

in	*u8MuInstance*	MU Instance number

Returns

void

6.3.6.7 Hse_Ip_GetHseStatus()

```
hseStatus_t Hse_Ip_GetHseStatus (
            uint8 u8MuInstance )
```

Returns the HSE firmware status.

This function retrieves the global status of the HSE firmware, read from MU_FSR register. As a note, this function can be called by the application on a MU instance even before initializing the Hse_IP layer by calling Hse_Ip_Init() on that particular MU instance.

Parameters

in	*u8MuInstance*	MU Instance number

Returns

void

6.3.6.8 Hse_Ip_RegisterGenericCallback()

```
void Hse_Ip_RegisterGenericCallback (
            uint8 u8MuInstance,
            uint32 u32NotifEventsMask,
            Hse_Ip_pfGenericPurposeCallbackType pfCallback )
```

Registers a general purpose callback.

This function saves the reference to a generic callback to be called whenever an error is reported by HSE. The signature of the callback should be: void callback(uint8 u8MuInstance, uint32 u32HseNotifEvents)

Parameters

in	*u8MuInstance*	MU Instance number
in	*notifEventsMask*	HSE Errors to be enabled (see definition of hseError_t).
in	*callback*	Pointer to the callback function.

Returns

void

6.3.6.9 Hse_Ip_RxIrqHandler()

```
void Hse_Ip_RxIrqHandler (
            uint8 u8MuInstance )
```

Rx interrupt handler.

This function processes the RX related interrupts from MU Ip layer

Parameters

in	*u8MuInstance*	MU Instance number

Returns

void

6.3.6.10 Hse_Ip_GeneralPurposeIrqHandler()

```
void Hse_Ip_GeneralPurposeIrqHandler (
            uint8 u8MuInstance )
```

General Purpose interrupt handler.

This function processes the General Purpose related interrupts from MU Ip layer

Parameters

`in`	*u8MuInstance*	MU Instance number

Returns

void