Document Title	Specification of Crypto Driver
Document Owner	AUTOSAR
Document Responsibility	AUTOSAR
Document Identification No	807

Document Status	published
Part of AUTOSAR Standard	Classic Platform
Part of Standard Release	R22-11

Document Change History

Date	Release	Changed by	Change Description
2022-11-24	R22-11	AUTOSAR Release Management	• Clarification on handling the Key state during Crypto_KeyElementSet API • Add support for custom service and related API
2021-11-25	R21-11	AUTOSAR Release Management	• Clarification of Sync/Async for APIs • Clarification of key format description for SHE-Keys • Clarification about key state after Crypto_KeyElementSet() API. • Input and Output be optional for AEAD encrypt and decrypt in update mode
2020-11-30	R20-11	AUTOSAR Release Management	• Minor corrections and editorial changes. • Support to save and restore workspace. • Add function to get and invalidate the overall key status. • Harmonize and extend crypto config • Consistent custom configuration for CryptoAlgorithmFamily and -Mode. • Key element handling in NVM.

Document Change History

Date	Release	Changed by	Change Description
2019-11-28	R19-11	AUTOSAR Release Management	• Minor corrections and editorial changes • Cleanup of return code and DET error • Default RNG configuration for CryptoDriver Objects • Clarifcation on Read/Write access for key element. • Remove certificate support functions • Remove virtual key references • Changed Document Status from Final to published
2018-10-31	4.4.0	AUTOSAR Release Management	• Remove secure counter • Align return values of interface functions. • Support source and destination buffers for crypto operations located in crypto driver. • Support key management operation in asynchronous mode
2017-12-08	4.3.1	AUTOSAR Release Management	• Rollout of ‘Runtime Errors’ • minor corrections, clarifications and editorial changes; For details please refer to the ChangeDocumentation
2016-11-30	4.3.0	AUTOSAR Release Management	• Initial Release

Table of Contents

1 Introduction and functional overview

This specification specifies the functionality, API and the configuration of the AUTOSAR Basic Software module Crypto Driver.

The Crypto Drivers are located in the Microcontroller Abstraction Layer, which is below the Crypto Hardware Abstraction Layer (Crypto Interface [4]) and the upper service layer (Crypto Service Manager [5]). The Crypto Driver is a driver for a specific device, that is only abstracting the features supported by the hardware.

The Crypto Drivers allow defining of different Crypto Driver Objects (i.e. AES accelerator, SW component, etc), which shall be used for concurrent requests in different buffers. For each hardware object a priority-dependent job processing shall be supported. A crypto software solution (i.e. software-based CDD) can define interfaces identical to the Crypto Drivers for interacting with the upper layers, which shall provide an interface to the applications.

2 Acronyms and abbreviations

Abbreviation / Acronym:	***Description:***
CDD	Complex Device Driver
CSM	Crypto Service Manager
CRYIF	Crypto Interface
CRYPTO	Crypto Driver
DET	Default Error Tracer
HSM	Hardware Security Module
HW	Hardware
SHE	Security Hardware Extension
SW	Software

2.1 Glossary of Terms

Terms:	***Description:***
Crypto Driver Object	A Crypto Driver implements one or more Crypto Driver Objects. The Crypto Driver Object can offer different crypto primitives in hardware or software. The Crypto Driver Objects of one Crypto Driver are independent of each other. There is only one workspace for each Crypto Driver Object (i.e. only one crypto primitive can be performed at the same time) The only exception of independency between Crypto Driver Object is the usage of a default Random Number Generator (see [SWS_Crypto_00225]).
Key	A Key can be referenced by a job in the Csm. In the Crypto Driver, the key references a specific key type.
Key Type	A key type consists of references to key elements. The key types are typically pre-configured by the vendor of the Crypto Driver.
Key Element	Key elements are used to store data. This data can be e.g. key material or the IV needed for AES encryption. It can also be used to configure the behaviour of the key management functions. Key elements from different keys have different memory area (both NV and RAM area).
Channel	A channel is the path from a Crypto Service Manager queue via the Crypto Interface to a specific Crypto Driver Object.
Job	A 'Job' is a configured 'CsmJob'. Among others, it refers to a key, a cryptographic primitive and a reference channel.
Crypto Primitive	'Primitive' is an instance of a configured cryptographic algorithm realized in a Crypto Driver Object. Among others it refers to a functionality provided by the CSM to the application, the concrete underlining 'algorithmfamily' (e.g. AES, MD5, RSA, ...), and a 'algorithmmode' (e.g. ECB, CBC, ...).
Operation	An operation of a crypto primitive declares what part of the crypto primitive shall be performed. There are three different operation

	modes:	
	START	Operation mode indicates a new request of a crypto primitive, and it shall cancel all previous requests of the same job and primitive.
	UPDATE	Operation mode indicates, that the crypto primitive expects input data.
	FINISH	Operation mode indicates, that after this part all data are fed completely and the crypto primitive can finalize the calculations.
	It is also possible to perform more than one operation at once by concatenating the corresponding bits of the operation mode argument.	
Priority	The priority of a job defines the importance of it. The higher the priority (as well in value), the more immediate the job will be executed. The priority of a cryptographic job is part of the configuration.	
Service	A 'Service' shall be understood as defined in the TR_Glossary document: A service is a type of operation that has a published specification of interface and behavior, involving a contract between the provider of the capability and the potential clients.	

3 Related documentation

3.1 Input documents

[1] AUTOSAR Layered Software Architecture
AUTOSAR_EXP_LayeredSoftwareArchitecture.pdf

[2] AUTOSAR General Requirements on Basic Software Modules
AUTOSAR_SRS_BSWGeneral.pdf

[3] AUTOSAR General Specification for Basic Software Modules
AUTOSAR_SWS_BSWGeneral.pdf

[4] AUTOSAR Specification of Crypto Interface
AUTOSAR_SWS_CryptoInterface.pdf

[5] AUTOSAR Specification of Crypto Service Manager
AUTOSAR_SWS_CryptoServiceManager.pdf

[6] AUTOSAR Requirements on Crypto Modules
AUTOSAR_SRS_CryptoStack.pdf

[7] AUTOSAR Specification of Secure Hardware Extension
AUTOSAR_TR_SecureHardwareExtension.pdf

[8] Glossary
AUTOSAR_TR_Glossary

[9] Specification of the 3GPP Confidentiality and Integrity Algorithms 128-EEA3 & 128-EIA3:
Document 1: 128-EEA3 and 128-EIA3 Specification, Version 1.7, 30th Dec 2011
Document 2: ZUC Specification, Version 1.6, 28th June 2011

[10] ISO/IEC 10118-3:2018 Part 3: Dedicated hash-functions (SM3)

[11] ISO/IEC 14888-3:2018 IT Security techniques — Digital signatures with appendix — Part 3: Discrete logarithm based mechanisms (SM2)

3.2 Related standards and norms

[12] IEC 7498-1 The Basic Model, IEC Norm, 1994

3.3 Related specification

AUTOSAR provides a General Specification on Basic Software (SWS BSW General) [3] which is also valid for Crypto Driver

Thus, the specification SWS BSW General [3] shall be considered as additional and required specification for Crypto Driver.

4 Constraints and assumptions

4.1 Limitations

n.a.

4.2 Applicability to car domains

The Crypto Driver can be used for all domain applications when security features are to be used.

5 Dependencies to other modules

[SWS_Crypto_00003] ⌈ If an off-chip crypto hardware module (e.g. external HSM) is used, the Crypto Driver shall use services of other MCAL drivers (e.g. SPI).
⌋
Hint: If the Crypto Driver uses services of other MCAL drivers (e.g. SPI), it must be ensured that these drivers are up and running before initializing the Crypto Driver module.

[SWS_Crypto_00116] ⌈The Crypto Driver shall be able to store key material in a non-volatile way if supported by the dedicated crypto hardware.
⌋

Note:
The Crypto Drivers are called by the Crypto Interface (CRYIF), which is implemented according to the cryptographic interface specification [4].

The Crypto Drivers access the underlying hardware and software objects, to calculate results with their cryptographic primitives. The results shall be forwarded to the CRYIF.

5.1 File structure

5.1.1 Code File Structure

The code file structure is not defined within this specification completely.

[SWS_Crypto_00005] ⌈ The code file structure shall contain a source file Crypto.c and a code file Crypto_KeyManagement.c.
⌋()

6 Requirements traceability

Requirement	Description	Satisfied by
SRS_BSW_00101	The Basic Software Module shall be able to initialize variables and hardware in a separate initialization function	SWS_Crypto_91000
SRS_BSW_00358	The return type of init() functions implemented by AUTOSAR Basic Software Modules shall be void	SWS_Crypto_91000
SRS_BSW_00407	Each BSW module shall provide a function to read out the version information of a dedicated module implementation	SWS_Crypto_91001
SRS_BSW_00414	Init functions shall have a pointer to a configuration structure as single parameter	SWS_Crypto_91000, SWS_Crypto_91016
SRS_CryptoStack_00008	The Crypto Stack shall allow static configuration of keys used for cryptographic jobs	SWS_Crypto_00184, SWS_Crypto_00185, SWS_Crypto_00186, SWS_Crypto_00187, SWS_Crypto_00188, SWS_Crypto_00189, SWS_Crypto_00190, SWS_Crypto_00191, SWS_Crypto_00192, SWS_Crypto_00193, SWS_Crypto_00243, SWS_Crypto_00244, SWS_Crypto_00245, SWS_Crypto_00246, SWS_Crypto_00247, SWS_Crypto_00251, SWS_Crypto_00254
SRS_CryptoStack_00061	The Crypto Stack shall support detection of invalid keys	SWS_Crypto_00242, SWS_Crypto_00248, SWS_Crypto_00249, SWS_Crypto_00250, SWS_Crypto_00251
SRS_CryptoStack_00086	The CSM module shall distinguish between error types	SWS_Crypto_00040
SRS_CryptoStack_00095	The Crypto Driver module shall strictly separate error and status information	SWS_Crypto_91018
SRS_CryptoStack_00098	The Crypto Driver shall provide access to all cryptographic algorithms supported by the hardware	SWS_Crypto_00013
SRS_CryptoStack_00117	Keys shall not be used if they are empty or corrupted	SWS_Crypto_00248, SWS_Crypto_00249, SWS_Crypto_00250, SWS_Crypto_00251, SWS_Crypto_91023, SWS_Crypto_91024
SRS_CryptoStack_00118	Key material shall be securely stored either	SWS_Crypto_00243, SWS_Crypto_00246, SWS_Crypto_91023, SWS_Crypto_91024,

	in NVM or CSM	SWS_Crypto_91025, SWS_Crypto_91026
SRS_CryptoStack_00119	Provide a proof that the key has been programmed correctly	SWS_Crypto_91026
SRS_CryptoStack_00120	Cleanup all key material on shutdown operation	SWS_Crypto_00243
SWS_BSW_00050	Check parameters passed to Initialization functions	SWS_Crypto_00215

7 Functional specification

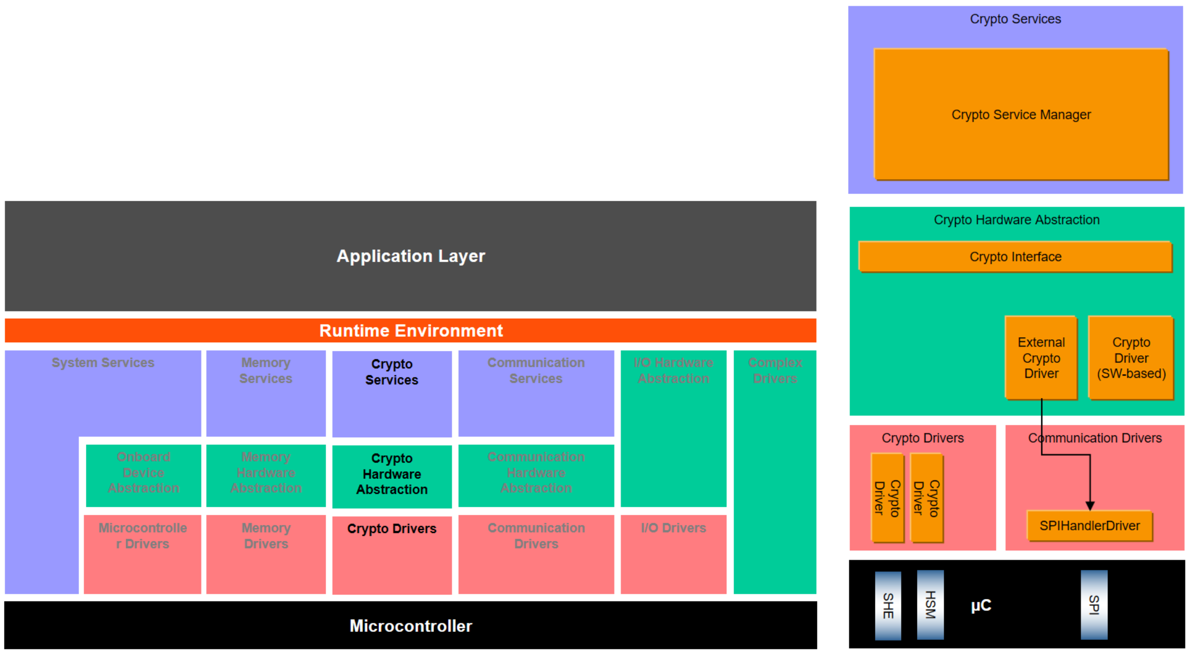


Figure 7-1: AUTOSAR Layered View with Crypto Driver Module

The Crypto Driver module is located in the micro controller abstraction layer and is below the Crypto Interface module and Crypto Service Manager module. It implements a generic interface for synchronous and asynchronous cryptographic primitives. It also supports key storage, key configuration, and key management for cryptographic services.

To provide cryptographic functionalities an ECU needs to integrate one unique Crypto Service Manager module and one Crypto Interface. However, the Crypto Interface can access several Crypto Drivers, each of them is configured according to the underlying Crypto Driver Object.

A Crypto Driver Object represents an instance of independent crypto hardware “device” (e.g. AES accelerator). There could be a channel for fast AES and CMAC calculations on an HSM for jobs with high priority, which ends on a native AES calculation service in the Crypto Driver. But it is also possible, that a Crypto Driver Object is a piece of software, e.g. for RSA calculations where jobs are able to encrypt, decrypt, sign or verify data. The Crypto Driver Object is the endpoint of a crypto channel.

7.1 Pre-Configuration

The vendor of the Crypto Driver has to provide a pre-configuration for the Crypto Driver which represents the capabilities of the Crypto Driver. The pre-configuration shall be delivered with the BSWMD-file of the Crypto Driver.

7.1.1 CryptoPrimitive configuration

The Crypto Driver contains one or more crypto driver objects that each supports one or more crypto primitive service. Algorithm family and mode provides details on how to operate crypto primitive services. For example, the crypto primitive service “Encrypt” supports the Family “AES” and the mode “CBC”. Further refinements are required by setting secondary family values.
The CryptoPrimitive configuration of a crypto driver object defines the possible options for a crypto primitive service. From the example above, the driver can support the modes CBC or ECB for an AES algorithm. This is sometimes referred as the “capabilities” of the driver for a specific crypto primitive service. Thus, a CryptoPrimitive may configure several family and mode values if it is applicable for a crypto service primitive.
The job configuration of the CSM contains a reference to a CsmPrimitive. The corresponding Config section of this CsmPrimitive also defines the families and mode, but with multiplicity of 1, to identify uniquely what combination of family and mode shall be used for a job. At least, the associated crypto driver object must contain a CryptoPrimitive that supports the configuration. This shall be ensured on configuration level and at least checked at runtime.
To support new features for a crypto service which aren’t yet modelled in the specification (in the various enumerations of family and mode), custom specific algorithm families and modes can be configured in the crypto driver. The CryptoPrimitive references to these custom values to indicate what combination of crypto primitive service, (custom-)family and (custom-)mode are possible.
The CSM job configuration need to reference to these custom configuration items.

[**SWS_Crypto_00239**] ⌈ A crypto driver that supports algorithm families and algorithm modes beyond the pre-defined ones available in [**ECUC_Crypto_00035**], [**ECUC_Crypto_00036**] and/or [**ECUC_Crypto_00037**] shall define custom specific value algorithm families and algorithm modes through the containers *CryptoPrimitiveAlgorithmFamilyCustom* and *CryptoPrimitiveAlgorithmModeCustom*.
⌋()

[**SWS_Crypto_00240**] ⌈ The *CryptoPrimitiveAlgorithmModeCustom/CryptoPrimitiveAlgorithmModeCustomId* and *CryptoPrimitiveAlgorithmFamilyCustom/CryptoPrimitiveAlgorithmFamilyCustomId* shall use the reserved range from 128..254.
⌋()

[**SWS_Crypto_00241**] ⌈ A *CryptoPrimitive* shall define one *CryptoPrimitiveService* and can define one or more items for *CryptoPrimitiveAlgorithmFamily*, *CryptoPrimitiveAlgorithmMode*, *CryptoPrimitiveAlgorithmSecondaryFamily*, *CryptoPrimitiveAlgorithmFamilyCustomRef* and *CryptoPrimitiveAlgorithmModeCustomRef* as long as all permutations of the multiple items can be supported. Otherwise, further `CryptoPrimitive(s)` for this CryptoPrimitiveService must be defined.

⌋()

Example:
A crypto primitive service ENCRYPT may support AES and 3DES with mode ECB and CBC in all permutations (AES-CBC, AES-ECB, 3DES-CBC and 3DES-ECB). But a crypto primitive service ENCRYPT, with support of family items for AES and RSA and mode CBC are not valid, because RSA-CBC is not reasonably supported. In this case, a new CryptoPrimitive for RSA shall be defined.

7.1.2 Cryptographic capabilities

The capabilities of a Crypto Driver can be divided in the two main topics: key storage and supported algorithms. The supported algorithms can be pre-configured by creating a new CryptoPrimitive container (e.g. MacGenerate). In this container the vendor can now specify that the Crypto Driver is for example only capable of doing a CMAC. In this case, an example configuration would be:

```
CryptoPrimitiveAlgorithmFamily  = CRYPTO_ALGOFAM_AES
CryptoPrimitiveAlgorithmMode  = CRYPTO_ALGOMODE_CMAC
CryptoPrimitiveAlgorithmSecondaryFamily =
CRYPTO_ALGOMODE_NOT_SET
CryptoPrimitiveService = MacGenerate
```

The primitive MacGenerate can then be referenced by the Crypto Driver Object to show, that it is capable of doing a CMAC. If no other primitives a pre-configured, the Crypto Driver Object is not able to perform e.g. an AES encryption.

If all primitives are independent from each other, a vendor would pre-configure one Crypto Driver Object for each primitive. Otherwise, there would be one Crypto Driver Object, which would reference all primitives.

7.1.3 Available Keys

The keys, which are provided by the Crypto Driver can also be pre-configured. A CryptoKey container references a specific CryptoKeyType. The CryptoKeyType provides the information which key elements are contained in a CryptoKey referencing this CryptoKeyType.

The vendor also pre-configures the key elements to define:
- read/write access
- the maximum size of the element
- if the element can be read/written with data smaller than the maximum size
- the init value after startup if the element is not already initialized

The init value is the value, which is stored into the key element at the initialization of the crypto driver when the key element is empty. It is e.g. used for the key element with the id CRYPTO_KE_<Service>_ALGORITHM. This way, the key management functions can be configured. To provide e.g. different key exchange algorithms in one Crypto Driver, the vendor can pre-configure the following containers and set the init values of the CRYPTO_KE_<Service>_ALGORITHM key element to a vendor specific value:

CryptoKeyElement_KeyExchange_Algorithm_RSA
- ID = 11
- Init value = 0x00
- Size = 1
- Read Access = RA_NONE
- Write Access = WA_NONE

CryptoKeyElement_KeyExchange_Algorithm_Ed25519
- ID = 11
- Init value = 0x01
- Size = 1
- Read Access = RA_NONE
- Write Access = WA_NONE

CryptoKeyType_KeyExchange_RSA
- CryptoKeyElement_KeyExchange_Algorithm_RSA
- CryptoKeyElement_KeyExchange_PartnerPubKey
- CryptoKeyElement_KeyExchange_OwnPubKey
- CryptoKeyElement_KeyExchange_Base
- CryptoKeyElement_KeyExchange_PrivKey
- CryptoKeyElement_KeyExchange_SharedValue

CryptoKeyType_KeyExchange_Ed25519
- CryptoKeyElement_KeyExchange_Algorithm_Ed25519
- CryptoKeyElement_KeyExchange_PartnerPubKey
- CryptoKeyElement_KeyExchange_OwnPubKey
- CryptoKeyElement_KeyExchange_Base
- CryptoKeyElement_KeyExchange_PrivKey
- CryptoKeyElement_KeyExchange_SharedValue

When a key exchange should be performed with a CryptoKey of type CryptoKeyType_KeyExchange_Ed25519, the Crypto Driver knows with the value stored in the key element CRYPTO_KE_KEYEXCHANGE_ALGORITHM that Ed25519 shall be used as underlying cryptographic primitive.

If a key should be used in more than one primitive e.g. KeyExchange and AES-Encrypt-CBC, the CryptoKeyType could be extended by needed elements:

CryptoKeyType_KeyExchange_Cipher_combined
- CryptoKeyElement_KeyExchange_Algorithm_Ed25519
- CryptoKeyElement_KeyExchange_PartnerPubKey
- CryptoKeyElement_KeyExchange_OwnPubKey
- CryptoKeyElement_KeyExchange_Base
- CryptoKeyElement_KeyExchange_PrivKey
- CryptoKeyElement_KeyExchange_SharedValue
    - ID = 1
- CryptoKeyElement_Cipher_IV

Note that CryptoKeyElement_KeyExchange_SharedValue has the id set to 1. When calling the encrypt service with a key of CryptoKeyType

CryptoKeyType_KeyExchange_Cipher_combined, the shared value of the key exchange is automatically used as encryption key.

7.2 General Behavior

The Crypto Driver can have one or more Crypto Driver Objects.

[SWS_Crypto_00012] ⌈ In case several Crypto Driver instances (of same or different vendor) are implemented in one ECU the file names, API names, and published parameters must be distinguished such that no two definitions with the same name are generated.
The name shall be formatted according to **SWS_BSW_00102:** `Crypto_<vi>_<ai>`, where `<vi>` is the `vendorId` and `<ai>` is the `vendorApiInfix`.
⌋()

[SWS_Crypto_00013] ⌈ The Crypto Driver may support all crypto primitives that are supported by the underlying hardware object.
⌋(SRS_CryptoStack_00098)

A job, declared in CSM specification [5], is an instance of a configured cryptographic primitive.

[SWS_Crypto_00014] ⌈ A Crypto Driver Object shall only support processing one job at one time.
⌋()

[SWS_Crypto_00117] ⌈ A Crypto Driver with n Crypto Driver Objects shall be able to process n jobs in parallel.
⌋()

Hint: Jobs, that are in the job queue (described in chapter 7.2.3.1), do not count as in processing.

[SWS_Crypto_00224] ⌈ If a Crypto Driver requires Random Number Generator services for internal use (e.g. Crypto_KeyExchangeCalcPubVal), it shall configure the first Crypto Driver Object (object number 0) for this purpose. The configuration is done in ECUC_Crypto_00044 and ECUC_Crypto_00045.
⌋()

It is also possible to configure and use other Crypto Driver Objects with its own default Random Number Generator.

[SWS_Crypto_00225] ⌈ If a Crypto Driver Object has no default Random Number Generator but requires Random Number values, it shall use Crypto Driver Object 0 to request the Random Numbers.
⌋()

It should be noted, that this can lead to a conflict that must be arbitrated within the crypto driver and its objects. Especially, additional delays for a crypto service operation can be expected due to the generation of random number values.

[SWS_Crypto_00226] ⌈ A Crypto Driver Object shall only call a default random number generator of its own Crypto Driver.
⌋()

7.2.1 Normal Operation

[SWS_Crypto_00017] ⌈
"`START`" indicates a new request of a crypto primitive, and it shall cancel all previous requests of the same job.
⌋()

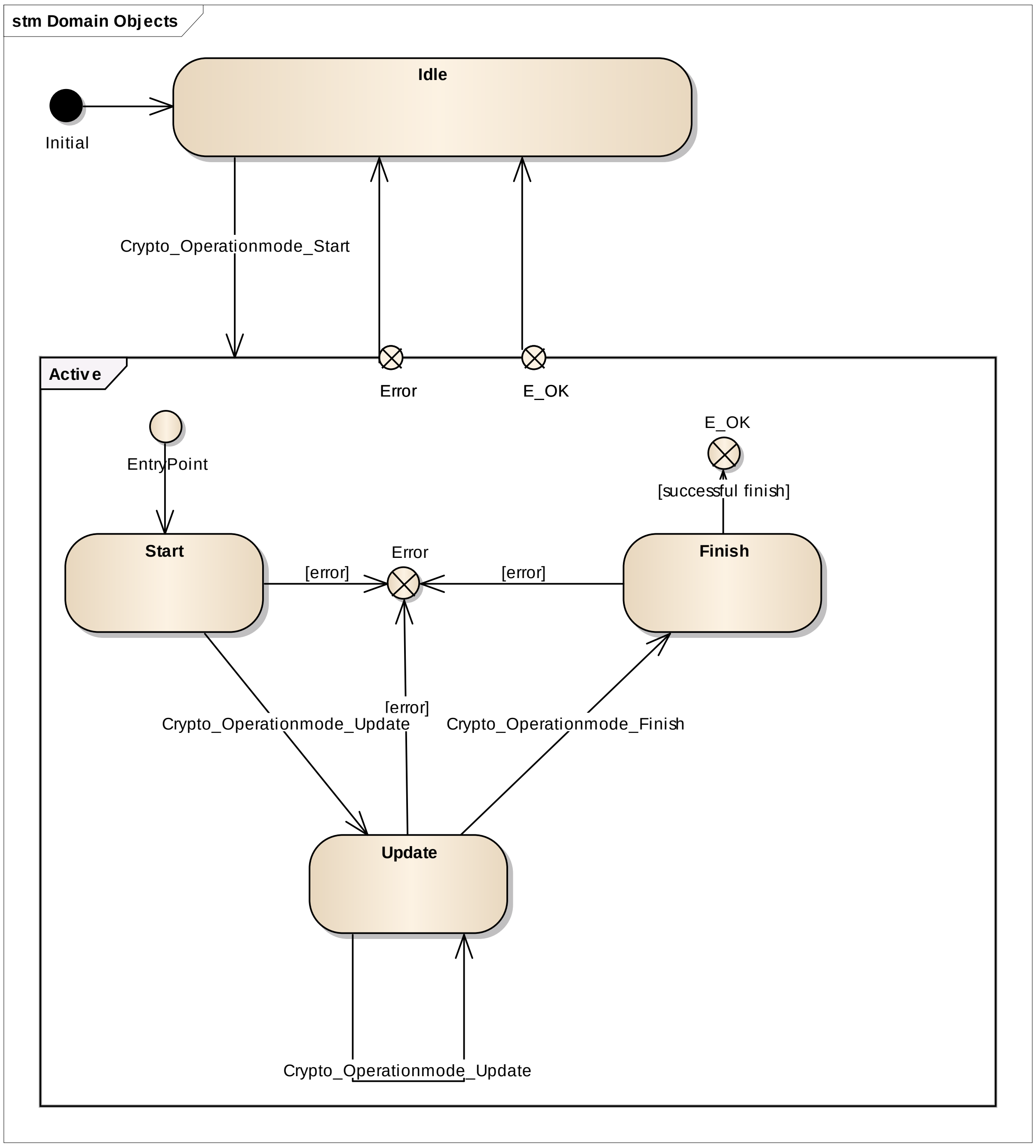
Note:
"job is being processed" means that the corresponding crypto driver object is currently and actively processing this job. When a job is not finished but the crypto driver object is not active with it (because, e.g., the operation "FINISH" is outstanding) this does not mean that this job is being processed.

Note:
To unite a single call function and a streaming approach for the crypto services, there is one interface `Crypto_ProcessJob()` with a service operation parameter (embedded in job structure parameter). This service operation is a flag field, that indicates the operation modes "`START`", "`UPDATE`" or "`FINISH`". It declares explicitly which operation will be performed.
If the "`UPDATE`" flag is set, the crypto primitive expects input data. "`FINISH`" indicates, that after this function call, all data are fed completely and the crypto primitive can finalize the calculations.
These operations can be combined to execute multiple operations at once. Then, the operations are performed in the order "`START`", "`UPDATE`", "`FINISH`".

The coherent single call approach could improve the performance due to less overhead. Instead of calling the explicit API multiple times, only one call is necessary. This approach is intended to be used with small data input, which demand fast processing.

The diagram in SWS_Crypto_00018 shows the state machine of a job of this design without considering the transitions because of errors.

[SWS_Crypto_00018]
⌈



⌋()

[SWS_Crypto_00019] ⌈ After initialization the crypto driver is in “idle” state.
⌋()

[SWS_Crypto_00020] ⌈ If `Crypto_ProcessJob()` is called while in “Idle” or “Active” state and with the operation mode “`START`”, the previous request shall be cancelled. That means, that all previously buffered data for this job shall be reset, and the job shall switch to “Active” state and process the new one.
⌋()

Note:

Resetting a job using "`START`" is only possible when the job is not actively being processed.

