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

Document Status	Final
Part of AUTOSAR Standard	Classic Platform
Part of Standard Release	4.4.0

Document Change History

Date	Release	Changed by	Change Description
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 5
2 Acronyms and abbreviations 6
2.1 Glossary of Terms 6
3 Related documentation 8
3.1 Input documents 8
3.2 Related standards and norms 8
3.3 Related specification 8
4 Constraints and assumptions 9
4.1 Limitations 9
4.2 Applicability to car domains 9
5 Dependencies to other modules 10
5.1 File structure 10
5.1.1 Code File Structure 10
6 Requirements traceability 11
7 Functional specification 12
7.1 Pre-Configuration 12
7.1.1 Cryptographic capabilities 13
7.1.2 Available Keys 13
7.2 General Behavior 14
7.2.1 Normal Operation 15
7.2.2 Functional Requirements 18
7.2.3 Design Notes 18
7.2.4 Key Management 20
7.2.5 Key Formats 21
7.3 Error classification 25
7.3.1 Development Errors 25
7.3.2 Runtime Errors 25
7.3.3 Transient Faults 25
7.3.4 Production Errors 25
7.3.5 Extended Production Errors 26
8 API specification 27
8.1 Imported types 27
8.2 Type Definitions 28
8.3 Function definitions 28
8.3.1 General API 28
8.3.2 Job Processing Interface 30
8.3.3 Job Cancellation Interface 36
8.3.4 Key Management Interface 37
8.4 Scheduled functions 55
8.5 Expected Interfaces 55
8.5.1 Interfaces to Standard Software Modules 56

8.5.2 Mandatory Interfaces 56
8.5.3 Optional Interfaces 56

9 Sequence diagrams 57

10 Configuration specification 58

10.1 Containers and configuration parameters 58
10.1.1 Crypto 58
10.1.2 CryptoGeneral 59
10.1.3 CryptoDriverObjects 61
10.1.4 CryptoDriverObject 62
10.1.5 CryptoKeys 64
10.1.6 CryptoKey 65
10.1.7 CryptoKeyElements 66
10.1.8 CryptoKeyElement 68
10.1.9 CryptoKeyTypes 71
10.1.10 CryptoKeyType 72
10.1.11 CryptoPrimitives 72
10.1.12 CryptoPrimitive 73
10.2 Published Information 81

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)	
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.	
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 an instance of a job's configured cryptographic primitive.	
Crypto Primitive	A crypto primitive is an instance of a configured cryptographic algorithm realized in a Crypto Driver Object.	
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.	

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] Glossary
AUTOSAR_TR_Glossary

3.2 Related standards and norms

[8] 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
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
SRS_CryptoStack_00086	The CSM module shall distinguish between error types	SWS_Crypto_00040
SRS_CryptoStack_00098	The Crypto Driver shall provide access to all cryptographic algorithms supported by the hardware	SWS_Crypto_00013
SWS_BSW_00050	Check parameters passed to Initialization functions	SWS_Crypto_00215
SWS_BSW_00216	-	SWS_Crypto_91016

7 Functional specification

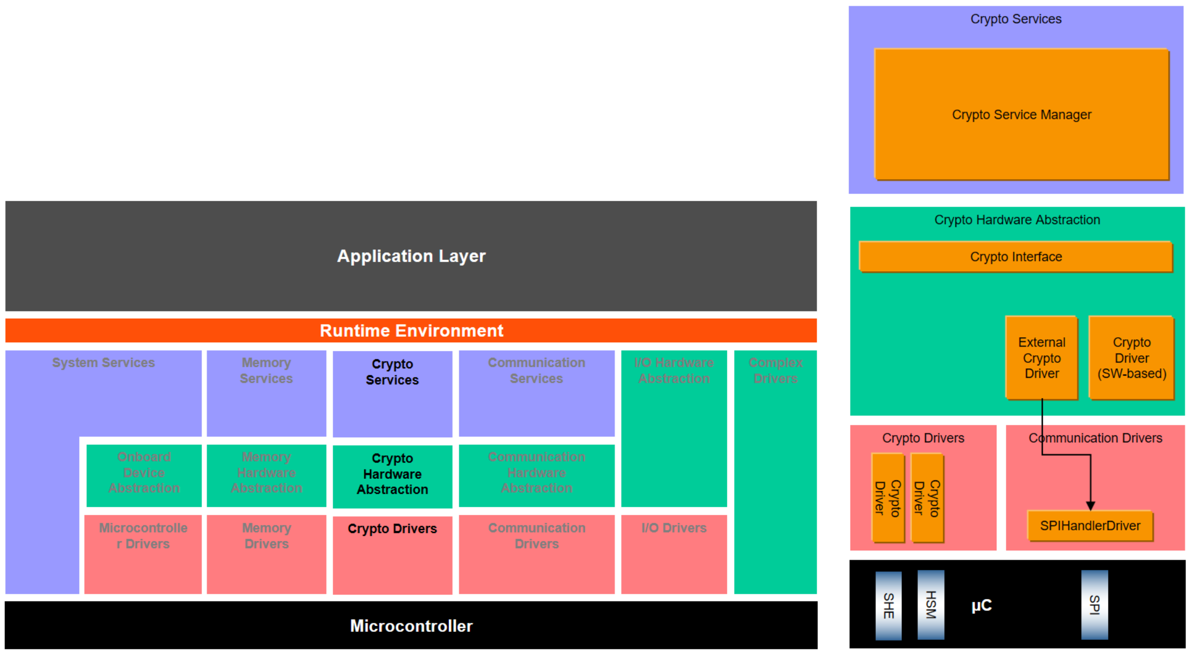


Figure 7.1: AUTOSAR Layered View with Crypto Driver Module

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 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.

7.1.2 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
- if the element is a virtual element

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.

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.
⌋()

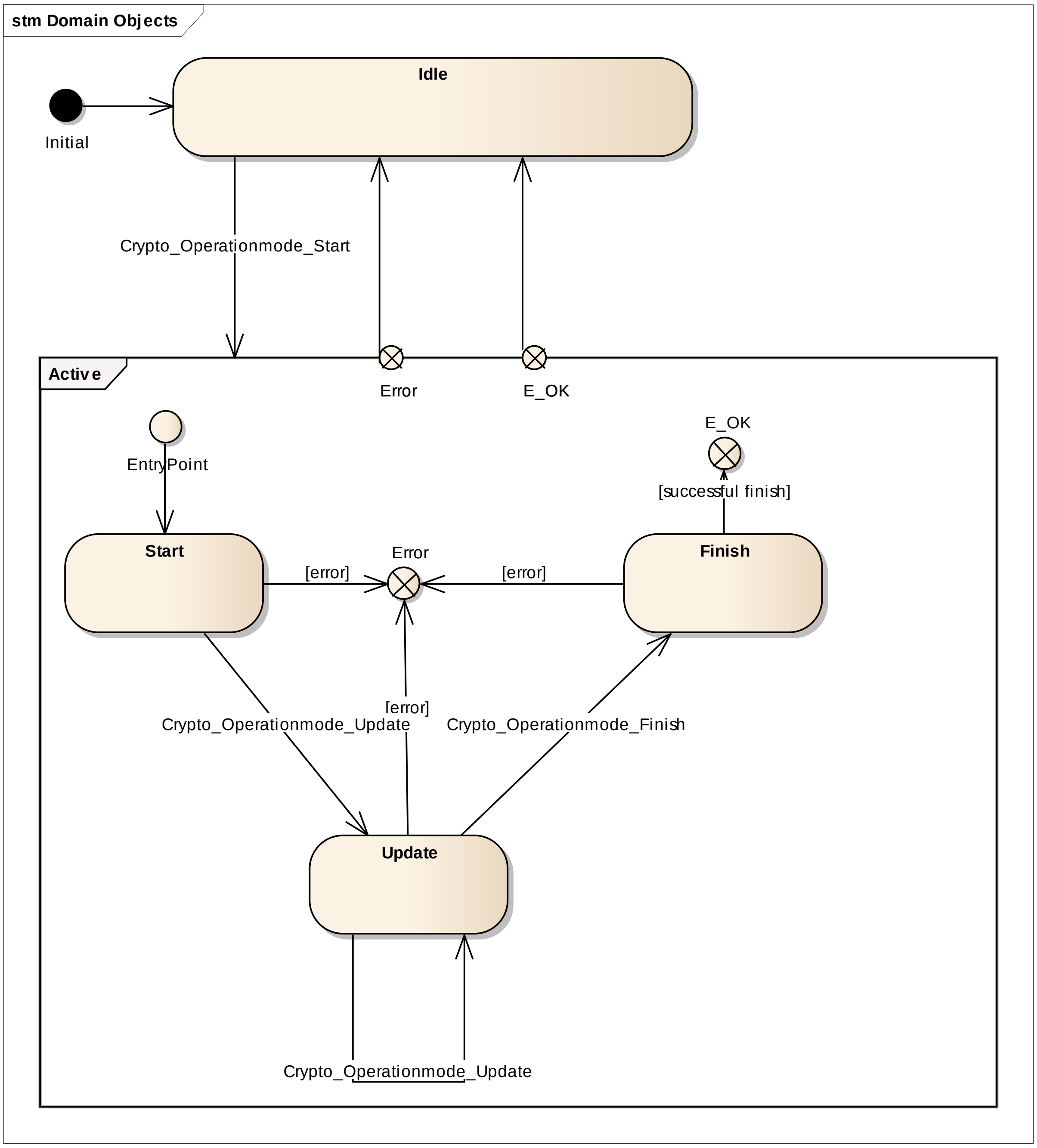
Note:
"job is being processed" means that the corresponding crypto driver object is currently and actively processing this job. When a job is not finished but the crypto driver object is not active with it (because, e.g., the operation "FINISH" is outstanding) this does not mean that this job is being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

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 `CRYPTO_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.

[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`.
⌋()

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. 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.

A key element can be configured as virtual. This way it behaves like a pointer to data which is stored in another element.

Example is the certificate. The certificate data is stored in one element. Elements like the issuer, public key of the certificate and signature only reference to the data somewhere in the main element with an offset without allocating own memory.

