User Manual

for S32K1 CRYPTO Driver

Document Number: UM2CRYPTOASR4.4 Rev0000R1.0.1 Rev. 1.0

Chapter 1

Revision History

Revision	Date	Author	Description
1.0	24.02.2022	NXP RTD Team	Prepared for release RTD S32K1 Version 1.0.1

Chapter 2

Introduction

This User Manual describes NXP Semiconductor AUTOSAR Crypto driver for S32K1 platforms.

AUTOSAR CRYPTO driver configuration parameters and deviations from the specification are described in CR↩ YPTO 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:

- s32k116_qfn32
- s32k116_lqfp48
- s32k118_lqfp48
- s32k118_lqfp64
- s32k142_lqfp48
- s32k142_lqfp64
- s32k142_lqfp100
- s32k142w_lqfp48
- s32k142w_lqfp64

- s32k144_lqfp48
- s32k144_lqfp64
- s32k144_lqfp100
- s32k144_mapbga100
- s32k144w_lqfp48
- s32k144w_lqfp64
- s32k146_lqfp64
- s32k146_lqfp100
- s32k146_mapbga100
- s32k146_lqfp144
- s32k148_lqfp100
- s32k148_mapbga100
- s32k148_lqfp144
- s32k148_lqfp176

All of the above microcontroller devices are collectively named as S32K1.

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
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)
ECU	Electronic Control Unit
FLS	Flash
MAC	Message Authentication Code
N/A	Not Applicable
NVM	Non-Volatile Memory
RAM	Random Access Memory
RNG	Random number generator
ROM	Read-only Memory
SHE	Secure Hardware Extension

- 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	S32K1xx Series Reference Manual	Rev. 14, 09/2021
2	S32K1xx Data Sheet	Rev. 14, 08/2021
3	Errata S32K116 (0N96V)	Rev. 22/OCT/2021
4	Errata S32K118 (0N97V)	Rev. 22/OCT/2021
5	Errata S32K142 (0N33V)	Rev. 22/OCT/2021
6	Errata S32K144 (0N57U)	Rev. 22/OCT/2021
7	Errata S32K144W (0P64A)	Rev. 22/OCT/2021
8	Errata S32K146 (0N73V)	Rev. 22/OCT/2021
9	Errata S32K148 (0N20V)	Rev. 22/OCT/2021

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 described in the Secure Hardware Extension (SHE) Functional Specification Version 1.1 by accessing the CSEc hardware IP functionalities.

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

- Loading SHE keys in plain and encrypted format
- Export SHE RAM
- AES-ECB encrypt/decrypt one pass, synchronous or asynchronous (polling or interrupt)
- AES-CBC encrypt/decrypt one pass, synchronous or asynchronous (polling or interrupt)
- AES-CMAC generate/verify one pass, synchronous or asynchronous (polling or interrupt)
- Miyaguchi–Preneel compression
- Random number generation

3.3 Hardware Resources

The Crypto driver implements its functionality by communicating with the CSEc hardware IP.

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 driver.

Requirement	Status	Description	Notes
SWS_Crypto_00017	N/S	"START" indicates a new request of a crypto primitive, and it shall cancel all previous requests of the same job.	CSEc IP does not have support for streaming, therefore Crypto Driver on S32K1 supports only SINGLECALL requests.
SWS_Crypto_00020	N/S	If Crypto_ProcessJob() is called while in "Idle" or "Active" state and with the operation mode "↩ START", the previous request shall be cancelled. That . . .	CSEc IP does not have support for streaming, therefore Crypto Driver on S32K1 supports only SINGLECALL requests.
SWS_Crypto_00118	N/S	If Crypto_ProcessJob() is called while the job is in state "Idle" and the "START" flag in the operation mode is not set, the function shall return wit. . .	CSEc IP does not have support for streaming, therefore Crypto Driver on S32K1 supports only SINGLECALL requests.
SWS_Crypto_00023	N/S	If Crypto_ProcessJob() is called while in "Active" state and with the operation mode "FINIS↩ H", the cryptographic calculations shall be finalized. Addi. . .	CSEc IP does not have support for streaming, therefore Crypto Driver on S32K1 supports only SINGLECALL requests.

Requirement	Status	Description	Notes
SWS_Crypto_00121	N/I	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 ...	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_00071	N/S	MemberService* - inputPtr / **redirected input - input↩ Length - secondaryInputPtr / **redirected input - secondary↩ InputLength - tertiaryInputPtr / **re...	CSEc IP does not have support for streaming, therefore Crypto Driver on S32K1 supports only SINGLECALL requests. Also, there is currently no support for redirection.
SWS_Crypto_00134	N/F	If the crypto primitive requires input data, its memory location is referred by the pointer job->jobPrimitive↩ Input.inputPtr.On calling Crypto_ProcessJ...	This requirement is partially implemented and for this reason is marked as not fulfilled in. The first part of the requirement is implemented. The second part which is related to redirection is not implemented.
SWS_Crypto_00135	N/F	If the crypto primitive requires a buffer for the result, its memory location is referred by the pointer job->jobPrimitive↩ Input.outputPtr. On calling ...	This requirement is partially implemented and for this reason is marked as not fulfilled in. The first part of the requirement is implemented. The second part which is related to redirection is not implemented.
SWS_Crypto_00136	N/F	If the buffer job->job↩ PrimitiveInput.outputPtr or job->jobPrimitiveInput.↩ secondaryOutputPtr is too small, or in case of input/output re-direction the ...	This requirement is partially implemented and for this reason is marked as not fulfilled in. The first part of the requirement is implemented. The second part which is related to redirection is not implemented.
SWS_Crypto_00141	N/S	If the random generator service is chosen and the corresponding entropy, the function shall return CRYPTO_E_ENTROP↩ Y_EXHAUSTED. The function Crypto_Pro...	This requirement can not be fulfilled as CSEc is not reporting or signaling entropy exhaustion.

Requirement	Status	Description	Notes
SWS_Crypto_00145	N/S	If the underlying crypto hardware does not allow execution of key management functions at the same time as processing a job, the key management functi...	CSEc does not support execution of key management functions in parallel with processing a job. There is no mechanism implemented in the driver for allowing a key management function to wait until the current job is processed, therefore this requirement is not fulfilled in on S32K1.
SWS_Crypto_00165	N/S	If no errors are detected by Crypto Driver, the service Crypto_KeyGenerate() generates the corresponding key.	There is no support in CSEc for key generation.
SWS_Crypto_00166	N/S	If no errors are detected by Crypto Driver, the service Crypto_KeyDerive() derives a key element with the aid of a salt and a password.	There is no support in CSEc for key derivation.
SWS_Crypto_00167	N/S	If no errors are detected by Crypto Driver, the service Crypto_KeyExchangeCalcPubVal() calculates the public value of the current job for the key exch...	There is no support in CSEc for key exchange.
SWS_Crypto_00109	N/S	The pointer publicValuePtr holds the memory location, where the data of the public value shall be stored. On calling this function, publicValue↩ LengthP...	There is no support in CSEc for key exchange.
SWS_Crypto_00110	N/S	If the buffer publicValuePtr is too small to store the result of the request, CRYPTO_E_S↩ MALL_BUFFER shall be returned and the function shall additiona...	There is no support in CSEc for key exchange.
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 eleme...	There is no support in C↩ SEc IP for signature generation/verification.
SWS_Crypto_00176	N/S	If the key element CRYPTO↩ _KE_CERTIFICATE_CU↩ RRENT_TIME is used during verification and the format of this timestamp does not match with the format of t...	There is no support in C↩ SEc IP for signature generation/verification.

Requirement	Status	Description	Notes
SWS_Crypto_00177	N/S	If no errors are detected by Crypto Driver, the service Crypto_CertificateVerify() uses the key element CRY↩ PTO_KE_CERT_PARS↩ EDPUBLICKEY of the key ref...	There is no support in C↩ SEc IP for signature generation/verification.
SWS_Crypto_00178	N/S	If certificate identified by verifyCryptoKeyId is verified successfully, the key identified by validateCryptoKeyId shall be set to valid.	There is no support in C↩ SEc IP for signature generation/verification.
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↩ _K...	There is no support in CSEc IP for asymmetric cryptographic primitives.
SWS_Crypto_00185	N/S	For CRYPTO_KE_FORMA↩ T_BIN_ RSA_PRIVATEKEY the parameter 'KeyMaterial O↩ CTET STRING' for RSA private keys is defined according to RFC3447 and has the fol...	There is no support in CSEc IP for asymmetric cryptographic primitives.
SWS_Crypto_00186	N/S	The RSA public key in the format CRYPTO_KE_FOR↩ MAT_BIN _RSA_PUBLIC↩ KEY is provided as follows↩ :RSAPublicKey ::= BIT_ST↩ RING {modulus INTEGER, – npublicEx...	There is no support in CSEc IP for asymmetric cryptographic primitives.
SWS_Crypto_00187	N/S	The RSA public key in the format CRYPTO_KE_FO↩ RMAT_BIN _IDENT_RS↩ A_PUBLICKEY is provided as follows:PublicKeyInfo ::= SEQUENCE {KeyAlgorithm↩ Identifier :...	There is no support in CSEc IP for asymmetric cryptographic primitives.
SWS_Crypto_00188	N/S	The algorithm identifier for R↩ SA 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 0...	There is no support in CSEc IP for asymmetric cryptographic primitives.
SWS_Crypto_00189	N/S	Due to a lack of clear and efficient standard definition for E↩ CC keys, key material for ECC is defined as binary information in the format definition ...	CSEc IP does not support ECC keys.

Requirement	Status	Description	Notes
SWS_Crypto_00190	N/S	Public keys for NIST and Brainpool ECC curves are provided with their X and Y coordinates:ECC Public Key = Point X / Point Y.The points are stored in ...	CSEc IP does not support ECC keys.
SWS_Crypto_00191	N/S	Private keys for NIST and Brainpool ECC curves are provided with their X and Y coordinates and an additional scalar↩ :ECC Private Key = Point X / Point ...	CSEc IP does not support ECC keys.
SWS_Crypto_00192	N/S	The public key information for ED25519 contains a point on the curve:ED25519 Public Key = Point X The point is stored in little endian format.Example↩ :...	CSEc IP does not support ECC keys.
SWS_Crypto_00193	N/S	The private key information for ED25519 contains a random constant and the point X on the curve:ED25519 Private Key = Seed K / Point XThe point and th...	CSEc IP does not support ECC keys.
SWS_Crypto_00198	N/S	If during initialization of the Crypto Driver the value of a persistent key could not be loaded, the Crypto Driver shall set the state of the correspo...	CSEc IP does not offer a mechanism for informing if at initialization a key was successfully loaded or not.
SWS_Crypto_00199	N/I	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 ...	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.
SWS_Crypto_00203	N/I	If job->jobRedirectionInfoRef is not a NULLPTR and the configuration bit for the input↩ Redirection, secondaryInput↩ Redirection and/or tertiary↩ InputRedir...	For this release, this feature is not implemented.
SWS_Crypto_00204	N/I	If job->jobRedirectionInfo↩ Ref is not a NULLPTR and the configuration bit for the outputRedirection and/or secondaryoutputRedirection is set within job...	For this release, this feature is not implemented.