[SWS_Crypto_00118] ⌈ 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 with `E_NOT_OK`.
⌋()

Note:
If `Crypto_ProcessJob()` is called while in "Active" state and with the operation mode "`UPDATE`", the crypto primitive is fed with input data. In terms of streaming of arbitrary amounts of user data multiple calls with operation mode "`UPDATE`" is used, to feed more input data to the previously ones. In the "Update" state, there are usually also calculations of intermediate results of cryptographic primitives. Actually, in some cases (e.g. AES Encryption in CBC mode) there is also the generation of output data. While operating with the streaming approach ("Start", "Update", "Finish") the Crypto Driver Object is waiting for further input ("Update") until the "Finish" state has been reached. No other job could be processed meanwhile.

[SWS_Crypto_00023] ⌈ If `Crypto_ProcessJob()` is called while in "Active" state and with the operation mode "`FINISH`", the cryptographic calculations shall be finalized. Additional data (i.e. the MAC to be tested on a MAC verification service) shall be available at this point to process this job successfully. The results of the calculations shall be stored in the output buffers. At end of the processing the Crypto Driver shall switch to "Idle" state.
⌋()

To process a crypto service with a single call with `Crypto_ProcessJob()` the operation mode "`CRYPTO_OPERATIONMODE_SINGLECALL`" is a disjunction (bitwise OR) of the 3 modes "`START`", "`UPDATE`" and "`FINISH`".

[SWS_Crypto_00025] ⌈ If an internal error occurs, the corresponding job state shall be set to "Idle" and all input data and intermediate results shall be discarded.
⌋()

[SWS_Crypto_00119] ⌈ If an internal error occurs while processing an asynchronous job, the corresponding job state shall be set to "Idle" and all input data and intermediate results shall be discarded. Further, the callback notification shall be called with an appropriate error code.
⌋()

7.2.2 Functional Requirements

Note: The information whether the job shall be processed synchronously or asynchronously is part of the `Crypto_JobType`.

7.2.2.1 Synchronous Job Processing

[SWS_Crypto_00026] ⌈ When the synchronous job processing is used, the corresponding interface functions shall compute the result synchronously within the context of this function call.
⌋()

[SWS_Crypto_00199] ⌈ If the Crypto Driver has a queue and if a synchronous job is issued and the priority is greater than the highest priority available in the queue, the Crypto Driver shall disable processing new jobs from the queue until the next call of the main function has finished that follows after completion of the currently processed job.
⌋()

Note: Channels may hold jobs of both asynchronous and synchronous processing type. If so, a synchronous job might not be accepted for processing although its job's priority is higher than those of all asynchronous jobs.

7.2.2.2 Asynchronous Job Processing

[SWS_Crypto_00027] ⌈ If the asynchronous job processing is used, the interface functions shall only hand over the necessary information to the primitive. The actual computation may be kicked-off by the main function.
⌋()

[SWS_Crypto_00028] ⌈ For each asynchronous request the Crypto Driver shall notify CRYIF about the completion of the job by calling the CRYIF_CallbackNotification function passing on the job information and the result of cryptographic operation.
⌋()

7.2.3 Design Notes

The Crypto Driver provides two services: (1) the crypto services itself and (2) key management.

7.2.3.1 Priority-dependent Job Queue

[SWS_Crypto_00029] ⌈ Optionally, every Crypto Driver Object shall be able to line up jobs into a queue to process them one after the other.
⌋()

[SWS_Crypto_00179] ⌈ The Crypto Driver Object shall disable queueing when the size of the crypto driver queue is set to 0.
⌋()

[SWS_Crypto_00030] ⌈ The queue shall sort the jobs according to the configured jobs' priority.
⌋()

The higher the job priority value, the higher the job's priority.

[SWS_Crypto_00031] ⌈ If `Crypto_ProcessJob()` is called, when the queue is empty and the Crypto Driver Object is not busy the Job shall switch to the state 'active' and execute the crypto primitive.
⌋()

[SWS_Crypto_00032] ⌈ If `Crypto_ProcessJob()` is called and the queue is full, the function shall return with `CRYPTO_E_BUSY`.
⌋()

Note:
It has to be ensured, that the asynchronous jobs are processed fast enough to avoid that the synchronous job has to wait for a long time.
It is also recommended to use CRYPTO_OPERATIONMODE_SINGLECALL for the asynchronous jobs.

Note:
A Crypto Driver Object can handle different jobs with synchronous and asynchronous job processing at the same time. However, synchronous job processing and job-queuing might not be useful. So, if synchronous job processing is chosen, the job queue will not be used, and a job will only be processed, when the Crypto Driver Object is not busy.

[SWS_Crypto_00121] ⌈ If `Crypto_ProcessJob()` is called and the Job is in "ACTIVE" state, the `Crypto_ProcessJob()` shall check if the requested job matches the current job in the Crypto Driver Object and if yes, bypass it from queueing.
⌋()

This implicates that only jobs with operation mode „START" shall be queued. If a job with operation mode "START" has been finished, the Crypto Driver Object is waiting for input. The callback function indicates the callee that an "UPDATE" or "FINISH" call shall be performed.

[SWS_Crypto_00033] ⌈ If `Crypto_ProcessJob()` is called with asynchronous job processing and the queue is not full, but the Crypto Driver Object is busy and if the job has the operation mode "START", the Crypto Driver Object shall put the job into the queue and return `E_OK.`
⌋()

[SWS_Crypto_00034] ⌈ If `Crypto_ProcessJob()` is called with synchronous job processing and the queue is not full, but the Crypto Driver Object is busy, the Crypto Driver Object shall not queue the job and return `CRYPTO_E_BUSY`. No job shall be put in any queue.
⌋()

7.2.4 Key Management

A key consists of one or more key elements.
Examples of key elements are the key material itself, an initialization vector, a seed state for random number generation, or the proof of the SHE standard.

Each key element has a defined access right for read or write access. The access right itself is defined by enumerations with an assigned value (see [ECUC_Crypto_00024] or [ECUC_Crypto_00027]) in the configuration of a CryptoKeyElement. The integer values of these enumerations are chosen in a way, that the assignment of an access right to a key element also deduces further right accesses. The lower the value, the higher the access right. This allows an easy compare of the assigned value to a required right access by numbers.

Example:

If a key element has access rights of CRYPTO_RA_INTERNAL_COPY=2, the right permission also applies to CRYPTO_RA_ENCRYPTED=1 and CRYPTO_RA_ALLOWED=0, because both rights of the assigned enumeration values are lower and therefore are lower in the ranking of the access rights.

[SWS_Crypto_00219] ⌈ Access rights shall be taken into account when direct access to key elements are required. This applies for read and write access.
⌋ ()

Keys or key elements can be copied using the key copy interface functions. This allows, for example, to store keys in one driver channel that provides secure key storage and to distribute them to other driver implementations that have no such capabilities. However, it is important that access rights to key elements are guarded within the copy operation to avoid unintended disclosure of stored keys.

[SWS_Crypto_00220] ⌈ A source key element must have higher or the same rights than the destination key element when key elements are copied using the key copying interfaces. A source key must have at least the rights CRYPTO_RA_INTERNAL_COPY or lower to get copied.
⌋ ()

Info:
An internal copy operation can only be performed if the key elements are located in the same crypto driver object.

[SWS_Crypto_00221] ⌈ Jobs shall use the assigned key elements without guarding the key access rights with the following exceptions:

- If a key element is used for input using the input re-direction, the key element must have access rights CRYPTO_RA_INTERNAL_COPY or lower. If input re-direction is used for CryptoPrimitiveService ENCRYPT/DECRYPT or AEAD_ENCRYPT/AEAD_DECRYPT, the access rights must be set to RA_ENCRYPTED or lower.[1)]

- If a key element is used for output re-direction, the key element must have access rights CRYPTO_WA_INTERNAL_COPY or lower.
- Any key element that is used to generate keys using Key Exchange operation shall have access rights of at least CRYPTO_RA_INTERNAL_COPY or lower.
- For Key Derivation, the source key shall have access rights of at least CRYPTO_RA_INTERNAL_COPY or lower. The destination key shall have at least the access right of its source key or lower[2)].

⌋ ()

[1)] Rationale: This is to avoid using a key element as input and (another for) output for encrypt and successively decrypt that key which would allow to extract the secret key in plaintext at the end.

[2)] This is to avoid to deduce the source key when using weak key derivation algorithms.

[SWS_Crypto_00037] ⌈ The index of the different key elements from the different crypto services are defined as in imported types table SWS_Csm_01022.
⌋()

[SWS_Crypto_00038] ⌈ A key has a state which is either “valid” or “invalid”.
⌋()

[SWS_Crypto_00039] ⌈ If a key is in the state “invalid”, crypto services which make use of that key, shall return with `CRYPTO_E_KEY_NOT_VALID`.
⌋()

Note: In case of error observed during Crypto_KeyElementSet() API call, the status of the key element needs to be considered as unknown.

If a key (or key element) is currently in use by a crypto service, the state of the key has to be “valid”. When the `KeyElementSet()` is called, the key state is set to “invalid”. So, the job which is currently running will probably work with an inconsistent key. It is up to the application to only change key, if currently no primitive works with that key (element).

Note: The mapping of keys and key elements to SHE hardware functionality is possible without being subject to any restrictions except that read and write access through key management interfaces must be performed in encrypted form due to hardware constrains. To provide an environment for legacy software the single key used by the hardware can be placed in a key element referenced by several keys. Every key has also a unique reference to a key element containing an identifier. The driver implemented according to this specification can hence wrap existing SHE hard- and software and pass the data from the key elements to the existing SHE driver. In this use case one key element could contain a counter that could be read and written by the driver as well as the application. This counter could be used to

detect if the key was overwritten. The loading of a key into the actual hardware key slot could be done immediately before the key is used, which would result in a combined loading and processing of the key, as well as a separate operation following the writing of a key into a key element. This would result in separate operations for loading and processing the key.
If a new driver is to be implemented, it would also be possible to configure keys with completely independent key elements. These independent keys can be stored in RAM and passed to the hardware key slot only when required for an operation. The number of keys stored in the driver can be independent of (and much larger than) the number of hardware key slots. This requires, of course, a handling and storing of keys in software with all potential drawbacks.
Storing keys permanently is done by calling Crypto_KeySetValid with the configuration parameter CryptoKeyElementPersist set. As in most cases writing operation takes some time it is recommended to store key permanently using the CRYPTO_KEYSETVALID job interface.

Different key types can have compatible key elements. In this case the keyElementId has the same value. Key elements with the same keyElementId may be regarded as compatible. This way, the same key can be used for different services.
The key material therefore shall always have the keyElementId 1.

Example is the generation of a key with the Key Management Interface and usage of the same key in a primitive like MacGenerate afterwards.

A key element may not be fully written. In some cases, the size of data to be stored in the key element can vary, e.g. asymmetric key elements. The Crypto Driver shall store the actually written size of data for internal usage and for exporting the element with Crypto_KeyElementGet(). If the key element shall allow to be not fully read or written can be configured with the parameter CryptoKeyElementAllowPartialAccess in the CryptoKeyElement container.

7.2.5 Key Formats

The key element with ID 1 is a particular key that provides the cryptographic key for the related cryptographic primitive.
In IETF RFC, keys are described in a specific format to provide a common and compact way to transport them between entities.
To support this standard also within AUTOSAR, the following chapter describes key formats that are supported by AUTOSAR.
This allows to provide a given key in such a format to the crypto driver, e.g. from parts of a certificate without the needs for an application to analyse the structure and to construct that data.
By supporting the IETF RFC within the crypto driver, the key material can be provided as is.

If supported, the extraction of the required key elements, such as private or public key fields, is accomplished in the driver.

It is unlikely to compose such formats back to the application by the crypto driver. Thus, such a key element might only be temporarily stored and may not persisted in "composed" format (M1M2M3), e.g. to retrieve that after reset.

[SWS_Crypto_00184]⌈ Asymmetric key material with identification is specified in accordance to RFC5958 in ASN.1 format. The key material with the format specifier CRYPTO_KE_FORMAT_BIN_IDENT_PRIVATEKEY_ PKCS8 needs to follow this format specification:

```
  OneAsymmetricKey ::= SEQUENCE {
    version              Version,
    KeyAlgorithm         KeyAlgorithmIdentifier,
    keyMaterial          KeyMaterial,
    attributes*          [0] Attributes OPTIONAL,
    ...,
    [[2: publicKey*      [1] PublicKey OPTIONAL ]],
    ...
  }
```

* The optional values for key attributes and the PublicKey are currently not used within the crypto driver and is listed here just for compatibility reason to RFC5958. A driver shall tolerate the provision of this information but doesn’t need to evaluate its contents.

The elements have the following meaning:

```
Version ::= INTEGER { v1(0), v2(1) } (v1, ..., v2)

 KeyAlgorithmIdentifier ::= AlgorithmIdentifier
                              { PUBLIC-KEY,
                              { PrivateKeyAlgorithms } }

KeyMaterial ::= OCTET STRING
                 -- Content varies based on the type of the key and is specified by its
AlgorithmIdentifier.
                  -- The KeyAlgorithmIdentifier defines which format specifier for KeyMaterial
shall be applied.
```

AlgorithmIdentifier: A value that identifies the format by its object identifier (OID).

⌋(SRS_CryptoStack_00008)

[SWS_Crypto_00254]⌈ Public key material with identification is specified in accordance to RFC5280, section 4.1. The key material with the format specifier CRYPTO_KE_FORMAT_BIN_IDENT_PUBLICKEY needs to follow this format specification:

```
SubjectPublicKeyInfo ::= SEQUENCE
                          { algorithm AlgorithmIdentifier,
                            subjectPublicKey BIT STRING }
```

The elements have the following meaning:

AlgorithmIdentifier:

A value that identifies the format by its object identifier (OID).
subjectPublicKey:
The public key itself.
⌋(SRS_CryptoStack_00008)

7.2.5.1 Definition of RSA Key Material

[SWS_Crypto_00185]⌈ For CRYPTO_KE_FORMAT_BIN_ RSA_PRIVATEKEY the parameter 'KeyMaterial OCTET STRING' for RSA private keys is defined according to IETF RFC8017 and has the following contents:

```
KeyMaterial ::= RSAPrivateKey

  RSAPrivateKey ::= SEQUENCE {
    version Version,
    modulus INTEGER, -- n
    publicExponent INTEGER, -- e
    privateExponent INTEGER, -- d
    prime1 INTEGER, -- p
    prime2 INTEGER, -- q
    exponent1 INTEGER, -- d mod (p-1)
    exponent2 INTEGER, -- d mod (q-1)
    coefficient INTEGER -- (inverse of q) mod p }

  Version ::= INTEGER { two-prime(0), multi(1) }
```

The fields of type RSAPrivateKey have the following meanings:

- version is the version number, for compatibility with future revisions of this document. It shall be 0 for this version of the document.
- modulus is the modulus n.
- publicExponent is the public exponent e.
- privateExponent is the private exponent d.
- prime1 is the prime factor p of n.
- prime2 is the prime factor q of n.
- exponent1 is d mod (p-1).
- exponent2 is d mod (q-1).
- coefficient is the Chinese Remainder Theorem coefficient (inverse of q) mod p.

⌋(SRS_CryptoStack_00008)

Note:
The values for prime1, prime2, exponent1, exponent2 and coefficient are optional. If prime1 is not provided, none of the following values in the list shall be provided. Otherwise, the key shall be rejected.

[SWS_Crypto_00186]⌈ The RSA public key in the format CRYPTO_KE_FORMAT_BIN _RSA_PUBLICKEY is provided as follows:

```
    RSAPublicKey ::= SEQUENCE {
        modulus INTEGER, -- n
    publicExponent INTEGER, -- e
  }
```

The fields of type RSAPublicKey have the following meanings:

- modulus is the modulus n.
- publicExponent is the public exponent e.

⌋(SRS_CryptoStack_00008)

[SWS_Crypto_00187]⌈ For the RSA public key in the format CRYPTO_KE_FORMAT_BIN_IDENT_PUBLICKEY the “subjectPublicKey BIT STRING” is defined as “RSAPublicKey”.

Explanation:
Considering RFC5280, section 4.1, the SubjectPublicKeyInfo follows directly the definition described above. Thus, a key type of CRYPTO_KE_FORMAT_BIN_IDENT_PUBLICKEY matches SubjectPublicKeyInfo and CRYPTO_KE_FORMAT_BIN _RSA_PUBLICKEY matches the subjectPublicKey in this definition.

⌋(SRS_CryptoStack_00008)

[SWS_Crypto_00188]⌈ The algorithm identifier for RSA keys shall have the value 1.2.840.113549.1.1.1. This corresponds to the ASN.1 coded OID value “2A 86 48 86 F7 0D 01 01 01”. This OID shall be provided whenever an AlgorithmIdentifier for RSA is required. In other words, when a key has the format CRYPTO_KE_FORMAT_BIN_IDENT_PRIVATEKEY_ PKCS8 or CRYPTO_KE_FORMAT_BIN_IDENT_PUBLICKEY and is used for RSA, the AlgorithmIdentifier must have this value.
Note: In some cases, a NULL value is followed directly to the OID. So, a value that follows directly after this OID in the same sequence is optional and should be tolerated.

⌋(SRS_CryptoStack_00008)

7.2.5.2 Definition of ECC Key Material

[SWS_Crypto_00189]⌈ Due to a lack of clear and efficient standard definition for ECC keys, key material for ECC is defined as binary information in the format definition of CRYPTO_KE_FORMAT_BIN_OCTET. The length of data depends on the assigned curve operation.

⌋(SRS_CryptoStack_00008)

[SWS_Crypto_00190]⌈ 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 little endian format.

The number of bytes for the key depends on the implementation of the curve.

Examples:
NIST curve P(256) public key = X(32) | Y(32)
NIST curve P(192) public key = X(24) | Y(24)
⌋(SRS_CryptoStack_00008)

[SWS_Crypto_00191]⌈ 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 Y | Scalar.

The points and the scalar are stored in little endian format.

Example:
Brainpool curve P(256) = X(32) | Y(32) | SCALAR(32)
⌋(SRS_CryptoStack_00008)

[SWS_Crypto_00192]⌈ 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:
ED25519 Public Key = X(32).
⌋(SRS_CryptoStack_00008)

[SWS_Crypto_00193]⌈ The private key information for ED25519 contains a random constant and the point X on the curve:
ED25519 Private Key = Seed K | Point X

The point and the seed are stored in little endian format.

Example:
ED25519 Private Key = Seed K(32) | X(32).
⌋(SRS_CryptoStack_00008)

7.2.5.3 Definition of SHE key material

[SWS_Crypto_00255]⌈ If a crypto key element is used as a secret key of a SHE hardware, then the configuration of CryptoKeyFormat/CryptoKeyElementFormat shall be set to CRYPTO_KE_FORMAT_BIN_SHEKEY. This indicates that the key element shall be presented in M1M2M3 format as specified in [7]. The function `Crypto_KeyElementSet()` is used to load the key into the SHE hardware (not directly on a call to this function but rather through key-set-valid operation).

With `Crypto_KeyElementSet()`, the parameter 'keyElementId' shall be set to 1. This can be taken from e.g. CRYPTO_KE_MAC_KEY or CRYPTO_KE_CIPHER_KEY (see SWS_Csm_01022).
The SHE hardware provides M4M5 as the proof of the correct key processing and to confirm the operation. To extract this key information, an additional key element is needed. To read the proof after the key-set-valid operation, the function `Crypto_KeyElementGet()` with key element ID value '2' shall be used, e.g. with pre-defined macros CRYPTO_KE_MAC_PROOF or CRYPTO_KE_CIPHER_PROOF. The proof will only be available directly after a successful key-element-set followed by key-set-valid operation, e.g. within an ignition cycle. This means, the proof must not be stored, e.g. in NV-RAM, to provide this information later on.⌋()

7.2.6 Key Storage in NVM

To allow a crypto driver to persist key elements in non-volatile memory, access to NVM is required. For this, the configuration of the crypto driver needs a reference to one or more NVM blocks. Keys with key elements that shall be persisted are to be assigned to these blocks. The elements that are to be persisted will then be written by the driver on validation and values are retrieved during initialization of the driver. The principal relation to configuration items is shown in Figure 7-2.

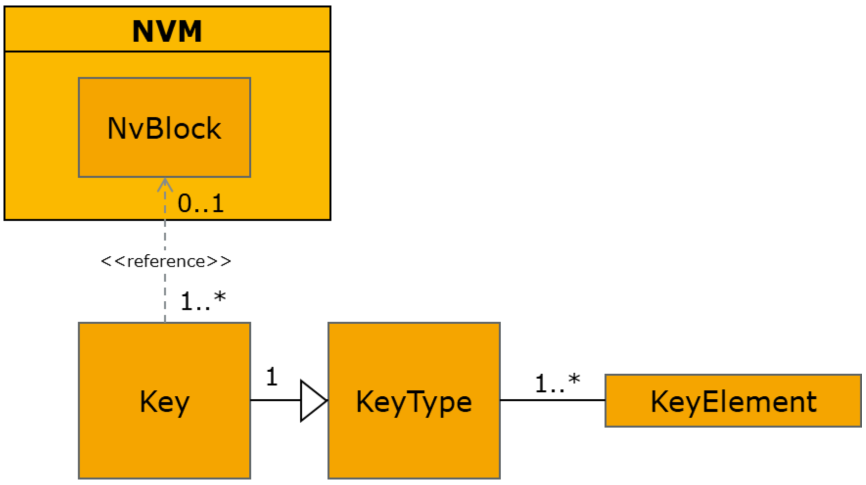


Figure 7-2: NvBlock configuration for key persistence

<u>**Important note**</u>:

The crypto driver only deals with the functional operation to store data to and read data from an NVM block. Any error handling like defects of NVM blocks must be captured and dealt with in the application.

7.2.6.1 Writing keys to NVM

[**SWS_Crypto_00242**] ⌈ The storage of keys resp. their elements shall be initiated only when KEYSETVALID” is started, either by a call to `Crypto_KeySetValid()`

or to `Crypto_ProcessJob()` with primitive service `CRYPTO_KEYSETVALID` or `CRYPTO_KEYSETINVALID` set. Thus, key element storage in NVM shall not be initiated on a call to `Crypto_KeyElementSet()` or similar operation. The storage operation shall only take place if the validation was successfully finished and the operation returns `E_OK` (either directly by a synchronous call or asynchronously through the callback).
⌋(SRS_CryptoStack_00061)

Rationale:

Especially when key elements of a key have interdependencies to one or more other key element(s) of the same key, writing the key element immediately when it is set has the risk, that the key itself can be inconsistent. In addition, the explicit initiation of the operation with KEYSETVALID allows the crypto driver to check the consistency and interdependency of all elements and storing will be denied if the check fails. Also, updating several elements of a key will result in a single write operation to NVM and not multiple times for each key element, which can reduce the total number of write operations to NVM.

[**SWS_Crypto_00243**] ⌈ On KEYSETVALID operation, writing the data block to NVM will be initiated by a call of the crypto driver to `NvM_SetRamBlockStatus()`. This is the typical operation when the configuration `CryptoNvBlock/CryptoNvBlockProcessing` is set to `DEFERRED`.
If `CryptoNvBlock/CryptoNvBlockProcessing` is set to `IMMEDIATE`, the Crypto Driver will, besides the call to `NvM_SetRamBlockStatus()`, also call `NvM_WriteBlock()` immediately afterwards. This will trigger an immediate write of the NV RAM Block to non-volatile memory.
⌋(SRS_CryptoStack_00008, SRS_CryptoStack_00118, SRS_CryptoStack_00120)

Note:

For proper operation it is recommended to set the parameter `NvMBlockUseSetRamBlockStatus` and `NvMSelectBlockForReadAll` of the NVM to `TRUE`. The value `NvMSelectBlockForWriteAll` shall be set to `TRUE` if `CryptoNvBlockProcessing` is set to `DEFERRED`.

[**SWS_Crypto_00244**] ⌈ If a call to an NVM service (`NvM_SetRamBlockStatus()` or `NvM_WriteBlock()`) returns with `E_NOT_OK`, or when the callback function `Crypto_<vi>_<ai>_NvBlock_Callback_<NvBlock>` (see [**SWS_Crypto_91026**]) indicates with `Nvm_RequestResultType` that the block write operation has failed, the runtime error `CRYPTO_E_RE_NVM_ACCESS_FAILED` shall be reported to the DET. In addition, the service call shall be requested again on the next execution of `Crypto_MainFunction()`. The number of retries can be limited by the parameter `CryptoNvBlock/CryptoNvBlockFailedRetries`.
⌋(SRS_CryptoStack_00008)

[**SWS_Crypto_00245**] ⌈ If a key is currently updated in NV RAM Block and has not yet been written by NVM, then this shall be reflected in the key status with `CRYPTO_KEYSTATUS_UPDATE_IN_PROGRESS` through the `Crypto_KeyGetStatus()` interface.
⌋(SRS_CryptoStack_00008)

[**SWS_Crypto_00246**] ⌈ For each configured `CryptoNvBlock` the Crypto driver shall provide a separate set of `Crypto_<vi>_<ai>_NvBlock_xxx_<NvBlock>()` functions (xxx refers to Init, ReadFrom, WriteTo and Callback) (see chapter Callback notification). The short name of the `CryptoNvBlock` container replaces the tag <NvBlock>. The tags <vi> and <ai> are replaced as described in [**SWS_Crypto_00012**]. These functions shall be set as callbacks in the respective fields of the NvmBlockDescriptor.
⌋(SRS_CryptoStack_00008, SRS_CryptoStack_00118)

7.2.6.2 Reading Keys from NVM

[**SWS_Crypto_00247**] ⌈ On initialization of the crypto driver, the callback function `Crypto_<vi>_<ai>_NvBlock_ReadFrom_<NvBlock>()` shall be called to retrieve the previously persisted key elements for the associated Keys.
⌋(SRS_CryptoStack_00008)

Note:

The format of the data within the NvBlock are local to the driver. At least, the driver needs to make sure that key elements and keys can be uniquely assigned within the NV RAM Block. Also, versioning of NV RAM Block data for future updates of the format shall be considered. Data consistency of the block may be considered.

[**SWS_Crypto_00248**] ⌈ On initialization, if an `NvBlock` could be read successfully and the internal consistency check of the `NvBlock` data was successful, all key elements of the referencing keys to this block shall be set with the value from the NV RAM Block data.
⌋(SRS_CryptoStack_00061, SRS_CryptoStack_00117)

[**SWS_Crypto_00249**] ⌈
On initialization, when reading an `NvBlock` for keys and the block is available but corrupted (driver internal data consistency check failed), all related keys shall be set to invalid. The length value of all associated key elements, where `CryptoKeyElementPersist` is set to `TRUE`, shall be set to 0 even if `CryptoKeyElementInitValue` is set for this key.
⌋(SRS_CryptoStack_00061, SRS_CryptoStack_00117)

Rationale:

This shall prevent an attack where `NvBlocks` are manipulated to force the driver back to an initial state.

[**SWS_Crypto_00250**] ⌈ If an `NvBlock` has no data, respectively the `NvBlock` is empty, key elements of all referenced keys, where `CryptoKeyElementInitValue` is set, shall use this configured init value (as if no key storage is configured for the key).
⌋(SRS_CryptoStack_00061, SRS_CryptoStack_00117)

[**SWS_Crypto_00251**] ⌈ After initialization of all key elements at startup (either with default or with persisted data), the crypto driver shall check all keys and its key elements if they are valid (at least one key element has a value). If so, the corresponding key shall be set to VALID state.
⌋(SRS_CryptoStack_00008, SRS_CryptoStack_00061, SRS_CryptoStack_00117)

7.2.7 Crypto Profiles

7.2.7.1 Overview of custom service and function profiles

The Crypto Driver can support vendor specific custom services and custom synchronous API functions, triggered by Csm_CustomService and Csm_CustomSync (Crypto_CustomSync). In order to align the realization by different Crypto vendors for a particular use case, a mapping of parameters is required. This mapping is defined by the definition of profiles for particular use case.

[SWS_Crypto_00258]⌈ A Crypto profile shall define the mapping for the following parameters for Csm_CustomService:

```
uint32 JobId
Crypto_OperationModeType mode
uint32 targetKeyId
const uint8* inputPtr
uint32 inputLength
const uint8 * secondaryInputPtr
uint32 secondaryInputLength
const uint8* tertiaryInputPtr
uint32 tertiaryInputLength
uint8* outputPtr
uint32* outputLengthPtr
uint8* secondaryOutputPtr
uint32* secondaryOutputLengthPtr
Crypto_VerifyResultType* verifyPtr⌋()
```

[SWS_Crypto_00259]⌈ A Crypto profile shall define the mapping for the following parameters for Csm_CustomSync:

`uint32 dispatchId` **** - unique id to identify the request

```
uint32 keyId
uint32 keyElementId
uint32 targetKeyId
uint32 targetKeyElementId
const uint8* inputPtr
uint32 inputLength
uint8* outputPtr
uint32* outputLengthPtr
uint8* secondaryOutputPtr
uint32* secondaryOutputLengthPtr
```

**** dispatchId shall be set according to the rule:

```
uint32 dispacthId = (uint32)(
      ((ServiceInfoType                        << 24) & 0xFF000000) ||
      ((CryptoPrimitiveAlgorithmFamilyCustomId << 16) & 0x00FF0000) ||
      ((Crypto_AlgorithmModeType               <<  8) & 0x0000FF00) ||
      ((ServiceId                                   ) & 0x000000FF))⌋()
```

7.2.7.2 Custom service and function profile 1 (KeyM and certificate management)

This profile defines how to use a custom service with the certificate management services specified for the KeyM. Using this profile enables the KeyM to forward the actual certificate processing to the Crypto Driver and its associated HSM. The Crypto profile provides a consistent mapping of the parameters from the KeyM services and functions to the Csm job configured for the custom service and to the custom synchronous API function Crypto_CustomSync.
The ServiceInfoType is defined as CUSTOM_SERVICE with value 0x15.
The Crypto_AlgorithmFamilyType is defined as CRYPTO_ALGOFAM_KEYM with value 0x80 and need to be configured via [ECUC_Crypto_00047] CryptoPrimitiveAlgorithmFamilyCustomId.
The Crypto_AlgorithmModeType is defined as CRYPTO_ALGOMODE_NOT_SET with value 0x00.
The KeyM_ServiceCertificate (ServiceId: 0x09) can be mapped by the KeyM to call KeyM_ServiceCertificateByCertId (ServiceId: 0x13).
In case the data type mapped to a parameter of type unit8* have a bigger size (e.g. uint16) the MSB should be the first byte.