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. certificates. 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

[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)

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 RFC3447 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 q-1 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 ::= BIT_STRING {
   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]⌈ The RSA public key in the format CRYPTO_KE_FORMAT_BIN _IDENT_RSA_PUBLICKEY is provided as follows:

```
PublicKeyInfo ::= SEQUENCE {
   KeyAlgorithmIdentifier ::= AlgorithmIdentifier,
   publicKey ::= 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)

7.3 Error classification

7.3.1 Development Errors

[SWS_Crypto_00040] Development Error Types⌈

Type of error	*Related error code*	*Value [hex]*
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

⌋(SRS_CryptoStack_00086)

7.3.2 Runtime Errors

[SWS_Crypto_00194] Runtime Error Types⌈

Type of error	*Related error code*	*Value [hex]*
Buffer is too small for operation	CRYPTO_E_RE_SMALL_BUFFER	0x00
Requested key is not available	CRYPTO_E_RE_KEY_NOT_AVAILABLE	0x01
Key cannot be read	CRYPTO_E_RE_KEY_READ_FAIL	0x02
Entropy is too low	CRYPTO_E_RE_ENTROPY_EXHAUSTED	0x03

⌋ ()

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.

8 API specification

8.1 Imported types

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

[SWS_Crypto_00042] Imported Types

Module	*Header File*	*Imported Type*
Csm	<none>	Crypto_JobInfoType
	<none>	Crypto_JobType
	<none>	Crypto_VerifyResultType
Std_Types	StandardTypes.h	Std_ReturnType
	StandardTypes.h	Std_VersionInfoType

⌋()

The Crypto Stack API uses the following extension to Std_ReturnType:
[SWS_Crypto_00043] ⌈

Range:	`CRYPTO_E_BUSY`	`0x02`	The service request failed because the service is still busy
	`CRYPTO_E_SMALL_BUFFER`	`0x03`	The service request failed because the provided buffer is too small to store the result
	`CRYPTO_E_ENTROPY_EXHAUSTION`	`0x04`	The service request failed because the entropy of the random number generator is exhausted
	`CRYPTO_E_QUEUE_FULL`	`0x05`	The service request failed because the queue is full
	`CRYPTO_E_KEY_READ_FAIL`	`0x06`	The service request failed, because key element extraction is not allowed
	`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 the key 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.
Description:	--		
Available via:	`CryIf.h`		

⌋ ()

Note:

`CRYPTO_E_KEY_NOT_AVAILABLE` is meant to indicate that the key has been programmed before but cannot be accessed at the moment (for instance it is

temporarily not accessible, e.g. when the key is disabled due to debugger connection or parameters are wrong).
`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".)

The Crypto Stack API uses the key element index definition from the CSM module.
Type definitions
N/A.

8.2 Type Definitions

[SWS_Crypto_91016] ⌈

Name:	`Crypto_ConfigType`	
Type:	`Structure`	
Range:	`implementation specific`	The content of the configuration data structure is implementation specific.
Description:	Configuration data structure of CryIf module	
Available via:	`Crypto.h`	

⌋ (SWS_BSW_00216)

8.3 Function definitions

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

[SWS_Crypto_00195]⌈ If a Crypto API is called with a buffer too small to perform the desires operation CRYPTO_E_RE_SMALL_BUFFER shall be reported to the DET and the operation shall not be performed.
⌋ ()

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:	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:	Sync or Async, depends on the job 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_ReturnType	E_OK: Request successful E_NOT_OK: Request failed CRYPTO_E_BUSY: Request failed, Crypro Driver Object is busy CRYPTO_E_KEY_NOT_VALID: Request failed, the key is not valid CRYPTO_E_KEY_SIZE_MISMATCH: Request failed, a key element has the wrong size CRYPTO_E_QUEUE_FULL: Request failed, the queue is full CRYPTO_E_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: The service request failed because the key is not available CRYPTO_E_ENTROPY_EXHAUSTION: Request failed, the entropy is exhausted CRYPTO_E_SMALL_BUFFER: The provided buffer is too small to store the result CRYPTO_E_JOB_CANCELED: The service request failed because the synchronous Job has been canceled CRYPTO_E_KEY_EMPTY: Request failed because of uninitialized source key element
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] ⌈ If 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_00201]⌈
If the parameter job->jobPrimitiveInfo->primitiveInfo->service is set to CRYPTO_KEYSETVALID,
CRYPTO_RANDOMSEED, CRYPTO_KEYGENERATE, CRYPTO_KEYDERIVE, CRYPTO_KEYEXCHANGECALCPUBVAL,
CRYPTO_KEYEXCHANGECALCSECRET, CRYPTO_CERTIFICATEPARSE or CRYPTO_CERTIFICATEVERIFY, the parameter job->cryptoKeyId must be in range; else the function Crypto_ProcessJob shall report CRYPTO_E_PARAM_HANDLE to DET and return E_NOT_OK.
⌋()

[SWS_Crypto_00202]⌈
If the parameter job->jobPrimitiveInfo->primitiveInfo->service is set to CRYPTO_KEYDERIVE or CRYPTO_CERTIFICATEVERIFY, the parameter job->cryptoTargetKeyId 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`, the parameter `job->jobPrimitiveInfo->primitiveInfo->resultLength` is required.