Requirement	Status	Description	Notes
SWS_Crypto_00212	N/S	Draft: The Crypto Driver module shall reject configurations with partition mappings which are not supported by the implementation.	S32K1 is a single core platform so there is no support for multicore.
SWS_Crypto_CONSTR_↩ 00001	N/S	Draft: The Crypto Driver module will operate as an independent instance in each of the partitions, means the called API will only target the partition...	S32K1 is a single core platform so there is no support for multicore.

3.5 Driver Limitations

- Infix is not supported and only one CRYPTO driver instance can be configured. Therefore, the parameter CryptoInstanceId is always set to 0.
- CSEc IP does not have support for streaming, therefore Crypto Driver on S32K1 supports only SINGLECALL requests.

3.6 Driver usage and configuration tips

Crypto Driver Objects

A maximum of one Crypto Driver Object can be configured in Tresos allowing access to symmetric cryptographic primitives. Each primitive can be executed with the help of Crypto_ProcessJob() API either in synchronous or asynchronous mode. The user can opt to enable the Crypto Driver Object's software queue to be setting the CryptoQueueSize attribute of the CDO to a non zero value. If multiple Crypto_ProcessJob() in asynchronous mode are received, the asynchronous job process requests that find Csec Ip busy will be put in the software queue and processed later, when Csec Ip becomes free.

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 responses are read when available from the Csec Ip and also to ensure that the jobs waiting in the CDO queue are being dequeued and sent to Csec Ip. In 'Interrupt' mode, there is no need to call the 'Crypto_MainFunction' API periodically. Both reading of responses from Csec Ip 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 represents a key material and cannot be set by using the field CryptoKeyElement↩ InitValue. 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 Csec so UseCsecKey field from Tresos should be enabled. All the other key elements that are different than key material are stored by Crypto Driver and Use Csec key field should be disabled. Csec key elements have one specific attribute that is used to identify a corresponding memory slot:

- CSEc Key Slot will be used in order to set a specific slot from the available slots defined by SHE spec.

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 SHE keys, the following sequence should be followed:

- The containers CryptoKeyElements, CryptoKeyTypes, CryptoKeys should be enabled.
- 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 (CryptoKey↩ ElementId = 1), mac proof (CryptoKeyElementId = 2) and cipher proof (CryptoKeyElementId = 6).

- CryptoKeyElements extended fields:
  - 'Use Csec Key' enables Csec key usage and management.
  - 'Csec Key Slot' selects the key slot inside the Csec Ip.

Processing a primitive

To process a primitive (random number generation, MAC generation or verification, AES encrypt/decrypt), the following sequence should be followed:

- If keys are needed, the containers CryptoKeyElements, CryptoKeyTypes, CryptoKeys should be enabled.
- Suppose AES CBC 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_OCT↩ ET, 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.
- In Tresos, a symmetric Crypto Driver Object should be configured, having a CryptoDriverObjectId set and also having the required primitive configured. For instance, in case AES CBC encryption is wanted, on the Crypto Primitives container the following should be set:
  - CryptoPrimitiveService set as CRYPTO_ENCRYPT
  - CryptoPrimitiveAlgorithmFamily set as CRYPTO_ALGOFAM_AES

  - CryptoPrimitiveAlgorithmMode set as CRYPTO_ALGOMODE_CBC
  - CryptoPrimitiveAlgorithmSecondaryFamily set as CRYPTO_ALGOFAM_NOT_SET

This Crypto primitive ref should be linked to Crypto Driver Object with symmetric primitives.

- 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 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_B↩IN_SHEKEYS, CryptoKeyElementInitValue should be left blank. Supposing the SHE RAM key will be used, the checkbox Use Csec Key should be enabled and the Csec Key Slot should be set to CSEC_IP_RAM_KEY.
- 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 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_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.

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_KeyeSet↩Valid(), Crypto_CertificateParse(), Crypto_CertificateVerify() should be available in Crypto_ProcessJob(), when the job is configured to one of the services in the list above. 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 CSEc service and are synchronous (eg. key management APIs, synchronous jobs, etc) are writting the request in the CSEc registers and after that remain in a loop, waiting for the CSEc to respond. In case that for some reason CSEc 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

'Csec 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 'Csec 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 'Csec 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_OPERA↩ TION_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.

If a synchronous command is requested and CSEc Ip does not provide an answer in the timeout window, a Cancel command is issued by the Csec_Ip driver in order to abort the execution of the command and leave the CSEc Ip ready to receive the next request. The Cancel command is a synchronous one and has its own, not configurable timeout. This timeout has the value of 100 milliseconds in case the 'Timeout Counter Type' attribute in the 'CryptoGeneral' tab of the plugin is configured to 'OSIF_COUNTER_SYSTEM' and a value of 10.000.000 ticks in case the 'Timeout Counter Type' attribute in the 'CryptoGeneral' tab of the plugin is configured to 'OSIF_COUNTER_DUMMY'. If for some reason, the Cancel command is not completed in the timeout window defined above, the CSEc Ip driver remains in a state where it cannot process any further commands and a platform reset would be required in order to get it back on the normal functioning path.

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' field 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_au8↩ NvramBlob1[] arrays, depending on the value of the u32BlobId received parameter, CRYPTO_NVRAM_B↩ LOB_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.

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 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.

ASR Extension services offered by 'Csec_Ip' interface

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

1. Csec_Ip_Init() - Must be called prior to any other service request from Csec_Ip layer. It is responsible with initializing the internal variables of the layer.
2. Csec_Ip_Deinit() - It is responsible with deinitializing the internal variables of the layer.
3. Csec_Ip_EncryptEcb() - Performs a synchronous or asynchronous AES-128 encryption in ECB mode.
4. Csec_Ip_DecryptEcb() - Performs a synchronous or asynchronous AES-128 decryption in ECB mode.
5. Csec_Ip_EncryptCbc() - Performs a synchronous or asynchronous AES-128 encryption in CBC mode.
6. Csec_Ip_DecryptCbc() - Performs a synchronous or asynchronous AES-128 decryption in CBC mode.
7. Csec_Ip_GenerateMac() - Calculates the MAC of a given message using CMAC with AES-128.
8. Csec_Ip_VerifyMac() - Verifies the MAC of a given message using CMAC with AES-128.

9. Csec_Ip_VerifyMacAddrMode() - Verifies the MAC of a given message (located in Flash) using CMAC with AES-128.
10. Csec_Ip_LoadKey() - Updates an internal key per the SHE specification
11. Csec_Ip_LoadPlainKey() - Updates the RAM key memory slot with a 128-bit plaintext.
12. Csec_Ip_ExportRamKey() - Exports the RAM_KEY into a format protected by SECRET_KEY.
13. Csec_Ip_InitRng() - Initializes the seed and derives a key for the PRNG.
14. Csec_Ip_ExtendSeed() - Extends the seed of the PRNG.
15. Csec_Ip_GenerateRnd() - Generates a vector of 128 random bits.
16. Csec_Ip_BootFailure() - Signals a failure detected during later stages of the boot process.
17. Csec_Ip_BootOk() - Marks a successful boot verification during later stages of the boot process.
18. Csec_Ip_BootDefine() - Implements an extension of the SHE standard to define both the user boot size and boot method.
19. Csec_Ip_GetStatus() - Returns the content of the status register.
20. Csec_Ip_GetId() - Returns the identity (UID) and the value of the status register protected by a MAC over a challenge and the data.
21. Csec_Ip_DbgChal() - Obtains a random number which the user shall use along with the MASTER_ECU_↩ KEY and UID to return an authorization request.
22. Csec_Ip_DbgAuth() - Erases all keys (actual and outdated) stored in NVM Memory if the authorization is confirmed by CSEc.
23. Csec_Ip_MpCompress() - Compresses the given messages by accessing the Miyaguchi-Prenell compression feature from the CSEc feature set.
24. Csec_Ip_MainFunction() - Main function of the Csec_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. Csec_Ip_Main↩ Function must not be called after requesting an asynchronous service with interrupt enabled because it may lead to requesting a CSEc command during the execution of another CSEc command which is not allowed by the CSEc.
25. Csec_Ip_CancelCommand() - Cancels a previously launched asynchronous command().
26. Csec_Ip_SetSynchronousCmdTimeout() - Updates the timeout for the synchronous commands

ASR Extension services offered by 'Crypto_ASRExtension' interface

The Crypto driver code encapsulates one layer called 'Crypto_ASRExtension' which allows an upper entity to use 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_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:
  - STATUS_BUSY is set when a command is processed.
  - SECURE_BOOT is set if the secure booting is activated.
  - BOOT_INIT is set if the secure booting has been personalized during the boot sequence.
  - BOOT_FINISHED is set when the secure booting has been finished by calling either CMD_BOOT_F↩ AILURE 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.

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_E_RE_SMALL_BU↩ FFER	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_Init(), Crypto_KeyCopy(), Crypto_KeyElementSet(), Crypto_KeyElementCopy(), Crypto_KeyElementCopyPartial(), Crypto_ProcessJob(), Crypto_KeySetValid()	CRYPTO_E_RE_NVRAM_O↩ PERATION_FAIL	Calling the upper layer services for reading/writting NVRAM information has failed
All Crypto APIs	CRYPTO_E_RE_OPERATIO↩ N_TIMEOUT	Timeout occured while waiting for a response from CSEc IP

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
      - Reference CryptoPrimitiveRef
      - Reference CryptoDriverObjectEcucPartitionRef
  - Container CryptoGeneral
    - Parameter CryptoDevErrorDetect
    - Parameter CryptoVersionInfoApi
    - Parameter CryptoInstanceId
    - Parameter CryptoMainFunctionPeriod
    - Parameter CryptoMulticoreSupport
    - Parameter CsecIpDevErrorDetect
    - Parameter CryptoTimeoutMethod
    - Parameter CsecTimeoutDuration
    - Parameter CryptoJobKeyManagement
    - Parameter CryptoEnableRedirection
    - Parameter CryptoEnableUserModeSupport
    - Parameter CryptoAlternateJobKeyMapping
    - 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 UseCsecKey
        - Parameter CsecKeySlot
        - Reference CryptoKeyElementVirtualTargetRef
  - 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

4.1 Module Crypto

Configuration of the Crypto (CryptoDriver) module

Included containers:

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

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 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

4.7 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
postBuildVariantValue	false
valueConfigClasses	VARIANT-PRE-COMPILE: PRE-COMPILE
requiresSymbolicNameValue	False
destination	/AUTOSAR/EcucDefs/EcuC/EcucPartitionCollection/EcucPartition

4.8 Container CryptoGeneral

Container for common configuration options

Included subcontainers:

- None

Property	Value
type	ECUC-PARAM-CONF-CONTAINER-DEF
lowerMultiplicity	1

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

4.9 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

4.10 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

4.11 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

4.12 Parameter CryptoMainFunctionPeriod

Specifies the period of main function Crypto_MainFunction in seconds.