[SWS_Crypto_00260]⌈ Each Crypto profile shall define the mapping of the parameters of Crypto_CustomSync to the using KeyM service/function.⌋()

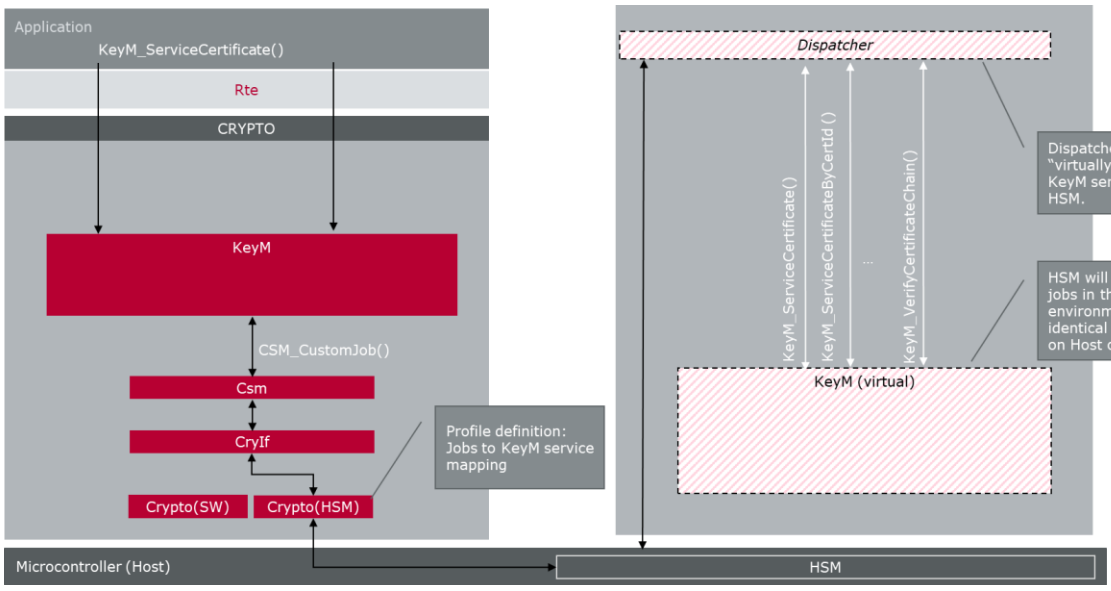
Mapping table for Csm_CustomService:

service Id	0x13	0x0C	0x0D	0x0E
processing type	async	async	async	async
Return type	Std_ReturnType	Std_ReturnType	Std_ReturnType	Std_ReturnType

Csm_CustomService	**KeyM_Service-Certificate-ByCertId**	**KeyM_VerifyCertificates**	**KeyM_VerifyCertificate**	**KeyM_Verify-CertificateChain**
uint32 JobId	JobId of the referenced custom CSM job from the given certificate configuration			
Crypto_Operation-ModeType mode	empty*	empty*	empty*	empty*
uint32 targetKeyId	uint8 SvcId**	uint8 SvcId**	uint8 SvcId**	uint8 SvcId**
const uint8* inputPtr	uint8 Service***	uint16 CertUpperId	empty*	uint8 NumberOfCertificates
uint32 inputLength	1	2	0	1
const uint8 * secondaryInputPtr	const unit8 * RequestData	empty*	empty*	KeyM_CertDataType[] certChainData
uint32 secondaryInput-Length	uint32 RequestDataLength	empty*	empty*	NumberOfCertificates * sizeof(KeyM_CertDataType)
const uint8* tertiaryInputPtr	empty*	empty*	empty*	empty*
uint32 tertiaryInputLength	empty*	empty*	empty*	empty*
uint8* outputPtr	uint8 ResponseData	empty*	empty*	empty*
uint32* outputLengthPtr	unit32 ResponseDataLength	empty*	empty*	empty*
uint8* secondaryOutputPtr	<specific return value for that custom service>			
uint32* secondaryOutput-LengthPtr	< sizeof(Std_ReturnType)>			
Crypto_VerifyResultType* verifyPtr	empty*			

* “empty”: it’s up to the vendor how to set the value of the parameter, but will not be used
** “SvcId”: is Service ID of the original KeyM API function; casted to uint32
*** “Service” is the parameter KeyM_ServiceCertificateType Service from the caller

The picture below shows the usage of the crypto driver (on HSM). Service requests from the application go to the standardized KeyM interfaces and if a crypto profile for custom processing defines the handling on HSM the crypto driver transfers the execution onto the HSM.


Application
KeyM_ServiceCertificate()
Rte
CRYPTO
KeyM
CSM_CustomJob()
Csm
CryIf
Crypto(SW)
Crypto(HSM)
Profile definition: Jobs to KeyM service mapping
Dispatcher
KeyM_ServiceCertificate()
KeyM_ServiceCertificateByCertId ()
...
KeyM_VerifyCertificateChain()
Dispatcher would "virtually" call the KeyM service on HSM.
HSM will execute the jobs in the secure environment identical to a KeyM on Host controller
KeyM (virtual)
Microcontroller (Host)
HSM

Mapping table for Csm_CustomSync:

service Id	0x0F	0x12	0x1B	0x1C	0x0A	0x0B
processing type	sync					
Return type	Std_ReturnType					
Csm_CustomSync	**KeyM_CertElementGet**	**KeyM_CertGetStatus**	**KeyM_CertificateElementGetByIndex**	**KeyM_CertificateElementGetCount**	**KeyM_SetCertificate**	**KeyM_GetCertificate**
uint32 dispatchId****	0x1580000F	0x15800012	0x1580001B	0x1580001C	0x1580000A	0x1580000B
uint32 keyId	keyId of the referenced key used in the custom CSM job from the given certificate configuration					
uint32 keyElementId	mapped keyId from CertElementId	empty*	mapped keyId from CertElementId	mapped keyId from CertElementId	empty*	empty*
uint32 targetKeyId	empty*	empty*	uint32 index	empty*	empty*	empty*
uint32 targetKeyElementId	empty*	empty*	empty*	empty*	empty*	empty*

const uint8* inputPtr	empty*	empty*	empty*	empty*		KeyM_CertDataType* CertificateDataPtr
uint32 inputLength	empty*	empty*	empty*	empty*	empty*	empty*
uint8* outputPtr	uint8* CertElementData	KeyM_CertificateStatusType* Status	uint8* CertElementDataPtr	uint16* CountPtr	empty*	empty*
uint32* outputLengthPtr	uint32* CertElementDataLength	sizeof(KeyM_CertificateStatusType)	uint32* CertElementDataLengthPtr	empty*	empty*	empty*
uint8* secondaryOutputPtr	Std_ReturnType					
uint32* secondaryOutputLengthPtr	sizeof(Std_ReturnType)					

**** dispatchId: see [SWS_Crypto_00259]

[SWS_Crypto_00261]⌈ Certificate element names and Ids:

Crypto Service	**key element Name**	**key element ID**	**Mandatory**
Certificate Parsing			
Certificate	CRYPTO_KE_CERTIFICATE_DATA	0	x
Format	CRYPTO_KE_CERTIFICATE_PARSING_FORMAT	18	
Version	CRYPTO_KE_CERTIFICATE_VERSION	20	
Serial Number	CRYPTO_KE_CERTIFICATE_SERIALNUMBER	21	
Signature Algorithm	CRYPTO_KE_CERTIFICATE_SIGNATURE_ALGORITHM	22	
Issuer	CRYPTO_KE_CERTIFICATE_ISSUER	23	
Validity start	CRYPTO_KE_CERTIFICATE_VALIDITY_NOT_BEFORE	24	
Validity end	CRYPTO_KE_CERTIFICATE_VALIDITY_NOT_AFTER	25	
Subject	CRYPTO_KE_CERTIFICATE_SUBJECT	26	
Subject Public Key	CRYPTO_KE_CERTIFICATE_SUBJECT_PUBLIC_KEY	1	

Extensions	CRYPTO_KE_CERTIFICATE_EXTENSIONS	27	
Signature	CRYPTO_KE_CERTIFICATE_SIGNATURE	28	

⌋()

7.3 Error classification

Section 7.2 "Error Handling" of the document "General Specification of Basic Software Modules" describes the error handling of the Basic Software in detail. Above all, it constitutes a classification scheme consisting of five error types which may occur in BSW modules.

Based on this foundation, the following section specifies particular errors arranged in the respective subsections below.

7.3.1 Development Errors

[SWS_Crypto_00040]⌈

Type of error	*Related error code*	*Error value*
API request called before initialization of Crypto Driver.	CRYPTO_E_UNINIT	0x00
Initialization of Crypto Driver failed	CRYPTO_E_INIT_FAILED	0x01
API request called with invalid parameter (Nullpointer without redirection).	CRYPTO_E_PARAM_POINTER	0x02
API request called with invalid parameter (out of range).	CRYPTO_E_PARAM_HANDLE	0x04
API request called with invalid parameter (invalid value).	CRYPTO_E_PARAM_VALUE	0x05
Buffer is too small for operation	CRYPTO_E_SMALL_BUFFER	0x06

⌋(SRS_CryptoStack_00086)

7.3.2 Runtime Errors

[SWS_Crypto_00194]⌈

Type of error	*Related error code*	*Error value*
Entropy is too low	CRYPTO_E_RE_ENTROPY_EXHAUSTED	0x03
NVM access has failed	CRYPTO_E_RE_NVM_ACCESS_FAILED	0x04

⌋()

7.3.3 Transient Faults

There are no transient faults.

7.3.4 Production Errors

There are no production errors.

7.3.5 Extended Production Errors

There are no production errors.

7.4 Error detection

[SWS_Crypto_00217]⌈ The crypto Driver shall check if `job->cryptoKeyId` and, if applicable, `job->targetCryptoKeyId` are in range, before it executes a job. If the check fails, the function `Crypto_ProcessJob()` shall report `CRYPTO_E_PARAM_HANDLE` to DET and return `E_NOT_OK`.
⌋()

[SWS_Crypto_00195]⌈ If a Crypto Driver API is called and any buffer addressed during the operation is too small, then the operation shall not be performed. If development error detection for the Crypto Driver is enabled, then the API function shall report `CRYPTO_E_SMALL_BUFFER` to the DET, else return `E_NOT_OK`.
⌋()

Note:
For CRYPTO_HASH, CRYPTO_MACGENERATE and CRYPTO_RANDOMGENERATE services, truncated results are allowed (see **[SWS_Crypto_00065]**, **[SWS_Crypto_00252]**).

8 API specification

8.1 Imported types

In this chapter all types included from the following modules are listed:

[SWS_Crypto_00042]⌈

Module	***Header File***	***Imported Type***
Csm	Crypto_GeneralTypes.h	Crypto_AlgorithmFamilyType
	Crypto_GeneralTypes.h	Crypto_AlgorithmInfoType
	Crypto_GeneralTypes.h	Crypto_AlgorithmModeType
	Crypto_GeneralTypes.h	Crypto_JobPrimitiveInfoType
	Crypto_GeneralTypes.h	Crypto_JobPrimitiveInputOutputType
	Crypto_GeneralTypes.h	Crypto_JobRedirectionInfoType
	Crypto_GeneralTypes.h	Crypto_JobStateType
	Crypto_GeneralTypes.h	Crypto_JobType
	Crypto_GeneralTypes.h	Crypto_PrimitiveInfoType
	Crypto_GeneralTypes.h	Crypto_ProcessingType
	Crypto_GeneralTypes.h	Crypto_ServiceInfoType
	Rte_Csm_Type.h	Crypto_KeyStatusType
	Rte_Csm_Type.h	Crypto_OperationModeType
	Rte_Csm_Type.h	Crypto_VerifyResultType
NvM	Rte_NvM_Type.h	NvM_BlockIdType
	Rte_NvM_Type.h	NvM_BlockRequestType
	Rte_NvM_Type.h	NvM_InitBlockRequestType
	Rte_NvM_Type.h	NvM_RequestResultType
Std	Std_Types.h	Std_ReturnType
	Std_Types.h	Std_VersionInfoType

⌋()

8.2 Type Definitions

8.2.1 Extension to Std_ReturnType

[SWS_Crypto_91018]⌈

Range	CRYPTO_E_BUSY	0x02	The service request failed because the service is still busy
	CRYPTO_E_ENTROPY_EXHAUSTED	0x04	The service request failed because the entropy of the random number generator is exhausted
	CRYPTO_E_KEY_READ_FAIL	0x06	The service request failed because read access was denied
	CRYPTO_E_KEY_WRITE_FAIL	0x07	The service request failed because the writing access failed
	CRYPTO_E_KEY_NOT_AVAILABLE	0x08	The service request failed because at least one required key element is not available.
	CRYPTO_E_KEY_NOT_VALID	0x09	The service request failed because the key is invalid.
	CRYPTO_E_KEY_SIZE_MISMATCH	0x0A	The service request failed because the key size does not match.
	CRYPTO_E_JOB_CANCELED	0x0C	The service request failed because the Job has been canceled.
	CRYPTO_E_KEY_EMPTY	0x0D	The service request failed because of uninitialized source key element.
	CRYPTO_E_CUSTOM_ERROR	0x0E	Custom processing failed.
Description	--		
Available via	Crypto_GeneralTypes.h		

⌋(SRS_CryptoStack_00095)

Note:

`CRYPTO_E_KEY_NOT_AVAILABLE` is meant to indicate that required key elements of a key in the context of a specific Crypto Primitive or key management function have been programmed before but at least one of these key elements cannot be accessed at the moment (for instance it is temporarily not accessible, e.g. when the key is disabled due to debugger connection).

`CRYPTO_E_KEY_EMPTY` is meant to indicate that the referred key content has not been written so far and has no default value (For example, in SHE 1.1, the error code ERC_KEY_EMPTY would be returned then, "if the application attempts to use a key that has not been initialized".)

Furthermore, it should be noted, that the Crypto Stack API uses the key element index definition from the CSM module (see SWS_Csm_00122).

8.2.2 Crypto_ConfigType

[SWS_Crypto_91016]⌈

Name	Crypto_ConfigType	
Kind	Structure	
Elements	implementation specific	
	Type	--
	Comment	The content of the configuration data structure is implementation specific.
Description	Configuration data structure of CryIf module	
Available via	Crypto.h	

⌋(SRS_BSW_00414)

8.3 Function definitions

This is a list of functions provided for upper layer modules.

8.3.1 General API

8.3.1.1 Crypto_Init

[SWS_Crypto_91000]⌈

Service Name	Crypto_Init

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

Service ID [hex]	0x00
Sync/Async	Synchronous
Reentrancy	Non Reentrant

Parameters (in)	configPtr	Pointer to a selected configuration structure
Parameters (inout)	None	
Parameters (out)	None	
Return value	void	--
Description	Initializes the Crypto Driver.	
Available via	Crypto.h	

⌋(SRS_BSW_00101, SRS_BSW_00358, SRS_BSW_00414)

[SWS_Crypto_00215] ⌈ The Configuration pointer `configPtr` shall always have a null pointer value.
⌋ (SWS_BSW_00050)

The Configuration pointer `configPtr` is currently not used and shall therefore be set to null pointer value.

[SWS_Crypto_00198]⌈
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 corresponding key to invalid.
⌋()

Note: After initialization of the Crypto Driver and before the application starts, the application should consider to check the state of the configured keys and to implement an appropriate handling if the key's state is invalid.

[SWS_Crypto_00045] ⌈ If the initialization of the Crypto Driver fails, the Crypto shall report `CRYPTO_E_INIT_FAILED` to the DET.
⌋()

8.3.1.2 Crypto_GetVersionInfo

[SWS_Crypto_91001]⌈

Service Name	Crypto_GetVersionInfo	

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

Service ID [hex]	0x01	
Sync/Async	Synchronous	
Reentrancy	Reentrant	
Parameters (in)	versioninfo	Pointer to where to store the version information of this module.
Parameters (inout)	None	
Parameters (out)	None	

Return value	void	--
Description	Returns the version information of this module.	
Available via	Crypto.h	

⌋(SRS_BSW_00407)

[SWS_Crypto_00047] ⌈ If the parameter `versioninfo` is a null pointer and if development error detection for the Crypto Driver is enabled, the function `Crypto_GetVersionInfo` shall report `CRYPTO_E_PARAM_POINTER` to the DET.
⌋()

8.3.2 Job Processing Interface

8.3.2.1 Crypto_ProcessJob

[SWS_Crypto_91003]⌈

Service Name	Crypto_ProcessJob	

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

Service ID [hex]	0x03	
Sync/Async	Depends on configuration	
Reentrancy	Reentrant	
Parameters (in)	objectId	Holds the identifier of the Crypto Driver Object.
Parameters (inout)	job	Pointer to the configuration of the job. Contains structures with job and primitive relevant information but also pointer to result buffers.
Parameters (out)	None	
Return value	Std_-Return-Type	E_OK: Request successful E_NOT_OK: Request failed CRYPTO_E_BUSY: Request failed, Crypro Driver Object is busy or queue is full 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_KEY_READ_FAIL: The service request failed, because key element extraction is not allowed CRYPTO_E_KEY_WRITE_FAIL: The service request failed because the writing access failed CRYPTO_E_KEY_NOT_AVAILABLE: Request failed, at least one required key element is not available. CRYPTO_E_ENTROPY_EXHAUSTED: Request failed, the entropy is exhausted

		CRYPTO_E_JOB_CANCELED: The service request failed because the synchronous Job has been canceled CRYPTO_E_KEY_EMPTY: Request failed because of uninitialized source key element CRYPTO_E_CUSTOM_ERROR: Custom processing failed
Description	Performs the crypto primitive, that is configured in the job parameter.	
Available via	Crypto.h	

⌋()

This Interface has a different behavior depending on the content of the `job` parameter (i.e. the type of crypto service).
Depending on this configuration, other input parameters within the `job` need to be set, in order to call this function successfully. I.e. the MAC Generate crypto primitive requires a key, a plaintext to be used, and a buffer for the generated MAC.

[SWS_Crypto_00057] ⌈ If the module is not initialized and if development error detection for the Crypto Driver is enabled, the function `Crypto_ProcessJob` shall report `CRYPTO_E_UNINIT` to the DET and return `E_NOT_OK`.
⌋()

[SWS_Crypto_00058] ⌈ If the parameter `objectId` is out of range and if development error detection for the Crypto Driver is enabled, the function `Crypto_ProcessJob` shall report `CRYPTO_E_PARAM_HANDLE` to the DET and return `E_NOT_OK`.
⌋()

[SWS_Crypto_00059] ⌈ **I**f the parameter `job` is a null pointer and if development error detection for the Crypto Driver is enabled, the function `Crypto_ProcessJob` shall report `CRYPTO_E_PARAM_POINTER` to the DET and return `E_NOT_OK`.
⌋()

[SWS_Crypto_00064] ⌈ If the parameter `job->jobPrimitiveInfo->primitiveInfo->service` is not supported by the Crypto Driver Object and if development error detection for the Crypto Driver is enabled, the function `Crypto_ProcessJob` shall report `CRYPTO_E_PARAM_HANDLE` to the DET and return `E_NOT_OK`
⌋()

[SWS_Crypto_00202]⌈ If the parameter job->jobPrimitiveInfo->primitiveInfo->service is set to CRYPTO_KEYDERIVE, the parameter job->targetCryptoKeyId must be in range; else the function Crypto_ProcessJob shall report CRYPTO_E_PARAM_HANDLE to DET and return E_NOT_OK
⌋()

[SWS_Crypto_00065] ⌈ If `job->jobPrimitiveInfo->primitiveInfo->service` is set to `CRYPTO_HASH` or `CRYPTO_MACGENERATE`, and the parameter `job->jobPrimitiveInputOutput->outputLengthPtr` is smaller than the

result length of the chosen algorithm, the most significant bits of the result shall be placed to the available buffer referenced by job->jobPrimitiveInputOutput->outputPtr as a truncated output.
⌋()

[SWS_Crypto_00252] ⌈ If job->jobPrimitiveInfo->primitiveInfo->service is set to CRYPTO_RANDOMGENERATE and the parameter job->jobPrimitiveInputOutput->outputLengthPtr is smaller than the result length of the chosen algorithm, arbitrary bits of the result of the algorithm shall be placed to the available buffer referenced by job->jobPrimitiveInputOutput->outputPtr as a truncated output.
⌋()

[SWS_Crypto_00067] ⌈ If the parameter job->jobPrimitiveInfo->primitiveInfo->algorithm (with its variation in family, keyLength and mode) is not supported by the Crypto Driver Object and if development error detection for the Crypto Driver is enabled, the function Crypto_ProcessJob shall report CRYPTO_E_PARAM_HANDLE to the DET and return E_NOT_OK.
⌋()

Depending of the crypto service configured in job->jobPrimitiveInfo->primitiveInfo->service, different parameters of job->jobPrimitiveInput are required to be set with valid values. The table in SWS_Crypto_00071 specifies which parameters are required or optional for a service in different modes. The following requirements specify the behavior if a required member is a null pointer.

[SWS_Crypto_00070] ⌈ If a pointer to a buffer is required as an argument, but it is a null pointer, the Crypto_ProcessJob() function shall report CRYPTO_E_PARAM_POINTER to the DET if development error detection for the Crypto Driver is enabled, and return E_NOT_OK.
⌋()

[SWS_Crypto_00142] ⌈ If a length information is required for processing a service request, either as variable or pointer, but the indicated length value is zero, and if development error detection for the Crypto Driver is enabled, the Crypto_ProcessJob() function report CRYPTO_E_PARAM_VALUE to the DET and return E_NOT_OK.
⌋()

[**SWS_Crypto_00228**] ⌈ If job->jobPrimitiveInputOutput->mode is set to CRYPTO_OPERATIONMODE_SAVE_CONTEXT or CRYPTO_OPERATIONMODE_RESTORE_CONTEXT the crypto driver shall check if CryptoPrimitive/CryptoPrimitiveSupportContext is set to TRUE. If not, the function shall return E_NOT_OK.
⌋()

[**SWS_Crypto_00229**] ⌈ If job->jobPrimitiveInputOutput->mode is set to CRYPTO_OPERATIONMODE_SAVE_CONTEXT the crypto driver is called, then first the

length of `job->outputPtr` provided with `job->outputLengthPtr` shall be checked if all context relevant data can be stored. If not, the function shall return `E_NOT_OK`.
⌋()

[**SWS_Crypto_00230**] ⌈ To save the context the crypto driver object shall store all relevant workspace data of the currently active primitive that was modified by previously processed START and UPDATE operation and shall place them `into job->outputPtr` (so-called snapshot of context data). The length of the provided data shall be written to the memory where `job->outputLengthPtr` references to. The function shall return `E_OK` if the operation was successful.
⌋()

Note:
The context data shall contain all data that is needed to restore a workspace with this data, so that the crypto primitive can resume the operation at exactly this point. For example, if a MAC calculation is currently processed, all internal data of the workspace shall be provided so that subsequent update and finish operation provides the same MAC as if the job is newly setup, e.g. with a START operation, the context is restored and the same set of data is provided when the context snapshot was saved. This shall result in the same MAC.

[**SWS_Crypto_00231**] ⌈ If job->jobPrimitiveInputOutput->mode is set to CRYPTO_OPERATIONMODE_RESTORE_CONTEXT when the crypto driver is called, then first job->inputLength shall be checked if the length of the provided context data is large enough for the currently active primitive. If not, the function shall return E_NOT_OK and shall leave the current state untouched. If enough data are provided, the data from job->inputPtr shall be extracted and copied to the right place in the current workspace. If the workspace has been successfully restored the function shall return E_ OK.

⌋()

Note:

The provision of context data to software components can have high security impacts. A thorough security analysis should be made before this feature is activated and used. At least, it bears the risk of leaking key material to a malicious caller of this function.
It is up to the crypto driver (object) to perform a consistency check or to encrypt, respectively decrypt the context data. But for security reason it is highly recommended to use encryption.

[SWS_Crypto_00071] ⌈

Member / Service*	inputPtr / **redirected input	inputLength	secondaryInputPtr / **redirected input	secondaryInputLength	tertiaryInputPtr / **redirected input	tertiaryInputLength	outputPtr / ***redirected output	outputLengthPtr	secondaryOutputPtr/ ***redirected output	secondaryOutputLengthPtr	verifyPtr	mode
HASH	UGC	UGC					FC	FC				SUFC
MACGENERATE	UGC	UGC					FC	FC				SUFC
MACVERIFY	UGC	UGC	F	F				C			F	SUFC
ENCRYPT	UGC	UGC					UFC	UFC				SUFC
DECRYPT	UGC	UGC					UFC	UFC				SUFC
AEADENCRYPT	VGC	VGC	VG	VG			UFC	UFC	F	F		SUFC
AEADDECRYPT	UGC	UGC	VG	VG	F	F	VFC	VFC			F	SUFC
SIGNATUREGENERATE	UGC	UGC					FC	FC				SUFC
SIGNATUREVERIFY	UGC	UGC	F	F			C	C			F	SUFC
RANDOMGENERATE							FC	FC				
CUSTOM	ZGV	ZGV	ZGV	ZGV	ZGV	ZGV	ZGV	ZGV	ZGV	ZGV	ZGV	ZGV

***:** Service names are derived from `Crypto_ServiceInfoType` (part of job struct)
**: In case of input redirection the corresponding key element is used as input instead of the inputBuffer.
***: In case of output redirection the corresponding key element is used as output instead of the outputBuffer
***: Which parameter is optional or not may depend on the actual algorithm implementation and it can also be influenced by the parameter processing order.

S: member required in Start mode.
U: member required in Update mode.
F: member required in Finish mode.
C: member required for Context Save/Restore operation.
Z: member optional in Start mode.
G: member optional in Finish mode.
V: member optional in Update mode.
⌋()

[SWS_Crypto_00072] ⌈ All crypto services listed in `Crypto_ServiceInfoType` except of `CRYPTO_HASH`, and `CRYPTO_RANDOMGENERATE` require a key represented as a key identifier.
⌋()

[SWS_Crypto_00073] ⌈ In the following table the content of the different input and output buffers of `job.jobPrimitiveInputOutputType` are specified:

Parameter / **Service***	Input**	Secondary Input**	Tertiary Input**	Output***	Secondary Output***	CryIf KeyId	Target CryIf KeyId
HASH	plaintext			generated hash			
MACGENERATE	plaintext			generated MAC			
MACVERIFY	plaintext	MAC to be verified					
ENCRYPT	plaintext			encrypted ciphertext			
DECRYPT	ciphertext			decrypted plaintext			
AEADENCRYPT	plaintext	associated Data		encrypted ciphertext	generated Tag		
AEADDECRYPT	ciphertext	associated Data	Tag to be verified	decrypted Plaintext			
SIGNATUREGENERATE	plaintext			generated signature			
SIGNATUREVERIFY	plaintext	signature to be verified					
RANDOMGENERATE				Generated random			
RANDOMSEED	Seed					KeyId	
KEYGENERATE						KeyId	
KEYDERIVE						KeyId	Target KeyId
KEYEXCHANGE CALCPUBVAL				Public Value		KeyId	
KEYEXCHANGE CALCSECRET	Partner's Public Value					KeyId	
KEYSETVALID						KeyId	
KEYSETINVALID						KeyId	
CUSTOM	Input	Secondary Input	Tertiary Input	Output	Secondary Output	KeyId	Target KeyId

*****: Service names are derived from `Crypto_ServiceInfoType`.
**: In case of input redirection the corresponding key element is used as input instead of the inputBuffer.
***: In case of output redirection the corresponding key element is used as output instead of the output buffer

⌋()

If no errors are detected by the Crypto Driver, the Crypto Driver processes the crypto service, configured in `job`, with the underlying hardware or software solutions.

[SWS_Crypto_00134] ⌈ If the crypto primitive requires input data, its memory location is referred by the pointer `job->jobPrimitiveInput.inputPtr`.
On calling `Crypto_ProcessJob`, the length of this data is stored in `job->jobPrimitiveInput.inputLength`.

This applies analogously to `job->jobPrimitiveInput.secondaryInputPtr` and `job->jobPrimitiveInput.secondaryInputLength` respectively `job->jobPrimitiveInput.tertiaryinputPtr` and `job->jobPrimitiveInput.tertiaryInputLength`, if they shall be used for the chosen crypto primitive.