If the configured result length of the job is smaller than the result length of the chosen algorithm, the most significant bits of the result shall be truncated to the configured result length.
⌋()

[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 pointer is required as an argument, but the value, which is pointed by the length pointer 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_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	UG	UG					F	F				SUF
MACGENERATE	UG	UG					F	F				SUF
MACVERIFY	UG	UG	F	F							F	SUF
ENCRYPT	UG	UG					UF	UF				SUF
DECRYPT	UG	UG					UF	UF				SUF
AEADENCRYPT	UG	UG	F	F			UF	UF	F	F		SUF
AEADDECRYPT	UG	UG	F	F	F	F	UF	UF			F	SUF
SIGNATUREGENERATE	UG	UG					F	F				SUF
SIGNATUREVERIFY	UG	UG	F	F							F	SUF

RANDOMGENERATE							F	F				

***:** 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

S: member required in Start mode.
U: member required in Update mode.
F: member required in Finish mode.
G: member optional in Finish 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	Target KeyId
KEYGENERATE						KeyId	
KEYDERIVE						KeyId	Target KeyId
KEYEXCHANGE CALCPUBVAL		Public Value				KeyId	
KEYEXCHANGE	Partner's						

CALCSECRET	Public Value					KeyId	
CERTIFICATE-PARSE						KeyId	
CERTIFICATE-VERIFY			Verify Ptr		Verify Ptr	KeyId	Target KeyId
KEYSETVALID						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.
If in addition input data is provided together with valid re-direction information and the crypto operation allows a data operation on both input data and key element data (e.g. for hash or MAC operation), then both data shall be processed. The key element data is always processed first, followed by the input data.

⌋()

Rationale: Some key generation operations require to combine a continuous key secret with additional data in a hash or MAC operation. The result of the operation is used as a secret key. The intention of this requirement is to leave all data in the crypto driver respectively in the HSM.

[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_00136] ⌈ If the buffer `job->jobPrimitiveInput.outputPtr` or `job->jobPrimitiveInput.secondaryOutputPtr` is too small, or in case of input/output re-direction the corresponding key element buffer is too small, to store the result of the request, `CRYPTO_E_SMALL_BUFFER` shall be returned and the function shall additionally report the runtime error `CRYPTO_E_RE_SMALL_BUFFER`.
⌋()

[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_JobInfoType* 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_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 (in):	cryptoKeyId	Holds the identifier of the key whose key element shall be set.
	keyElementId	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_ReturnType	E_OK: Request successful E_NOT_OK: Request failed CRYPTO_E_BUSY: Request failed, Crypto Driver Object is busy CRYPTO_E_KEY_WRITE_FAIL:Request failed because write access was denied CRYPTO_E_KEY_NOT_AVAILABLE: Request failed because the key is not available CRYPTO_E_KEY_SIZE_MISMATCH: Request failed, key element size does not match size of provided data
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_ReturnType	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”.

8.3.4.2 Key Extraction Interface
8.3.4.2.1 Crypto_KeyElementGet
[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):	cryptoKeyId	Holds the identifier of the key whose key element shall be returned.
	keyElementId	Holds the identifier of the key element which shall be returned.
Parameters (inout):	resultLengthPtr	Holds a pointer to a memory location in which the length information is stored. On calling this function this parameter shall contain the size of the buffer provided by resultPtr. If the key element is configured to allow partial access, this parameter contains the amount of data which should be read from the key element. The size may not be equal to the size of the provided buffer anymore. When the request has finished, the amount of data that has been stored shall be stored.
Parameters (out):	resultPtr	Holds the pointer of the buffer for the returned key element
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 is not available CRYPTO_E_KEY_READ_FAIL: Request failed because read access was denied CRYPTO_E_SMALL_BUFFER: The provided buffer is too small to store the result CRYPTO_E_KEY_EMPTY: Request failed because of uninitialized source key element
Description:	This interface shall be used to get a key element of the key identified by the cryptoKeyId and store the key element in the memory location pointed by the result pointer. Note: If the actual key element is directly mapped to flash memory, there could be a bigger delay when calling this function (synchronous operation).	
Available via:	`Crypto.h`	

⌋ ()

[SWS_Crypto_00140] ⌈ If the function `Crypto_KeyElementGet` returns `CRYPTO_E_KEY_NOT_AVAILABLE`, the function shall additionally report the runtime error `CRYPTO_E_RE_KEY_NOT_AVAILABLE`.
⌋()