Property	Value
type	ECUC-FLOAT-PARAM-DEF
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	0.01
max	9.9999999E7
min	0.0

4.13 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

4.14 Parameter CsecIpDevErrorDetect

Vendor specific: Switches the CSEc 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

4.15 Parameter CryptoTimeoutMethod

Vendor specific: Counter type used in timeout detection for CSEc 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
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_COU↩ NTER_CUSTOM']

4.16 Parameter CsecTimeoutDuration

Vendor specific: Timeout duration defines the waiting period for CSEc 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 - Csec Timeout is interpreted as ticks.

OSIF_COUNTER_SYSTEM - Csec Timeout is interpreted as microseconds.

OSIF_COUNTER_CUSTOM - Csec 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

4.17 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

4.18 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

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

4.19 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

4.20 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

4.21 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']

4.22 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_au8NvramBl or Crypto_au8NvramBlob1[] arrays, depending on the value of the u32BlobId received parameter, CRYPTO_NVRAM_BLOB 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

4.23 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
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

4.24 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

4.25 Container CryptoKeyElement

Configuration of a CryptoKeyElement

Included subcontainers:

- None

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

4.26 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

4.27 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_SHEKEYS
literals	['CRYPTO_KE_FORMAT_BIN_SHEKEYS', 'CRYPTO_KE_FORMAT_↩ BIN_OCTET']

4.28 Parameter CryptoKeyElementId

Identifier of the CRYPTO key element.

Property	Value
type	ECUC-INTEGER-PARAM-DEF

Property	Value
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

4.29 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	

4.30 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 CSEc key' is checked, the value in the checkbox 'CryptoKeyElementPersist' is ignored and the key element value will be stored inside CSEc.

If the checkbox 'Use CSEc 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

4.31 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_ENCRYPTED
literals	['CRYPTO_RA_ALLOWED', 'CRYPTO_RA_DENIED', 'CRYPTO_RA_↩ ENCRYPTED', 'CRYPTO_RA_INTERNAL_COPY']

4.32 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 CSEc 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

4.33 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_ENCRYPTED
literals	['CRYPTO_WA_ALLOWED', 'CRYPTO_WA_DENIED', 'CRYPTO_WA↩ _ENCRYPTED', 'CRYPTO_WA_INTERNAL_COPY']

4.34 Parameter UseCsecKey

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

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	true

4.35 Parameter CsecKeySlot

Vendor specific: Slot of the key inside CSEc.

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	CSEC_IP_MASTER_ECU_KEY
literals	['CSEC_IP_MASTER_ECU_KEY', 'CSEC_IP_BOOT_MAC_KEY', 'CSE↩C_IP_BOOT_MAC', 'CSEC_IP_KEY_1', 'CSEC_IP_KEY_2', 'CSEC_I↩P_KEY_3', 'CSEC_IP_KEY_4', 'CSEC_IP_KEY_5', 'CSEC_IP_KEY_↩6', 'CSEC_IP_KEY_7', 'CSEC_IP_KEY_8', 'CSEC_IP_KEY_9', 'CSEC_↩IP_KEY_10', 'CSEC_IP_RAM_KEY', 'CSEC_IP_KEY_11', 'CSEC_IP_↩KEY_12', 'CSEC_IP_KEY_13', 'CSEC_IP_KEY_14', 'CSEC_IP_KEY_15', 'CSEC_IP_KEY_16', 'CSEC_IP_KEY_17']

4.36 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

Property	Value
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

4.37 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

4.38 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

4.39 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

4.40 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

4.41 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

4.42 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

4.43 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

4.44 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

4.45 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

4.46 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_AES
literals	['CRYPTO_ALGOFAM_NOT_SET', 'CRYPTO_ALGOFAM_AES', 'CRY↩PTO_ALGOFAM_RNG', 'CRYPTO_ALGOFAM_CUSTOM']

4.47 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_ECB
literals	['CRYPTO_ALGOMODE_NOT_SET', 'CRYPTO_ALGOMODE_ECB', 'C↩RYPTO_ALGOMODE_CBC', 'CRYPTO_ALGOMODE_CMAC', 'CRYPT↩O_ALGOMODE_CTRDRBG', 'CRYPTO_ALGOMODE_CUSTOM']

4.48 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

Property	Value
multiplicityConfigClasses	N/A
postBuildVariantValue	false
valueConfigClasses	VARIANT-PRE-COMPILE: PRE-COMPILE
defaultValue	CRYPTO_ALGOFAM_NOT_SET
literals	['CRYPTO_ALGOFAM_NOT_SET', 'CRYPTO_ALGOFAM_AES', 'CRY↩ PTO_ALGOFAM_RNG', 'CRYPTO_ALGOFAM_CUSTOM']

4.49 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	ENCRYPT
literals	['ENCRYPT', 'DECRYPT', 'MAC_GENERATE', 'MAC_VERIFY', 'RANDOM']

4.50 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

4.51 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

4.52 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

4.53 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

4.54 Parameter ModuleId

Vendor specific: Module ID of this module.

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	114
max	114
min	114

4.55 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

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

4.56 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
postBuildVariantMultiplicity	N/A
multiplicityConfigClasses	N/A
postBuildVariantValue	false
valueConfigClasses	VARIANT-PRE-COMPILE: PUBLISHED-INFORMATION
defaultValue	0
max	0
min	0

4.57 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

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

4.58 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	

4.59 Parameter VendorId

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

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: PUBLISHED-INFORMATION
defaultValue	43
max	43
min	43

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_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 C↩ RYPTO_KE_CERT_SIGNEDDATA, CRYPTO_KE_CERT_PARSEDPUBLICKEY and CRYPTO_KE_CE↩ RT_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 148 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 153 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 158 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 163 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 168 of file Crypto.h.

6.1.2.6 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 180 of file Crypto.h.

6.1.2.7 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 185 of file Crypto.h.

6.1.2.8 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 196 of file Crypto.h.

6.1.2.9 CRYPTO_E_RE_KEY_NOT_AVAILABLE

```
#define CRYPTO_E_RE_KEY_NOT_AVAILABLE
```

Requested key is not available.

Definition at line 202 of file Crypto.h.

6.1.2.10 CRYPTO_E_RE_KEY_READ_FAIL

```
#define CRYPTO_E_RE_KEY_READ_FAIL
```

Key cannot be read.

Definition at line 207 of file Crypto.h.

6.1.2.11 CRYPTO_E_RE_ENTROPY_EXHAUSTED

```
#define CRYPTO_E_RE_ENTROPY_EXHAUSTED
```

Entropy is too low.

Definition at line 213 of file Crypto.h.

6.1.2.12 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 218 of file Crypto.h.

6.1.2.13 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 223 of file Crypto.h.

6.1.2.14 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 229 of file Crypto.h.

6.1.2.15 CRYPTO_INIT_ID

```
#define CRYPTO_INIT_ID
```

AUTOSAR API's service IDs.

API service ID for Crypto_Init function.

Definition at line 239 of file Crypto.h.

6.1.2.16 CRYPTO_GETVERSIONINFO_ID

```
#define CRYPTO_GETVERSIONINFO_ID
```

API service ID for Crypto_GetVersionInfo function.

Definition at line 246 of file Crypto.h.

6.1.2.17 CRYPTO_PROCESSJOB_ID

```
#define CRYPTO_PROCESSJOB_ID
```

API service ID for Crypto_ProcessJob function.

Definition at line 252 of file Crypto.h.

6.1.2.18 CRYPTO_CANCELJOB_ID

```
#define CRYPTO_CANCELJOB_ID
```

API service ID for Crypto_CancelJob function.

Definition at line 257 of file Crypto.h.

6.1.2.19 CRYPTO_KEYSETVALID_ID

```
#define CRYPTO_KEYSETVALID_ID
```

API service ID for Crypto_KeySetValid function.

Definition at line 263 of file Crypto.h.

6.1.2.20 CRYPTO_KEYELEMENTSET_ID

```
#define CRYPTO_KEYELEMENTSET_ID
```

API service ID for Crypto_KeyElementSet function.

Definition at line 269 of file Crypto.h.

6.1.2.21 CRYPTO_KEYELEMENTCOPY_ID

```
#define CRYPTO_KEYELEMENTCOPY_ID
```

API service ID for Crypto_KeyElementCopy function.

Definition at line 274 of file Crypto.h.

6.1.2.22 CRYPTO_KEYCOPY_ID

```
#define CRYPTO_KEYCOPY_ID
```

API service ID for Crypto_KeyCopy function.

Definition at line 279 of file Crypto.h.

6.1.2.23 CRYPTO_KEYELEMENTCOPYPARTIAL_ID

```
#define CRYPTO_KEYELEMENTCOPYPARTIAL_ID
```

API service ID for Crypto_KeyElementCopyPartial function.

Definition at line 284 of file Crypto.h.

6.1.2.24 CRYPTO_KEYELEMENTIDSGET_ID

```
#define CRYPTO_KEYELEMENTIDSGET_ID
```

API service ID for Crypto_KeyElementIdsGet function.

Definition at line 290 of file Crypto.h.

6.1.2.25 CRYPTO_CERTIFICATEPARSE_ID

```
#define CRYPTO_CERTIFICATEPARSE_ID
```

API service ID for Crypto_CertificateParse function.

Definition at line 295 of file Crypto.h.

6.1.2.26 CRYPTO_CERTIFICATEVERIFY_ID

```
#define CRYPTO_CERTIFICATEVERIFY_ID
```

API service ID for Crypto_CertificateVerify function.

Definition at line 300 of file Crypto.h.

6.1.2.27 CRYPTO_KEYDERIVE_ID

```
#define CRYPTO_KEYDERIVE_ID
```

API service ID for Crypto_KeyDerive function.

Definition at line 306 of file Crypto.h.

6.1.2.28 CRYPTO_KEYEXCHANGECALCSECRET_ID

```
#define CRYPTO_KEYEXCHANGECALCSECRET_ID
```

API service ID for Crypto_KeyExchangeCalcSecret function.

Definition at line 311 of file Crypto.h.

6.1.2.29 CRYPTO_KEYGENERATE_ID

```
#define CRYPTO_KEYGENERATE_ID
```

API service ID for Crypto_KeyGenerate function.

Definition at line 316 of file Crypto.h.

6.1.2.30 CRYPTO_RANDOMSEED_ID

```
#define CRYPTO_RANDOMSEED_ID
```

API service ID for Crypto_RandomSeed function.

Definition at line 321 of file Crypto.h.

6.1.2.31 CRYPTO_KEYELEMENTGET_ID

```
#define CRYPTO_KEYELEMENTGET_ID
```

API service ID for Crypto_KeyElementGet function.

Definition at line 326 of file Crypto.h.