If the input is redirected to a key element, the input buffer of the respective key element has to be used.
⌋()

[SWS_Crypto_00203] ⌈ If `job->jobRedirectionInfoRef` is not a `NULLPTR` and the configuration bit for the inputRedirection, secondaryInputRedirection and/or tertiaryInputRedirection is set within `job-> jobRedirectionInfoRef->redirectionConfig`, then the corresponding key element buffer located by `job-> jobRedirectionInfoRef->inputKeyId+ job-> jobRedirectionInfoRef->inputKeyElementId, job-> jobRedirectionInfoRef->secondaryInputKeyId+ job-> jobRedirectionInfoRef->secondaryInputKeyElementId,` and/or `jobRedirectionInfoRef->tertiaryInputKeyId+ job-> jobRedirectionInfoRef->tertiaryInputKeyElementId` and its length shall be used.
Any data provided by the input parameter of the function interface shall be ignored.
⌋()

[SWS_Crypto_00135] ⌈ If the crypto primitive requires a buffer for the result, its memory location is referred by the pointer `job->jobPrimitiveInput.outputPtr`. On calling this function, `job->jobPrimitiveInput. outputLengthPtr` shall contain the size of the associated buffer. When the request has finished, the actual length of the returned value shall be stored.

This applies analogously to `job->jobPrimitiveInput.secondaryOutputPtr` and `job->jobPrimitiveInput.secondaryOutputLengthPtr`, if they shall be used for the chosen crypto primitive.

If the output is redirected to a key element, the output buffer of the respective key element has to be used instead.
⌋()

[SWS_Crypto_00204] ⌈ If `job->jobRedirectionInfoRef` is not a `NULLPTR` and the configuration bit for the outputRedirection and/or secondaryoutputRedirection is set within `job-> jobRedirectionInfoRef->redirectionConfig`, then the corresponding key element buffer located by `job-> jobRedirectionInfoRef->outputKeyId + job-> jobRedirectionInfoRef->outputKeyElementId` and/or `job-> jobRedirectionInfoRef->secondaryOutputKeyId + job-> jobRedirectionInfoRef->secondaryOutputKeyElementId` shall be used as output. The length of the respective key element shall be set according to the length of the output.
⌋()

[SWS_Crypto_00141] ⌈ If the random generator service is chosen and the corresponding entropy, the function shall return `CRYPTO_E_ENTROPY_EXHAUSTED`. The function `Crypto_ProcessJob` shall additionally report the runtime error `CRYPTO_E_RE_ENTROPY_EXHAUSTED`.
⌋()

8.3.3 Job Cancellation Interface

8.3.3.1 Crypto_CancelJob

[SWS_Crypto_00122]⌈

Service Name	Crypto_CancelJob	

Syntax	`Std_ReturnType Crypto_CancelJob (`
	`  uint32 objectId,`
	`  Crypto_JobType* job`
	`)`	

Service ID [hex]	0x0e	
Sync/Async	Synchronous	
Reentrancy	Reentrant but not for same Crypto Driver Object	
Parameters (in)	objectId	Holds the identifier of the Crypto Driver Object.
Parameters (inout)	job	Pointer to the configuration of the job. Contains structures with job and primitive relevant information.
Parameters (out)	None	
Return value	Std_-ReturnType	E_OK: Request successful, job has been removed. E_NOT_OK: Request failed, job couldn't be removed. CRYPTO_E_JOB_CANCELED: The job has been cancelled but is still processed. No results will be returned to the application.
Description	This interface removes the provided job from the queue and cancels the processing of the job if possible.	
Available via	Crypto.h	

⌋()

[SWS_Crypto_00123] ⌈ If development error detection for the Crypto Driver is enabled: The function Crypto_CancelJob shall raise the error CRYPTO_E_UNINIT and return E_NOT_OK if the module is not yet initialized.
⌋ ( )

[SWS_Crypto_00124] ⌈ If development error detection for the Crypto Driver is enabled: The function Crypto_CancelJob shall raise the error CRYPTO_E_PARAM_HANDLE and return E_NOT_OK if the parameter objectId is out or range.
⌋ ( )

[SWS_Crypto_00125] ⌈ If development error detection for the Crypto Driver is enabled: The function Crypto_CancelJob shall raise the error CRYPTO_E_PARAM_POINTER and return E_NOT_OK if the parameter job is a null pointer.
⌋ ( )

[SWS_Crypto_00214] ⌈ If no errors are detected by Crypto Driver and the driver does currently not process this job, the service Crypto_CancelJob() shall return E_OK without any processing.
⌋()

[SWS_Crypto_00143] ⌈ If no errors are detected by Crypto Driver and the driver is able to cancel the job immediately, the service Crypto_CancelJob() shall remove the job from the queue and cancel the job in the hardware. If the cancellation is successful E_OK shall be returned, otherwise it shall return E_NOT_OK.
⌋()

Note:

Especially hardware implementations may not support a cancelation. If Crypto_CancelJob() is called and immediate cancelation is not possible at least all results and notifications of the job shall be suppressed. The caller can be sure, that there will be no (intermediate) results by callback or synchronous result value.

[SWS_Crypto_00183] ⌈ If no errors are detected by Crypto Driver and the driver is not able to cancel the job (e.g. due to hardware limitations), the service Crypto_CancelJob() shall return CRYPTO_E_JOB_CANCELED.
⌋()

Note:

SWS_Crypto_00183 should not have any effect on the job processing in the Crypto Driver. The processing should be completed as any other regular job. The CSM guarantees that the result buffer pointer is valid until the job is finished.

8.3.4 Key Management Interface

Note:

If the actual key element to be modified is directly mapped to flash memory, there could be a bigger delay when calling the key management functions (synchronous operation)

[SWS_Crypto_00145] ⌈ If the underlying crypto hardware does not allow execution of key management functions at the same time as processing a job, the key management functions shall wait while the current job is executed and start the processing of the key management function afterwards.

⌋()

Note:

It has to be ensured, that the jobs are processed fast enough to avoid that the key management function has to wait for a long time.
It is also recommended to use CRYPTO_OPERATIONMODE_SINGLECALL for the jobs.

8.3.4.1 Key Setting Interface

8.3.4.1.1 Crypto_KeyElementSet

[SWS_Crypto_00223] ⌈ The crypto driver shall only perform this operation if `CryptoKeyElement/CryptoKeyElementWriteAccess` is set to `CRYPTO_WA_ALLOWED` or `CRYPTO_WA_ENCRYPTED`.

⌋ ()

Note: If the key element references a SHE-key, it is recommended to set `CryptoKeyElementWriteAccess` to `CRYPTO_WA_ENCRYPTED`. The key itself must be provided as SHE-keys in the format M1M2M3.

[SWS_Crypto_91004]⌈

Service Name	Crypto_KeyElementSet	

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

Service ID [hex]	0x04	
Sync/Async	Synchronous	
Reentrancy	Non Reentrant	
Parameters	cryptoKeyId	Holds the identifier of the key whose key element shall be set.

(in)	keyElement Id	Holds the identifier of the key element which shall be set.
	keyPtr	Holds the pointer to the key data which shall be set as key element.
	keyLength	Contains the length of the key element in bytes.
Parameters (inout)	None	
Parameters (out)	None	
Return value	Std_Return-Type	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, the requested key element of the requested key is not available CRYPTO_E_KEY_SIZE_MISMATCH: Request failed, key element size does not match size of provided data
Description	Sets the given key element bytes to the key identified by cryptoKeyId.	
Available via	Crypto.h	

⌋()

Note:

This service works synchronously. However, it is possible that the underlying key material is resident in the flash memory. Hence it may take some time to execute this function.

[SWS_Crypto_00075] ⌈ If the Crypto Driver is not yet initialized and if development error detection for the Crypto Driver is enabled, the function `Crypto_KeyElementSet` shall report `CRYPTO_E_UNINIT` to the DET and return `E_NOT_OK`.
⌋()

[SWS_Crypto_00076] ⌈ If `cryptoKeyId` is out of range and if development error detection for the Crypto Driver is enabled, the function `Crypto_KeyElementSet` shall report `CRYPTO_E_PARAM_HANDLE` to the DET and return `E_NOT_OK`.
⌋()

[SWS_Crypto_00077] ⌈ If parameter `keyElementId` is out of range and if development error detection for the Crypto Driver is enabled, the function `Crypto_KeyElementSet` shall report `CRYPTO_E_PARAM_HANDLE` to the DET and return `E_NOT_OK`.
⌋()

[SWS_Crypto_00078] ⌈ If the parameter `keyPtr` is a null pointer and if development error detection for the Crypto Driver is enabled, the function

Crypto_KeyElementSet shall report CRYPTO_E_PARAM_POINTER to the DET and return E_NOT_OK.
⌋()

[SWS_Crypto_00079] ⌈ If keyLength is zero and if development error detection for the Crypto Driver is enabled, the function Crypto_KeyElementSet shall report CRYPTO_E_PARAM_VALUE to the DET and return E_NOT_OK.
⌋()

[SWS_Crypto_00146] ⌈ If keyLength is smaller than the size of the key element, and the key element is not configured to allow partial access, the function Crypto_KeyElementSet shall return CRYPTO_E_KEY_SIZE_MISMATCH.
⌋()

8.3.4.1.2 Crypto_KeySetValid

[SWS_Crypto_91014]⌈

Service Name	Crypto_KeySetValid	

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

Service ID [hex]	0x05	
Sync/Async	Synchronous	
Reentrancy	Non Reentrant	
Parameters (in)	cryptoKeyId	Holds the identifier of the key which shall be set to valid.
Parameters (inout)	None	
Parameters (out)	None	
Return value	Std_Return-Type	E_OK: Request successful E_NOT_OK: Request failed CRYPTO_E_BUSY: Request failed, Crypro Driver Object is busy
Description	Sets the key state of the key identified by cryptoKeyId to valid.	
Available via	Crypto.h	

⌋()

[SWS_Crypto_00196]⌈ If the module is not yet initialized and development error detection for the Crypto Driver is enabled, the function Crypto_KeySetValid() shall report CRYPTO_E_UNINIT to the DET and return E_NOT_OK.
⌋()

[SWS_Crypto_00197]⌈ If parameter `cryptoKeyId` is out of range and if development error detection for the Crypto Driver is enabled, the function `Crypto_KeySetValid()` shall report `CRYPTO_E_PARAM_HANDLE` to the DET and return `E_NOT_OK`.
⌋()

If no errors are detected by Crypto Driver, the service `Crypto_KeySetValid()` sets the key `cryptoKeyId` to “valid”.

8.3.4.1.3 Crypto_KeySetInvalid

[SWS_Crypto_91020]⌈

Service Name	Crypto_KeySetInvalid	

Syntax	`Std_ReturnType Crypto_KeySetInvalid (`
	`  uint32 cryptoKeyId`
	`)`	

Service ID [hex]	0x15	
Sync/Async	Synchronous	
Reentrancy	Non Reentrant	
Parameters (in)	cryptoKeyId	Holds the identifier of the key for which the status shall be set to invalid.
Parameters (inout)	None	
Parameters (out)	None	
Return value	Std_Return-Type	E_OK: Request successful E_NOT_OK: Request failed CRYPTO_E_BUSY: Request failed, Crypro Driver Object is busy
Description	Sets invalid for the status of the key identified by cryptoKeyId.	
Available via	Crypto.h	

⌋()

[SWS_Crypto_00236] ⌈ If the module is not yet initialized and development error detection for the Crypto Driver is enabled, the function `Crypto_KeySetInvalid()` shall report `CRYPTO_E_UNINIT` to the DET.
⌋()

[SWS_Crypto_00237] ⌈ If parameter `cryptoKeyId` is out of range and if development error detection for the Crypto Driver is enabled, the function `Crypto_KeySetInvalid()` shall report `CRYPTO_E_PARAM_HANDLE` to the DET.
⌋()

[**SWS_Crypto_00238**] ⌈ If no errors are detected by Crypto Driver, the service `Crypto_KeySetInvalid()` sets the status of key `cryptoKeyId` to invalid.
⌋()

8.3.4.2 Key Extraction Interface

8.3.4.2.1 Crypto_KeyElementGet

[SWS_Crypto_00222] ⌈ The crypto driver shall only perform this operation if `CryptoKeyElement/CryptoKeyElementReadAccess` is set to `CRYPTO_RA_ALLOWED` or `CRYPTO_RA_ENCRYPTED`.
⌋ ()

Note: Reading keys from SHE hardware is not possible, except if it is a SHE RAM-Key. In this case, reading the SHE key will provide M1M2M3. The key element should be set to CRYPTO_RA_ENCRYPTED in this case.

[SWS_Crypto_91006]⌈

Service Name	Crypto_KeyElementGet	

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

Service ID [hex]	0x06	
Sync/Async	Synchronous	
Reentrancy	Reentrant	
Parameters (in)	crypto KeyId	Holds the identifier of the key whose key element shall be returned.
	key Element Id	Holds the identifier of the key element which shall be returned.
Parameters (inout)	result Length Ptr	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.

Parameters (out)	resultPtr	Holds the pointer of the buffer for the returned key element
Return value	Std_-Return-Type	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 of the requested key is not available CRYPTO_E_KEY_READ_FAIL: Request failed because read access was denied CRYPTO_E_KEY_EMPTY: Request failed because of uninitialized source key element
Description	This interface shall be used to get a key element of the key identified by the crypto KeyId 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).	
Available via	Crypto.h	

⌋()

[SWS_Crypto_00085] ⌈ If the module is not yet initialized and if development error detection for the Crypto Driver is enabled, the function `Crypto_KeyElementGet` shall report `CRYPTO_E_UNINIT` to the DET and return `E_NOT_OK`.
⌋()

[SWS_Crypto_00086] ⌈ **I**f the parameter `cryptoKeyId` is out of range and if development error detection for the Crypto Driver is enabled, the function `Crypto_KeyElementGet` shall report `CRYPTO_E_PARAM_HANDLE` to the DET and return `E_NOT_OK`.
⌋()

[SWS_Crypto_00087] ⌈ If the parameter `keyElementId` is not contained in the respective key type and if development error detection for the Crypto Driver is enabled, the function `Crypto_KeyElementGet` shall report `CRYPTO_E_PARAM_HANDLE` to the DET and return `E_NOT_OK`.
⌋()

[SWS_Crypto_00088] ⌈ If the parameter `resultPtr` is a null pointer and if development error detection for the Crypto Driver is enabled, the function `Crypto_KeyElementGet` shall report `CRYPTO_E_PARAM_POINTER` the DET and return `E_NOT_OK`.
⌋()

[SWS_Crypto_00089] ⌈ If the parameter `resultLengthPtr` is a null pointer and if development error detection for the Crypto Driver is enabled, the function `Crypto_KeyElementGet` shall report `CRYPTO_E_PARAM_POINTER` to the DET and return `E_NOT_OK`.
⌋()

[SWS_Crypto_00090] ⌈ If the value, which is pointed by `resultLengthPtr` is zero and if development error detection for the Crypto Driver is enabled, the function `Crypto_KeyElementGet` shall report `CRYPTO_E_PARAM_VALUE` to the DET and return `E_NOT_OK`.
⌋()

If no errors are detected by Crypto Driver, the service `Crypto_KeyElementGet()` retrieves the value of the key element and store it in the buffer, which is pointed by the `resultPtr`.

[SWS_Crypto_00092] ⌈ The pointer `resultPtr` holds the memory location, where the data of the key element shall be stored. On calling this function, `resultLengthPtr` shall contain the size of the buffer provided by `resultPtr`. When the request has finished, the actual length of the returned value shall be stored.
⌋()

8.3.4.3 Key Status Interface

8.3.4.3.1 Crypto_KeyGetStatus

[SWS_Crypto_91019]⌈

Service Name	Crypto_KeyGetStatus	

Syntax	`Std_ReturnType Crypto_KeyGetStatus (`
	`  uint32 cryptoKeyId,`
	`  Crypto_KeyStatusType* keyStatusPtr`
	`)`	

Service ID [hex]	0x14	
Sync/Async	Synchronous	
Reentrancy	Non Reentrant	
Parameters (in)	cryptoKeyId	Holds the identifier of the key for which the key state shall be returned.
Parameters (inout)	None	
Parameters (out)	keyStatusPtr	Contains the pointer to the data where the status of the key shall be stored.
Return value	Std_Return-Type	E_OK: Request successful E_NOT_OK: Request failed
Description	Returns the key state of the key identified by cryptoKeyId.	
Available via	Crypto.h	

⌋()

[**SWS_Crypto_00232**] ⌈ If the module is not yet initialized and development error detection for the Crypto Driver is enabled, the function Crypto_KeyGetStatus shall report `CRYPTO_E_UNINIT` to the DET and return `E_NOT_OK`.
⌋()

[**SWS_Crypto_00233**] ⌈ If parameter cryptoKeyId is out of range and if development error detection for the Crypto Driver is enabled, the function Crypto_KeyGetStatus shall report `CRYPTO_E_PARAM_HANDLE` to the DET and return `E_NOT_OK`.
⌋()

[**SWS_Crypto_00234**] ⌈ If the parameter keyPtr is a null pointer and if development error detection for the Crypto Driver is enabled, the function Crypto_KeyGetStatus shall report `CRYPTO_E_PARAM_POINTER` to the DET and return `E_NOT_OK`.
⌋()

[**SWS_Crypto_00235**] ⌈ If no errors are detected by the Crypto Driver, the status of the key shall be placed into keyStatusPtr. If the key is in valid state, the value `CRYPTO_KEYSTATUS_VALID` shall be reported. If the key is currently not valid, the status `CRYPTO_KEYSTATUS_INVALID` shall be reported.
⌋()

8.3.4.4 Key Copying Interface

8.3.4.4.1 Crypto_KeyElementCopy

[SWS_Crypto_00148]⌈

Service Name	Crypto_KeyElementCopy	

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

Service ID [hex]	0x0f	
Sync/Async	Synchronous	
Reentrancy	Reentrant but not for the same cryptoKeyId	
Parameters (in)	cryptoKeyId	Holds the identifier of the key whose key element shall be the source element.
	keyElementId	Holds the identifier of the key element which shall be the source for the copy operation.
	targetCrypto KeyId	Holds the identifier of the key whose key element shall be the destination element.
	targetKey ElementId	Holds the identifier of the key element which shall be the destination for the copy operation.

Parameters (inout)	None	
Parameters (out)	None	
Return value	Std_Return-Type	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 of at least one requested key 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 because of uninitialized source key element
Description	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)	
Available via	Crypto.h	

⌋()

[SWS_Crypto_00149] ⌈ If the Crypto Driver is not yet initialized and if development error detection for the Crypto Driver is enabled, the function `Crypto_KeyElementCopy` shall report `CRYPTO_E_UNINIT` to the DET and return `E_NOT_OK`.
⌋()

[SWS_Crypto_00150] ⌈ If `cryptoKeyId` is out of range and if development error detection for the Crypto Driver is enabled, the function `Crypto_KeyElementCopy` shall report `CRYPTO_E_PARAM_HANDLE` to the DET and return `E_NOT_OK.`
⌋()

[SWS_Crypto_00151] ⌈ If `targetCryptoKeyId` is out of range and if development error detection for the Crypto Driver is enabled, the function `Crypto_KeyElementCopy` shall report `CRYPTO_E_PARAM_HANDLE` to the DET and return `E_NOT_OK.`
⌋()

[SWS_Crypto_00152] ⌈ If parameter `keyElementId` is out of range and if development error detection for the Crypto Driver is enabled, the function `Crypto_KeyElementCopy` shall report `CRYPTO_E_PARAM_HANDLE` to the DET and return `E_NOT_OK.`
⌋()

[SWS_Crypto_00153] ⌈ If parameter `targetKeyElementId` is out of range and if development error detection for the Crypto Driver is enabled, the function `Crypto_KeyElementCopy` shall report `CRYPTO_E_PARAM_HANDLE` to the DET and return `E_NOT_OK.`
⌋()

[SWS_Crypto_00154] If no errors are detected by the Crypto Driver, the function shall copy the key element referenced by `keyElementId` in the key referenced by `cryptoKeyId` to the key element referenced by `targetKeyElementId` in the key referenced by `targetCryptoKeyId.`

8.3.4.4.2 Crypto_KeyElementCopyPartial

[SWS_Crypto_91015]⌈

Service Name	Crypto_KeyElementCopyPartial	

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

Service ID [hex]	0x13	
Sync/Async	Synchronous	
Reentrancy	Reentrant but not for the same cryptoKeyId	
Parameters (in)	cryptoKeyId	Holds the identifier of the key whose key element shall be the source element.
	keyElementId	Holds the identifier of the key element which shall be the source for the copy operation.
	keyElement SourceOffset	This is the offset of the of the source key element indicating the start index of the copy operation.
	keyElement TargetOffset	This is the offset of the of the target key element indicating the start index of the copy operation.
	keyElementCopy Length	Specifies the number of bytes that shall be copied.
	targetCryptoKey Id	Holds the identifier of the key whose key element shall be the destination element.
	targetKey ElementId	Holds the identifier of the key element which shall be the destination for the copy operation.
Parameters (inout)	None	

Parameters (out)	None	
Return value	Std_ReturnType	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 of at least one requested key 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 because of uninitialized source key element
Description	Copies a key element to another key element in the same crypto driver. The key ElementSourceOffset and keyElementCopyLength 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. Note: If the actual key element is directly mapped to flash memory, there could be a bigger delay when calling this function (synchronous operation).	
Available via	Crypto.h	

⌋()

[SWS_Crypto_00205] ⌈ If the Crypto Driver is not yet initialized and if development error detection for the Crypto Driver is enabled, the function `Crypto_KeyElementCopyPartial` shall report `CRYPTO_E_UNINIT` to the DET and return `E_NOT_OK`.
⌋()

[SWS_Crypto_00206] ⌈ If `cryptoKeyId, keyElementId ,` `targetKeyElementId` or `targetCryptoKeyId` is out of range and if development error detection for the Crypto Driver is enabled, the function `Crypto_KeyElementCopyPartial` shall report `CRYPTO_E_PARAM_HANDLE` to the DET and return `E_NOT_OK.`
⌋()

[SWS_Crypto_00207] ⌈ If the total length of the key element specified with `keyElementId` of the key referenced by `cryptoKeyId` is smaller than `keyElementSourceOffset` + `keyElementCopyLength` `Crypto_KeyElementCopyPartial` shall return `CRYPTO_E_KEY_SIZE_MISMATCH.`
⌋()

[SWS_Crypto_00208] ⌈ If the maximum available buffer of the key element specified with `targetKeyElementId` of the key referenced by `targetCryptoKeyId` is smaller than `keyElementTargetOffset` + `keyElementCopyLength,` the

function `Crypto_KeyElementCopyPartial` shall return
`CRYPTO_E_KEY_SIZE_MISMATCH`.
⌋()

[SWS_Crypto_00209] ⌈ If no errors are detected by the Crypto Driver, the function `Crypto_KeyElementCopyPartial` shall copy a part of the key element referenced by `keyElementId` of the key referenced by `cryptoKeyId` with the offset of `keyElementSourceOffset` and with the length specified by `keyElementCopyLength` to the key element referenced by `targetKeyElementId` of the key referenced by `targetCryptoKeyId`.
⌋()

[SWS_Crypto_00210] ⌈ If the current length of the target key element is greater or equal than (`keyElementTargetOffset` + `keyElementCopyLength`), the key element length remains unchanged.
⌋()

[SWS_Crypto_00211] ⌈ If the current length of the target key element is lower than (`keyElementTargetOffset` + `keyElementCopyLength`) and the maximum length of the key element is greater or equal than (`keyElementTargetOffset` + `keyElementCopyLength`), then the source data shall be copied into the target key element and the length shall be set to (`keyElementTargetOffset` + `keyElementCopyLength`).
⌋()

8.3.4.4.3 Crypto_KeyCopy

[SWS_Crypto_00155]⌈

Service Name	Crypto_KeyCopy	

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

Service ID [hex]	0x10	
Sync/Async	Synchronous	
Reentrancy	Reentrant but not for the same cryptoKeyId	
Parameters (in)	cryptoKeyId	Holds the identifier of the key whose key element shall be the source element.
	targetCrypto KeyId	Holds the identifier of the key whose key element shall be the destination element.
Parameters (inout)	None	
Parameters (out)	None	

Return value	Std_Return-Type	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, at least one of the requested keys 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 because of uninitialized source key element
Description	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)	
Available via	Crypto.h	

⌋()

[SWS_Crypto_00156] ⌈ If the Crypto Driver is not yet initialized and if development error detection for the Crypto Driver is enabled, the function `Crypto_KeyCopy` shall report `CRYPTO_E_UNINIT` to the DET and return `E_NOT_OK`.
⌋()

[SWS_Crypto_00157] ⌈ If `cryptoKeyId` is out of range and if development error detection for the Crypto Driver is enabled, the function `Crypto_KeyCopy` shall report `CRYPTO_E_PARAM_HANDLE` to the DET and return `E_NOT_OK`.
⌋()

[SWS_Crypto_00158] ⌈ If `targetCryptoKeyId` is out of range and if development error detection for the Crypto Driver is enabled, the function `Crypto_KeyCopy` shall report `CRYPTO_E_PARAM_HANDLE` to the DET and return `E_NOT_OK`.
⌋()

[SWS_Crypto_00159] If no errors are detected by the Crypto Driver, the function shall copy all key elements in the key referenced by `cryptoKeyId` to the key the key referenced by `targetCryptoKeyId`.
⌋()

8.3.4.4.4 Crypto_KeyElementIdsGet

[SWS_Crypto_00160]⌈

Service Name	Crypto_KeyElementIdsGet

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

	)	
Service ID [hex]	0x11	
Sync/Async	Synchronous	
Reentrancy	Reentrant but not for the same cryptoKeyId	
Parameters (in)	cryptoKey Id	Holds the identifier of the key whose available element ids shall be exported.
Parameters (inout)	keyElement IdsLength Ptr	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 keyElement IdsPtr. When the request has finished, the actual number of key elements shall be stored.
Parameters (out)	keyElement IdsPtr	Contains the pointer to the array where the ids of the key elements shall be stored.
Return value	Std_-ReturnType	E_OK: Request successful E_NOT_OK: Request failed CRYPTO_E_BUSY: Request failed, Crypto Driver Object is busy
Description	Used to retrieve information which key elements are available in a given key.	
Available via	Crypto.h	

⌋()

[SWS_Crypto_00161] ⌈ If the Crypto Driver is not yet initialized and if development error detection for the Crypto Driver is enabled, the function `Crypto_KeyElementIdsGet` shall report `CRYPTO_E_UNINIT` to the DET and return `E_NOT_OK`.
⌋()

[SWS_Crypto_00162] ⌈ If `cryptoKeyId` is out of range and if development error detection for the Crypto Driver is enabled, the function `Crypto_KeyElementIdsGet` shall report `CRYPTO_E_PARAM_HANDLE` to the DET and return `E_NOT_OK`.
⌋()

If no errors are detected by the Crypto Driver, the function stores all ids of the key elements available in the key identified by `cryptoKeyId` to an array provided by `keyElementIdsPtr`. It also stores the number of elements to the value, which is pointed by `keyElementIdsLengthPtr`.

Note:
This function is needed by the CRYIF when a whole key should be copied from one Crypto Driver to another Crypto Driver by the CRYIF.

8.3.4.5 Key Generation Interface

8.3.4.5.1 Crypto_RandomSeed

[SWS_Crypto_91013]⌈

Service Name	Crypto_RandomSeed	

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

Service ID [hex]	0x0d	
Sync/Async	Synchronous	
Reentrancy	Reentrant, but not for the same cryptoKeyId	
Parameters (in)	cryptoKeyId	Holds the identifier of the key for which a new seed shall be generated.
	seedPtr	Holds a pointer to the memory location which contains the data to feed the seed.
	seedLength	Contains the length of the seed in bytes.
Parameters (inout)	None	
Parameters (out)	None	
Return value	Std_Return-Type	E_OK: Request successful E_NOT_OK: Request failed CRYPTO_E_KEY_NOT_VALID: Request failed, the key state of the key identified by cryIfKeyId is "invalid".
Description	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	
Available via	Crypto.h	

⌋()

[SWS_Crypto_00128] ⌈ If the module is not yet initialized and if development error detection for the Crypto Driver is enabled, the function `Crypto_RandomSeed` shall report `CRYPTO_E_UNINIT` to the DET and return `E_NOT_OK`.
⌋()

[SWS_Crypto_00129] ⌈ If the parameter `cryptoKeyId` is out of range and if development error detection for the Crypto Driver is enabled, the function `Crypto_RandomSeed` shall report `CRYPTO_E_PARAM_HANDLE` to the DET and return `E_NOT_OK`.
⌋()

[SWS_Crypto_00130] ⌈ If the parameter `seedPtr` is a null pointer and if development error detection for the Crypto Driver is enabled, the function

Crypto_RandomSeed shall report CRYPTO_E_PARAM_POINTER to the DET and return E_NOT_OK.
⌋()

[SWS_Crypto_00131] ⌈ If seedLength is zero and if development error detection for the Crypto Driver is enabled, the function Crypto_RandomSeed shall report CRYPTO_E_PARAM_VALUE to the DET and return E_NOT_OK.
⌋()

If no errors are detected by Crypto Driver, the service Crypto_RandomSeed() feeds the given key with a seed state derived from the entropy source. The internal state of the random generator is stored in the key element CRYPTO_KE_RANDOM_SEED.