[SWS_Crypto_00139] ⌈ If the function `Crypto_KeyElementGet` returns `CRYPTO_E_KEY_READ_FAIL`, the function shall additionally report the runtime error `CRYPTO_E_RE_KEY_READ_FAIL`.
⌋()

[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] ⌈ If 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 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_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 Copying Interface

8.3.4.3.1 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.
	targetCryptoKeyId	Holds the identifier of the key whose key element shall be the destination element.
	targetKeyElementId	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 E_BUSY: Request failed, Crypto Driver Object is busy CRYPTO_E_KEY_NOT_AVAILABLE: Request failed, the requested key element is not available CRYPTO_E_KEY_READ_FAIL: Request failed, not allowed to extract key element CRYPTO_E_KEY_WRITE_FAIL: Request failed, not allowed to write key element CRYPTO_E_KEY_SIZE_MISMATCH: Request failed, key element sizes are not compatible CRYPTO_E_KEY_EMPTY: Request failed 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`.

8.3.4.3.2 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.
	keyElementSourceOffset	This is the offset of the of the source key element indicating the start index of the copy operation.
	keyElementTargetOffset	This is the offset of the of the target key element indicating the start index of the copy operation.
	keyElementCopyLength	Specifies the number of bytes that shall be copied.
	targetCryptoKeyId	Holds the identifier of the key whose key element shall be the destination element.
	targetKeyElementId	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 E_BUSY: Request failed, Crypto Driver Object is busy CRYPTO_E_KEY_NOT_AVAILABLE: Request failed, the requested key element is not available

		CRYPTO_E_KEY_READ_FAIL: Request failed, not allowed to extract key element CRYPTO_E_KEY_WRITE_FAIL: Request failed, not allowed to write key element CRYPTO_E_KEY_SIZE_MISMATCH: Request failed, key element sizes are not compatible CRYPTO_E_KEY_EMPTY: Request failed because of uninitialized source key element
Description:	Copies a key element to another key element in the same crypto driver. The keyElementSourceOffset 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`).
⌋()

8.3.4.3.3 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.
	targetCryptoKeyId	Holds the identifier of the key whose key element shall be the destination element.
Parameters (inout):	None	
Parameters (out):	None	
Return value:	Std_ReturnType	E_OK: Request successful E_NOT_OK: Request failed E_BUSY: Request failed, Crypto Driver Object is busy CRYPTO_E_KEY_NOT_AVAILABLE: Request failed, the requested key element is not available CRYPTO_E_KEY_READ_FAIL: Request failed, not allowed to extract key element CRYPTO_E_KEY_WRITE_FAIL: Request failed, not allowed to write key element CRYPTO_E_KEY_SIZE_MISMATCH: Request failed, key element sizes are not compatible CRYPTO_E_KEY_EMPTY: Request failed 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.`
⌋()

8.3.4.3.4 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):	cryptoKeyId	Holds the identifier of the key whose available element ids shall be exported.
	keyElementIdsLengthPtr	Holds a pointer to the memory location in which the number of key elements in the given key is stored. On calling this function, this parameter shall contain the size of the buffer provided by keyElementIdsPtr. When the request has finished, the actual number of key elements shall be stored.
Parameters (inout):	None	
Parameters (out):	keyElementIdsPtr	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 E_BUSY: Request failed, Crypto Driver Object is busy CRYPTO_E_SMALL_BUFFER: The provided buffer is too small to store the result
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.

8.3.4.4 Key Generation Interface

8.3.4.4.1 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_ReturnType	E_OK: Request successful E_NOT_OK: Request failed
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`.

8.3.4.4.2 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_ReturnType	E_OK: Request successful E_NOT_OK: Request failed E_BUSY: Request failed, Crypto Driver Object is busy CRYPTO_E_KEY_EMPTY: Request failed because of uninitialized source key element
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.
⌋()

8.3.4.5 Key Derivation Interface

8.3.4.5.1 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.
	targetCryptoKeyId	Holds the identifier of the key which is used to store the derived key.
Parameters (inout):	None	
Parameters (out):	None	
Return value:	Std_ReturnType	E_OK: Request successful E_NOT_OK: Request failed E_BUSY: Request failed, Crypto Driver Object is busy CRYPTO_E_KEY_EMPTY: Request failed because of uninitialized source key element
Description:	Derives a new key by using the key elements in the given key identified by the cryptoKeyId. The given key contains the key elements for the password, salt. The derived key is stored in the key element with the id 1 of the key identified by targetCryptoKeyId. The number of iterations is given in the key element CRYPTO_KE_KEYDERIVATION_ITERATIONS.	
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`.