6.1.2.32 CRYPTO_KEYEXCHANGECALCPUBVAL_ID

```
#define CRYPTO_KEYEXCHANGECALCPUBVAL_ID
```

API service ID for Crypto_KeyExchangeCalcPubVal function.

Definition at line 330 of file Crypto.h.

6.1.2.33 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 338 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 353 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

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

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	*objectId*	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

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	*objectId*	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

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	*keyElementId*	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

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

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

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
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

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

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
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

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
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

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	*cryptoKeyId*	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

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

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

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
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

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

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 C↩ RYPTO_KE_CERT_SIGNEDDATA, CRYPTO_KE_CERT_PARSEDPUBLICKEY and CRYPTO_KE_CE↩ RT_SIGNATURE.

Parses the certificate data stored in the key element CRYPTO_KE_CERT_DATA and fills the key elements C↩ RYPTO_KE_CERT_SIGNEDDATA, CRYPTO_KE_CERT_PARSEDPUBLICKEY and CRYPTO_KE_CE↩ RT_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

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

6.2 CRYPTO_ASR_EXTENSIONS

6.2.1 Detailed Description

Macros

- #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_SIPHASH_2_4_32
  Defines for Crypto ASR extension functionality.

Function Reference

- 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, uint8 *pId, uint8 *pSreg, uint8 *pMac)
  SHE get id service.
- Std_ReturnType Crypto_Exts_SHE_DebugChal (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, uint8 *pResult, const uint32 *pResultLen)
  Miyaguchi-Preneel Compression.

6.2.2 Macro Definition Documentation

6.2.2.1 CRYPTO_EXTS_SETSYNCREQUESTSTIMEOUT_ID

```
#define CRYPTO_EXTS_SETSYNCREQUESTSTIMEOUT_ID
```

API service ID for Crypto_Exts_SetSynchronousRequestsTimeout function.

Definition at line 94 of file Crypto_ASRExtension.h.

6.2.2.2 CRYPTO_EXTS_SHE_BOOTFAILURE_ID

```
#define CRYPTO_EXTS_SHE_BOOTFAILURE_ID
```

API service ID for Crypto_Exts_She_BootFailure function.

Definition at line 99 of file Crypto_ASRExtension.h.

6.2.2.3 CRYPTO_EXTS_SHE_BOOTOK_ID

```
#define CRYPTO_EXTS_SHE_BOOTOK_ID
```

API service ID for Crypto_Exts_She_BootOk function.

Definition at line 104 of file Crypto_ASRExtension.h.

6.2.2.4 CRYPTO_EXTS_SHE_GETSTATUS_ID

```
#define CRYPTO_EXTS_SHE_GETSTATUS_ID
```

API service ID for Crypto_Exts_She_GetStatus function.

Definition at line 109 of file Crypto_ASRExtension.h.

6.2.2.5 CRYPTO_EXTS_SHE_GETID_ID

```
#define CRYPTO_EXTS_SHE_GETID_ID
```

API service ID for Crypto_Exts_She_GetId function.

Definition at line 114 of file Crypto_ASRExtension.h.

6.2.2.6 CRYPTO_EXTS_SHE_DEBUGCHAL_ID

```
#define CRYPTO_EXTS_SHE_DEBUGCHAL_ID
```

API service ID for Crypto_Exts_She_DebugChal function.

Definition at line 119 of file Crypto_ASRExtension.h.

6.2.2.7 CRYPTO_EXTS_SHE_DEBUGAUTH_ID

```
#define CRYPTO_EXTS_SHE_DEBUGAUTH_ID
```

API service ID for Crypto_Exts_She_DebugAuth function.

Definition at line 124 of file Crypto_ASRExtension.h.

6.2.2.8 CRYPTO_EXTS_SHE_MPCOMPRESSION_ID

```
#define CRYPTO_EXTS_SHE_MPCOMPRESSION_ID
```

API service ID for Crypto_Exts_She_MPCompression function.

Definition at line 129 of file Crypto_ASRExtension.h.

6.2.2.9 CRYPTO_ALGOMODE_SIPHASH_2_4_32

```
#define CRYPTO_ALGOMODE_SIPHASH_2_4_32
```

Defines for Crypto ASR extension functionality.

Definition at line 169 of file Crypto_ASRExtension.h.

6.2.3 Function Reference

6.2.3.1 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

6.2.3.2 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

6.2.3.3 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

6.2.3.4 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

6.2.3.5 Crypto_Exts_SHE_GetId()

```
Std_ReturnType Crypto_Exts_SHE_GetId (
            const uint8 * pChallenge,
            uint8 * pId,
```

```
            uint8 * pSreg,
            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

6.2.3.6 Crypto_Exts_SHE_DebugChal()

```
Std_ReturnType Crypto_Exts_SHE_DebugChal (
            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