8.3.4.5.2 Crypto_KeyGenerate

[SWS_Crypto_91007]⌈

Service Name	Crypto_KeyGenerate	

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

Service ID [hex]	0x07	
Sync/Async	Synchronous	
Reentrancy	Reentrant but not for the same cryptoKeyId	
Parameters (in)	cryptoKeyId	Holds the identifier of the key which is to be updated with the generated value.
Parameters (inout)	None	
Parameters (out)	None	
Return value	Std_Return-Type	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 because of uninitialized source key element CRYPTO_E_KEY_NOT_VALID: Request failed, the key state of the key identified by cryIfKeyId is "invalid".
Description	Generates new key material store it in the key identified by cryptoKeyId.	
Available via	Crypto.h	

⌋()

[SWS_Crypto_00094] ⌈ If the module is not yet initialized and if development error detection for the Crypto Driver is enabled, the function Crypto_KeyGenerate shall report CRYPTO_E_UNINIT to the DET and return E_NOT_OK.

⌋()

[SWS_Crypto_00095] ⌈ If the parameter `cryptoKeyId` is out of range and if development error detection for the Crypto Driver is enabled, the function `Crypto_KeyGenerate` shall report `CRYPTO_E_PARAM_HANDLE` to the DET and return `E_NOT_OK`.
⌋()

[SWS_Crypto_00165] ⌈ If no errors are detected by Crypto Driver, the service `Crypto_KeyGenerate()` generates the corresponding key.
⌋()

8.3.4.6 Key Derivation Interface

8.3.4.6.1 Crypto_KeyDerive

[SWS_Crypto_91008]⌈

Service Name	Crypto_KeyDerive	

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

Service ID [hex]	0x08	
Sync/Async	Synchronous	
Reentrancy	Reentrant but not for the same cryptoKeyId	
Parameters (in)	cryptoKeyId	Holds the identifier of the key which is used for key derivation.
	targetCryptoKey Id	Holds the identifier of the key which is used to store the derived key.
Parameters (inout)	None	
Parameters (out)	None	
Return value	Std_Return-Type	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 because of uninitialized source key element CRYPTO_E_KEY_NOT_VALID: Request failed, the key state of the key identified by cryIfKeyId is "invalid".
Description	Derives a new key by using the key elements in the given key identified by the crypto KeyId. 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 targetCryptoKey Id. The number of iterations is given in the key element CRYPTO_KE_ KEYDERIVATION_ITERATIONS.	

Available via	Crypto.h

⌋()

[SWS_Crypto_00097] ⌈ If the module is not yet initialized and if development error detection for the Crypto Driver is enabled, the function `Crypto_KeyDerive` shall report `CRYPTO_E_UNINIT` to the DET and return `E_NOT_OK`.
⌋()

[SWS_Crypto_00098] ⌈ If the parameter `cryptoKeyId` is out of range and if development error detection for the Crypto Driver is enabled, the function `Crypto_KeyDerive` shall report `CRYPTO_E_PARAM_HANDLE` to the DET and return `E_NOT_OK`.
⌋()

[SWS_Crypto_00180]⌈ If the parameter `targetCryptoKeyId` is out of range and if development error detection for the Crypto Driver is enabled, the function `Crypto_KeyDerive` shall report `CRYPTO_E_PARAM_HANDLE` to the DET and return `E_NOT_OK`.⌋()

[SWS_Crypto_00166] ⌈ 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.
⌋()

The key derivation service needs a salt and password to derivate a new key. The salt and the password therefore are stored as key elements in the key referred by `cryptoKeyId`.

8.3.4.7 Key Exchange Interface

8.3.4.7.1 Crypto_KeyExchangeCalcPubVal

[SWS_Crypto_91009]⌈

Service Name	Crypto_KeyExchangeCalcPubVal	

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

Service ID [hex]	0x09	
Sync/Async	Synchronous	
Reentrancy	Reentrant but not for the same cryptoKeyId	
Parameters (in)	cryptoKeyId	Holds the identifier of the key which shall be used for the key exchange protocol.
Parameters	public	Holds a pointer to the memory location in which the public value length

(inout)	Value LengthPtr	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.
Parameters (out)	public ValuePtr	Contains the pointer to the data where the public value shall be stored.
Return value	Std_-Return-Type	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 because of uninitialized source key element CRYPTO_E_KEY_NOT_VALID: Request failed, the key state of the key identified by cryIfKeyId is "invalid".
Description	Calculates the public value for the key exchange and stores the public key in the memory location pointed by the public value pointer.	
Available via	Crypto.h	

⌋()

[SWS_Crypto_00103] ⌈ If the module is not yet initialized and if development error detection for the Crypto Driver is enabled: The function `Crypto_KeyExchangeCalcPubVal` shall report `CRYPTO_E_UNINIT` to the DET and return `E_NOT_OK`.
⌋()

[SWS_Crypto_00104] ⌈ If the parameter `cryptoKeyId` is out of range and if development error detection for the Crypto Driver is enabled, the function `Crypto_KeyExchangeCalcPubVal` shall report `CRYPTO_E_PARAM_HANDLE` to the DET and return `E_NOT_OK`.
⌋()

[SWS_Crypto_00105] ⌈ If the parameter `publicValuePtr` is a null pointer and if development error detection for the Crypto Driver is enabled, the function `Crypto_KeyExchangeCalcPubVal` shall report `CRYPTO_E_PARAM_POINTER` to the DET and return `E_NOT_OK`.
⌋()

[SWS_Crypto_00106] ⌈ If the parameter `pubValueLengthPtr` is a null pointer and if development error detection for the Crypto Driver is enabled, the function `Crypto_KeyExchangeCalcPubVal` shall report `CRYPTO_E_PARAM_POINTER` to the DET and return `E_NOT_OK`.
⌋()

[SWS_Crypto_00107] ⌈ If the value, which is pointed by `pubValueLengthPtr` is zero and if development error detection for the Crypto Driver is enabled, the function `Crypto_KeyExchangeCalcPubVal` shall report `CRYPTO_E_PARAM_VALUE` to the DET and return `E_NOT_OK`.
⌋()

[SWS_Crypto_00167] ⌈ If no errors are detected by Crypto Driver, the service `Crypto_KeyExchangeCalcPubVal()` calculates the public value of the current job for the key exchange.
⌋()

[SWS_Crypto_00109] ⌈ The pointer `publicValuePtr` holds the memory location, where the data of the public value shall be stored. On calling this function, `publicValueLengthPtr` 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.
⌋()

8.3.4.7.2 Crypto_KeyExchangeCalcSecret

[SWS_Crypto_91010]⌈

Service Name	Crypto_KeyExchangeCalcSecret	

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

Service ID [hex]	0x0a	
Sync/Async	Synchronous	
Reentrancy	Reentrant but not for the same cryptoKeyId	
Parameters (in)	cryptoKeyId	Holds the identifier of the key which shall be used for the key exchange protocol.
	partnerPublic ValuePtr	Holds the pointer to the memory location which contains the partner's public value.
	partnerPublic ValueLength	Contains the length of the partner's public value in bytes.
Parameters (inout)	None	
Parameters (out)	None	
Return value	Std_ReturnType	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 because of uninitialized source key element CRYPTO_E_KEY_NOT_VALID: Request failed, the key state of the key identified by cryIfKeyId is "invalid".
Description	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.
Available via	Crypto.h

⌋()

[SWS_Crypto_00111] ⌈ If the module is not yet initialized and if development error detection for the Crypto Driver is enabled, the function `Crypto_KeyExchangeCalcSecret` shall report `CRYPTO_E_UNINIT` to the DET and return `E_NOT_OK`.
⌋()

[SWS_Crypto_00112] ⌈ If the parameter `cryptoKeyId` is out of range and if development error detection for the Crypto Driver is enabled, the function `Crypto_KeyExchangeCalcSecret` shall report `CRYPTO_E_PARAM_HANDLE` to the DET and return `E_NOT_OK`
⌋()

[SWS_Crypto_00113] ⌈ If the parameter `partnerPublicValuePtr` is a null pointer and if development error detection for the Crypto Driver is enabled, the function `Crypto_KeyExchangeCalcSecret` shall report `CRYPTO_E_PARAM_POINTER` to the DET and return `E_NOT_OK`.
⌋()

[SWS_Crypto_00115] ⌈ If `partnerPublicValueLength` is zero and if development error detection for the Crypto Driver is enabled, the function `Crypto_KeyExchangeCalcSecret` shall report `CRYPTO_E_PARAM_VALUE` to the DET and return `E_NOT_OK`.
⌋()

If no errors are detected by Crypto, the service `Crypto_KeyExchangeCalcSecret()` calculated the shared secret key for the key exchange and store it as key element in `cryptoKeyId`.

8.3.5 Custom Service Interface

[SWS_Crypto_91027]⌈

Service Name	Crypto_CustomSync

Syntax	`Std_ReturnType Crypto_CustomSync (`
	`  uint32 dispatchId,`
	`  uint32 keyId,`
	`  uint32 keyElementId,`
	`  uint32 targetKeyId,`
	`  uint32 targetKeyElementId,`
	`  const uint8* inputPtr,`

	`uint32 inputLength,`
	`uint8* outputPtr,`
	`uint32* outputLengthPtr,`
	`uint8* secondaryOutputPtr,`
	`uint32* secondaryOutputLengthPtr`
	`)`	

Service ID [hex]	0x1a	
Sync/Async	Synchronous	
Reentrancy	Non Reentrant	
Parameters (in)	dispatchId	unique id to identify the request
	keyId	key Id
	keyElementId	key element Id
	targetKeyId	--
	targetKeyElementId	--
	inputPtr	Pointer to the input data.
	inputLength	Contains the input length in bytes.
Parameters (inout)	None	
Parameters (out)	outputPtr	Pointer to the output data.
	outputLengthPtr	Contains the output length in bytes.
	secondaryOutputPtr	Pointer to the secondary output data.
	secondaryOutput LengthPtr	Contains the secondary output length in bytes.
Return value	Std_ReturnType	E_OK: Request successful E_NOT_OK: Request failed CRYPTO_E_BUSY: The service request failed because the service is still busy CRYPTO_E_CUSTOM_ERROR: Custom processing failed
Description	Requests the execution of a function that is specified by the given dispatch id.	
Available via	Crypto.h	

⌋()

[SWS_Crypto_00256]⌈ If the module is not yet initialized and if development error detection for the Crypto Driver is enabled,
the function Crypto_CustomSync shall report CRYPTO_E_UNINIT to the DET and return E_NOT_OK.⌋()

[SWS_Crypto_00257]⌈ If the parameter cryptoKeyId is out of range and if development error detection for the Crypto Driver is enabled,
the function Crypto_CustomSync shall report CRYPTO_E_PARAM_HANDLE to the DET and return E_NOT_OK.⌋()

8.4 Callback notification

8.4.1 Crypto_NvBlock_Init_<NvBlock>

[SWS_Crypto_91023]⌈

Service Name	Crypto_<vi>_<ai>_NvBlock_Init_<NvBlock>	

Syntax	`Std_ReturnType Crypto_<vi>_<ai>_NvBlock_Init_<NvBlock> (`
	`  NvM_InitBlockRequestType initBlockRequest`
	`)`	

Service ID [hex]	0x16	
Sync/Async	Synchronous	
Reentrancy	Non Reentrant	
Parameters (in)	initBlockRequest	The request type
Parameters (inout)	None	
Parameters (out)	None	
Return value	Std_ReturnType	E_OK: callback function has been processed successfully any other: callback function has been processed unsuccessfully
Description	This function will be called by NVM. The crypto driver has the task to initialize the NVRAM block. The resulting function name shall be set to the container associated with the Nvm BlockDescriptor:{CryptoNvBlock/CryptoNvBlockDescriptorRef}/NvmInitBlock Callback}/NvmInitBlockCallbackFnc	
Available via	Crypto_cfg.h	

⌋(SRS_CryptoStack_00117, SRS_CryptoStack_00118)

8.4.2 Crypto_NvBlock_ReadFrom_<NvBlock>

[SWS_Crypto_91024]⌈

Service Name	Crypto_<vi>_<ai>_NvBlock_ReadFrom_<NvBlock>	
Syntax	`Std_ReturnType Crypto_<vi>_<ai>_NvBlock_ReadFrom_<NvBlock> ( const void* NvmBuffer )`	
Service ID [hex]	0x17	
Sync/Async	Synchronous	
Reentrancy	Non Reentrant	
Parameters (in)	NvmBuffer	The address of the buffer where the data can be read from.
Parameters (inout)	None	
Parameters (out)	None	
Return value	Std_Return-Type	E_OK: callback function has been processed successfully any other: callback function has been processed unsuccessfully
Description	This function is called by NVM to let the crypto driver copy the key data from the mirror of the NVM ram block to an internal RAM buffer of the crypto driver. The resulting function name shall be set to the container associated with the Nvm BlockDescriptor: {CryptoNvBlock/{CryptoNvBlockDescriptorRef} / NvMReadRam BlockFromNvCallback	
Available via	Crypto_cfg.h	

⌋(SRS_CryptoStack_00117, SRS_CryptoStack_00118)

8.4.3 Crypto_NvBlock_WriteTo_<NvBlock>

[SWS_Crypto_91025]⌈

Service Name	Crypto_<vi>_<ai>_NvBlock_WriteTo_<NvBlock>
Syntax	`Std_ReturnType Crypto_<vi>_<ai>_NvBlock_WriteTo_<NvBlock> ( void* NvmBuffer )`
Service ID [hex]	0x18
Sync/Async	Synchronous
Reentrancy	Non Reentrant
Parameters (in)	None
Parameters (inout)	None

Parameters (out)	NvmBuffer	The address of the buffer where the data shall be written to
Return value	Std_Return-Type	E_OK: callback function has been processed successfully any other: callback function has been processed unsuccessfully
Description	This function is called by NVM to let the crypto driver copy the internal key element data to a mirror of the NVM ram block shortly before the data are written to NVM. The resulting function name shall be set to the container associated with the Nvm BlockDescriptor: {CryptoNvBlock/{CryptoNvBlockDescriptorRef} / NvMWriteRam BlockToNvCallback	
Available via	Crypto_cfg.h	

⌋(SRS_CryptoStack_00118)

8.4.4 Crypto_NvBlock_Callback_<NvBlock>

[SWS_Crypto_91026]⌈

Service Name	Crypto_<vi>_<ai>_NvBlock_Callback_<NvBlock>	

Syntax	`Std_ReturnType Crypto_<vi>_<ai>_NvBlock_Callback_<NvBlock> (`
	`  NvM_BlockRequestType BlockRequest,`
	`  NvM_RequestResultType JobResult`
	`)`	

Service ID [hex]	0x19	
Sync/Async	Synchronous	
Reentrancy	Non Reentrant	
Parameters (in)	Block Request	The request type (read, write, ... etc.) of the previous processed block job
	JobResult	The request result of the previous processed block job.
Parameters (inout)	None	
Parameters (out)	None	
Return value	Std_Return-Type	E_OK: callback function has been processed successfully any other: callback function has been processed unsuccessfully
Description	This function is called from NVM to notify the crypto driver that an asynchronous single block request has been finished. The resulting function name shall be set to the container associated with the Nvm BlockDescriptor: {CryptoNvBlock/{CryptoNvBlockDescriptorRef} /NvmSingleBlock Callback}/NvmSingleBlockCallbackFnc	
Available via	Crypto_cfg.h	

⌋(SRS_CryptoStack_00118, SRS_CryptoStack_00119)

8.5 Scheduled functions

8.5.1.1 Crypto_MainFunction

The `Crypto_MainFunction()` is necessary for asynchronous job processing. For synchronous job processing providing the main function is optional.

[SWS_Crypto_91012]⌈

Service Name	Crypto_MainFunction

Syntax	`void Crypto_MainFunction (`
	`  void`
	`)`

Service ID [hex]	0x0c
Description	If asynchronous job processing is configured and there are job queues, the function is called cyclically to process queued jobs.
Available via	SchM_Crypto.h

⌋()

8.6 Expected Interfaces

In this section, all interfaces required from other modules are listed.

8.6.1 Interfaces to Standard Software Modules

[SWS_Crypto_00126] ⌈ The Crypto Driver shall use an AUTOSAR DET module for development error notification.
⌋()

8.6.2 Mandatory Interfaces

[]⌈

API Function	***Header File***	***Description***
There are no mandatory interfaces.		

⌋()

8.6.3 Optional Interfaces

[SWS_Crypto_91017]⌈

API Function	***Header File***	***Description***
Det_ReportError	Det.h	Service to report development errors.
NvM_SetRam-BlockStatus	NvM.h	Service for setting the RAM block status of a permanent RAM block or the status of the explicit synchronization of a NVRAM block.
NvM_WriteBlock	NvM.h	Service to copy the data of the RAM block to its corresponding NV block.

⌋()

9 Sequence diagrams

n/a

10 Configuration specification

Chapter 10.1 specifies the structure (containers) and the parameters of the module Crypto.

Chapter 10.2 specifies additionally published information of the module Crypto.

10.1 Containers and configuration parameters

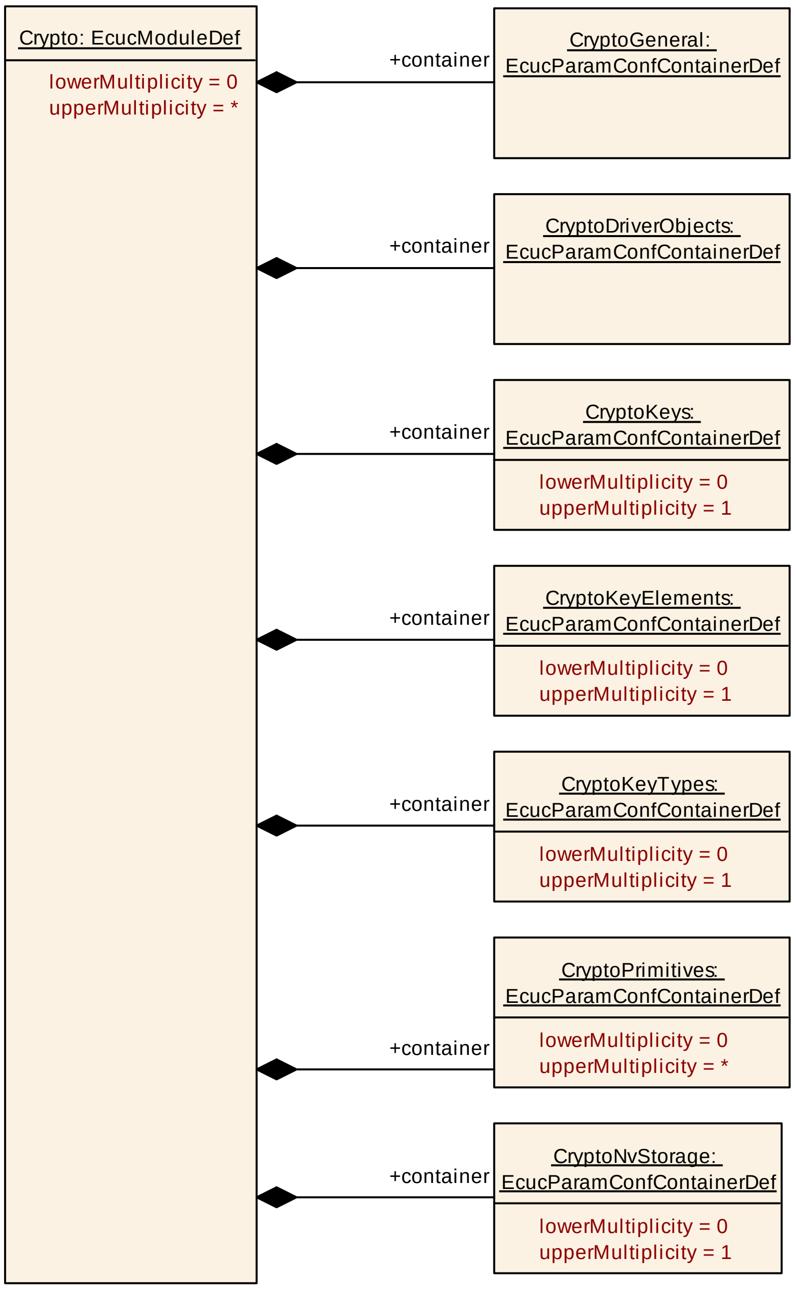
The following chapters summarize all configuration parameters. The detailed meanings of the parameters describe Chapters 7 and Chapter 8.

Note: The Ids in the configuration containers shall be consecutive, gapless and shall start from zero.

10.1.1 Crypto

SWS Item	[ECUC_Crypto_00001]
Module Name	Crypto
Description	Configuration of the Crypto (CryptoDriver) module
Post-Build Variant Support	false
Supported Config Variants	VARIANT-PRE-COMPILE

Included Containers		
Container Name	**Multiplicity**	**Scope / Dependency**
CryptoDriver-Objects	1	Container for CRYPTO Objects
CryptoGeneral	1	Container for common configuration options
CryptoKey-Elements	0..1	Container for Crypto key elements
CryptoKeyTypes	0..1	Container for CRYPTO key types
CryptoKeys	0..1	Container for CRYPTO keys
CryptoNv-Storage	0..1	Container of NV block storage. Contains a collection of all NV storage blocks used for key storage.
CryptoPrimitives	0..*	Container for CRYPTO primitives



10.1.2 CryptoGeneral

SWS Item	[ECUC_Crypto_00002]
Container Name	CryptoGeneral
Parent Container	Crypto
Description	Container for common configuration options
Post-Build Variant Multiplicity	false

Multiplicity Configuration Class	**Pre-compile time**	X	All Variants
	Link time	--	
	Post-build time	--	
Configuration Parameters			

SWS Item	[ECUC_Crypto_00006]		
Parameter Name	CryptoDevErrorDetect		
Parent Container	CryptoGeneral		
Description	Switches the development error detection and notification on or off. true: detection and notification is enabled. false: detection and notification is disabled		
Multiplicity	1		
Type	EcucBooleanParamDef		
Default value	false		
Multiplicity Configuration Class	**Pre-compile time**	X	All Variants
	Link time	--	
	Post-build time	--	
Value Configuration Class	**Pre-compile time**	X	All Variants
	Link time	--	
	Post-build time	--	
Scope / Dependency	scope: local		

SWS Item	[ECUC_Crypto_00040]		
Parameter Name	CryptoInstanceId		
Parent Container	CryptoGeneral		
Description	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.		
Multiplicity	1		
Type	EcucIntegerParamDef		
Range	0 .. 255		
Default value	--		
Post-Build Variant Value	false		
Value	**Pre-compile time**	X	All Variants

Configuration Class	**Link time**	--	
	Post-build time	--	
Scope / Dependency	scope: local		

SWS Item	[ECUC_Crypto_00038]		
Parameter Name	CryptoMainFunctionPeriod		
Parent Container	CryptoGeneral		
Description	Specifies the period of main function Crypto_MainFunction in seconds.		
Multiplicity	0..1		
Type	EcucFloatParamDef		
Range	]0 .. INF[		
Default value	--		
Multiplicity Configuration Class	**Pre-compile time**	X	All Variants
	Link time	--	
	Post-build time	--	
Value Configuration Class	**Pre-compile time**	X	All Variants
	Link time	--	
	Post-build time	--	
Scope / Dependency	scope: local		

SWS Item	[ECUC_Crypto_00007]		
Parameter Name	CryptoVersionInfoApi		
Parent Container	CryptoGeneral		
Description	Pre-processor switch to enable and disable availability of the API Crypto_GetVersionInfo(). True: API Crypto_GetVersionInfo() is available False: API Crypto_GetVersionInfo() is not available.		
Multiplicity	1		
Type	EcucBooleanParamDef		
Default value	false		
Multiplicity Configuration Class	**Pre-compile time**	X	All Variants
	Link time	--	
	Post-build time	--	

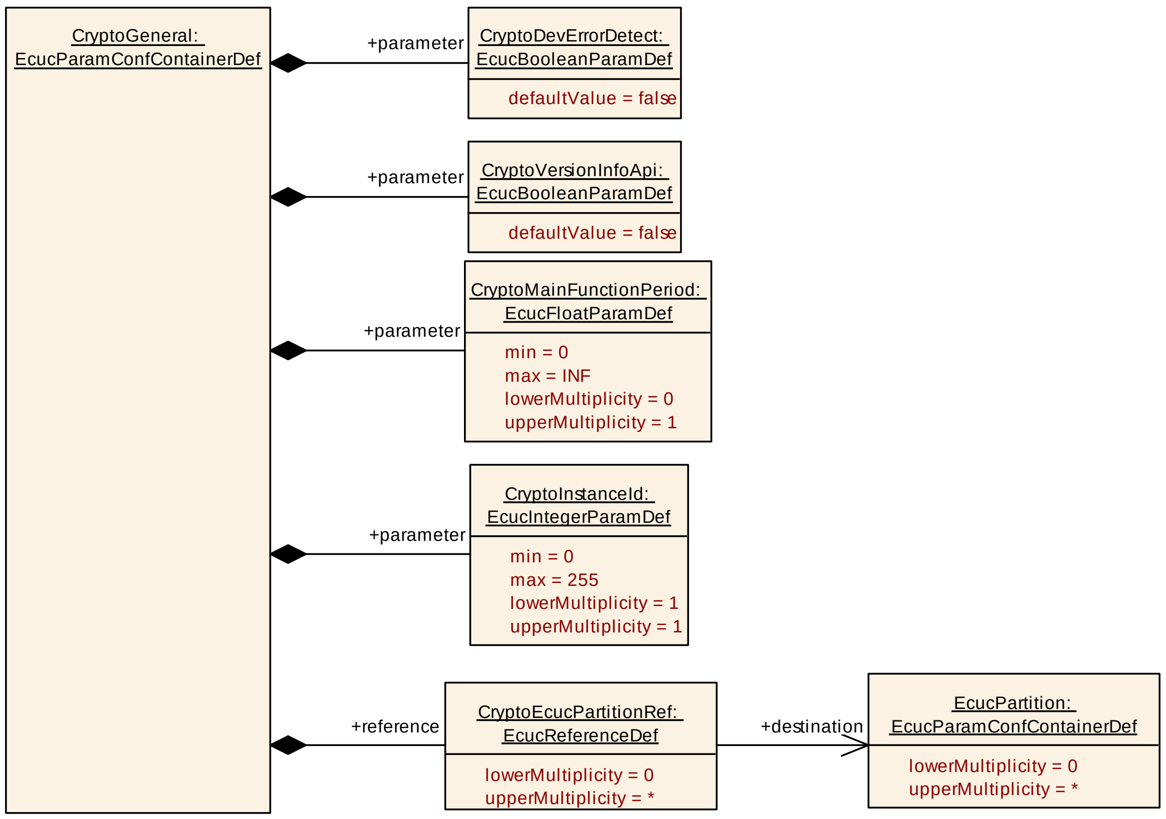
Value Configuration Class	**Pre-compile time**	X	All Variants
	Link time	--	
	Post-build time	--	
Scope / Dependency	scope: local		

SWS Item	[ECUC_Crypto_00042]		
Parameter Name	CryptoEcucPartitionRef		
Parent Container	CryptoGeneral		
Description	Maps the Crypto driver to zero or multiple ECUC partitions to make the modules API available in this partition. The module will operate as an independent instance in each of the partitions.		
Multiplicity	0..*		
Type	Reference to EcucPartition		
Post-Build Variant Multiplicity	false		
Post-Build Variant Value	false		
Multiplicity Configuration Class	**Pre-compile time**	X	All Variants
	Link time	--	
	Post-build time	--	
Value Configuration Class	**Pre-compile time**	X	All Variants
	Link time	--	
	Post-build time	--	
Scope / Dependency	scope: ECU		

No Included Containers

[SWS_Crypto_00212] ⌈ The Crypto Driver module shall reject configurations with partition mappings which are not supported by the implementation.
⌋()