8.3.4.6 Key Exchange Interface

8.3.4.6.1 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 (inout):	publicValueLengthPtr	Holds a pointer to the memory location in which the public value length information is stored. On calling this function, this parameter shall contain the size of the buffer provided by publicValuePtr. When the request has finished, the actual length of the returned value shall be stored.
Parameters (out):	publicValuePtr	Contains the pointer to the data where the public value shall be stored.
Return value:	Std_ReturnType	E_OK: Request successful E_NOT_OK: Request failed E_BUSY: Request failed, Crypto Driver Object is busy CRYPTO_E_SMALL_BUFFER: The provided buffer is too small to store the result CRYPTO_E_KEY_EMPTY: Request failed because of uninitialized source key element
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.
⌋()

[SWS_Crypto_00110] ⌈ If the buffer `publicValuePtr` is too small to store the result of the request, `CRYPTO_E_SMALL_BUFFER` shall be returned and the function shall additionally report the runtime error `CRYPTO_E_RE_SMALL_BUFFER`.
⌋()

8.3.4.6.2 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.
	partnerPublicValuePtr	Holds the pointer to the memory location which contains the partner's public value.
	partnerPublicValueLength	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 E_BUSY: Request failed, Crypto Driver Object is busy CRYPTO_E_SMALL_BUFFER: The provided buffer is too small to store the result CRYPTO_E_KEY_EMPTY: Request failed because of uninitialized source key element
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`.

8.3.4.7 Certificate Interface

8.3.4.7.1 Crypto_CertificateParse

[SWS_Crypto_91011] ⌈

Service name:	Crypto_CertificateParse	
Syntax:	`Std_ReturnType Crypto_CertificateParse( uint32 cryptoKeyId )`	
Service ID[hex]:	0x0b	
Sync/Async:	Synchronous	
Reentrancy:	Reentrant, but not for the same cryptoKeyId	
Parameters (in):	cryptoKeyId	Holds the identifier of the key which shall be parsed.
Parameters (inout):	None	
Parameters (out):	None	
Return value:	Std_ReturnType	E_OK: Request successful E_NOT_OK: Request failed E_BUSY: Request failed, Crypto Driver Object is busy CRYPTO_E_KEY_EMPTY: Request failed because of uninitialized source key element
Description:	Parses the certificate data stored in the key element CRYPTO_KE_CERT_DATA and fills the key elements CRYPTO_KE_CERT_SIGNEDDATA, CRYPTO_KE_CERT_PARSEDPUBLICKEY and CRYPTO_KE_CERT_SIGNATURE.	
Available via:	`Crypto.h`	

⌋ ()

[SWS_Crypto_00168] ⌈ If the module is not yet initialized and if development error detection for the Crypto Driver is enabled, the function `Crypto_CertificateParse` shall report `CRYPTO_E_UNINIT` to the DET and return `E_NOT_OK`.
⌋()

[SWS_Crypto_00169] ⌈ If the parameter `cryptoKeyId` is out of range and if development error detection for the Crypto Driver is enabled, the function `Crypto_CertificateParse` shall report `CRYPTO_E_PARAM_HANDLE` to the DET and return `E_NOT_OK`.
⌋()

[SWS_Crypto_00170] ⌈ If no errors are detected by Crypto Driver, the service `Crypto_CertificateParse()` parses the certificate which is stored in the certificate data element and fills at least the key elements `CRYPTO_KE_CERT_SIGNEDDATA`, `CRYPTO_KE_CERT_PARSEDPUBLICKEY` and `CRYPTO_KE_CERT_SIGNATURE` with the corresponding data.
⌋()

Note: These key elements may be virtual and point with an offset to the certificate data, which is stored in the key element `CRYPTO_KE_CERT_DATA`, in order to save memory. The offset can not be read or written by the CRYIF.

8.3.4.7.2 Crypto_CertificateVerify

[SWS_Crypto_00171] ⌈

Service name:	Crypto_CertificateVerify	

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

Service ID[hex]:	0x12	
Sync/Async:	Synchronous	
Reentrancy:	Reentrant, but not for the same cryptoKeyId	
Parameters (in):	cryptoKeyId	Holds the identifier of the key which shall be used to validate the certificate.
	verifyCryptoKeyId	Holds the identifier of the key contain
Parameters (inout):	None	
Parameters (out):	verifyPtr	Holds a pointer to the memory location which will contain the result of the certificate verification.
Return value:	Std_ReturnType	E_OK: Request successful E_NOT_OK: Request failed E_BUSY: Request failed, Crypto Driver Object is busy CRYPTO_E_KEY_EMPTY: Request failed because of uninitialized source key element
Description:	Verifies the certificate stored in the key referenced by cryptoValidateKeyId with the certificate stored in the key referenced by cryptoKeyId.	
Available via:	`Crypto.h`	

⌋ ()

[SWS_Crypto_00172] ⌈ If the module is not yet initialized and if development error detection for the Crypto Driver is enabled, the function `Crypto_CertificateVerify` shall report `CRYPTO_E_UNINIT` to the DET and return `E_NOT_OK`.
⌋()

[SWS_Crypto_00173] ⌈ If the parameter `cryptoKeyId` is out of range and if development error detection for the Crypto Driver is enabled, the function `Crypto_CertificateVerify` shall report `CRYPTO_E_PARAM_HANDLE` to the DET and return `E_NOT_OK`.
⌋()