6.2.3.7 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

6.2.3.8 Crypto_Exts_MPCompression()

```
Std_ReturnType Crypto_Exts_MPCompression (
            const uint8 * pInput,
            uint32 u32InputLen,
            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

6.3 CSEC_IP

6.3.1 Detailed Description

Data Structures

- struct Csec_Ip_StateType

  Structure defining driver state. More...
- struct Csec_Ip_ReqType

  Structure defining request parameters. More...
- struct Csec_Ip_PramType

Macros

- #define CSEC_IP_STATUS_BUSY_U8

  The bit is set when CSEC is processing a command.
- #define CSEC_IP_STATUS_SECURE_BOOT_U8

  The bit is set if the secure booting is activated.
- #define CSEC_IP_STATUS_BOOT_INIT_U8

  The bit is set if the secure booting has been personalized during the boot sequence.
- #define CSEC_IP_STATUS_BOOT_FINISHED_U8

  The bit is set when the secure booting has been finished by calling either CMD_BOOT_FAILURE or CMD_BO↩ OT_OK or if CMD_SECURE_BOOT failed in verifying BOOT_MAC.
- #define CSEC_IP_STATUS_BOOT_OK_U8

  The bit is set if the secure booting (CMD_SECURE_BOOT) succeeded. If CMD_BOOT_FAILURE is called the bit is erased.
- #define CSEC_IP_STATUS_RND_INIT_U8

  The bit is set if the random number generator has been initialized.
- #define CSEC_IP_STATUS_EXT_DEBUGGER_U8

  The bit is set if an external debugger is connected to the chip.
- #define CSEC_IP_STATUS_INT_DEBUGGER_U8

  The bit is set if the internal debugging mechanisms are activated.
- #define CSEC_IP_ERC_NO_ERROR

  No error has occurred.
- #define CSEC_IP_ERC_SEQUENCE_ERROR

  The call sequence of the commands is invalid.
- #define CSEC_IP_ERC_KEY_NOT_AVAILABLE

  The used key has DBG Attached flag and debugger is active.
- #define CSEC_IP_ERC_KEY_INVALID

  A function is called to perform an operation with a key that is not allowed for the given operation.
- #define CSEC_IP_ERC_KEY_EMPTY

  Key slot is empty (not initialized)/not present or higher slot (not partitioned).
- #define CSEC_IP_ERC_NO_SECURE_BOOT

  Not applicable, BOOT_DEFINE once configured, will automatically run secure boot.
- #define CSEC_IP_ERC_KEY_WRITE_PROTECTED

A key update is attempted on a write protected key slot or the debugger is started while a key is write-protected.

- #define CSEC_IP_ERC_KEY_UPDATE_ERROR

  A key update did not succeed due to errors in verification of the messages.

- #define CSEC_IP_ERC_RNG_SEED

  The PRNG seed has not yet been initialized.

- #define CSEC_IP_ERC_NO_DEBUGGING

  Internal debugging is not possible because the authentication did not succeed.

- #define CSEC_IP_ERC_MEMORY_FAILURE

  General memory technology failure (multi-bit ECC error, common fault detection).

- #define CSEC_IP_ERC_GENERAL_ERROR

  Detected error that is not covered by the other error codes.

- #define CSEC_IP_ERC_NO_RESPONSE

  No response received from Csec Ip in the timeout window.

- #define CSEC_IP_ERC_STATUS_BUSY

  Another command is in progress.

Types Reference

- typedef uint8 Csec_Ip_StatusType

  Status of the CSEC cryptographic related feature set. Provides one bit for each status code as per SHE specification. CSEC IP status masks can be used for status verification.

- typedef uint16 Csec_Ip_ErrorCodeType

  Unsigned integer defining the CSEC error codes.

- typedef void(∗ pfCsecIpResponseCallbackType) (Csec_Ip_ErrorCodeType ErrCode, Csec_Ip_CmdType e↩ CompletedCmd, void ∗pCallbackParam)

  Callback for asynchronous command.

Enum Reference

- enum Csec_Ip_KeyIdType

  Enum defining the Key IDs and key memory slot idenfitication.

- enum Csec_Ip_CmdType

  Enum defining SHE compliant commands present in the CSEc command set.

- enum Csec_Ip_CallSequenceType

  Enum defining call sequence.

- enum Csec_Ip_BootFlavorType

  Enum defining the boot type for the BOOT_DEFINE command.

- enum Csec_Ip_ReqTypeType

  Enum defining the possible asynchronous types of service requests.

Function Reference

- void Csec_Ip_Init (Csec_Ip_StateType *pState)

  Initializes the internal state of the driver.
- void Csec_Ip_Deinit (void)

  Clears the internal state of the driver.
- Csec_Ip_ErrorCodeType Csec_Ip_EncryptEcb (const Csec_Ip_ReqType *pRequest, Csec_Ip_KeyIdType eKeyId, const uint8 *pPlainText, uint32 u32Length, uint8 *pCipherText)

  Performs the AES-128 encryption in ECB mode.
- Csec_Ip_ErrorCodeType Csec_Ip_DecryptEcb (const Csec_Ip_ReqType *pRequest, Csec_Ip_KeyIdType eKeyId, const uint8 *pCipherText, uint32 u32Length, uint8 *pPlainText)

  Performs the AES-128 decryption in ECB mode.
- Csec_Ip_ErrorCodeType Csec_Ip_EncryptCbc (const Csec_Ip_ReqType *pRequest, Csec_Ip_KeyIdType eKeyId, const uint8 *pPlainText, uint32 u32Length, const uint8 *pIV, uint8 *pCipherText)

  Performs the AES-128 encryption in CBC mode.
- Csec_Ip_ErrorCodeType Csec_Ip_DecryptCbc (const Csec_Ip_ReqType *pRequest, Csec_Ip_KeyIdType eKeyId, const uint8 *pCipherText, uint32 u32Length, const uint8 *pIV, uint8 *pPlainText)

  Performs the AES-128 decryption in CBC mode.
- Csec_Ip_ErrorCodeType Csec_Ip_GenerateMac (const Csec_Ip_ReqType *pRequest, Csec_Ip_KeyIdType eKeyId, const uint8 *pMsg, uint32 u32MsgLen, uint8 *pCmac)

  Calculates the MAC of a given message using CMAC with AES-128.
- Csec_Ip_ErrorCodeType Csec_Ip_VerifyMac (const Csec_Ip_ReqType *pRequest, Csec_Ip_KeyIdType eKeyId, const uint8 *pMsg, uint32 u32MsgLen, const uint8 *pMac, uint16 u16MacLen, boolean *pbVerifStatus)

  Verifies the MAC of a given message using CMAC with AES-128.
- Csec_Ip_ErrorCodeType Csec_Ip_LoadKey (Csec_Ip_KeyIdType eKeyId, const uint8 *pM1, const uint8 *pM2, const uint8 *pM3, uint8 *pM4, uint8 *pM5)

  Updates an internal key per the SHE specification.
- Csec_Ip_ErrorCodeType Csec_Ip_LoadPlainKey (const uint8 *pPlainKey)

  Updates the RAM key memory slot with a 128-bit plaintext.
- Csec_Ip_ErrorCodeType Csec_Ip_ExportRamKey (uint8 *pM1, uint8 *pM2, uint8 *pM3, uint8 *pM4, uint8 *pM5)

  Exports the RAM_KEY into a format protected by SECRET_KEY.
- Csec_Ip_ErrorCodeType Csec_Ip_InitRng (void)

  Initializes the seed and derives a key for the PRNG.
- Csec_Ip_ErrorCodeType Csec_Ip_ExtendSeed (const uint8 *pEntropy)

  Extends the seed of the PRNG.
- Csec_Ip_ErrorCodeType Csec_Ip_GenerateRnd (const Csec_Ip_ReqType *pRequest, uint8 *pRnd)

  Generates a vector of 128 random bits.
- Csec_Ip_ErrorCodeType Csec_Ip_BootFailure (void)

  Signals a failure detected during later stages of the boot process.
- Csec_Ip_ErrorCodeType Csec_Ip_BootOk (void)

  Marks a successful boot verification during later stages of the boot process.
- Csec_Ip_ErrorCodeType Csec_Ip_BootDefine (uint32 u32BootSize, Csec_Ip_BootFlavorType eBootFlavor)

  Implements an extension of the SHE standard to define both the user boot size and boot method.
- Csec_Ip_StatusType Csec_Ip_GetStatus (void)

  Returns the content of the status register.
- Csec_Ip_ErrorCodeType Csec_Ip_GetId (const uint8 *pChallenge, uint8 *pUid, uint8 *pSreg, uint8 *pMac)

Returns the identity (UID) and the value of the status register protected by a MAC over a challenge and the data.

- Csec_Ip_ErrorCodeType Csec_Ip_DbgChal (uint8 *pChallenge)

  Obtains a random number which the user shall use along with the MASTER_ECU_KEY and UID to return an authorization request.
- Csec_Ip_ErrorCodeType Csec_Ip_DbgAuth (const uint8 *pAuthorization)

  Erases all keys (actual and outdated) stored in NVM Memory if the authorization is confirmed by CSEc.
- Csec_Ip_ErrorCodeType Csec_Ip_MpCompress (const uint8 *pMsg, uint16 u16MsgLen, uint8 *pMp↩ Compress)

  Compresses the given messages by accessing the Miyaguchi-Preneel compression feature from the CSEc feature set.
- void Csec_Ip_MainFunction (void)

  Function that should be called cyclically to process the requests sent using asynchronous poll method.
- void Csec_Ip_CancelCommand (void)

  Cancels a previously launched asynchronous command.
- void Csec_Ip_SetSynchronousCmdTimeout (uint32 u32Timeout)

  Updates the timeout for the synchronous commands.
- void Csec_Ip_IrqHandler (void)

  Interrupt handler.
- Csec_Ip_ErrorCodeType Csec_Ip_VerifyMacAddrMode (Csec_Ip_KeyIdType eKeyId, const uint8 *pMsg, uint32 u32MsgLen, const uint8 *pMac, uint16 u16MacLen, boolean *pbVerifStatus)

  Verifies the MAC of a given message (located in Flash) using CMAC with AES-128.
- Csec_Ip_ErrorCodeType Csec_Ip_GenerateMacAddrMode (Csec_Ip_KeyIdType eKeyId, const uint8 *p↩ Msg, uint32 u32MsgLen, uint8 *pCmac)

  Calculates the MAC of a given message (located in Flash) using CMAC with AES-128.

6.3.2 Data Structure Documentation

6.3.2.1 struct Csec_Ip_StateType

Structure defining driver state.

Definition at line 251 of file Csec_Ip.h.

Data Fields

- boolean bCmdInProgress
- Csec_Ip_CmdType eCmd
- const uint8 * pInputBuff
- uint8 * pOutputBuff
- uint32 u32Index
- uint32 u32InputSize
- uint8 u8PartialSize
- Csec_Ip_KeyIdType eKeyId
- Csec_Ip_ErrorCodeType ErrCode
- const uint8 * pIV
- Csec_Ip_CallSequenceType eSeq
- uint32 u32MsgLen
- boolean * pbVerifStatus

- boolean bMacWritten
- const uint8 * pMac
- uint16 u16MacLen
- pfCsecIpResponseCallbackType pfCallback
- void * pCallbackParam
- Csec_Ip_ReqTypeType eReqType
- uint32 u32Timeout

6.3.2.1.1 Field Documentation

6.3.2.1.1.1 bCmdInProgress `boolean bCmdInProgress`

Specifies if a command is in progress. If a command is in progress this boolean will be set to TRUE

Definition at line 253 of file Csec_Ip.h.

6.3.2.1.1.2 eCmd `Csec_Ip_CmdType eCmd`

Specifies the type of the command in execution

Definition at line 254 of file Csec_Ip.h.

6.3.2.1.1.3 pInputBuff `const uint8* pInputBuff`

Specifies the input pointer of the command in execution

Definition at line 255 of file Csec_Ip.h.

6.3.2.1.1.4 pOutputBuff `uint8* pOutputBuff`

Specifies the output pointer of the command in execution

Definition at line 256 of file Csec_Ip.h.

6.3.2.1.1.5 u32Index `uint32 u32Index`

Specifies the index in the input buffer of the command in execution

Definition at line 257 of file Csec_Ip.h.

6.3.2.1.1.6 u32InputSize `uint32 u32InputSize`

Specifies the size of the input of the command in execution

Definition at line 258 of file Csec_Ip.h.

6.3.2.1.1.7 u8PartialSize `uint8 u8PartialSize`

Specifies the size of the currently processed chunck of the input

Definition at line 259 of file Csec_Ip.h.

6.3.2.1.1.8 eKeyId `Csec_Ip_KeyIdType eKeyId`

Specifies the key used for the command in execution

Definition at line 260 of file Csec_Ip.h.

6.3.2.1.1.9 ErrCode `Csec_Ip_ErrorCodeType ErrCode`

Specifies the error code of the last executed command

Definition at line 261 of file Csec_Ip.h.

6.3.2.1.1.10 pIV `const uint8* pIV`

Specifies the IV of the command in execution (for encryption/decryption using CBC mode)

Definition at line 262 of file Csec_Ip.h.

6.3.2.1.1.11 eSeq `Csec_Ip_CallSequenceType eSeq`

Specifies if the information is the first or a following function call.

Definition at line 263 of file Csec_Ip.h.

6.3.2.1.1.12 u32MsgLen `uint32 u32MsgLen`

Specifies the message size (in bits) for the command in execution (for MAC generation/verification)

Definition at line 264 of file Csec_Ip.h.

6.3.2.1.1.13 pbVerifStatus `boolean* pbVerifStatus`

Specifies the result of the last executed MAC verification command

Definition at line 265 of file Csec_Ip.h.

6.3.2.1.1.14 bMacWritten `boolean bMacWritten`

Specifies if the MAC to be verified was written in CSE_PRAM for a MAC verification command

Definition at line 266 of file Csec_Ip.h.

6.3.2.1.1.15 pMac `const uint8* pMac`

Specifies the MAC to be verified for a MAC verification command

Definition at line 267 of file Csec_Ip.h.

6.3.2.1.1.16 u16MacLen `uint16 u16MacLen`

Specifies the number of bits of the MAC to be verified for a MAC verification command

Definition at line 268 of file Csec_Ip.h.

6.3.2.1.1.17 pfCallback `pfCsecIpResponseCallbackType pfCallback`

The callback invoked when an asynchronous command is completed

Definition at line 269 of file Csec_Ip.h.

6.3.2.1.1.18 pCallbackParam void* pCallbackParam

User parameter for the command completion callback

Definition at line 270 of file Csec_Ip.h.

6.3.2.1.1.19 eReqType Csec_Ip_ReqTypeType eReqType

Selects the request type, asynchronous using interrupts or asynchronous polling

Definition at line 271 of file Csec_Ip.h.

6.3.2.1.1.20 u32Timeout uint32 u32Timeout

Timeout for a command in ticks or microseconds depending on the value (TICKS or SYSTEM) of the 'CSEc Ip Timeout Counter Type' attribute in the configuration.

Definition at line 272 of file Csec_Ip.h.

6.3.2.2 struct Csec_Ip_ReqType

Structure defining request parameters.

Definition at line 279 of file Csec_Ip.h.

Data Fields

- Csec_Ip_ReqTypeType eReqType
- pfCsecIpResponseCallbackType pfCallback
- void * pCallbackParam

6.3.2.2.1 Field Documentation

6.3.2.2.1.1 eReqType Csec_Ip_ReqTypeType eReqType

Selects the request type (POLL/IRQ)

Definition at line 281 of file Csec_Ip.h.

6.3.2.2.1.2 pfCallback pfCsecIpResponseCallbackType pfCallback

The callback for asynchronous request

Definition at line 282 of file Csec_Ip.h.

6.3.2.2.1.3 pCallbackParam void* pCallbackParam

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

Definition at line 283 of file Csec_Ip.h.

6.3.2.3 struct Csec_Ip_PramType

CSE_PRAM - Register Layout Typedef

Definition at line 100 of file Csec_Ip_Pram.h.

Data Fields

Type	Name	Description
___IO uint32	Csec_Ip_aPramRegister[((uint8) 32U)]	CSE PRAM 0 Register to CSE PRAM 31 Register, array offset: 0x0, array step: 0x4

6.3.3 Macro Definition Documentation

6.3.3.1 CSEC_IP_STATUS_BUSY_U8

```
#define CSEC_IP_STATUS_BUSY_U8
```

The bit is set when CSEC is processing a command.

Definition at line 81 of file Csec_Ip.h.

6.3.3.2 CSEC_IP_STATUS_SECURE_BOOT_U8

```
#define CSEC_IP_STATUS_SECURE_BOOT_U8
```

The bit is set if the secure booting is activated.

Definition at line 83 of file Csec_Ip.h.

6.3.3.3 CSEC_IP_STATUS_BOOT_INIT_U8

```
#define CSEC_IP_STATUS_BOOT_INIT_U8
```

The bit is set if the secure booting has been personalized during the boot sequence.

Definition at line 85 of file Csec_Ip.h.

6.3.3.4 CSEC_IP_STATUS_BOOT_FINISHED_U8