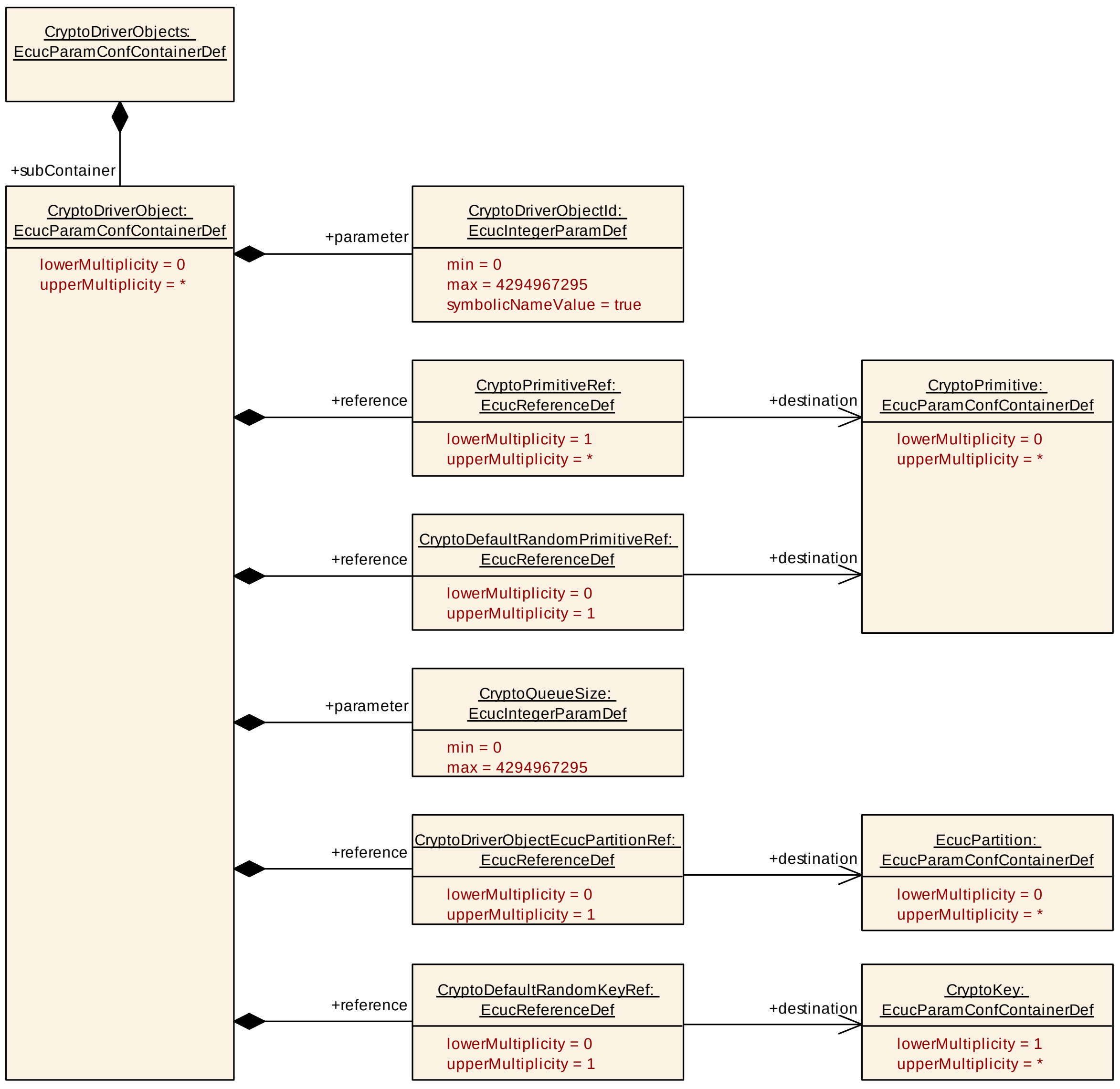
[SWS_Crypto_CONSTR_00001] ⌈ The Crypto Driver module will operate as an independent instance in each of the partitions, means the called API will only target the partition it is called in.
⌋()



10.1.3 CryptoDriverObjects

SWS Item	[ECUC_Crypto_00003]		
Container Name	CryptoDriverObjects		
Parent Container	Crypto		
Description	Container for CRYPTO Objects		
Post-Build Variant Multiplicity	false		
Multiplicity Configuration Class	**Pre-compile time**	X	All Variants
	Link time	--	
	Post-build time	--	
Configuration Parameters			

Included Containers		
Container Name	**Multiplicity**	**Scope / Dependency**
CryptoDriverObject	0..*	Configuration of a CryptoDriverObject



10.1.4 CryptoDriverObject

SWS Item	[ECUC_Crypto_00008]
Container Name	CryptoDriverObject
Parent Container	CryptoDriverObjects
Description	Configuration of a CryptoDriverObject
Configuration Parameters	

SWS Item	[ECUC_Crypto_00009]		
Parameter Name	CryptoDriverObjectId		
Parent Container	CryptoDriverObject		
Description	Identifier of the Crypto Driver Object. The Crypto Driver Object offers different crypto primitives.		
Multiplicity	1		
Type	EcucIntegerParamDef (Symbolic Name generated for this parameter)		
Range	0 .. 4294967295		
Default value	--		
Post-Build Variant Multiplicity	false		
Post-Build Variant Value	false		
Multiplicity Configuration Class	**Pre-compile time**	X	All Variants
	Link time	--	
	Post-build time	--	
Value Configuration Class	**Pre-compile time**	X	All Variants
	Link time	--	
	Post-build time	--	
Scope / Dependency	scope: local		

SWS Item	[ECUC_Crypto_00019]		
Parameter Name	CryptoQueueSize		
Parent Container	CryptoDriverObject		
Description	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.		
Multiplicity	1		
Type	EcucIntegerParamDef		
Range	0 .. 4294967295		
Default value	--		
Multiplicity Configuration Class	**Pre-compile time**	X	All Variants
	Link time	--	
	Post-build time	--	

Value Configuration Class	**Pre-compile time**	X	All Variants
	Link time	--	
	Post-build time	--	
Scope / Dependency	scope: local		

SWS Item	[ECUC_Crypto_00045]		
Parameter Name	CryptoDefaultRandomKeyRef		
Parent Container	CryptoDriverObject		
Description	This is a reference to the CryptoKey that is used by the CryptoDefault RandomPrimitiveRef. The key contains key elements that are necessary to seed the random number generator. This element shall only be set if the primitive referenced by CryptoDefault RandomPrimitiveRef requires a seed value.		
Multiplicity	0..1		
Type	Reference to CryptoKey		
Multiplicity Configuration Class	**Pre-compile time**	X	All Variants
	Link time	--	
	Post-build time	--	
Value Configuration Class	**Pre-compile time**	X	All Variants
	Link time	--	
	Post-build time	--	
Scope / Dependency	scope: local		

SWS Item	[ECUC_Crypto_00044]
Parameter Name	CryptoDefaultRandomPrimitiveRef
Parent Container	CryptoDriverObject
Description	This is a reference to a primitive that configures a default random number generator. If a crypto driver object needs to perform a crypto primitive that requires a random number generator, but the configuration of this primitive does not provide parameter for a random number generator, then this default random number generator shall be used (i.e. the elements of algorithm family, secondary algorithm family and algorithm mode do not provide this information). Example: The crypto driver shall generate a signature based on elliptic curve but the primitive for signature generation lacks information about a random number generator.
Multiplicity	0..1

Type	Reference to CryptoPrimitive		
Multiplicity Configuration Class	**Pre-compile time**	X	All Variants
	Link time	--	
	Post-build time	--	
Value Configuration Class	**Pre-compile time**	X	All Variants
	Link time	--	
	Post-build time	--	
Scope / Dependency	scope: local		

SWS Item	[ECUC_Crypto_00043]		
Parameter Name	CryptoDriverObjectEcucPartitionRef		
Parent Container	CryptoDriverObject		
Description	Maps a crypto driver object to zero or one ECUC partition. The ECUC partition referenced is a subset of the ECUC partitions where the Crypto driver is mapped to.		
Multiplicity	0..1		
Type	Reference to EcucPartition		
Post-Build Variant Multiplicity	false		
Post-Build Variant Value	false		
Multiplicity Configuration Class	**Pre-compile time**	X	All Variants
	Link time	--	
	Post-build time	--	
Value Configuration Class	**Pre-compile time**	X	All Variants
	Link time	--	
	Post-build time	--	
Scope / Dependency	scope: ECU		

SWS Item	[ECUC_Crypto_00018]
Parameter Name	CryptoPrimitiveRef
Parent Container	CryptoDriverObject
Description	Refers to primitive in the CRYPTO. The CryptoPrimitive is a pre-configured container of the crypto service that shall be used.

Multiplicity	1..*		
Type	Reference to CryptoPrimitive		
Post-Build Variant Multiplicity	false		
Post-Build Variant Value	false		
Multiplicity Configuration Class	**Pre-compile time**	X	All Variants
	Link time	--	
	Post-build time	--	
Value Configuration Class	**Pre-compile time**	X	All Variants
	Link time	--	
	Post-build time	--	
Scope / Dependency	scope: local		

No Included Containers

[SWS_Crypto_CONSTR_00002] ⌈ The ECUC partitions referenced by CryptoDriverObjectEcucPartitionRef shall be a subset of the ECUC partitions referenced by CryptoEcucPartitionRef.
⌋()

[SWS_Crypto_CONSTR_00003] ⌈ If the CryptoDriverObjectEcucPartitionRef shall be configured for an HSM it shall be mapped to 0 or 1 ECUC partitions only.
⌋()

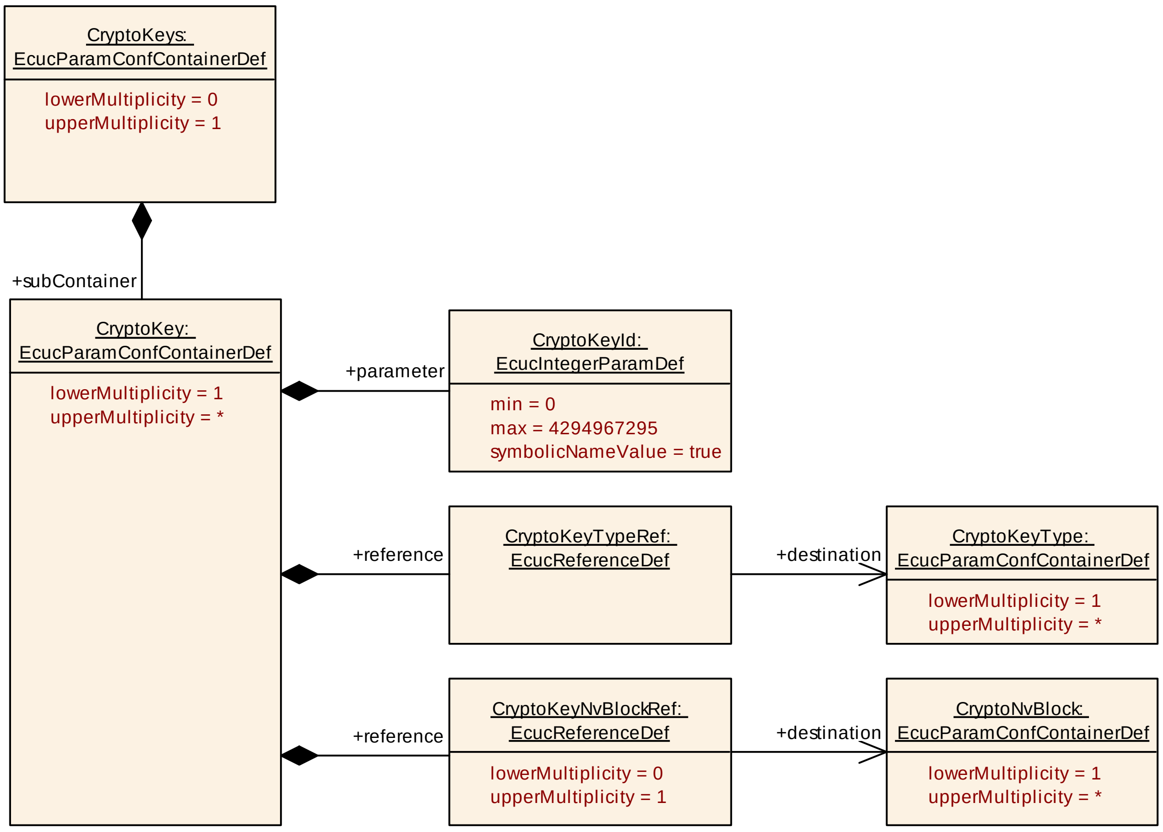
[SWS_Crypto_CONSTR_00004] ⌈ If CryptoEcucPartitionRef references one or more ECUC partitions, CryptoDriverObjectEcucPartitionRef shall have a multiplicity of greater than zero and reference one or several of these ECUC partitions as well.
⌋()

10.1.5 CryptoKeys

SWS Item	[ECUC_Crypto_00004]
Container Name	CryptoKeys
Parent Container	Crypto
Description	Container for CRYPTO keys

Configuration Parameters

Included Containers		
Container Name	**Multiplicity**	**Scope / Dependency**
CryptoKey	1..*	Configuration of a CryptoKey



10.1.6 CryptoKey

SWS Item	[ECUC_Crypto_00011]		
Container Name	CryptoKey		
Parent Container	CryptoKeys		
Description	Configuration of a CryptoKey		
Post-Build Variant Multiplicity	false		
Multiplicity Configuration Class	**Pre-compile time**	X	All Variants
	Link time	--	

	Post-build time	--	
Configuration Parameters			

SWS Item	[ECUC_Crypto_00012]		
Parameter Name	CryptoKeyId		
Parent Container	CryptoKey		
Description	Identifier of the CRYPTO Key		
Multiplicity	1		
Type	EcucIntegerParamDef (Symbolic Name generated for this parameter)		
Range	0 .. 4294967295		
Default value	--		
Post-Build Variant Multiplicity	false		
Post-Build Variant Value	false		
Multiplicity Configuration Class	**Pre-compile time**	X	All Variants
	Link time	--	
	Post-build time	--	
Value Configuration Class	**Pre-compile time**	X	All Variants
	Link time	--	
	Post-build time	--	
Scope / Dependency	scope: local		

SWS Item	[ECUC_Crypto_00059]		
Parameter Name	CryptoKeyNvBlockRef		
Parent Container	CryptoKey		
Description	Reference to the NV block where the persistent key elements of this key shall be stored to.		
Multiplicity	0..1		
Type	Reference to CryptoNvBlock		
Multiplicity Configuration Class	**Pre-compile time**	X	All Variants
	Link time	--	
	Post-build time	--	
Value Configuration Class	**Pre-compile time**	X	All Variants

	Link time	--	
	Post-build time	--	
Scope / Dependency	scope: local		

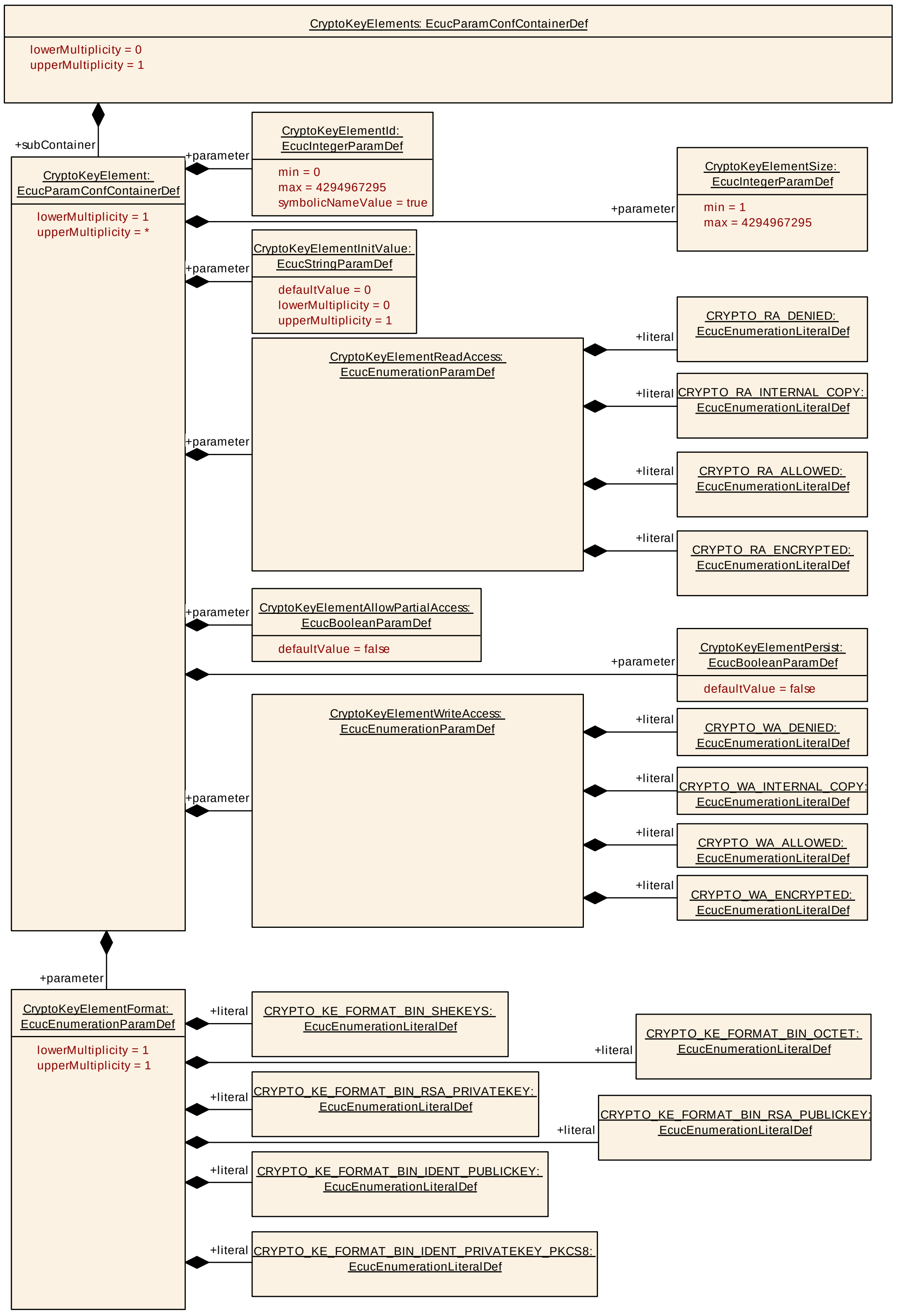
SWS Item	[ECUC_Crypto_00020]		
Parameter Name	CryptoKeyTypeRef		
Parent Container	CryptoKey		
Description	Refers to a pointer in the CRYPTOto a CryptoKeyType. The CryptoKeyType provides the information which key elements are contained in a CryptoKey.		
Multiplicity	1		
Type	Reference to CryptoKeyType		
Post-Build Variant Multiplicity	false		
Post-Build Variant Value	false		
Multiplicity Configuration Class	**Pre-compile time**	X	All Variants
	Link time	--	
	Post-build time	--	
Value Configuration Class	**Pre-compile time**	X	All Variants
	Link time	--	
	Post-build time	--	
Scope / Dependency	scope: local		

No Included Containers

10.1.7 CryptoKeyElements

SWS Item	[ECUC_Crypto_00005]
Container Name	CryptoKeyElements
Parent Container	Crypto
Description	Container for Crypto key elements
Configuration Parameters	

Included Containers		
Container Name	**Multiplicity**	**Scope / Dependency**
CryptoKeyElement	1..*	Configuration of a CryptoKeyElement


CryptoKeyElements: EcucParamConfContainerDef
lowerMultiplicity = 0
upperMultiplicity = 1
+subContainer
CryptoKeyElement: EcucParamConfContainerDef
lowerMultiplicity = 1
upperMultiplicity = *
+parameter
CryptoKeyElementId: EcucIntegerParamDef
min = 0
max = 4294967295
symbolicNameValue = true
CryptoKeyElementSize: EcucIntegerParamDef
min = 1
max = 4294967295
CryptoKeyElementInitValue: EcucStringParamDef
defaultValue = 0
lowerMultiplicity = 0
upperMultiplicity = 1
CryptoKeyElementReadAccess: EcucEnumerationParamDef
+literal
CRYPTO_RA_DENIED: EcucEnumerationLiteralDef
CRYPTO_RA_INTERNAL_COPY: EcucEnumerationLiteralDef
CRYPTO_RA_ALLOWED: EcucEnumerationLiteralDef
CRYPTO_RA_ENCRYPTED: EcucEnumerationLiteralDef
CryptoKeyElementAllowPartialAccess: EcucBooleanParamDef
defaultValue = false
CryptoKeyElementPersist: EcucBooleanParamDef
defaultValue = false
CryptoKeyElementWriteAccess: EcucEnumerationParamDef
CRYPTO_WA_DENIED: EcucEnumerationLiteralDef
CRYPTO_WA_INTERNAL_COPY: EcucEnumerationLiteralDef
CRYPTO_WA_ALLOWED: EcucEnumerationLiteralDef
CRYPTO_WA_ENCRYPTED: EcucEnumerationLiteralDef
CryptoKeyElementFormat: EcucEnumerationParamDef
lowerMultiplicity = 1
upperMultiplicity = 1
CRYPTO_KE_FORMAT_BIN_SHEKEYS: EcucEnumerationLiteralDef
CRYPTO_KE_FORMAT_BIN_OCTET: EcucEnumerationLiteralDef
CRYPTO_KE_FORMAT_BIN_RSA_PRIVATEKEY: EcucEnumerationLiteralDef
CRYPTO_KE_FORMAT_BIN_RSA_PUBLICKEY: EcucEnumerationLiteralDef
CRYPTO_KE_FORMAT_BIN_IDENT_PUBLICKEY: EcucEnumerationLiteralDef
CRYPTO_KE_FORMAT_BIN_IDENT_PRIVATEKEY_PKCS8: EcucEnumerationLiteralDef

10.1.8 CryptoKeyElement

SWS Item	[ECUC_Crypto_00014]
Container Name	CryptoKeyElement
Parent Container	CryptoKeyElements
Description	Configuration of a CryptoKeyElement
Configuration Parameters	

SWS Item	[ECUC_Crypto_00025]		
Parameter Name	CryptoKeyElementAllowPartialAccess		
Parent Container	CryptoKeyElement		
Description	Enable or disable writing and reading the key element with data smaller than the size of the element. True: enable partial access of the key element False: disable partial access of the key element		
Multiplicity	1		
Type	EcucBooleanParamDef		
Default value	false		
Multiplicity Configuration Class	**Pre-compile time**	X	All Variants
	Link time	--	
	Post-build time	--	
Value Configuration Class	**Pre-compile time**	X	All Variants
	Link time	--	
	Post-build time	--	
Scope / Dependency	scope: local		

SWS Item	[ECUC_Crypto_00041]	
Parameter Name	CryptoKeyElementFormat	
Parent Container	CryptoKeyElement	
Description	Defines the format for the key element. This is the format used to provide or extract the key data from the driver.	
Multiplicity	1	
Type	EcucEnumerationParamDef	
Range	CRYPTO_KE_FORMAT_BIN_IDENT_PRIVATEKEY_PKCS8	0x03

	CRYPTO_KE_FORMAT_BIN_IDENT_PUBLICKEY	0x04	
	CRYPTO_KE_FORMAT_BIN_OCTET	0x01	
	CRYPTO_KE_FORMAT_BIN_RSA_PRIVATEKEY	0x05	
	CRYPTO_KE_FORMAT_BIN_RSA_PUBLICKEY	0x06	
	CRYPTO_KE_FORMAT_BIN_SHEKEYS	0x02	
Multiplicity Configuration Class	**Pre-compile time**	X	All Variants
	Link time	--	
	Post-build time	--	
Value Configuration Class	**Pre-compile time**	X	All Variants
	Link time	--	
	Post-build time	--	
Scope / Dependency			

SWS Item	[ECUC_Crypto_00021]		
Parameter Name	CryptoKeyElementId		
Parent Container	CryptoKeyElement		
Description	Identifier of the CRYPTO key element		
Multiplicity	1		
Type	EcucIntegerParamDef (Symbolic Name generated for this parameter)		
Range	0 .. 4294967295		
Default value	--		
Post-Build Variant Multiplicity	false		
Post-Build Variant Value	false		
Multiplicity Configuration Class	**Pre-compile time**	X	All Variants
	Link time	--	
	Post-build time	--	
Value Configuration Class	**Pre-compile time**	X	All Variants
	Link time	--	
	Post-build time	--	
Scope / Dependency	scope: local		

SWS Item	[ECUC_Crypto_00023]		
Parameter Name	CryptoKeyElementInitValue		
Parent Container	CryptoKeyElement		
Description	Value which will be used to fill the key element during startup (i) for all non-persistent key elements, and (ii) for those persistent key elements that have never been written.		
Multiplicity	0..1		
Type	EcucStringParamDef		
Default value	0		
Regular Expression	--		
Multiplicity Configuration Class	**Pre-compile time**	X	All Variants
	Link time	--	
	Post-build time	--	
Value Configuration Class	**Pre-compile time**	X	All Variants
	Link time	--	
	Post-build time	--	
Scope / Dependency	scope: local		

SWS Item	[ECUC_Crypto_00026]		
Parameter Name	CryptoKeyElementPersist		
Parent Container	CryptoKeyElement		
Description	Enable or disable persisting of the key element in non-volatile storage. True: enable persisting of the key element. False: disable persisting of the key element.		
Multiplicity	1		
Type	EcucBooleanParamDef		
Default value	false		
Multiplicity Configuration Class	**Pre-compile time**	X	All Variants
	Link time	--	
	Post-build time	--	
Value Configuration Class	**Pre-compile time**	X	All Variants
	Link time	--	
	Post-build time	--	

Scope / Dependency	scope: local

SWS Item	[ECUC_Crypto_00024]		
Parameter Name	CryptoKeyElementReadAccess		
Parent Container	CryptoKeyElement		
Description	Define the reading access rights of the key element through external API. CRYPTO_RA_DENIED = key element cannot be read from outside the Crypto Driver CRYPTO_RA INTERNAL_COPY = key element can be copied to another key element in the same crypto driver. CRYPTO_RA_ALLOWED = key element can be read as plaintext CRYPTO_RA_ENCRYPTED = key element can be read encrypted. E.g. SHE Ram-Key export.		
Multiplicity	1		
Type	EcucEnumerationParamDef		
Range	CRYPTO_RA_ALLOWED	0x00	
	CRYPTO_RA_DENIED	0x03	
	CRYPTO_RA_ENCRYPTED	0x01	
	CRYPTO_RA_INTERNAL_COPY	0x02	
Multiplicity Configuration Class	**Pre-compile time**	X	All Variants
	Link time	--	
	Post-build time	--	
Value Configuration Class	**Pre-compile time**	X	All Variants
	Link time	--	
	Post-build time	--	
Scope / Dependency	scope: local dependency: The values for the enumeration types are chosen in a way that its value can be used to deduce a hierarchical access level.		

SWS Item	[ECUC_Crypto_00022]	
Parameter Name	CryptoKeyElementSize	
Parent Container	CryptoKeyElement	
Description	Maximum Size size of a CRYPTO key element in bytes	
Multiplicity	1	
Type	EcucIntegerParamDef	
Range	1 .. 4294967295	

Default value	--		
Post-Build Variant Multiplicity	false		
Post-Build Variant Value	false		
Multiplicity Configuration Class	**Pre-compile time**	X	All Variants
	Link time	--	
	Post-build time	--	
Value Configuration Class	**Pre-compile time**	X	All Variants
	Link time	--	
	Post-build time	--	
Scope / Dependency	scope: local		

SWS Item	[ECUC_Crypto_00027]		
Parameter Name	CryptoKeyElementWriteAccess		
Parent Container	CryptoKeyElement		
Description	Define the writing access rights of the key element through external API. CRYPTO_WA_DENIED = key element can not be written from outside the Crypto Driver CRYPTO_WA INTERNAL_COPY = key element can be filled with another key element in the same crypto driver. CRYPTO_WA_ALLOWED = key element can be rwritten as plaintext CRYPTO_WA_ENCRYPTED = key element can be written encrypted. E.g. SHE load key.		
Multiplicity	1		
Type	EcucEnumerationParamDef		
Range	CRYPTO_WA_ALLOWED	0x00	
	CRYPTO_WA_DENIED	0x03	
	CRYPTO_WA_ENCRYPTED	0x01	
	CRYPTO_WA_INTERNAL_COPY	0x02	
Multiplicity Configuration Class	**Pre-compile time**	X	All Variants
	Link time	--	
	Post-build time	--	
Value Configuration Class	**Pre-compile time**	X	All Variants
	Link time	--	
	Post-build time	--	
Scope / Dependency	scope: local dependency: The values for the enumeration types are chosen in a way that its		

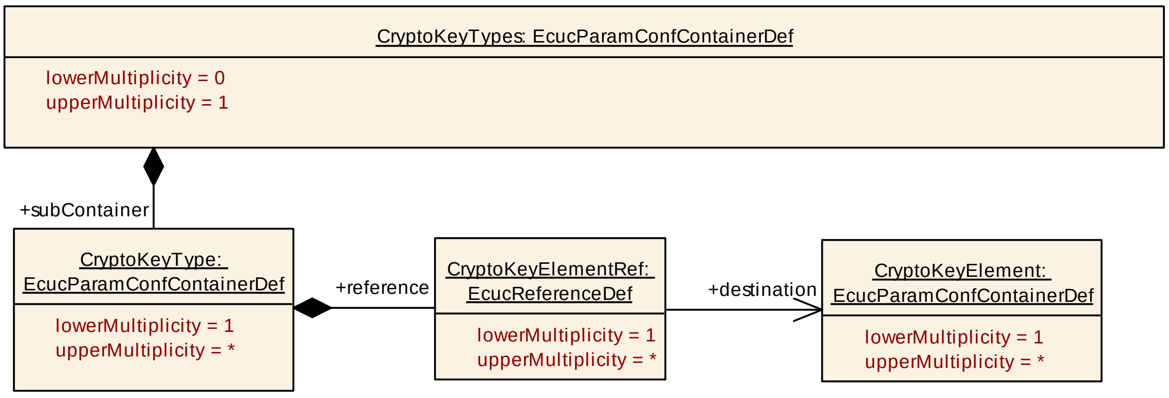
	value can be used to deduce a hierarchical access level.