[SWS_Crypto_00174] ⌈ If the parameter `verifyCryptoKeyId` is out of range and if development error detection for the Crypto Driver is enabled, the function `Crypto_CertificateVerify` shall report `CRYPTO_E_PARAM_HANDLE` to the DET and return `E_NOT_OK`.
⌋()

[SWS_Crypto_00175] ⌈ If the parameter `verifyPtr` is a null pointer and if development error detection for the Crypto Driver is enabled, the function `Crypto_CertificateVerify` shall report `CRYPTO_E_PARAM_POINTER` to the DET and return `E_NOT_OK`.
⌋()

[SWS_Crypto_00176] ⌈ If the key element `CRYPTO_KE_CERTIFICATE_CURRENT_TIME` is used during verification and the format of this timestamp does not match with the format of the timestamp of the certificate, the function `Crypto_CertificateVerify` shall report `CRYPTO_E_PARAM_HANDLE` to the DET and return `E_NOT_OK`.
⌋()

[SWS_Crypto_00177] ⌈ If no errors are detected by Crypto Driver, the service `Crypto_CertificateVerify()` uses the key element `CRYPTO_KE_CERT_PARSEDPUBLICKEY` of the key referenced by `cryptoKeyId` to do a signature verification. The data contained in the key element `CRYPTO_KE_CERT_SIGNEDDATA` and the signature contained in the key element `CRYPTO_KE_CERT_SIGNATURE` shall be available in the key referenced by `verifyCryptoKeyId`.
⌋()

[SWS_Crypto_00178] ⌈ If certificate identified by `verifyCryptoKeyId` is verified successfully, the key identified by `validateCryptoKeyId` shall be set to valid.
⌋()

Note: The Function may also do further certificate validation by checking if the current time is in the validity period of the certificate and if the issuer of the key referenced by `validateCryptoKeyId` is the same as the subject in the key referenced by `cryptoKeyId`.

8.4 Scheduled functions

8.4.1.1 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`

⌋ ()

8.5 Expected Interfaces

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

8.5.1 Interfaces to Standard Software Modules

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

8.5.2 Mandatory Interfaces

API function	***Header File***	***Description***

8.5.3 Optional Interfaces

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

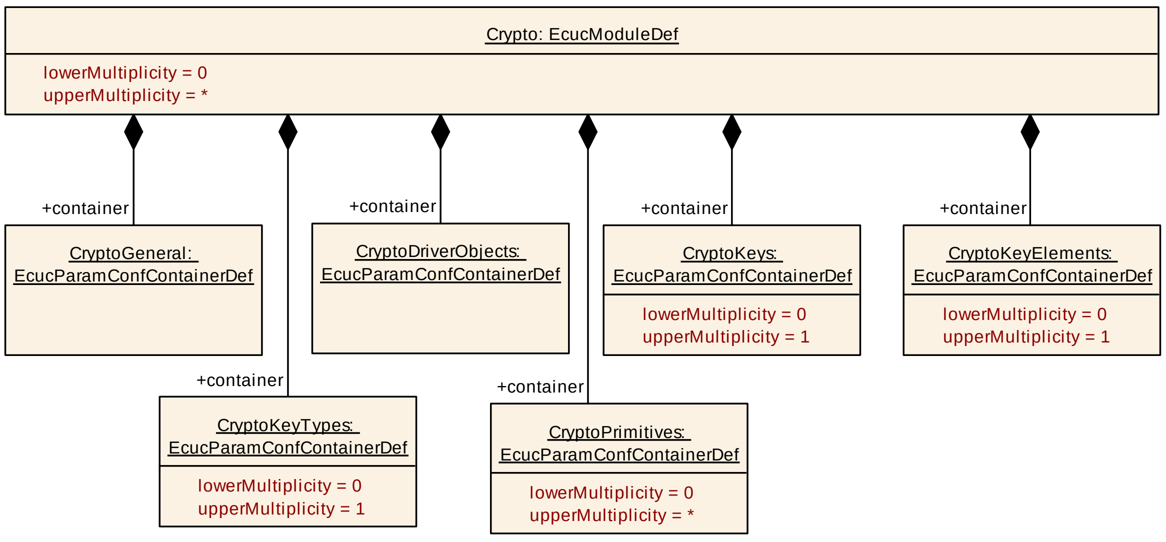
The following chapters summarize all configuration parameters. The detailed meanings of the parameters describe Chapters 7 and Chapter 8.

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*
Module 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***
CryptoDriverObjects	1	Container for CRYPTO Objects
CryptoGeneral	1	Container for common configuration options
CryptoKeyElements	0..1	Container for Crypto key elements
CryptoKeyTypes	0..1	Container for CRYPTO key types
CryptoKeys	0..1	Container for CRYPTO keys
CryptoPrimitives	0..*	Container for CRYPTO primitives



10.1.2 CryptoGeneral

SWS Item	**ECUC_Crypto_00002 :**		
Container Name	CryptoGeneral		
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 :**		
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 :**		
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 Configuration Class	***Pre-compile time***	X	All Variants
	Link time	--	
	Post-build time	--	
Scope / Dependency	scope: local		

SWS Item	**ECUC_Crypto_00038 :**		
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		

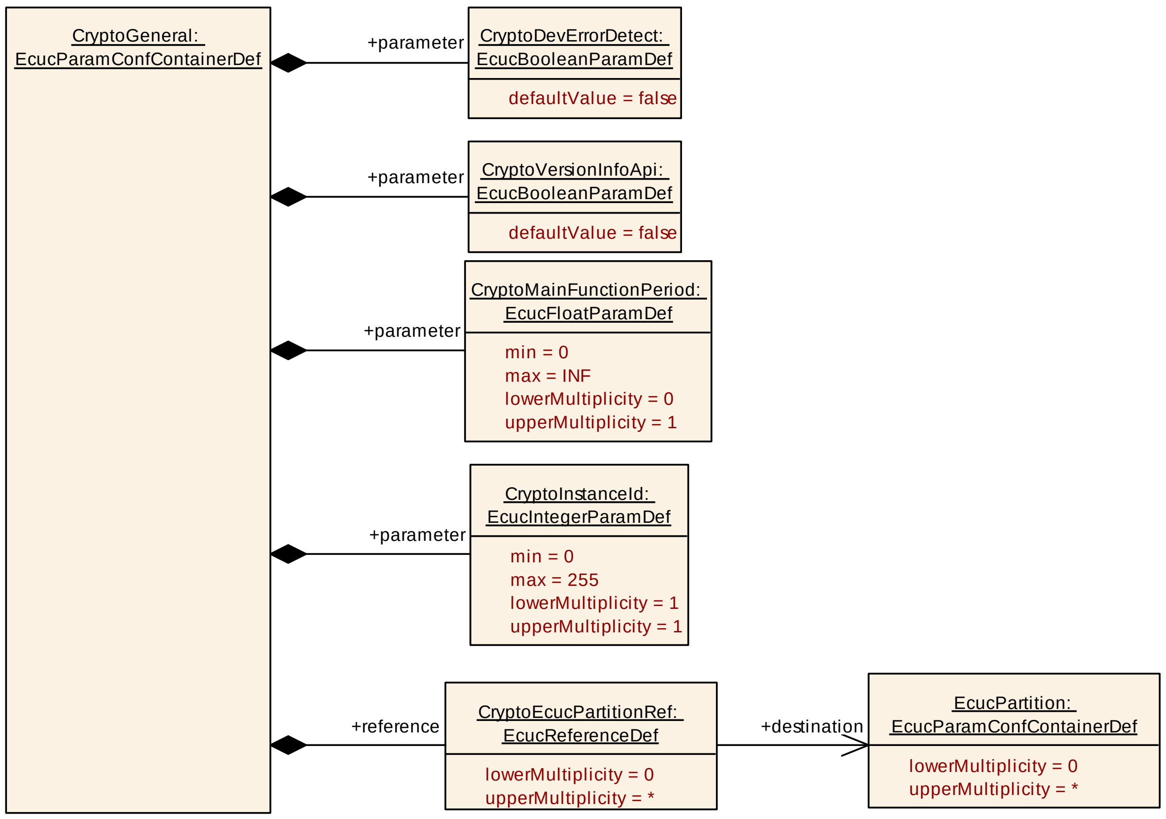
SWS Item	**ECUC_Crypto_00007 :**		
Name	CryptoVersionInfoApi		
Parent Container	CryptoGeneral		
Description	Pre-processor switch to enable and disable availability of the API Crypto_GetVersionInfo(). True: API Crypto_GetVersionInfo() is available False: API Cryptosm_GetVersionInfo() is not available.		
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 :**		
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. **Tags:** atp.Status=draft		
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] Draft ⌈ The Crypto Driver module shall reject configurations with partition mappings which are not supported by the implementation.
⌋()

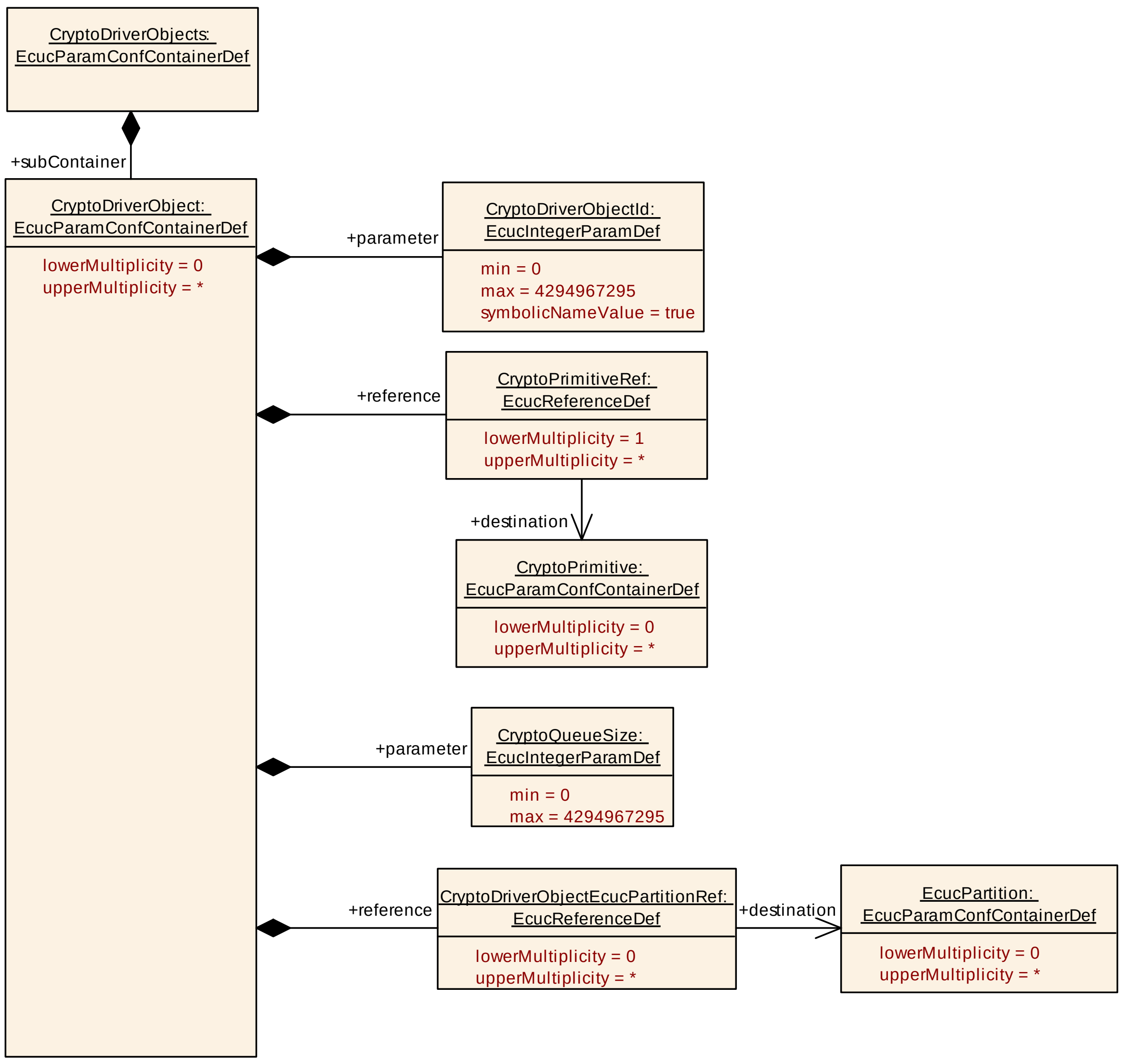
[SWS_Crypto_CONSTR_00001] Draft ⌈ The Crypto Driver module will operate as an independent instance in each of the partitions, means the called API will only target the partition it is called in.
⌋()



10.1.3 CryptoDriverObjects

SWS Item	**ECUC_Crypto_00003 :**		
Container Name	CryptoDriverObjects		
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
Description	Configuration of a CryptoDriverObject
Configuration Parameters	

SWS Item	**ECUC_Crypto_00009 :**	
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 :**		
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_00043 :**		
Name	CryptoDriverObjectEcucPartitionRef		
Parent Container	CryptoDriverObject		
Description	Maps a crypto driver object to zero or multiple ECUC partitions. The ECUC partition referenced is a subset of the ECUC partitions where the Crypto driver is mapped to. Note: CryptoDriverObjects such as a HSM shall be mapped to one partition only. **Tags:** atp.Status=draft		
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		

SWS Item	**ECUC_Crypto_00018 :**
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

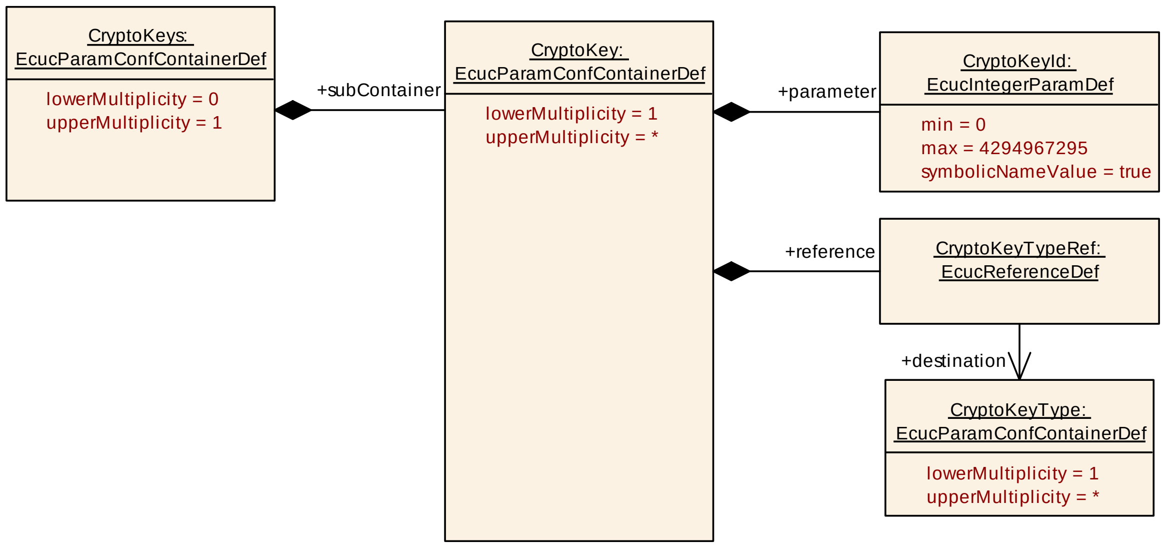
[SWS_Crypto_CONSTR_00002] Draft ⌈ The ECUC partitions referenced by CryptoDriverObjectEcucPartitionRef shall be a subset of the ECUC partitions referenced by CryptoEcucPartitionRef.