```
#define CSEC_IP_STATUS_BOOT_FINISHED_U8
```

The bit is set when the secure booting has been finished by calling either CMD_BOOT_FAILURE or CMD_B↩ OOT_OK or if CMD_SECURE_BOOT failed in verifying BOOT_MAC.

Definition at line 90 of file Csec_Ip.h.

6.3.3.5 CSEC_IP_STATUS_BOOT_OK_U8

```
#define CSEC_IP_STATUS_BOOT_OK_U8
```

The bit is set if the secure booting (CMD_SECURE_BOOT) succeeded. If CMD_BOOT_FAILURE is called the bit is erased.

Definition at line 95 of file Csec_Ip.h.

6.3.3.6 CSEC_IP_STATUS_RND_INIT_U8

```
#define CSEC_IP_STATUS_RND_INIT_U8
```

The bit is set if the random number generator has been initialized.

Definition at line 97 of file Csec_Ip.h.

6.3.3.7 CSEC_IP_STATUS_EXT_DEBUGGER_U8

```
#define CSEC_IP_STATUS_EXT_DEBUGGER_U8
```

The bit is set if an external debugger is connected to the chip.

Definition at line 99 of file Csec_Ip.h.

6.3.3.8 CSEC_IP_STATUS_INT_DEBUGGER_U8

```
#define CSEC_IP_STATUS_INT_DEBUGGER_U8
```

The bit is set if the internal debugging mechanisms are activated.

Definition at line 101 of file Csec_Ip.h.

6.3.3.9 CSEC_IP_ERC_NO_ERROR

```
#define CSEC_IP_ERC_NO_ERROR
```

No error has occurred.

Definition at line 211 of file Csec_Ip.h.

6.3.3.10 CSEC_IP_ERC_SEQUENCE_ERROR

```
#define CSEC_IP_ERC_SEQUENCE_ERROR
```

The call sequence of the commands is invalid.

Definition at line 212 of file Csec_Ip.h.

6.3.3.11 CSEC_IP_ERC_KEY_NOT_AVAILABLE

```
#define CSEC_IP_ERC_KEY_NOT_AVAILABLE
```

The used key has DBG Attached flag and debugger is active.

Definition at line 213 of file Csec_Ip.h.

6.3.3.12 CSEC_IP_ERC_KEY_INVALID

```
#define CSEC_IP_ERC_KEY_INVALID
```

A function is called to perform an operation with a key that is not allowed for the given operation.

Definition at line 214 of file Csec_Ip.h.

6.3.3.13 CSEC_IP_ERC_KEY_EMPTY

```
#define CSEC_IP_ERC_KEY_EMPTY
```

Key slot is empty (not initialized)/not present or higher slot (not partitioned).

Definition at line 215 of file Csec_Ip.h.

6.3.3.14 CSEC_IP_ERC_NO_SECURE_BOOT

```
#define CSEC_IP_ERC_NO_SECURE_BOOT
```

Not applicable, BOOT_DEFINE once configured, will automatically run secure boot.

Definition at line 216 of file Csec_Ip.h.

6.3.3.15 CSEC_IP_ERC_KEY_WRITE_PROTECTED

```
#define CSEC_IP_ERC_KEY_WRITE_PROTECTED
```

A key update is attempted on a write protected key slot or the debugger is started while a key is write-protected.

Definition at line 217 of file Csec_Ip.h.

6.3.3.16 CSEC_IP_ERC_KEY_UPDATE_ERROR

```
#define CSEC_IP_ERC_KEY_UPDATE_ERROR
```

A key update did not succeed due to errors in verification of the messages.

Definition at line 218 of file Csec_Ip.h.

6.3.3.17 CSEC_IP_ERC_RNG_SEED

```
#define CSEC_IP_ERC_RNG_SEED
```

The PRNG seed has not yet been initialized.

Definition at line 219 of file Csec_Ip.h.

6.3.3.18 CSEC_IP_ERC_NO_DEBUGGING

```
#define CSEC_IP_ERC_NO_DEBUGGING
```

Internal debugging is not possible because the authentication did not succeed.

Definition at line 220 of file Csec_Ip.h.

6.3.3.19 CSEC_IP_ERC_MEMORY_FAILURE

```
#define CSEC_IP_ERC_MEMORY_FAILURE
```

General memory technology failure (multi-bit ECC error, common fault detection).

Definition at line 221 of file Csec_Ip.h.

6.3.3.20 CSEC_IP_ERC_GENERAL_ERROR

```
#define CSEC_IP_ERC_GENERAL_ERROR
```

Detected error that is not covered by the other error codes.

Definition at line 222 of file Csec_Ip.h.

6.3.3.21 CSEC_IP_ERC_NO_RESPONSE

```
#define CSEC_IP_ERC_NO_RESPONSE
```

No response received from Csec Ip in the timeout window.

Definition at line 223 of file Csec_Ip.h.

6.3.3.22 CSEC_IP_ERC_STATUS_BUSY

```
#define CSEC_IP_ERC_STATUS_BUSY
```

Another command is in progress.

Definition at line 224 of file Csec_Ip.h.

6.3.4 Types Reference

6.3.4.1 Csec_Ip_StatusType

```
typedef uint8 Csec_Ip_StatusType
```

Status of the CSEC cryptographic related feature set. Provides one bit for each status code as per SHE specification. CSEC IP status masks can be used for status verification.

Definition at line 116 of file Csec_Ip.h.

6.3.4.2 Csec_Ip_ErrorCodeType

```
typedef uint16 Csec_Ip_ErrorCodeType
```

Unsigned integer defining the CSEC error codes.

A 16 bitfield that provides one bit for each error code.

Definition at line 210 of file Csec_Ip.h.

6.3.4.3 pfCsecIpResponseCallbackType

```
typedef void(* pfCsecIpResponseCallbackType) (Csec_Ip_ErrorCodeType ErrCode, Csec_Ip_CmdType eCompleted↩
Cmd, void *pCallbackParam)
```

Callback for asynchronous command.

Definition at line 230 of file Csec_Ip.h.

6.3.5 Enum Reference

6.3.5.1 Csec_Ip_KeyIdType

```
enum Csec_Ip_KeyIdType
```

Enum defining the Key IDs and key memory slot idenfitication.

Key ID values based on key bank select and key index

Enumerator

CSEC_IP_SECRET_KEY	Secret key.
CSEC_IP_MASTER_ECU_KEY	Master ECU key.
CSEC_IP_BOOT_MAC_KEY	Boot MAC key used by the secure booting mechanism to verify the authenticity of the software.
CSEC_IP_BOOT_MAC	Stores the MAC of the Bootloader for the secure booting mechanism.
CSEC_IP_KEY_1	User key 1.
CSEC_IP_KEY_2	User key 2.
CSEC_IP_KEY_3	User key 3.
CSEC_IP_KEY_4	User key 4.
CSEC_IP_KEY_5	User key 5.
CSEC_IP_KEY_6	User key 6.
CSEC_IP_KEY_7	User key 7.
CSEC_IP_KEY_8	User key 8.
CSEC_IP_KEY_9	User key 9.
CSEC_IP_KEY_10	User key 10.
CSEC_IP_RAM_KEY	A volatile key that can be used for arbitrary operations.
CSEC_IP_KEY_11	User key 11.
CSEC_IP_KEY_12	User key 12.
CSEC_IP_KEY_13	User key 13.
CSEC_IP_KEY_14	User key 14.
CSEC_IP_KEY_15	User key 15.
CSEC_IP_KEY_16	User key 16.
CSEC_IP_KEY_17	User key 17.
CSEC_IP_KEY_18	User key 18.
CSEC_IP_KEY_19	User key 19.
CSEC_IP_KEY_20	User key 20.
CSEC_IP_KEY_INVALID	Invalid key.

Definition at line 123 of file Csec_Ip.h.

6.3.5.2 Csec_Ip_CmdType

```
enum Csec_Ip_CmdType
```

Enum defining SHE compliant commands present in the CSEc command set.

Enumerator

CSEC_IP_CMD_ENC_ECB	AES-128 encryption in ECB mode.
CSEC_IP_CMD_ENC_CBC	AES-128 encryption in CBC mode.
CSEC_IP_CMD_DEC_ECB	AES-128 decryption in ECB mode.

Enumerator

CSEC_IP_CMD_DEC_CBC	AES-128 decryption in CBC mode.
CSEC_IP_CMD_GENERATE_MAC	AES-128 based CMAC generation.
CSEC_IP_CMD_VERIFY_MAC	AES-128 based CMAC verification.
CSEC_IP_CMD_LOAD_KEY	Internal key update.
CSEC_IP_CMD_LOAD_PLAIN_KEY	RAM key update.
CSEC_IP_CMD_EXPORT_RAM_KEY	RAM key export.
CSEC_IP_CMD_INIT_RNG	PRNG initialization.
CSEC_IP_CMD_EXTEND_SEED	PRNG seed entropy extension.
CSEC_IP_CMD_RND	Random number generation.
CSEC_IP_CMD_BOOT_FAILURE	Impose sanctions during invalid boot.
CSEC_IP_CMD_BOOT_OK	Finish boot verification.
CSEC_IP_CMD_GET_ID	Get UID.
CSEC_IP_CMD_BOOT_DEFINE	Secure boot configuration.
CSEC_IP_CMD_DBG_CHAL	Get debug challenge.
CSEC_IP_CMD_DBG_AUTH	Debug authentication.
CSEC_IP_CMD_MP_COMPRESS	Miyaguchi-Preneel compression.

Definition at line 157 of file Csec_Ip.h.

6.3.5.3 Csec_Ip_CallSequenceType

enum Csec_Ip_CallSequenceType

Enum defining call sequence.

Data can be processed in one function call or, if it is too large and can not be done in a single function call, a series of function calls can be used to process the data set. This enum will provide the information regarding the call sequence, if it is the first or a following function call.

Enumerator

CSEC_IP_CALL_SEQ_FIRST	1st function call
CSEC_IP_CALL_SEQ_SUBSEQUENT	2nd to nth function call

Definition at line 187 of file Csec_Ip.h.

6.3.5.4 Csec_Ip_BootFlavorType

enum Csec_Ip_BootFlavorType

Enum defining the boot type for the BOOT_DEFINE command.

Enumerator

CSEC_IP_BOOT_STRICT	Strict Boot method.
CSEC_IP_BOOT_SERIAL	Serial Boot method.
CSEC_IP_BOOT_PARALLEL	Parallel Boot method.
CSEC_IP_BOOT_NOT_DEFINED	No Boot defined or non-CSEC enabled part.

Definition at line 197 of file Csec_Ip.h.

6.3.5.5 Csec_Ip_ReqTypeType