No Included Containers

10.1.9 CryptoKeyTypes

SWS Item	[ECUC_Crypto_00017]		
Container Name	CryptoKeyTypes		
Parent Container	Crypto		
Description	Container for CRYPTO key types		
Post-Build Variant Multiplicity	false		
Multiplicity Configuration Class	**Pre-compile time**	X	All Variants
	Link time	--	
	Post-build time	--	
Configuration Parameters			

Included Containers		
Container Name	**Multiplicity**	**Scope / Dependency**
CryptoKeyType	1..*	Configuration of a CryptoKeyType



10.1.10 CryptoKeyType

SWS Item	[ECUC_Crypto_00030]
Container Name	CryptoKeyType
Parent Container	CryptoKeyTypes
Description	Configuration of a CryptoKeyType
Configuration Parameters	

SWS Item	[ECUC_Crypto_00031]		
Parameter Name	CryptoKeyElementRef		
Parent Container	CryptoKeyType		
Description	Refers to a pointer in the CRYPTOCrypto Key Element, which holds the data of the crypto key element.		
Multiplicity	1..*		
Type	Reference to CryptoKeyElement		
Multiplicity Configuration Class	**Pre-compile time**	X	All Variants
	Link time	--	
	Post-build time	--	
Value Configuration Class	**Pre-compile time**	X	All Variants
	Link time	--	
	Post-build time	--	
Scope / Dependency	scope: local		

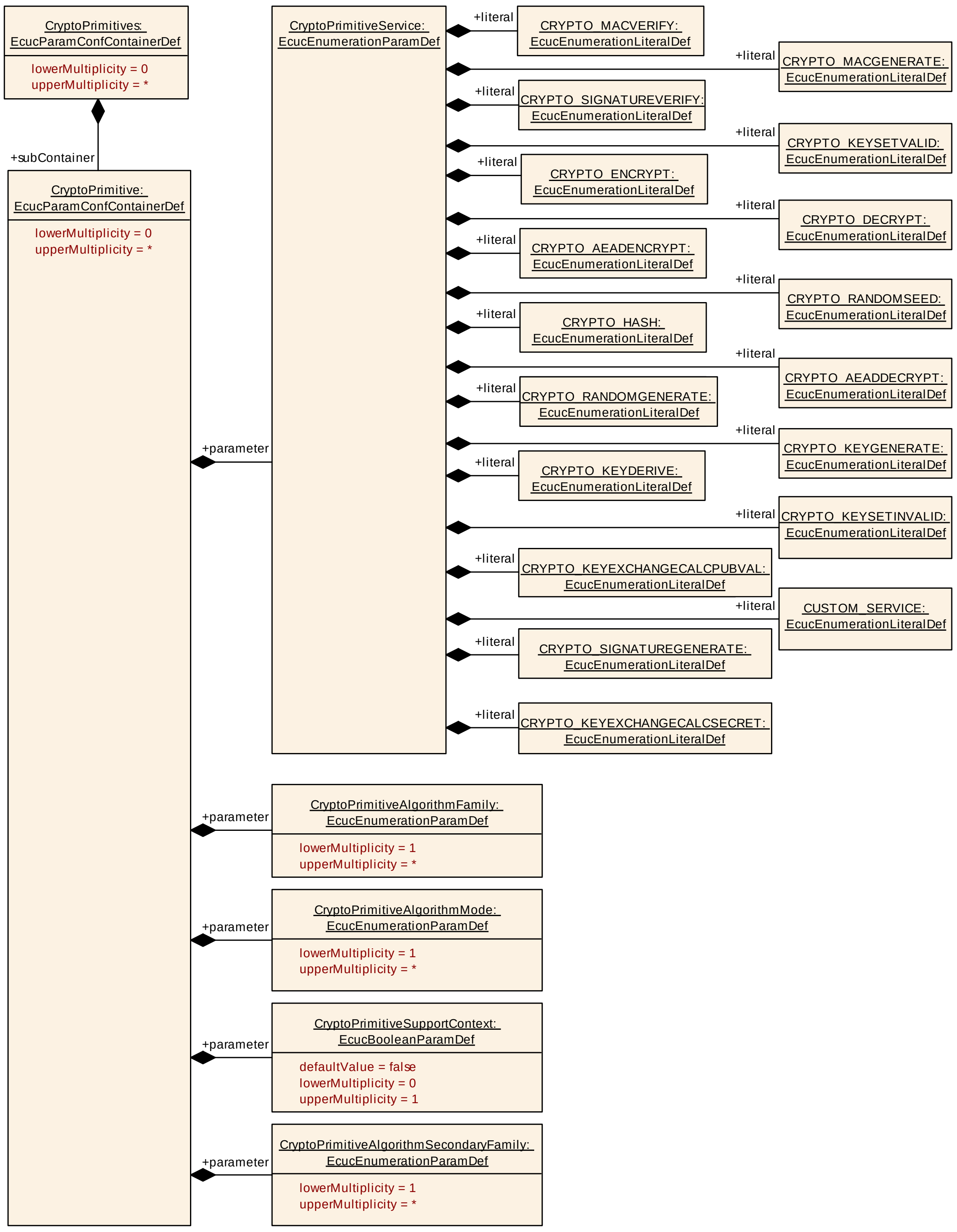
No Included Containers

10.1.11 CryptoPrimitives

SWS Item	[ECUC_Crypto_00032]		
Container Name	CryptoPrimitives		
Parent Container	Crypto		
Description	Container for CRYPTO primitives		
Multiplicity Configuration Class	**Pre-compile time**	X	All Variants

	Link time	--	
	Post-build time	--	
Configuration Parameters			

Included Containers		
Container Name	**Multiplicity**	**Scope / Dependency**
CryptoPrimitive	0..*	Configuration of a CryptoPrimitive
CryptoPrimitive-AlgorithmFamily-Custom	0..*	Container of custom algorithm family values. The container name serves as a symbolic name for the identifier of the custom algorithm family type.
CryptoPrimitive-AlgorithmMode-Custom	0..*	Container of custom algorithm family values. The container name serves as a symbolic name for the identifier of the custom algorithm family type.


CryptoPrimitives: EcucParamConfContainerDef
lowerMultiplicity = 0
upperMultiplicity = *
+subContainer
CryptoPrimitive: EcucParamConfContainerDef
lowerMultiplicity = 0
upperMultiplicity = *
+parameter
CryptoPrimitiveService: EcucEnumerationParamDef
+literal
CRYPTO_MACVERIFY: EcucEnumerationLiteralDef
CRYPTO_MACGENERATE: EcucEnumerationLiteralDef
CRYPTO_SIGNATUREVERIFY: EcucEnumerationLiteralDef
CRYPTO_KEYSETVALID: EcucEnumerationLiteralDef
CRYPTO_ENCRYPT: EcucEnumerationLiteralDef
CRYPTO_DECRYPT: EcucEnumerationLiteralDef
CRYPTO_AEADENCRYPT: EcucEnumerationLiteralDef
CRYPTO_RANDOMSEED: EcucEnumerationLiteralDef
CRYPTO_HASH: EcucEnumerationLiteralDef
CRYPTO_AEADDECRYPT: EcucEnumerationLiteralDef
CRYPTO_RANDOMGENERATE: EcucEnumerationLiteralDef
CRYPTO_KEYGENERATE: EcucEnumerationLiteralDef
CRYPTO_KEYDERIVE: EcucEnumerationLiteralDef
CRYPTO_KEYSETINVALID: EcucEnumerationLiteralDef
CRYPTO_KEYEXCHANGECALCPUBVAL: EcucEnumerationLiteralDef
CUSTOM_SERVICE: EcucEnumerationLiteralDef
CRYPTO_SIGNATUREGENERATE: EcucEnumerationLiteralDef
CRYPTO_KEYEXCHANGECALCSECRET: EcucEnumerationLiteralDef
CryptoPrimitiveAlgorithmFamily: EcucEnumerationParamDef
lowerMultiplicity = 1
upperMultiplicity = *
CryptoPrimitiveAlgorithmMode: EcucEnumerationParamDef
lowerMultiplicity = 1
upperMultiplicity = *
CryptoPrimitiveSupportContext: EcucBooleanParamDef
defaultValue = false
lowerMultiplicity = 0
upperMultiplicity = 1
CryptoPrimitiveAlgorithmSecondaryFamily: EcucEnumerationParamDef
lowerMultiplicity = 1
upperMultiplicity = *

10.1.12 CryptoPrimitive

SWS Item	[ECUC_Crypto_00033]
Container Name	CryptoPrimitive

Parent Container	CryptoPrimitives		
Description	Configuration of a CryptoPrimitive		
Multiplicity Configuration Class	**Pre-compile time**	X	All Variants
	Link time	--	
	Post-build time	--	
Configuration Parameters			

SWS Item	[ECUC_Crypto_00035]	
Parameter Name	CryptoPrimitiveAlgorithmFamily	
Parent Container	CryptoPrimitive	
Description	Determines the algorithm family used for the crypto service	
Multiplicity	1..*	
Type	EcucEnumerationParamDef	
Range	CRYPTO_ALGOFAM_3DES	--
	CRYPTO_ALGOFAM_AES	--
	CRYPTO_ALGOFAM_BLAKE_1_256	--
	CRYPTO_ALGOFAM_BLAKE_1_512	--
	CRYPTO_ALGOFAM_BLAKE_2s_256	--
	CRYPTO_ALGOFAM_BLAKE_2s_512	--
	CRYPTO_ALGOFAM_BRAINPOOL	--
	CRYPTO_ALGOFAM_CHACHA	--
	CRYPTO_ALGOFAM_CUSTOM	--
	CRYPTO_ALGOFAM_DH	--
	CRYPTO_ALGOFAM_DRBG	--
	CRYPTO_ALGOFAM_ECCANSI	--
	CRYPTO_ALGOFAM_ECCNIST	--
	CRYPTO_ALGOFAM_ECCSEC	--
	CRYPTO_ALGOFAM_ECDH	--
	CRYPTO_ALGOFAM_ECDSA	--
	CRYPTO_ALGOFAM_ED25519	--
	CRYPTO_ALGOFAM_EEA3	--

	CRYPTO_ALGOFAM_EIA3	--
	CRYPTO_ALGOFAM_FIPS186	--
	CRYPTO_ALGOFAM_HKDF	--
	CRYPTO_ALGOFAM_KDFX963	--
	CRYPTO_ALGOFAM_NOT_SET	--
	CRYPTO_ALGOFAM_PADDING_ ONEWITHZEROS	--
	CRYPTO_ALGOFAM_PADDING_PKCS7	--
	CRYPTO_ALGOFAM_PBKDF2	--
	CRYPTO_ALGOFAM_POLY1305	--
	CRYPTO_ALGOFAM_RIPEMD160	--
	CRYPTO_ALGOFAM_RNG	--
	CRYPTO_ALGOFAM_RSA	--
	CRYPTO_ALGOFAM_SHA1	--
	CRYPTO_ALGOFAM_SHA2_224	--
	CRYPTO_ALGOFAM_SHA2_256	--
	CRYPTO_ALGOFAM_SHA2_384	--
	CRYPTO_ALGOFAM_SHA2_512	--
	CRYPTO_ALGOFAM_SHA2_512_224	--
	CRYPTO_ALGOFAM_SHA2_512_256	--
	CRYPTO_ALGOFAM_SHA3_224	--
	CRYPTO_ALGOFAM_SHA3_256	--
	CRYPTO_ALGOFAM_SHA3_384	--
	CRYPTO_ALGOFAM_SHA3_512	--
	CRYPTO_ALGOFAM_SHAKE128	--
	CRYPTO_ALGOFAM_SHAKE256	--
	CRYPTO_ALGOFAM_SIPHASH	--
	CRYPTO_ALGOFAM_SM2	--
	CRYPTO_ALGOFAM_SM3	--
	CRYPTO_ALGOFAM_X25519	--
Post-Build Variant Value	false	

Multiplicity Configuration Class	**Pre-compile time**	X	All Variants
	Link time	--	
	Post-build time	--	
Value Configuration Class	**Pre-compile time**	X	All Variants
	Link time	--	
	Post-build time	--	
Scope / Dependency	scope: local		

SWS Item	[ECUC_Crypto_00036]	
Parameter Name	CryptoPrimitiveAlgorithmMode	
Parent Container	CryptoPrimitive	
Description	Determines the algorithm mode used for the crypto service	
Multiplicity	1..*	
Type	EcucEnumerationParamDef	
Range	CRYPTO_ALGOMODE_12ROUNDS	--
	CRYPTO_ALGOMODE_20ROUNDS	--
	CRYPTO_ALGOMODE_8ROUNDS	--
	CRYPTO_ALGOMODE_CBC	--
	CRYPTO_ALGOMODE_CFB	--
	CRYPTO_ALGOMODE_CMAC	--
	CRYPTO_ALGOMODE_CTR	--
	CRYPTO_ALGOMODE_CTRDRBG	--
	CRYPTO_ALGOMODE_CUSTOM	--
	CRYPTO_ALGOMODE_ECB	--
	CRYPTO_ALGOMODE_GCM	--
	CRYPTO_ALGOMODE_GMAC	--
	CRYPTO_ALGOMODE_HMAC	--
	CRYPTO_ALGOMODE_NOT_SET	--
	CRYPTO_ALGOMODE_OFB	--
	CRYPTO_ALGOMODE_PXXXR	--

	CRYPTO_ALGOMODE_RSAES_OAEP	--	
	CRYPTO_ALGOMODE_RSAES_PKCS1_v1_5	--	
	CRYPTO_ALGOMODE_RSASSA_PKCS1_ v1_5	--	
	CRYPTO_ALGOMODE_RSASSA_PSS	--	
	CRYPTO_ALGOMODE_SIPHASH_2_4	--	
	CRYPTO_ALGOMODE_SIPHASH_4_8	--	
	CRYPTO_ALGOMODE_XTS	--	
Post-Build Variant Value	false		
Multiplicity Configuration Class	**Pre-compile time**	X	All Variants
	Link time	--	
	Post-build time	--	
Value Configuration Class	**Pre-compile time**	X	All Variants
	Link time	--	
	Post-build time	--	
Scope / Dependency	scope: local		

SWS Item	[ECUC_Crypto_00037]	
Parameter Name	CryptoPrimitiveAlgorithmSecondaryFamily	
Parent Container	CryptoPrimitive	
Description	Determines the algorithm secondary family used for the crypto service	
Multiplicity	1..*	
Type	EcucEnumerationParamDef	
Range	CRYPTO_ALGOFAM_3DES	--
	CRYPTO_ALGOFAM_AES	--
	CRYPTO_ALGOFAM_BLAKE_1_256	--
	CRYPTO_ALGOFAM_BLAKE_1_512	--
	CRYPTO_ALGOFAM_BLAKE_2s_256	--
	CRYPTO_ALGOFAM_BLAKE_2s_512	--
	CRYPTO_ALGOFAM_BRAINPOOL	--

	CRYPTO_ALGOFAM_CHACHA	--
	CRYPTO_ALGOFAM_CUSTOM	--
	CRYPTO_ALGOFAM_DRBG	--
	CRYPTO_ALGOFAM_ECCANSI	--
	CRYPTO_ALGOFAM_ECCNIST	--
	CRYPTO_ALGOFAM_ECCSEC	--
	CRYPTO_ALGOFAM_ED25519	--
	CRYPTO_ALGOFAM_FIPS186	--
	CRYPTO_ALGOFAM_NOT_SET	--
	CRYPTO_ALGOFAM_PADDING_ ONEWITHZEROS	--
	CRYPTO_ALGOFAM_PADDING_PKCS7	--
	CRYPTO_ALGOFAM_POLY1305	--
	CRYPTO_ALGOFAM_RIPEMD160	--
	CRYPTO_ALGOFAM_RNG	--
	CRYPTO_ALGOFAM_RSA	--
	CRYPTO_ALGOFAM_SHA1	--
	CRYPTO_ALGOFAM_SHA2_224	--
	CRYPTO_ALGOFAM_SHA2_256	--
	CRYPTO_ALGOFAM_SHA2_384	--
	CRYPTO_ALGOFAM_SHA2_512	--
	CRYPTO_ALGOFAM_SHA2_512_224	--
	CRYPTO_ALGOFAM_SHA2_512_256	--
	CRYPTO_ALGOFAM_SHA3_224	--
	CRYPTO_ALGOFAM_SHA3_256	--
	CRYPTO_ALGOFAM_SHA3_384	--
	CRYPTO_ALGOFAM_SHA3_512	--
	CRYPTO_ALGOFAM_SHAKE128	--
	CRYPTO_ALGOFAM_SHAKE256	--
	CRYPTO_ALGOFAM_SIPHASH	--
	CRYPTO_ALGOFAM_X25519	--

Post-Build Variant Value	false		
Multiplicity Configuration Class	**Pre-compile time**	X	All Variants
	Link time	--	
	Post-build time	--	
Value Configuration Class	**Pre-compile time**	X	All Variants
	Link time	--	
	Post-build time	--	
Scope / Dependency	scope: local		

SWS Item	[ECUC_Crypto_00034]	
Parameter Name	CryptoPrimitiveService	
Parent Container	CryptoPrimitive	
Description	Determines the crypto service used for defining the capabilities	
Multiplicity	1	
Type	EcucEnumerationParamDef	
Range	CRYPTO_AEADDECRYPT	0x06
	CRYPTO_AEADENCRYPT	0x05
	CRYPTO_DECRYPT	0x04
	CRYPTO_ENCRYPT	0x03
	CRYPTO_HASH	0x00
	CRYPTO_KEYDERIVE	0x0E
	CRYPTO_KEYEXCHANGECALCPUBVAL	0x0F
	CRYPTO_KEYEXCHANGECALCSECRET	0x10
	CRYPTO_KEYGENERATE	0x0D
	CRYPTO_KEYSETINVALID	0x14
	CRYPTO_KEYSETVALID	0x13
	CRYPTO_MACGENERATE	0x01
	CRYPTO_MACVERIFY	0x02
	CRYPTO_RANDOMGENERATE	0x0B
	CRYPTO_RANDOMSEED	0x0C

	CRYPTO_SIGNATUREGENERATE	0x07	
	CRYPTO_SIGNATUREVERIFY	0x8	
	CUSTOM_SERVICE	0x15	
Multiplicity Configuration Class	**Pre-compile time**	X	All Variants
	Link time	--	
	Post-build time	--	
Value Configuration Class	**Pre-compile time**	X	All Variants
	Link time	--	
	Post-build time	--	
Scope / Dependency	scope: local		

SWS Item	[ECUC_Crypto_00053]		
Parameter Name	CryptoPrimitiveSupportContext		
Parent Container	CryptoPrimitive		
Description	Configures if the crypto primitive supports to store or restore context data of the workspace. Since this option is vulnerable to security, it shall only set to TRUE if absolutely needed.		
Multiplicity	0..1		
Type	EcucBooleanParamDef		
Default value	false		
Multiplicity Configuration Class	**Pre-compile time**	X	All Variants
	Link time	--	
	Post-build time	--	
Value Configuration Class	**Pre-compile time**	X	All Variants
	Link time	--	
	Post-build time	--	
Scope / Dependency	scope: local		

SWS Item	[ECUC_Crypto_00050]
Parameter Name	CryptoPrimitiveAlgorithmFamilyCustomRef
Parent Container	CryptoPrimitive
Description	Reference to a customer specific algorithm family custom container

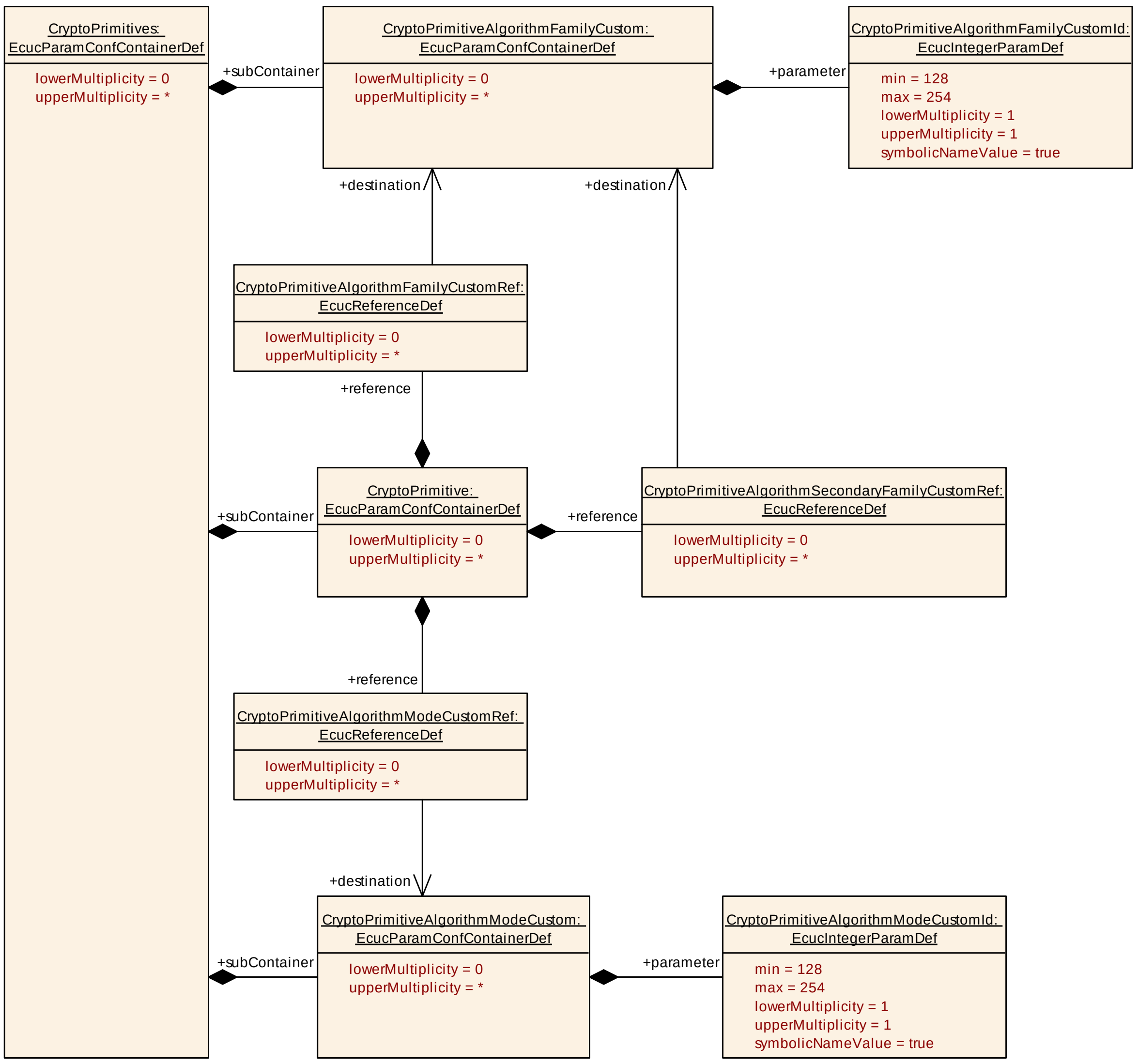
Multiplicity	0..*		
Type	Reference to CryptoPrimitiveAlgorithmFamilyCustom		
Multiplicity Configuration Class	**Pre-compile time**	X	All Variants
	Link time	--	
	Post-build time	--	
Value Configuration Class	**Pre-compile time**	X	All Variants
	Link time	--	
	Post-build time	--	
Scope / Dependency	scope: local dependency: This reference is only needed if the driver also supports custom specific features for PrimitiveAlgorithmFamily for the specific Crypto PrimitiveService.		

SWS Item	[ECUC_Crypto_00051]		
Parameter Name	CryptoPrimitiveAlgorithmModeCustomRef		
Parent Container	CryptoPrimitive		
Description	Reference to a customer specific algorithm mode custom container		
Multiplicity	0..*		
Type	Reference to CryptoPrimitiveAlgorithmModeCustom		
Multiplicity Configuration Class	**Pre-compile time**	X	All Variants
	Link time	--	
	Post-build time	--	
Value Configuration Class	**Pre-compile time**	X	All Variants
	Link time	--	
	Post-build time	--	
Scope / Dependency	dependency: This reference is only needed if the driver also supports custom specific features for PrimitiveAlgorithmMode for the specific Crypto PrimitiveService.		

SWS Item	[ECUC_Crypto_00052]
Parameter Name	CryptoPrimitiveAlgorithmSecondaryFamilyCustomRef
Parent Container	CryptoPrimitive
Description	Reference to a customer specific algorithm family custom container
Multiplicity	0..*

Type	Reference to CryptoPrimitiveAlgorithmFamilyCustom		
Multiplicity Configuration Class	**Pre-compile time**	X	All Variants
	Link time	--	
	Post-build time	--	
Value Configuration Class	**Pre-compile time**	X	All Variants
	Link time	--	
	Post-build time	--	
Scope / Dependency	scope: local dependency: This container is only needed if the driver also supports custom specific features for PrimitiveSecondaryAlgorithmFamily for the specific CryptoPrimitiveService.		

No Included Containers



10.1.13 CryptoPrimitiveAlgorithmFamilyCustom

SWS Item	[ECUC_Crypto_00046]
Container Name	CryptoPrimitiveAlgorithmFamilyCustom
Parent Container	CryptoPrimitives
Description	Container of custom algorithm family values. The container name serves as a symbolic name for the identifier of the custom algorithm family type.
Configuration Parameters	

SWS Item	[ECUC_Crypto_00047]
Parameter Name	CryptoPrimitiveAlgorithmFamilyCustomId

Parent Container	CryptoPrimitiveAlgorithmFamilyCustom		
Description	The custom value of this algorithm family		
Multiplicity	1		
Type	EcucIntegerParamDef (Symbolic Name generated for this parameter)		
Range	128 .. 254		
Default value	--		
Multiplicity Configuration Class	**Pre-compile time**	X	All Variants
	Link time	--	
	Post-build time	--	
Value Configuration Class	**Pre-compile time**	X	All Variants
	Link time	--	
	Post-build time	--	
Scope / Dependency	scope: local dependency: The CustomIds are used to distinguish the different custom algorithm family. Thus, every designated family shall have a unique ID that shall be fixed over updates and lifetime of the driver.		

No Included Containers

10.1.14 CryptoPrimitiveAlgorithmModeCustom

SWS Item	[ECUC_Crypto_00048]
Container Name	CryptoPrimitiveAlgorithmModeCustom
Parent Container	CryptoPrimitives
Description	Container of custom algorithm family values. The container name serves as a symbolic name for the identifier of the custom algorithm family type.
Configuration Parameters	

SWS Item	[ECUC_Crypto_00049]
Parameter Name	CryptoPrimitiveAlgorithmModeCustomId
Parent Container	CryptoPrimitiveAlgorithmModeCustom

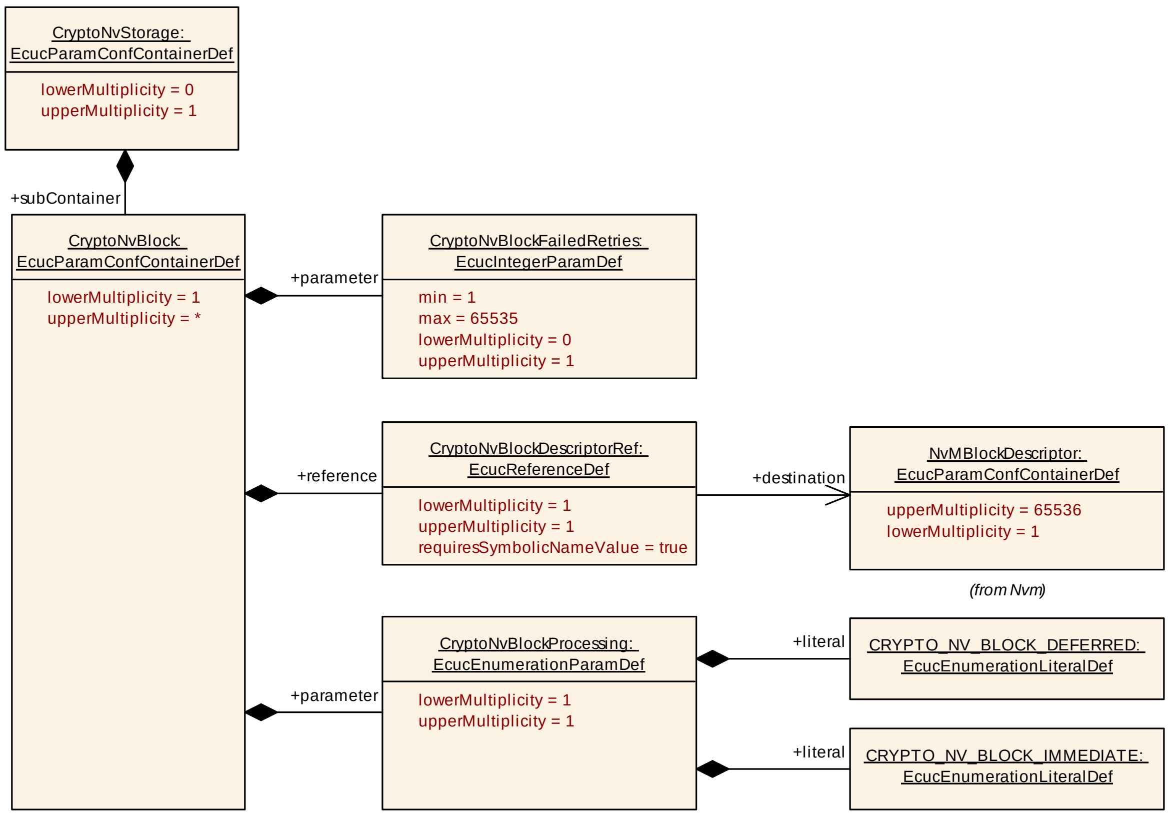
Description	The custom value of this algorithm mode		
Multiplicity	1		
Type	EcucIntegerParamDef (Symbolic Name generated for this parameter)		
Range	128 .. 254		
Default value	--		
Multiplicity Configuration Class	**Pre-compile time**	X	All Variants
	Link time	--	
	Post-build time	--	
Value Configuration Class	**Pre-compile time**	X	All Variants
	Link time	--	
	Post-build time	--	
Scope / Dependency	scope: local dependency: The CustomIds are used to distinguish the different custom algorithm mode. Thus, every designated mode shall have a unique ID that is fixed over updates and lifetime of the driver.		