⌋()

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

10.1.5 CryptoKeys

SWS Item	**ECUC_Crypto_00004 :**
Container Name	CryptoKeys
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		
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 :**		
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_00020 :**
Name	CryptoKeyTypeRef
Parent Container	CryptoKey

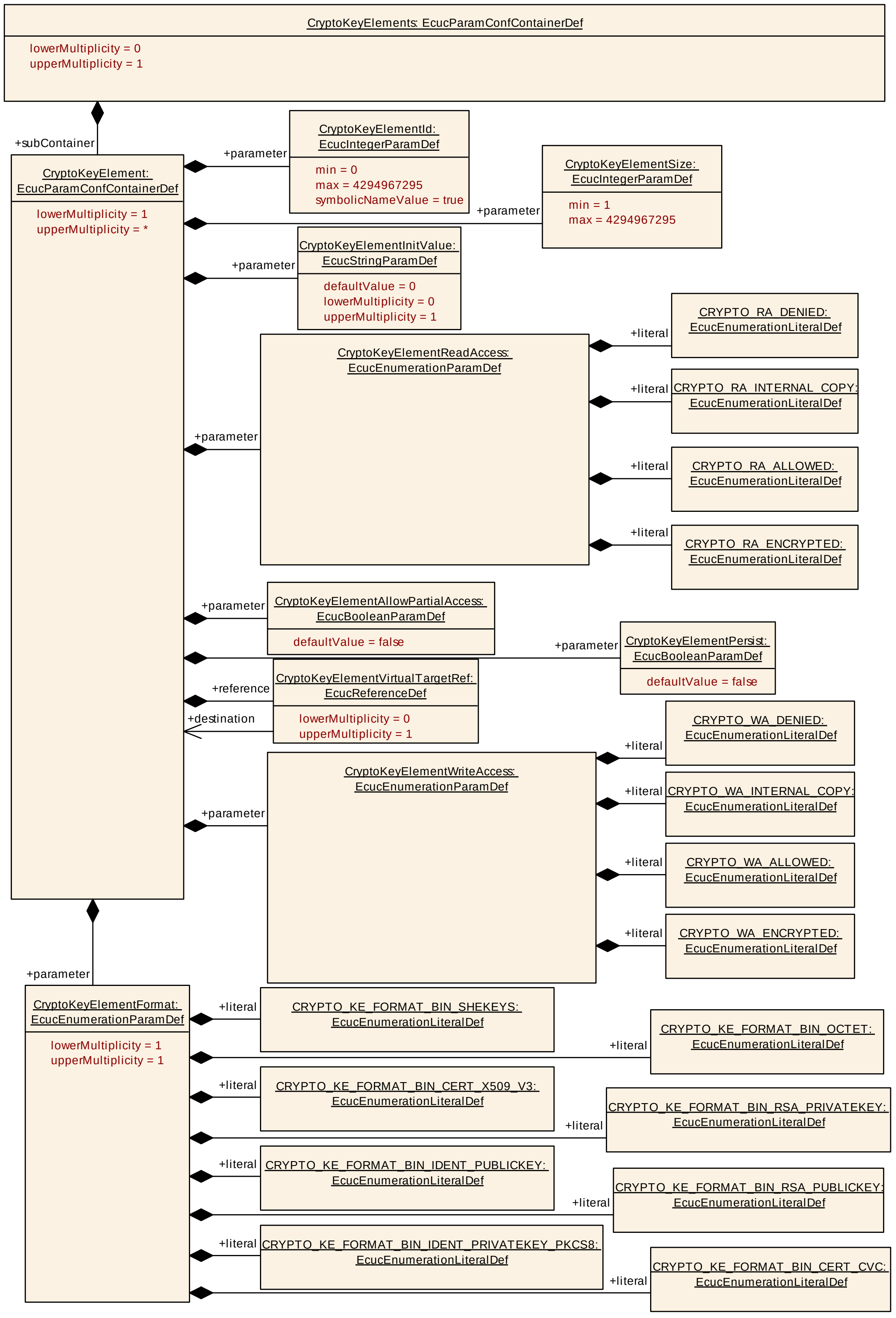
Description	Refers to a pointer in the CRYPTOto a CryptoKeyType. The CryptoKeyType provides the information which key elements are contained in a CryptoKey.		
Multiplicity	1		
Type	Reference to [CryptoKeyType]		
Post-Build Variant Multiplicity	false		
Post-Build Variant Value	false		
Multiplicity Configuration 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
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
+parameter
CryptoKeyElementSize: EcucIntegerParamDef
min = 1
max = 4294967295
+parameter
CryptoKeyElementInitValue: EcucStringParamDef
defaultValue = 0
lowerMultiplicity = 0
upperMultiplicity = 1
+parameter
CryptoKeyElementReadAccess: EcucEnumerationParamDef
+literal
CRYPTO_RA_DENIED: EcucEnumerationLiteralDef
+literal
CRYPTO_RA_INTERNAL_COPY: EcucEnumerationLiteralDef
+literal
CRYPTO_RA_ALLOWED: EcucEnumerationLiteralDef
+literal
CRYPTO_RA_ENCRYPTED: EcucEnumerationLiteralDef
+parameter
CryptoKeyElementAllowPartialAccess: EcucBooleanParamDef
defaultValue = false
+parameter
CryptoKeyElementPersist: EcucBooleanParamDef
defaultValue = false
+reference
CryptoKeyElementVirtualTargetRef: EcucReferenceDef
lowerMultiplicity = 0
upperMultiplicity = 1
+destination
+parameter
CryptoKeyElementWriteAccess: EcucEnumerationParamDef
+literal
CRYPTO_WA_DENIED: EcucEnumerationLiteralDef
+literal
CRYPTO_WA_INTERNAL_COPY: EcucEnumerationLiteralDef
+literal
CRYPTO_WA_ALLOWED: EcucEnumerationLiteralDef
+literal
CRYPTO_WA_ENCRYPTED: EcucEnumerationLiteralDef
+parameter
CryptoKeyElementFormat: EcucEnumerationParamDef
lowerMultiplicity = 1
upperMultiplicity = 1
+literal
CRYPTO_KE_FORMAT_BIN_SHEKEYS: EcucEnumerationLiteralDef
+literal
CRYPTO_KE_FORMAT_BIN_OCTET: EcucEnumerationLiteralDef
+literal
CRYPTO_KE_FORMAT_BIN_CERT_X509_V3: EcucEnumerationLiteralDef
+literal
CRYPTO_KE_FORMAT_BIN_RSA_PRIVATEKEY: EcucEnumerationLiteralDef
+literal
CRYPTO_KE_FORMAT_BIN_IDENT_PUBLICKEY: EcucEnumerationLiteralDef
+literal
CRYPTO_KE_FORMAT_BIN_RSA_PUBLICKEY: EcucEnumerationLiteralDef
+literal
CRYPTO_KE_FORMAT_BIN_IDENT_PRIVATEKEY_PKCS8: EcucEnumerationLiteralDef
+literal
CRYPTO_KE_FORMAT_BIN_CERT_CVC: EcucEnumerationLiteralDef

10.1.8 CryptoKeyElement

SWS Item	**ECUC_Crypto_00014 :**
Container Name	CryptoKeyElement
Description	Configuration of a CryptoKeyElement
Configuration Parameters	

SWS Item	**ECUC_Crypto_00025 :**		
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 :**		
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_CERT_CVC	0x08	
	CRYPTO_KE_FORMAT_BIN_CERT_X509_V3	0x07	
	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 :**
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 :**		
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		
maxLength	--		
minLength	--		
regularExpression	--		
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 :**		
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 :**
Name	CryptoKeyElementReadAccess
Parent Container	CryptoKeyElement
Description	Define the reading access rights of the key element. 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	0x03	
	CRYPTO_RA_DENIED	0x01	
	CRYPTO_RA_ENCRYPTED	0x04	
	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		

SWS Item	**ECUC_Crypto_00022 :**		
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 :**	
Name	CryptoKeyElementWriteAccess	
Parent Container	CryptoKeyElement	
Description	Define the writing access rights of the key element 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	0x03
	CRYPTO_WA_DENIED	0x01
	CRYPTO_WA_ENCRYPTED	0x04
	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		

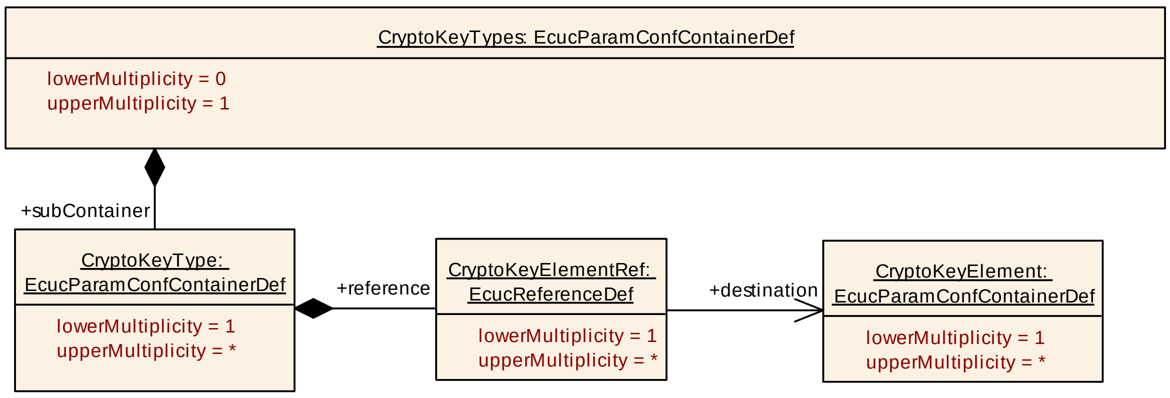
SWS Item	**ECUC_Crypto_00028 :**		
Name	CryptoKeyElementVirtualTargetRef		
Parent Container	CryptoKeyElement		
Description	Refers to a key element which will contain the actual data. If the Reference is configured, the key element will be a virtual key element.		
Multiplicity	0..1		
Type	Reference to [CryptoKeyElement]		
Post-Build Variant Value	true		
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

10.1.9 CryptoKeyTypes

SWS Item	**ECUC_Crypto_00017 :**		
Container Name	CryptoKeyTypes		
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
Description	Configuration of a CryptoKeyType
Configuration Parameters	

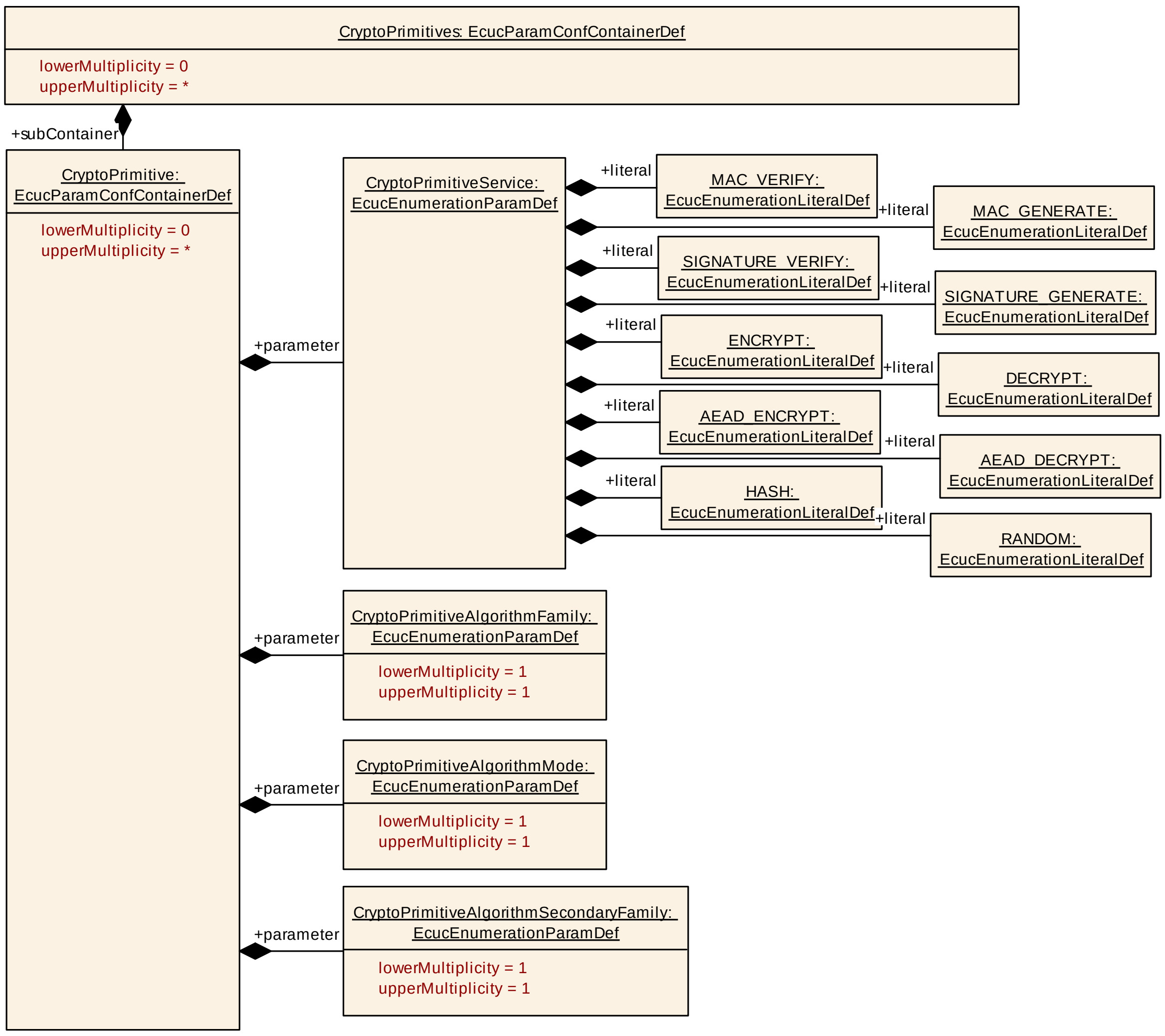
SWS Item	**ECUC_Crypto_00031 :**		
Name	CryptoKeyElementRef		
Parent Container	CryptoKeyType		
Description	Refers to a pointer in the CRYPTOCrypto Key Element, which holds the data of the crypto key element.		
Multiplicity	1..*		
Type	Reference to [CryptoKeyElement]		
Multiplicity Configuration Class	***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		
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


CryptoPrimitives: EcucParamConfContainerDef
lowerMultiplicity = 0