```
enum Csec_Ip_ReqTypeType
```

Enum defining the possible asynchronous types of service requests.

Enumerator

CSEC_IP_REQTYPE_SYNC	Synchronous - the service request function does not return until the CSEC completes the request, or the timeout expires
CSEC_IP_REQTYPE_ASYNC_IRQ	Asynchronous using interrupts - the service request function returns right after sending the request to CSEc; an interrupt is triggered when CSEc completes the request (application can be notified through the callback)
CSEC_IP_REQTYPE_ASYNC_POLL	Asynchronous polling - the service request function returns right after sending the request to CSEc; application must poll the driver by calling Csec_Ip_MainFunction

Definition at line 236 of file Csec_Ip.h.

6.3.6 Function Reference

6.3.6.1 Csec_Ip_Init()

```
void Csec_Ip_Init (
            Csec_Ip_StateType * pState )
```

Initializes the internal state of the driver.

Parameters

in	*pState*	Pointer to the state structure which will be used for holding the internal state of the driver.

Returns

void

6.3.6.2 Csec_Ip_Deinit()

```
void Csec_Ip_Deinit (
            void  )
```

Clears the internal state of the driver.

Returns

void

6.3.6.3 Csec_Ip_EncryptEcb()

```
Csec_Ip_ErrorCodeType Csec_Ip_EncryptEcb (
            const Csec_Ip_ReqType * pRequest,
            Csec_Ip_KeyIdType eKeyId,
            const uint8 * pPlainText,
            uint32 u32Length,
            uint8 * pCipherText )
```

Performs the AES-128 encryption in ECB mode.

This function performs the AES-128 encryption in ECB mode of the input plain text buffer The request can be performed synchronous or asynchronous.

Parameters

in	*pRequest*	Pointer to a structure that describes the request parameters, containing the request type (sync/async polling/async interrupt), pointer to the callback to be called when async operation completes, parameter to send to the callback when async operation completes.
in	*eKeyId*	KeyID used to perform the cryptographic operation.
in	*pPlainText*	Pointer to the plain text buffer.
in	*u32Length*	Number of bytes of plain text message to be encrypted. It should be multiple of 16 bytes.
out	*pCipherText*	Pointer to the cipher text buffer. The buffer shall have the same size as the plain text buffer.

Returns

Error Code after command execution. Output parameters are valid if the error code is CSEC_IP_ERC_N↩ O_ERROR.

6.3.6.4 Csec_Ip_DecryptEcb()

```
Csec_Ip_ErrorCodeType Csec_Ip_DecryptEcb (
            const Csec_Ip_ReqType * pRequest,
            Csec_Ip_KeyIdType eKeyId,
            const uint8 * pCipherText,
            uint32 u32Length,
            uint8 * pPlainText )
```

Performs the AES-128 decryption in ECB mode.

This function performs the AES-128 decryption in ECB mode of the input cipher text buffer. The request can be performed synchronous or asynchronous.

Parameters

in	*pRequest*	Pointer to a structure that describes the request parameters, containing the request type (sync/async polling/async interrupt), pointer to the callback to be called when async operation completes, parameter to send to the callback when async operation completes.
in	*eKeyId*	KeyID used to perform the cryptographic operation
in	*pCipherText*	Pointer to the cipher text buffer.
in	*u32Length*	Number of bytes of cipher text message to be decrypted. It should be multiple of 16 bytes.
out	*pPlainText*	Pointer to the plain text buffer. The buffer shall have the same size as the cipher text buffer.

Returns

Error Code after command execution. Output parameters are valid if the error code is CSEC_IP_ERC_N↩O_ERROR.

6.3.6.5 Csec_Ip_EncryptCbc()

```
Csec_Ip_ErrorCodeType Csec_Ip_EncryptCbc (
            const Csec_Ip_ReqType * pRequest,
            Csec_Ip_KeyIdType eKeyId,
            const uint8 * pPlainText,
            uint32 u32Length,
            const uint8 * pIV,
            uint8 * pCipherText )
```

Performs the AES-128 encryption in CBC mode.

This function performs the AES-128 encryption in CBC mode of the input plaintext buffer. The request can be performed synchronous or asynchronous.

Parameters

in	*pRequest*	Pointer to a structure that describes the request parameters, containing the request type (sync/async polling/async interrupt), pointer to the callback to be called when async operation completes, parameter to send to the callback when async operation completes.
in	*eKeyId*	KeyID used to perform the cryptographic operation.
in	*pPlainText*	Pointer to the plain text buffer.
in	*u32Length*	Number of bytes of plain text message to be encrypted. It should be multiple of 16 bytes.
in	*pIV*	Pointer to the initialization vector buffer.
out	*pCipherText*	Pointer to the cipher text buffer. The buffer shall have the same size as the plain text buffer.

Returns

Error Code after command execution. Output parameters are valid if the error code is CSEC_IP_ERC_N↩ O_ERROR.

6.3.6.6 Csec_Ip_DecryptCbc()

```
Csec_Ip_ErrorCodeType Csec_Ip_DecryptCbc (
            const Csec_Ip_ReqType * pRequest,
            Csec_Ip_KeyIdType eKeyId,
            const uint8 * pCipherText,
            uint32 u32Length,
            const uint8 * pIV,
            uint8 * pPlainText )
```

Performs the AES-128 decryption in CBC mode.

This function performs the AES-128 decryption in CBC mode of the input cipher text buffer. The request can be performed synchronous or asynchronous.

Parameters

in	*pRequest*	Pointer to a structure that describes the request parameters, containing the request type (sync/async polling/async interrupt), pointer to the callback to be called when async operation completes, parameter to send to the callback when async operation completes.
in	*eKeyId*	KeyID used to perform the cryptographic operation.
in	*pCipherText*	Pointer to the cipher text buffer.
in	*u32Length*	Number of bytes of cipher text message to be decrypted. It should be multiple of 16 bytes.
in	*pIV*	Pointer to the initialization vector buffer.
out	*pPlainText*	Pointer to the plain text buffer. The buffer shall have the same size as the cipher text buffer.

Returns

Error Code after command execution. Output parameters are valid if the error code is CSEC_IP_ERC_N↩ O_ERROR.

6.3.6.7 Csec_Ip_GenerateMac()

```
Csec_Ip_ErrorCodeType Csec_Ip_GenerateMac (
            const Csec_Ip_ReqType * pRequest,
            Csec_Ip_KeyIdType eKeyId,
            const uint8 * pMsg,
            uint32 u32MsgLen,
            uint8 * pCmac )
```

Calculates the MAC of a given message using CMAC with AES-128.

This function calculates the MAC of a given message using CMAC with AES-128. The request can be performed synchronous or asynchronous.

Parameters

in	*pRequest*	Pointer to a structure that describes the request parameters, containing the request type (sync/async polling/async interrupt), pointer to the callback to be called when async operation completes, parameter to send to the callback when async operation completes.
in	*eKeyId*	KeyID used to perform the cryptographic operation.
in	*pMsg*	Pointer to the message buffer.
in	*u32MsgLen*	Number of bits of message on which CMAC will be computed.
out	*pCmac*	Pointer to the buffer containing the result of the CMAC computation.

Returns

Error Code after command execution. Output parameters are valid if the error code is CSEC_IP_ERC_N↩ O_ERROR.

6.3.6.8 Csec_Ip_VerifyMac()

```
Csec_Ip_ErrorCodeType Csec_Ip_VerifyMac (
            const Csec_Ip_ReqType * pRequest,
            Csec_Ip_KeyIdType eKeyId,
            const uint8 * pMsg,
            uint32 u32MsgLen,
            const uint8 * pMac,
            uint16 u16MacLen,
            boolean * pbVerifStatus )
```

Verifies the MAC of a given message using CMAC with AES-128.

This function verifies the MAC of a given message using CMAC with AES-128. The request can be performed synchronous or asynchronous.

Parameters

in	*pRequest*	Pointer to a structure that describes the request parameters, containing the request type (sync/async polling/async interrupt), pointer to the callback to be called when async operation completes, parameter to send to the callback when async operation completes.
in	*eKeyId*	KeyID used to perform the cryptographic operation.
in	*pMsg*	Pointer to the message buffer.
in	*pMsg*	Number of bits of message on which CMAC will be computed.
in	*pMac*	Pointer to the buffer containing the CMAC to be verified.
in	*u16MacLen*	Number of bits of the CMAC to be compared. A macLength value of zero indicates that all 128-bits are compared.
out	*pbVerifStatus*	Status of MAC verification command (true: verification operation passed, false: verification operation failed).

Returns

Error Code after command execution. Output parameters are valid if the error code is CSEC_IP_ERC_N↩ O_ERROR.

6.3.6.9 Csec_Ip_LoadKey()

```
Csec_Ip_ErrorCodeType Csec_Ip_LoadKey (
            Csec_Ip_KeyIdType eKeyId,
            const uint8 * pM1,
            const uint8 * pM2,
            const uint8 * pM3,
            uint8 * pM4,
            uint8 * pM5 )
```

Updates an internal key per the SHE specification.

This function updates an internal key per the SHE specification.

Parameters

in	*e↩ KeyId*	KeyID of the key to be updated.
in	*pM1*	Pointer to the 128-bit M1 message containing the UID, Key ID and Authentication Key ID.
in	*pM2*	Pointer to the 256-bit M2 message contains the new security flags, counter and the key value all encrypted using a derived key generated from the Authentication Key.
in	*pM3*	Pointer to the 128-bit M3 message is a MAC generated over messages M1 and M2.
out	*pM4*	Pointer to a 256 bits buffer where the computed M4 parameter is stored.
out	*pM5*	Pointer to a 128 bits buffer where the computed M5 parameters is stored.

Returns

Error Code after command execution. Output parameters are valid if the error code is CSEC_IP_ERC_N↩ O_ERROR.

6.3.6.10 Csec_Ip_LoadPlainKey()

```
Csec_Ip_ErrorCodeType Csec_Ip_LoadPlainKey (
            const uint8 * pPlainKey )
```

Updates the RAM key memory slot with a 128-bit plaintext.

The function updates the RAM key memory slot with a 128-bit plaintext. The key is loaded without encryption and verification of the key, i.e. the key is handed over in plaintext. A plain key can only be loaded into the RAM_KEY slot.

Parameters

in	*pPlainKey*	Pointer to the 128-bit buffer containing the key that needs to be copied in RAM_KEY slot.

Returns

Error Code after command execution.

6.3.6.11 Csec_Ip_ExportRamKey()

```
Csec_Ip_ErrorCodeType Csec_Ip_ExportRamKey (
            uint8 * pM1,
            uint8 * pM2,
            uint8 * pM3,
            uint8 * pM4,
            uint8 * pM5 )
```

Exports the RAM_KEY into a format protected by SECRET_KEY.

This function exports the RAM_KEY into a format protected by SECRET_KEY.

Parameters

out	*pM1*	Pointer to a buffer where the M1 parameter will be exported.
out	*pM2*	Pointer to a buffer where the M2 parameter will be exported.
out	*pM3*	Pointer to a buffer where the M3 parameter will be exported.
out	*pM4*	Pointer to a buffer where the M4 parameter will be exported.
out	*pM5*	Pointer to a buffer where the M5 parameter will be exported.

Returns

Error Code after command execution. Output parameters are valid if the error code is CSEC_IP_ERC_N↩ O_ERROR.

6.3.6.12 Csec_Ip_InitRng()