No Included Containers

10.1.15 CryptoNvStorage

SWS Item	[ECUC_Crypto_00054]
Container Name	CryptoNvStorage
Parent Container	Crypto
Description	Container of NV block storage. Contains a collection of all NV storage blocks used for key storage.
Configuration Parameters	

Included Containers		
Container Name	**Multiplicity**	**Scope / Dependency**
CryptoNvBlock	1..*	Container to configure key storage in NVM


CryptoNvStorage: EcucParamConfContainerDef
lowerMultiplicity = 0
upperMultiplicity = 1
+subContainer
CryptoNvBlock: EcucParamConfContainerDef
lowerMultiplicity = 1
upperMultiplicity = *
+parameter
CryptoNvBlockFailedRetries: EcucIntegerParamDef
min = 1
max = 65535
lowerMultiplicity = 0
upperMultiplicity = 1
+reference
CryptoNvBlockDescriptorRef: EcucReferenceDef
lowerMultiplicity = 1
upperMultiplicity = 1
requiresSymbolicNameValue = true
+destination
NvMBlockDescriptor: EcucParamConfContainerDef
upperMultiplicity = 65536
lowerMultiplicity = 1
(from Nvm)
+parameter
CryptoNvBlockProcessing: EcucEnumerationParamDef
lowerMultiplicity = 1
upperMultiplicity = 1
+literal
CRYPTO_NV_BLOCK_DEFERRED: EcucEnumerationLiteralDef
+literal
CRYPTO_NV_BLOCK_IMMEDIATE: EcucEnumerationLiteralDef

10.1.16 CryptoNvBlock

SWS Item	[ECUC_Crypto_00055]
Container Name	CryptoNvBlock
Parent Container	CryptoNvStorage
Description	Container to configure key storage in NVM
Configuration Parameters	

SWS Item	[ECUC_Crypto_00057]	
Parameter Name	CryptoNvBlockFailedRetries	
Parent Container	CryptoNvBlock	
Description	Number of retries to request an NVM service operation.	
Multiplicity	0..1	
Type	EcucIntegerParamDef	
Range	1 .. 65535	

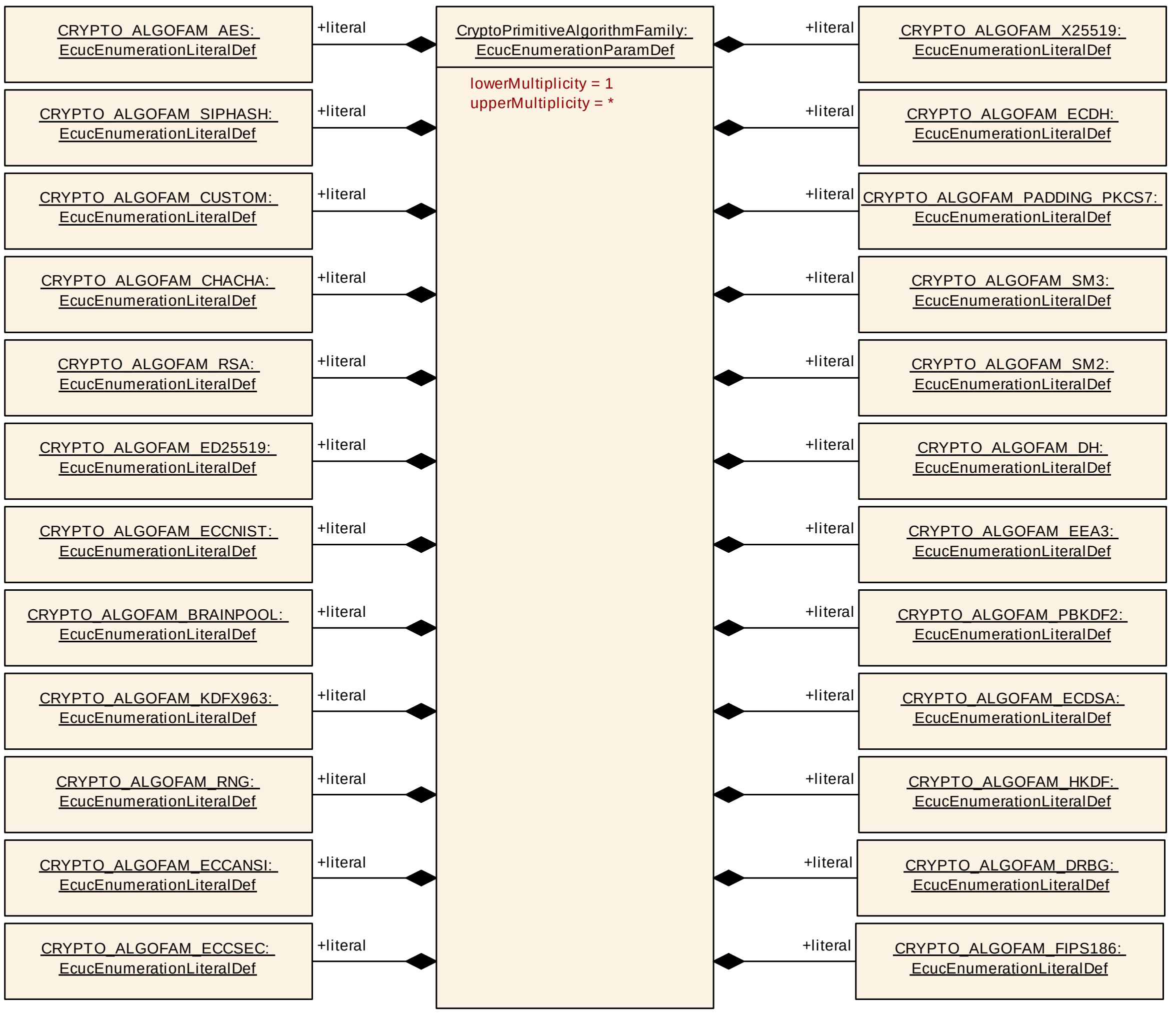
Default value	--		
Multiplicity Configuration Class	**Pre-compile time**	X	All Variants
	Link time	--	
	Post-build time	--	
Value Configuration Class	**Pre-compile time**	X	All Variants
	Link time	--	
	Post-build time	--	
Scope / Dependency	scope: local		

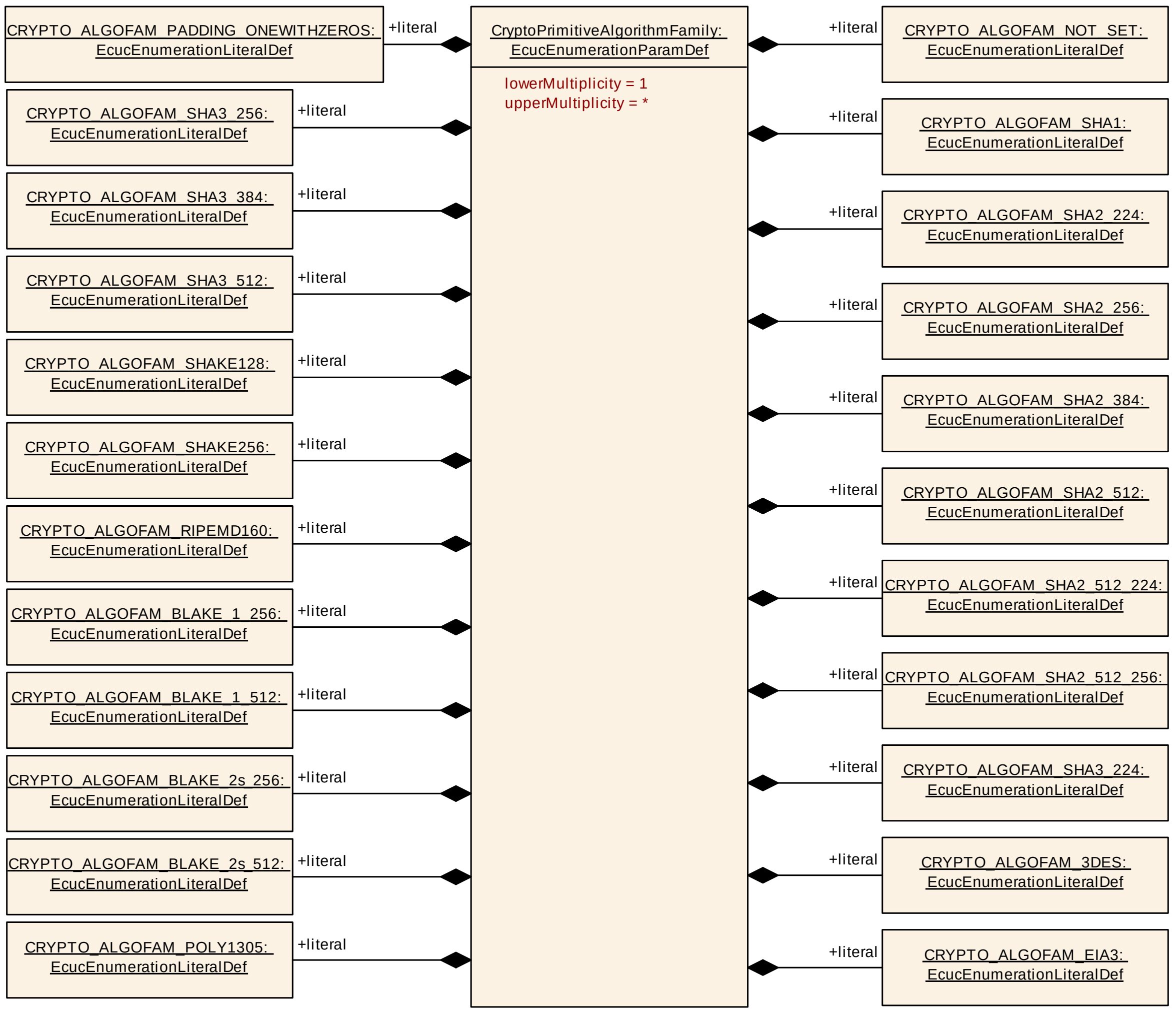
SWS Item	[ECUC_Crypto_00058]		
Parameter Name	CryptoNvBlockProcessing		
Parent Container	CryptoNvBlock		
Description	Selects the operation mode when an NV block shall be updated.		
Multiplicity	1		
Type	EcucEnumerationParamDef		
Range	CRYPTO_NV_BLOCK_DEFERRED	0x01	
	CRYPTO_NV_BLOCK_IMMEDIATE	0x02	
Multiplicity Configuration Class	**Pre-compile time**	X	All Variants
	Link time	--	
	Post-build time	--	
Value Configuration Class	**Pre-compile time**	X	All Variants
	Link time	--	
	Post-build time	--	
Scope / Dependency	scope: local		

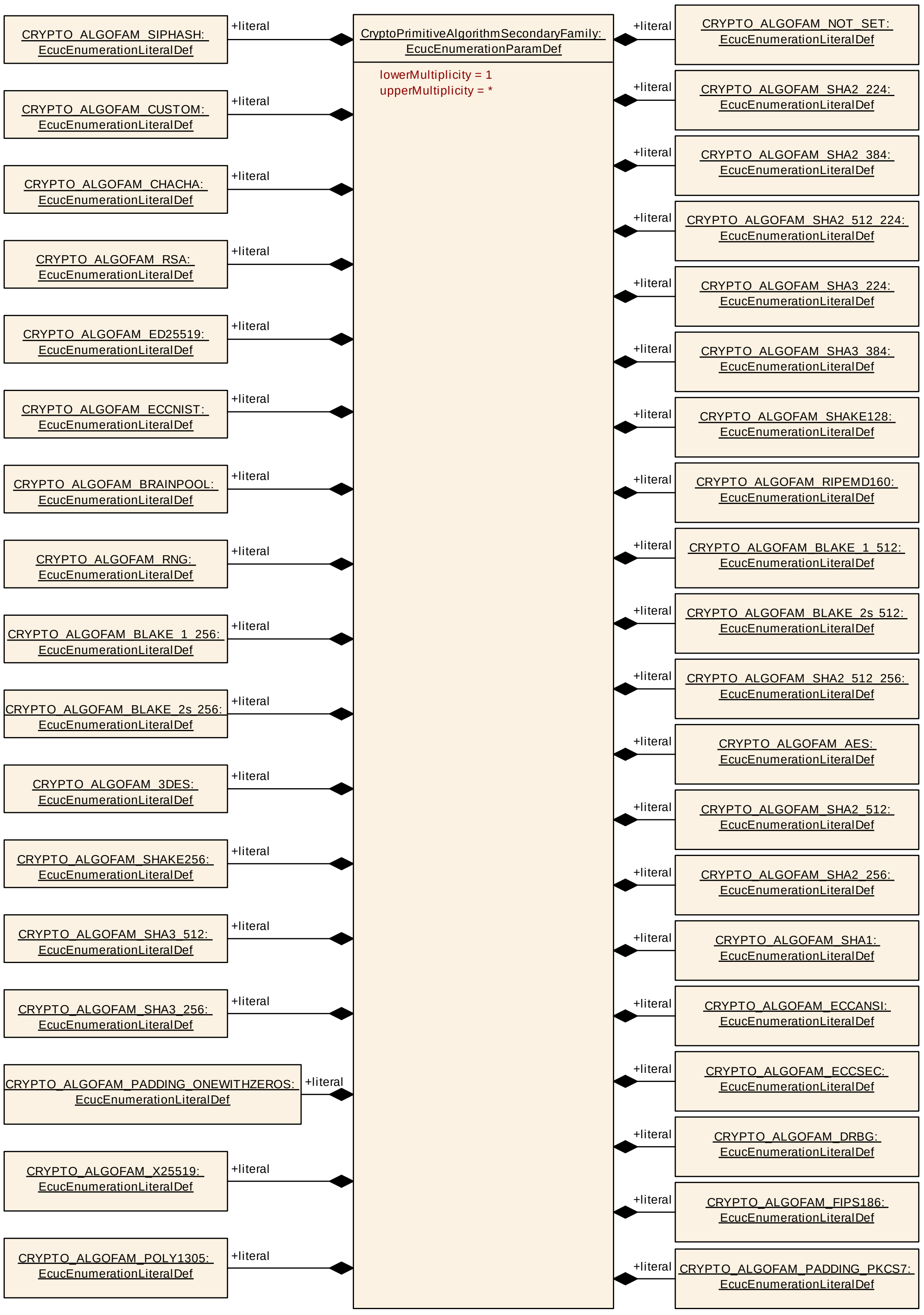
SWS Item	[ECUC_Crypto_00056]		
Parameter Name	CryptoNvBlockDescriptorRef		
Parent Container	CryptoNvBlock		
Description	Reference to an NvM block descriptor		
Multiplicity	1		
Type	Symbolic name reference to NvMBlockDescriptor		
Multiplicity Configuration Class	**Pre-compile time**	X	All Variants

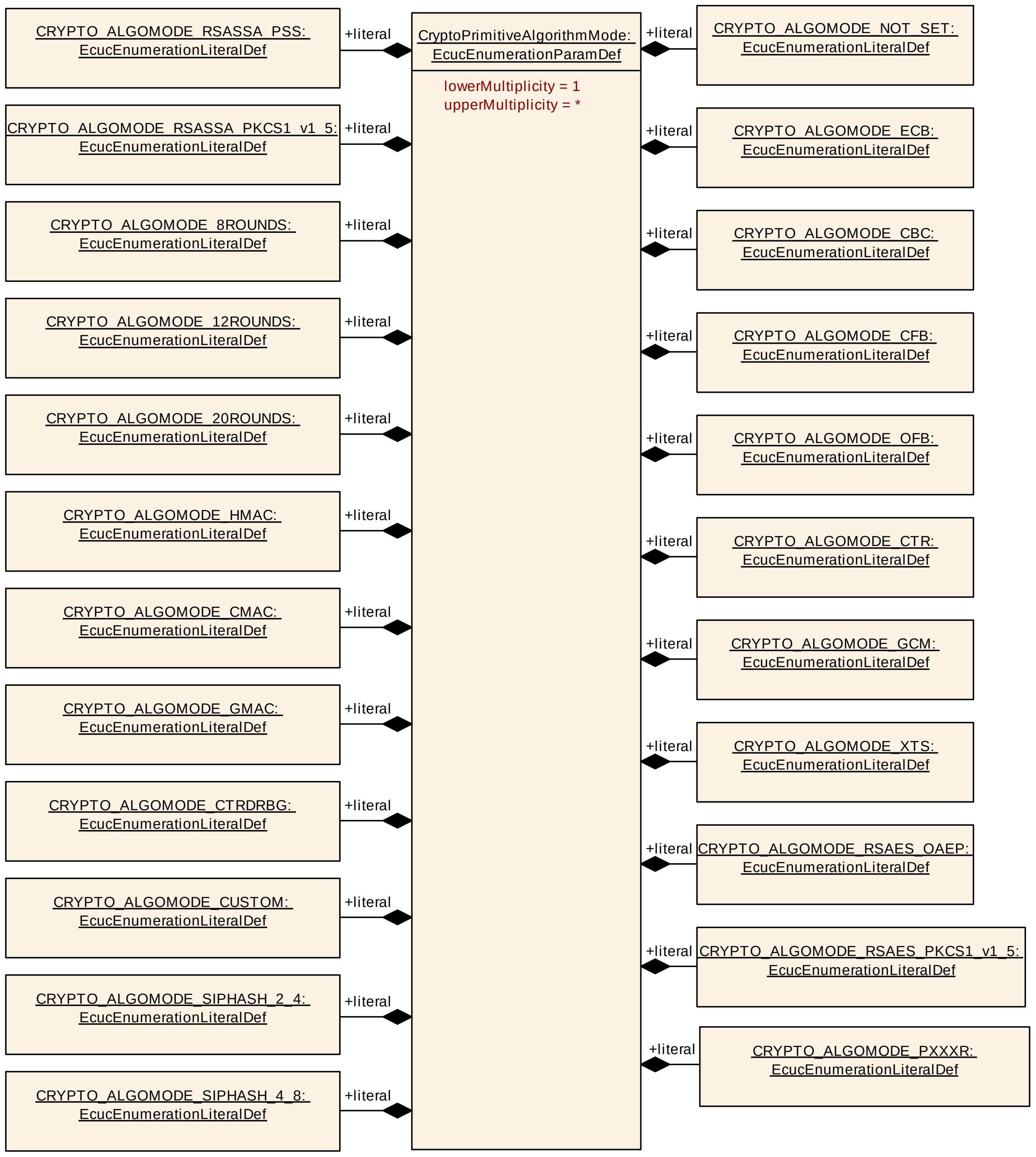
	Link time	--	
	Post-build time	--	
Value Configuration Class	**Pre-compile time**	X	All Variants
	Link time	--	
	Post-build time	--	
Scope / Dependency	scope: local		

No Included Containers


CRYPTO_ALGOFAM_AES: EcucEnumerationLiteralDef
CRYPTO_ALGOFAM_SIPHASH: EcucEnumerationLiteralDef
CRYPTO_ALGOFAM_CUSTOM: EcucEnumerationLiteralDef
CRYPTO_ALGOFAM_CHACHA: EcucEnumerationLiteralDef
CRYPTO_ALGOFAM_RSA: EcucEnumerationLiteralDef
CRYPTO_ALGOFAM_ED25519: EcucEnumerationLiteralDef
CRYPTO_ALGOFAM_ECCNIST: EcucEnumerationLiteralDef
CRYPTO_ALGOFAM_BRAINPOOL: EcucEnumerationLiteralDef
CRYPTO_ALGOFAM_KDFX963: EcucEnumerationLiteralDef
CRYPTO_ALGOFAM_RNG: EcucEnumerationLiteralDef
CRYPTO_ALGOFAM_ECCANSI: EcucEnumerationLiteralDef
CRYPTO_ALGOFAM_ECCSEC: EcucEnumerationLiteralDef
+literal
CryptoPrimitiveAlgorithmFamily: EcucEnumerationParamDef
lowerMultiplicity = 1
upperMultiplicity = *
CRYPTO_ALGOFAM_X25519: EcucEnumerationLiteralDef
CRYPTO_ALGOFAM_ECDH: EcucEnumerationLiteralDef
CRYPTO_ALGOFAM_PADDING_PKCS7: EcucEnumerationLiteralDef
CRYPTO_ALGOFAM_SM3: EcucEnumerationLiteralDef
CRYPTO_ALGOFAM_SM2: EcucEnumerationLiteralDef
CRYPTO_ALGOFAM_DH: EcucEnumerationLiteralDef
CRYPTO_ALGOFAM_EEA3: EcucEnumerationLiteralDef
CRYPTO_ALGOFAM_PBKDF2: EcucEnumerationLiteralDef
CRYPTO_ALGOFAM_ECDSA: EcucEnumerationLiteralDef
CRYPTO_ALGOFAM_HKDF: EcucEnumerationLiteralDef
CRYPTO_ALGOFAM_DRBG: EcucEnumerationLiteralDef
CRYPTO_ALGOFAM_FIPS186: EcucEnumerationLiteralDef


CryptoPrimitiveAlgorithmFamily: EcucEnumerationParamDef
lowerMultiplicity = 1
upperMultiplicity = *
+literal
CRYPTO_ALGOFAM_PADDING_ONEWITHZEROS: EcucEnumerationLiteralDef
CRYPTO_ALGOFAM_SHA3_256: EcucEnumerationLiteralDef
CRYPTO_ALGOFAM_SHA3_384: EcucEnumerationLiteralDef
CRYPTO_ALGOFAM_SHA3_512: EcucEnumerationLiteralDef
CRYPTO_ALGOFAM_SHAKE128: EcucEnumerationLiteralDef
CRYPTO_ALGOFAM_SHAKE256: EcucEnumerationLiteralDef
CRYPTO_ALGOFAM_RIPEMD160: EcucEnumerationLiteralDef
CRYPTO_ALGOFAM_BLAKE_1_256: EcucEnumerationLiteralDef
CRYPTO_ALGOFAM_BLAKE_1_512: EcucEnumerationLiteralDef
CRYPTO_ALGOFAM_BLAKE_2s_256: EcucEnumerationLiteralDef
CRYPTO_ALGOFAM_BLAKE_2s_512: EcucEnumerationLiteralDef
CRYPTO_ALGOFAM_POLY1305: EcucEnumerationLiteralDef
CRYPTO_ALGOFAM_NOT_SET: EcucEnumerationLiteralDef
CRYPTO_ALGOFAM_SHA1: EcucEnumerationLiteralDef
CRYPTO_ALGOFAM_SHA2_224: EcucEnumerationLiteralDef
CRYPTO_ALGOFAM_SHA2_256: EcucEnumerationLiteralDef
CRYPTO_ALGOFAM_SHA2_384: EcucEnumerationLiteralDef
CRYPTO_ALGOFAM_SHA2_512: EcucEnumerationLiteralDef
CRYPTO_ALGOFAM_SHA2_512_224: EcucEnumerationLiteralDef
CRYPTO_ALGOFAM_SHA2_512_256: EcucEnumerationLiteralDef
CRYPTO_ALGOFAM_SHA3_224: EcucEnumerationLiteralDef
CRYPTO_ALGOFAM_3DES: EcucEnumerationLiteralDef
CRYPTO_ALGOFAM_EIA3: EcucEnumerationLiteralDef


CryptoPrimitiveAlgorithmSecondaryFamily: EcucEnumerationParamDef
lowerMultiplicity = 1
upperMultiplicity = *
+literal
CRYPTO_ALGOFAM_SIPHASH: EcucEnumerationLiteralDef
CRYPTO_ALGOFAM_CUSTOM: EcucEnumerationLiteralDef
CRYPTO_ALGOFAM_CHACHA: EcucEnumerationLiteralDef
CRYPTO_ALGOFAM_RSA: EcucEnumerationLiteralDef
CRYPTO_ALGOFAM_ED25519: EcucEnumerationLiteralDef
CRYPTO_ALGOFAM_ECCNIST: EcucEnumerationLiteralDef
CRYPTO_ALGOFAM_BRAINPOOL: EcucEnumerationLiteralDef
CRYPTO_ALGOFAM_RNG: EcucEnumerationLiteralDef
CRYPTO_ALGOFAM_BLAKE_1_256: EcucEnumerationLiteralDef
CRYPTO_ALGOFAM_BLAKE_2s_256: EcucEnumerationLiteralDef
CRYPTO_ALGOFAM_3DES: EcucEnumerationLiteralDef
CRYPTO_ALGOFAM_SHAKE256: EcucEnumerationLiteralDef
CRYPTO_ALGOFAM_SHA3_512: EcucEnumerationLiteralDef
CRYPTO_ALGOFAM_SHA3_256: EcucEnumerationLiteralDef
CRYPTO_ALGOFAM_PADDING_ONEWITHZEROS: EcucEnumerationLiteralDef
CRYPTO_ALGOFAM_X25519: EcucEnumerationLiteralDef
CRYPTO_ALGOFAM_POLY1305: EcucEnumerationLiteralDef
CRYPTO_ALGOFAM_NOT_SET: EcucEnumerationLiteralDef
CRYPTO_ALGOFAM_SHA2_224: EcucEnumerationLiteralDef
CRYPTO_ALGOFAM_SHA2_384: EcucEnumerationLiteralDef
CRYPTO_ALGOFAM_SHA2_512_224: EcucEnumerationLiteralDef
CRYPTO_ALGOFAM_SHA3_224: EcucEnumerationLiteralDef
CRYPTO_ALGOFAM_SHA3_384: EcucEnumerationLiteralDef
CRYPTO_ALGOFAM_SHAKE128: EcucEnumerationLiteralDef
CRYPTO_ALGOFAM_RIPEMD160: EcucEnumerationLiteralDef
CRYPTO_ALGOFAM_BLAKE_1_512: EcucEnumerationLiteralDef
CRYPTO_ALGOFAM_BLAKE_2s_512: EcucEnumerationLiteralDef
CRYPTO_ALGOFAM_SHA2_512_256: EcucEnumerationLiteralDef
CRYPTO_ALGOFAM_AES: EcucEnumerationLiteralDef
CRYPTO_ALGOFAM_SHA2_512: EcucEnumerationLiteralDef
CRYPTO_ALGOFAM_SHA2_256: EcucEnumerationLiteralDef
CRYPTO_ALGOFAM_SHA1: EcucEnumerationLiteralDef
CRYPTO_ALGOFAM_ECCANSI: EcucEnumerationLiteralDef
CRYPTO_ALGOFAM_ECCSEC: EcucEnumerationLiteralDef
CRYPTO_ALGOFAM_DRBG: EcucEnumerationLiteralDef
CRYPTO_ALGOFAM_FIPS186: EcucEnumerationLiteralDef
CRYPTO_ALGOFAM_PADDING_PKCS7: EcucEnumerationLiteralDef


CryptoPrimitiveAlgorithmMode: EcucEnumerationParamDef
lowerMultiplicity = 1
upperMultiplicity = *
+literal
CRYPTO_ALGOMODE_RSASSA_PSS: EcucEnumerationLiteralDef
CRYPTO_ALGOMODE_RSASSA_PKCS1_v1_5: EcucEnumerationLiteralDef
CRYPTO_ALGOMODE_8ROUNDS: EcucEnumerationLiteralDef
CRYPTO_ALGOMODE_12ROUNDS: EcucEnumerationLiteralDef
CRYPTO_ALGOMODE_20ROUNDS: EcucEnumerationLiteralDef
CRYPTO_ALGOMODE_HMAC: EcucEnumerationLiteralDef
CRYPTO_ALGOMODE_CMAC: EcucEnumerationLiteralDef
CRYPTO_ALGOMODE_GMAC: EcucEnumerationLiteralDef
CRYPTO_ALGOMODE_CTRDRBG: EcucEnumerationLiteralDef
CRYPTO_ALGOMODE_CUSTOM: EcucEnumerationLiteralDef
CRYPTO_ALGOMODE_SIPHASH_2_4: EcucEnumerationLiteralDef
CRYPTO_ALGOMODE_SIPHASH_4_8: EcucEnumerationLiteralDef
CRYPTO_ALGOMODE_NOT_SET: EcucEnumerationLiteralDef
CRYPTO_ALGOMODE_ECB: EcucEnumerationLiteralDef
CRYPTO_ALGOMODE_CBC: EcucEnumerationLiteralDef
CRYPTO_ALGOMODE_CFB: EcucEnumerationLiteralDef
CRYPTO_ALGOMODE_OFB: EcucEnumerationLiteralDef
CRYPTO_ALGOMODE_CTR: EcucEnumerationLiteralDef
CRYPTO_ALGOMODE_GCM: EcucEnumerationLiteralDef
CRYPTO_ALGOMODE_XTS: EcucEnumerationLiteralDef
CRYPTO_ALGOMODE_RSAES_OAEP: EcucEnumerationLiteralDef
CRYPTO_ALGOMODE_RSAES_PKCS1_v1_5: EcucEnumerationLiteralDef
CRYPTO_ALGOMODE_PXXXR: EcucEnumerationLiteralDef

10.2 Published Information

Published information contains data defined by the implementer of the SW module that does not change when the module is adapted (i.e. configured) to the actual HW/SW environment. It thus contains version and manufacturer information.

Additional module-specific published parameters are listed below if applicable.