```
Csec_Ip_ErrorCodeType Csec_Ip_InitRng (
            void  )
```

Initializes the seed and derives a key for the PRNG.

The function initializes the seed and derives a key for the PRNG. The function must be called before CMD_RND after every power cycle/reset.

Returns

Error Code after command execution.

6.3.6.13 Csec_Ip_ExtendSeed()

```
Csec_Ip_ErrorCodeType Csec_Ip_ExtendSeed (
            const uint8 * pEntropy )
```

Extends the seed of the PRNG.

Extends the seed of the PRNG by compressing the former seed value and the supplied entropy into a new seed. This new seed is then to be used to generate a random number by invoking the CMD_RND command. The random number generator must be initialized by CMD_INIT_RNG before the seed may be extended.

Parameters

in	*pEntropy*	Pointer to a 128-bit buffer containing the entropy.

Returns

Error Code after command execution. Output parameters are valid if the error code is CSEC_IP_ERC_N↩ O_ERROR.

6.3.6.14 Csec_Ip_GenerateRnd()

```
Csec_Ip_ErrorCodeType Csec_Ip_GenerateRnd (
            const Csec_Ip_ReqType * pRequest,
            uint8 * pRnd )
```

Generates a vector of 128 random bits.

The function returns a vector of 128 random bits. The random number generator has to be initialized by calling Csec_Ip_InitRng before random numbers can be supplied.

Parameters

in	*pRequest*	Pointer to a structure that describes the request parameters, containing the request type (sync/async polling/async interrupt), pointer to the callback to be called when async operation completes, parameter to send to the callback when async operation completes.
out	*pRnd*	Pointer to a 128-bit buffer where the generated random number has to be stored.

Returns

Error Code after command execution. Output parameters are valid if the error code is CSEC_IP_ERC_N↩ O_ERROR.

6.3.6.15 Csec_Ip_BootFailure()

```
Csec_Ip_ErrorCodeType Csec_Ip_BootFailure (
            void  )
```

Signals a failure detected during later stages of the boot process.

The function is called during later stages of the boot process to detect a failure.

Returns

Error Code after command execution.

6.3.6.16 Csec_Ip_BootOk()

```
Csec_Ip_ErrorCodeType Csec_Ip_BootOk (
            void  )
```

Marks a successful boot verification during later stages of the boot process.

The function is called during later stages of the boot process to mark successful boot verification.

Returns

Error Code after command execution.

6.3.6.17 Csec_Ip_BootDefine()

```
Csec_Ip_ErrorCodeType Csec_Ip_BootDefine (
            uint32 u32BootSize,
            Csec_Ip_BootFlavorType eBootFlavor )
```

Implements an extension of the SHE standard to define both the user boot size and boot method.

The function implements an extension of the SHE standard to define both the user boot size and boot method.

Parameters

in	*u32BootSize*	Number of blocks of 128-bit data to check on boot. Maximum size is 512kBytes.
in	*eBootFlavor*	The boot method.

Returns

Error Code after command execution.

6.3.6.18 Csec_Ip_GetStatus()

```
Csec_Ip_StatusType Csec_Ip_GetStatus (
            void  )
```

Returns the content of the status register.

The function shall return the content of the status register.

Returns

Value of the status register.

6.3.6.19 Csec_Ip_GetId()

```
Csec_Ip_ErrorCodeType Csec_Ip_GetId (
            const uint8 * pChallenge,
            uint8 * pUid,
            uint8 * pSreg,
            uint8 * pMac )
```

Returns the identity (UID) and the value of the status register protected by a MAC over a challenge and the data.

This function returns the identity (UID) and the value of the status register protected by a MAC over a challenge and the data.

Parameters

in	*pChallenge*	Pointer to the 128-bit buffer containing Challenge data.
out	*pUid*	Pointer to 120 bit buffer where the UID will be stored.
out	*pSreg*	Value of the status register.
out	*pMac*	Pointer to the 128 bit buffer where the MAC generated over challenge and UID and status will be stored.

Returns

Error Code after command execution. Output parameters are valid if the error code is CSEC_IP_ERC_N↩ O_ERROR.

6.3.6.20 Csec_Ip_DbgChal()

```
Csec_Ip_ErrorCodeType Csec_Ip_DbgChal (
            uint8 * pChallenge )
```

Obtains a random number which the user shall use along with the MASTER_ECU_KEY and UID to return an authorization request.

This function obtains a random number which the user shall use along with the MASTER_ECU_KEY and UID to return an authorization request.

Parameters

out	*pChallenge*	Pointer to the 128-bit buffer where the challenge data will be stored.

Returns

Error Code after command execution. Output parameters are valid if the error code is CSEC_IP_ERC_N↩ O_ERROR.

6.3.6.21 Csec_Ip_DbgAuth()

```
Csec_Ip_ErrorCodeType Csec_Ip_DbgAuth (
            const uint8 * pAuthorization )
```

Erases all keys (actual and outdated) stored in NVM Memory if the authorization is confirmed by CSEc.

This function erases all keys (actual and outdated) stored in NVM Memory if the authorization is confirmed by CSEc.

Parameters

in	*pAuthorization*	Pointer to the 128-bit buffer containing the authorization value.

Returns

Error Code after command execution.

6.3.6.22 Csec_Ip_MpCompress()

```
Csec_Ip_ErrorCodeType Csec_Ip_MpCompress (
            const uint8 * pMsg,
            uint16 u16MsgLen,
            uint8 * pMpCompress )
```

Compresses the given messages by accessing the Miyaguchi-Preneel compression feature from the CSEc feature set.

This function accesses a Miyaguchi-Preneel compression feature within the CSEc feature set to compress the given messages.

Parameters

in	*pMsg*	Pointer to the messages to be compressed. Messages must be pre-processed per SHE specification if they do not already meet the full 128-bit block size requirement.
in	*u16MsgLen*	The number of 128 bit messages to be compressed.
out	*pMpCompress*	Pointer to the 128 bit buffer storing the compressed data.

Returns

Error Code after command execution. Output parameters are valid if the error code is CSEC_IP_ERC_N↩O_ERROR.

6.3.6.23 Csec_Ip_MainFunction()

```
void Csec_Ip_MainFunction (
            void  )
```

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

After an asynchronous poll request is sent, the layer on top of the Csec_Ip should call periodically the Csec_Ip_MainFunction() in order to retrieve message processing status from CSEc and when a response is received, call the callback sent at request time.

Returns

void.

6.3.6.24 Csec_Ip_CancelCommand()

```
void Csec_Ip_CancelCommand (
            void  )
```

Cancels a previously launched asynchronous command.

Returns

void

6.3.6.25 Csec_Ip_SetSynchronousCmdTimeout()

```
void Csec_Ip_SetSynchronousCmdTimeout (
            uint32 u32Timeout )
```

Updates the timeout for the synchronous commands.

Updates the internal driver state with the given timeout. After calling this function the synchronous API calls will use the new timeout value.

Parameters

in	*u32Timeout*	Timeout for a command in ticks or microseconds depending on the value (TICKS or SYSTEM) of the 'CSEc Ip Timeout Counter Type' attribute in the configuration.

Returns

void

6.3.6.26 Csec_Ip_IrqHandler()

```
void Csec_Ip_IrqHandler (
            void  )
```

Interrupt handler.

This function processes the related interrupts from CSEC

Returns

void

6.3.6.27 Csec_Ip_VerifyMacAddrMode()

```
Csec_Ip_ErrorCodeType Csec_Ip_VerifyMacAddrMode (
            Csec_Ip_KeyIdType eKeyId,
            const uint8 * pMsg,
            uint32 u32MsgLen,
            const uint8 * pMac,
            uint16 u16MacLen,
            boolean * pbVerifStatus )
```

Verifies the MAC of a given message (located in Flash) using CMAC with AES-128.

This function verifies the MAC of a given message using CMAC with AES-128. It is different from the Csec_Ip_↩ VerifyMac function in the sense that it does not involve an extra copy of the data on which the CMAC is computed and the message pointer should be a pointer to Flash memory.

Parameters

in	*eKeyId*	KeyID used to perform the cryptographic operation.
in	*pMsg*	Pointer to the message buffer (pointing to Flash memory).
in	*u32MsgLen*	Number of bits of message on which CMAC will be computed.
in	*pMac*	Pointer to the buffer containing the CMAC to be verified.
in	*u16MacLen*	Number of bits of the CMAC to be compared. A macLength value of zero indicates that all 128-bits are compared.
out	*pbVerifStatus*	Status of MAC verification command (true: verification operation passed, false: verification operation failed).

Returns

Error Code after command execution. Output parameters are valid if the error code is CSEC_IP_ERC_N↩ O_ERROR.

6.3.6.28 Csec_Ip_GenerateMacAddrMode()

```
Csec_Ip_ErrorCodeType Csec_Ip_GenerateMacAddrMode (
            Csec_Ip_KeyIdType eKeyId,
            const uint8 * pMsg,
            uint32 u32MsgLen,
            uint8 * pCmac )
```

Calculates the MAC of a given message (located in Flash) using CMAC with AES-128.

This function calculates the MAC of a given message using CMAC with AES-128. It is different from the Csec↩ _Ip_GenerateMac function in the sense that it does not involve an extra copy of the data on which the CMAC is computed and the message pointer should be a pointer to Flash memory.

Parameters

in	*eKeyId*	KeyID used to perform the cryptographic operation.
in	*pMsg*	Pointer to the message buffer (pointing to Flash memory).
in	*u32MsgLen*	Number of bits of message on which CMAC will be computed.
out	*pCmac*	Pointer to the buffer containing the result of the CMAC computation.

Returns

Error Code after command execution. Output parameters are valid if the error code is CSEC_IP_ERC_N↩ O_ERROR.

How to Reach Us:

Home Page:
nxp.com

Web Support:
nxp.com/support

Information in this document is provided solely to enable system and software implementers to use NXP products. There are no express or implied copyright licenses granted hereunder to design or fabricate any integrated circuits based on the information in this document. NXP reserves the right to make changes without further notice to any products herein.

NXP makes no warranty, representation, or guarantee regarding the suitability of its products for any particular purpose, nor does NXP assume any liability arising out of the application or use of any product or circuit, and specifically disclaims any and all liability, including without limitation consequential or incidental damages. "Typical" parameters that may be provided in NXP data sheets and/or specifications can and do vary in different applications, and actual performance may vary over time. All operating parameters, including "typicals," must be validated for each customer application by customer's technical experts. NXP does not convey any license under its patent rights nor the rights of others. NXP sells products pursuant to standard terms and conditions of sale, which can be found at the following address: nxp.com/SalesTermsandConditions.