upperMultiplicity = *
+subContainer
CryptoPrimitive: EcucParamConfContainerDef
lowerMultiplicity = 0
upperMultiplicity = *
+parameter
CryptoPrimitiveService: EcucEnumerationParamDef
+literal
MAC_VERIFY: EcucEnumerationLiteralDef
MAC_GENERATE: EcucEnumerationLiteralDef
SIGNATURE_VERIFY: EcucEnumerationLiteralDef
SIGNATURE_GENERATE: EcucEnumerationLiteralDef
ENCRYPT: EcucEnumerationLiteralDef
DECRYPT: EcucEnumerationLiteralDef
AEAD_ENCRYPT: EcucEnumerationLiteralDef
AEAD_DECRYPT: EcucEnumerationLiteralDef
HASH: EcucEnumerationLiteralDef
RANDOM: EcucEnumerationLiteralDef
CryptoPrimitiveAlgorithmFamily: EcucEnumerationParamDef
lowerMultiplicity = 1
upperMultiplicity = 1
CryptoPrimitiveAlgorithmMode: EcucEnumerationParamDef
lowerMultiplicity = 1
upperMultiplicity = 1
CryptoPrimitiveAlgorithmSecondaryFamily: EcucEnumerationParamDef
lowerMultiplicity = 1
upperMultiplicity = 1

10.1.12 CryptoPrimitive

SWS Item	**ECUC_Crypto_00033 :**		
Container Name	CryptoPrimitive		
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 :**	
Name	CryptoPrimitiveAlgorithmFamily	
Parent Container	CryptoPrimitive	
Description	Determines the algorithm family used for the crypto service	
Multiplicity	1	
Type	EcucEnumerationParamDef	
Range	CRYPTO_ALGOFAM_3DES	0x13
	CRYPTO_ALGOFAM_AES	0x14

	CRYPTO_ALGOFAM_BLAKE_1_256	0x0F	
	CRYPTO_ALGOFAM_BLAKE_1_512	0x10	
	CRYPTO_ALGOFAM_BLAKE_2s_256	0x11	
	CRYPTO_ALGOFAM_BLAKE_2s_512	0x12	
	CRYPTO_ALGOFAM_BRAINPOOL	0x18	
	CRYPTO_ALGOFAM_CHACHA	0x15	
	CRYPTO_ALGOFAM_CUSTOM	0xFF	
	CRYPTO_ALGOFAM_DH	0x26	
	CRYPTO_ALGOFAM_DRBG	0x20	
	CRYPTO_ALGOFAM_ECCANSI	0x1e	
	CRYPTO_ALGOFAM_ECCNIST	0x19	
	CRYPTO_ALGOFAM_ECCSEC	0x1f	
	CRYPTO_ALGOFAM_ECIES	0x1D	
	CRYPTO_ALGOFAM_ED25519	0x17	
	CRYPTO_ALGOFAM_FIPS186	0x21	
	CRYPTO_ALGOFAM_KDFX963	0x25	
	CRYPTO_ALGOFAM_NOT_SET	0x00	
	CRYPTO_ALGOFAM_PADDING_ONEWITHZEROS	0x23	
	CRYPTO_ALGOFAM_PADDING_PKCS7	0x22	
	CRYPTO_ALGOFAM_PBKDF2	0x24	
	CRYPTO_ALGOFAM_RIPEMD160	0x0E	
	CRYPTO_ALGOFAM_RNG	0x1B	
	CRYPTO_ALGOFAM_RSA	0x16	
	CRYPTO_ALGOFAM_SHA1	0x01	
	CRYPTO_ALGOFAM_SHA2_224	0x02	
	CRYPTO_ALGOFAM_SHA2_256	0x03	
	CRYPTO_ALGOFAM_SHA2_384	0x04	
	CRYPTO_ALGOFAM_SHA2_512	0x05	
	CRYPTO_ALGOFAM_SHA2_512_224	0x06	
	CRYPTO_ALGOFAM_SHA2_512_256	0x07	
	CRYPTO_ALGOFAM_SHA3_224	0x08	
	CRYPTO_ALGOFAM_SHA3_256	0x09	
	CRYPTO_ALGOFAM_SHA3_384	0x0A	
	CRYPTO_ALGOFAM_SHA3_512	0x0B	
	CRYPTO_ALGOFAM_SHAKE128	0x0C	
	CRYPTO_ALGOFAM_SHAKE256	0x0D	
	CRYPTO_ALGOFAM_SIPHASH	0x1C	
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 :**	
Name	CryptoPrimitiveAlgorithmMode	
Parent Container	CryptoPrimitive	
Description	Determines the algorithm mode used for the crypto service	
Multiplicity	1	
Type	EcucEnumerationParamDef	
Range	CRYPTO_ALGOMODE_12ROUNDS	0x0D
	CRYPTO_ALGOMODE_20ROUNDS	0x0E

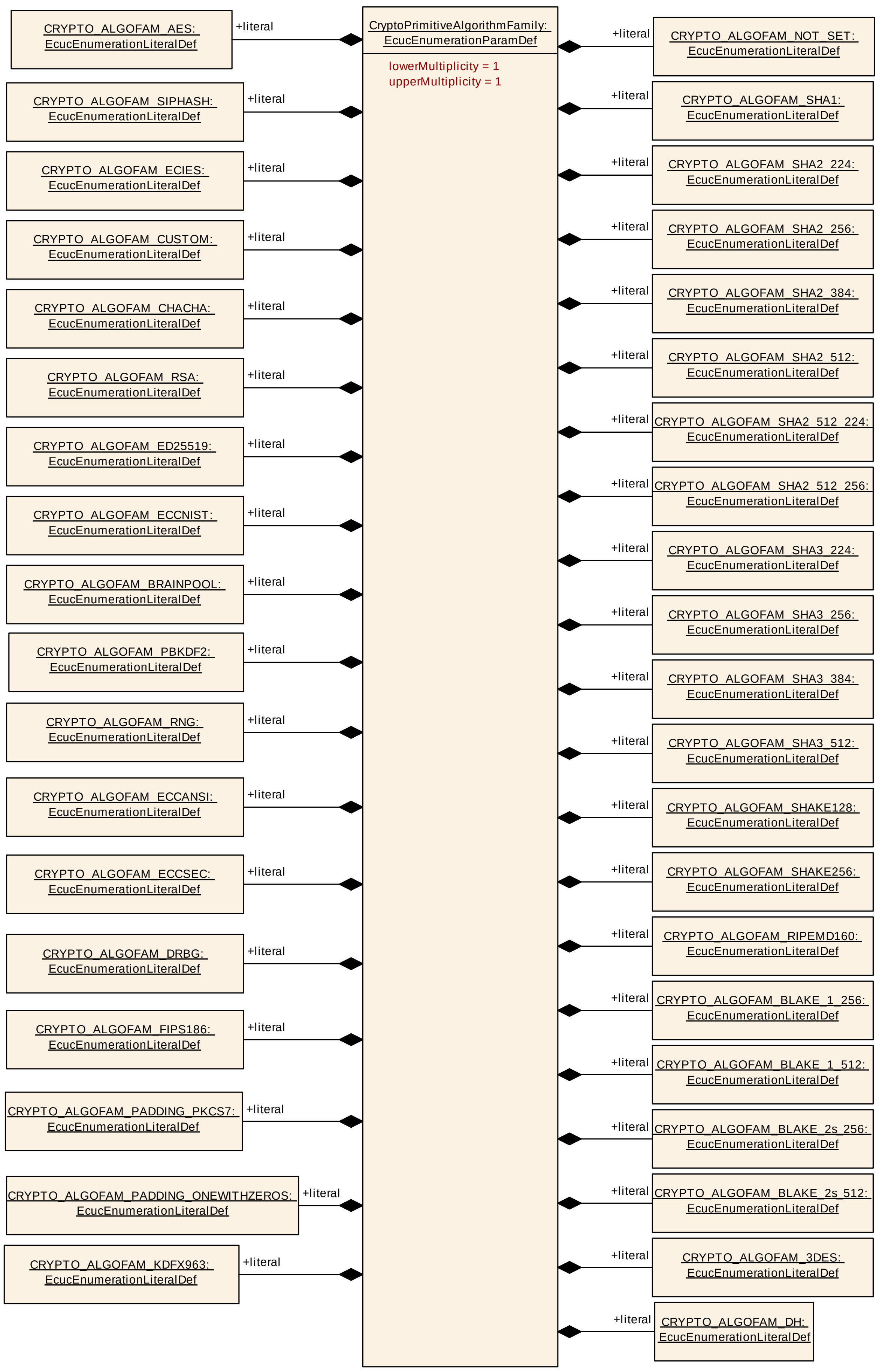
	CRYPTO_ALGOMODE_8ROUNDS	0x0C	
	CRYPTO_ALGOMODE_CBC	0x02	
	CRYPTO_ALGOMODE_CFB	0x03	
	CRYPTO_ALGOMODE_CMAC	0x10	
	CRYPTO_ALGOMODE_CTR	0x05	
	CRYPTO_ALGOMODE_CTRDRBG	0x12	
	CRYPTO_ALGOMODE_CUSTOM	0xFF	
	CRYPTO_ALGOMODE_ECB	0x01	
	CRYPTO_ALGOMODE_GCM	0x06	
	CRYPTO_ALGOMODE_GMAC	0x11	
	CRYPTO_ALGOMODE_HMAC	0x0F	
	CRYPTO_ALGOMODE_NOT_SET	0x00	
	CRYPTO_ALGOMODE_OFB	0x04	
	CRYPTO_ALGOMODE_PXXXR	0x09	
	CRYPTO_ALGOMODE_RSAES_OAEP	0x08	
	CRYPTO_ALGOMODE_RSAES_PKCS1_v1_5	0x09	
	CRYPTO_ALGOMODE_RSASSA_PKCS1_v1_5	0x0B	
	CRYPTO_ALGOMODE_RSASSA_PSS	0x0A	
	CRYPTO_ALGOMODE_SIPHASH_2_4	0x13	
	CRYPTO_ALGOMODE_SIPHASH_4_8	0x14	
	CRYPTO_ALGOMODE_XTS	0x07	
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		

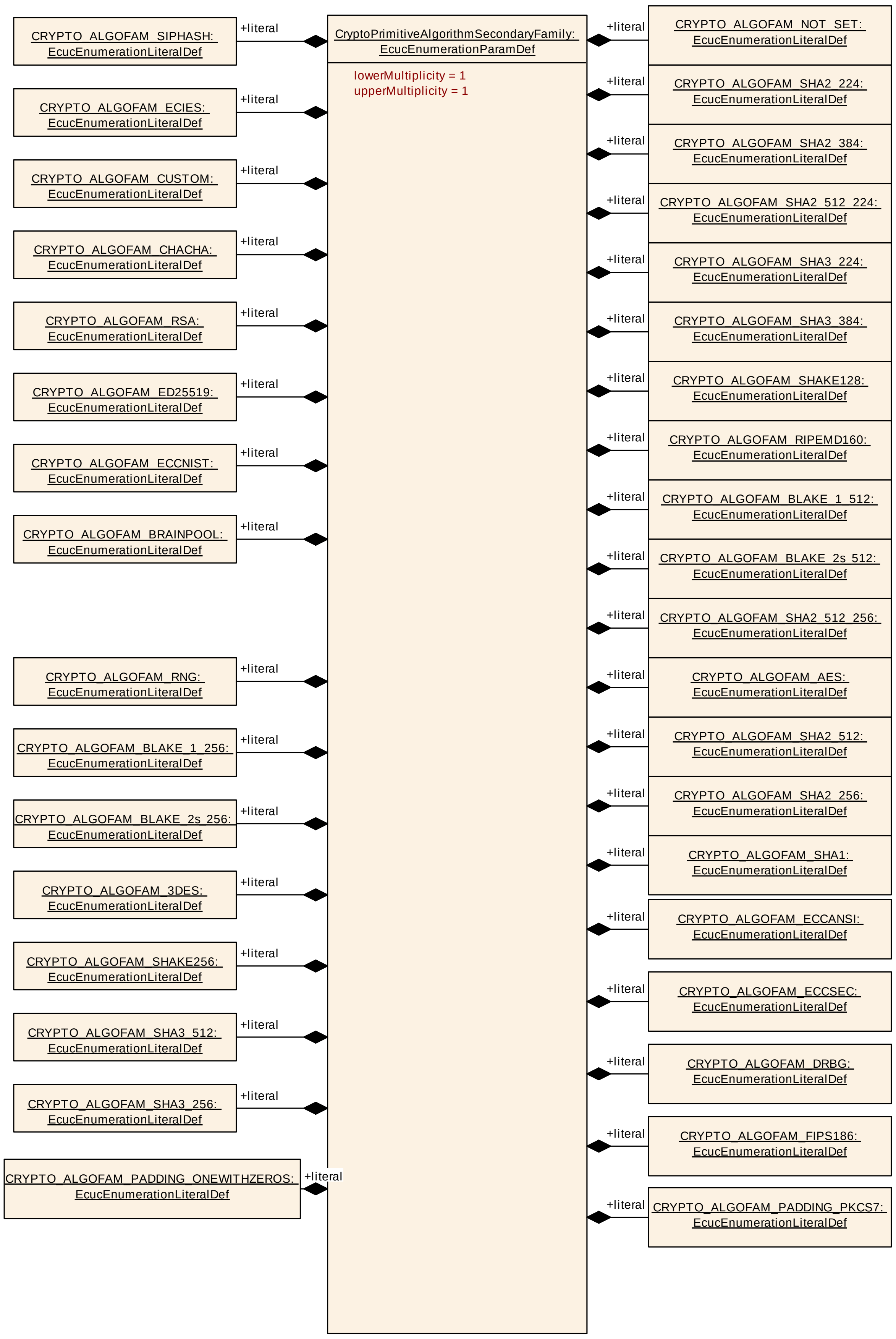
SWS Item	**ECUC_Crypto_00037 :**	
Name	CryptoPrimitiveAlgorithmSecondaryFamily	
Parent Container	CryptoPrimitive	
Description	Determines the algorithm secondary family used for the crypto service	
Multiplicity	1	
Type	EcucEnumerationParamDef	
Range	CRYPTO_ALGOFAM_3DES	0x13
	CRYPTO_ALGOFAM_AES	0x14
	CRYPTO_ALGOFAM_BLAKE_1_256	0x0F
	CRYPTO_ALGOFAM_BLAKE_1_512	0x10
	CRYPTO_ALGOFAM_BLAKE_2s_256	0x11
	CRYPTO_ALGOFAM_BLAKE_2s_512	0x12
	CRYPTO_ALGOFAM_BRAINPOOL	0x18
	CRYPTO_ALGOFAM_CHACHA	0x15
	CRYPTO_ALGOFAM_CUSTOM	0xFF
	CRYPTO_ALGOFAM_DRBG	0x20
	CRYPTO_ALGOFAM_ECCANSI	0x1e
	CRYPTO_ALGOFAM_ECCNIST	0x19
	CRYPTO_ALGOFAM_ECCSEC	0x1f
	CRYPTO_ALGOFAM_ECIES	0x1D
	CRYPTO_ALGOFAM_ED25519	0x17
	CRYPTO_ALGOFAM_FIPS186	0x21
	CRYPTO_ALGOFAM_NOT_SET	0x00
	CRYPTO_ALGOFAM_PADDING_ONEWITHZEROS	0x23

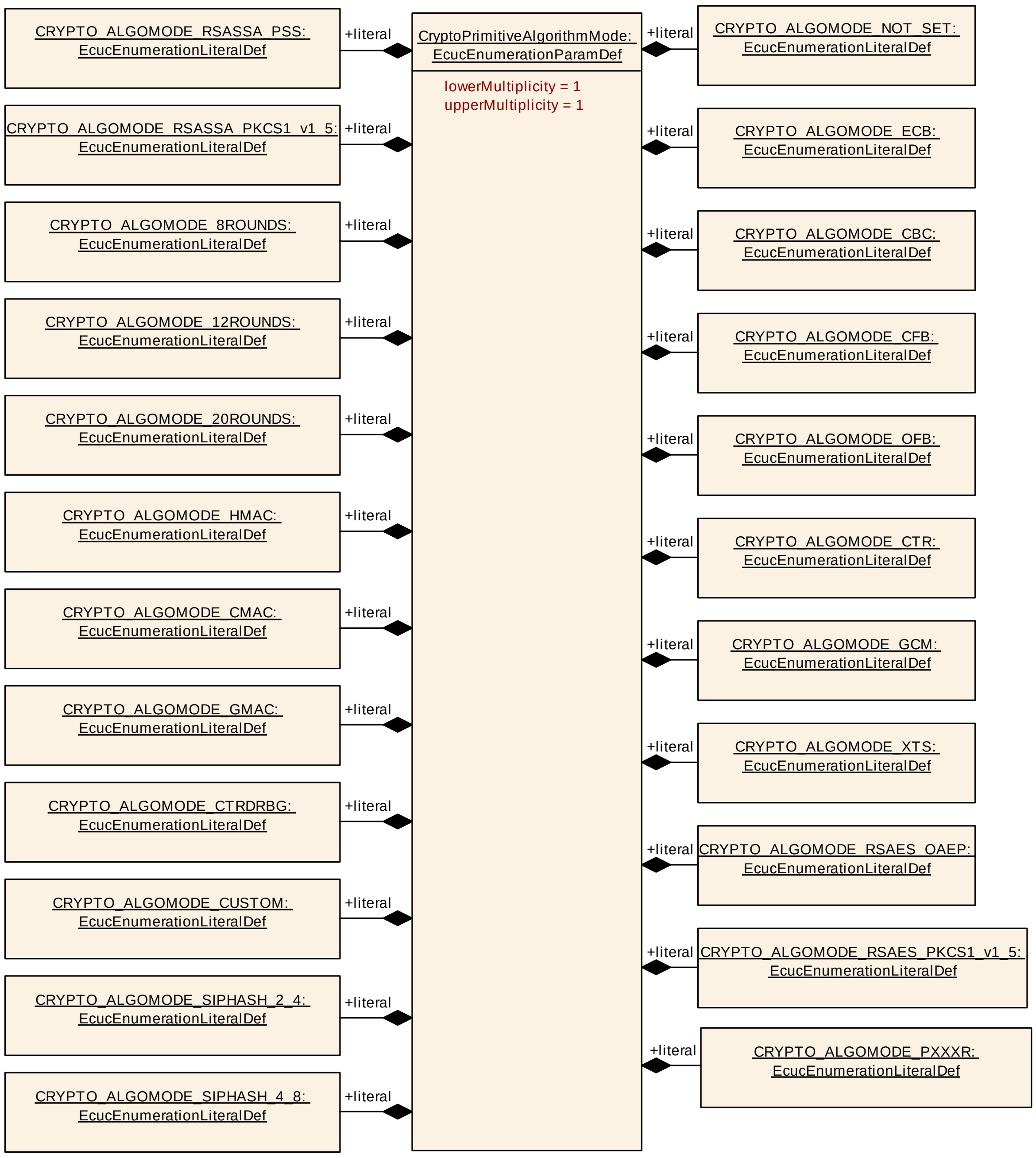
	CRYPTO_ALGOFAM_PADDING_PKCS7	0x22	
	CRYPTO_ALGOFAM_RIPEMD160	0x0E	
	CRYPTO_ALGOFAM_RNG	0x1B	
	CRYPTO_ALGOFAM_RSA	0x16	
	CRYPTO_ALGOFAM_SHA1	0x01	
	CRYPTO_ALGOFAM_SHA2_224	0x02	
	CRYPTO_ALGOFAM_SHA2_256	0x03	
	CRYPTO_ALGOFAM_SHA2_384	0x04	
	CRYPTO_ALGOFAM_SHA2_512	0x05	
	CRYPTO_ALGOFAM_SHA2_512_224	0x06	
	CRYPTO_ALGOFAM_SHA2_512_256	0x07	
	CRYPTO_ALGOFAM_SHA3_224	0x08	
	CRYPTO_ALGOFAM_SHA3_256	0x09	
	CRYPTO_ALGOFAM_SHA3_384	0x0A	
	CRYPTO_ALGOFAM_SHA3_512	0x0B	
	CRYPTO_ALGOFAM_SHAKE128	0x0C	
	CRYPTO_ALGOFAM_SHAKE256	0x0D	
	CRYPTO_ALGOFAM_SIPHASH	0x1C	
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 :**		
Name	CryptoPrimitiveService		
Parent Container	CryptoPrimitive		
Description	Determines the crypto service used for defining the capabilities		
Multiplicity	1		
Type	EcucEnumerationParamDef		
Range	AEAD_DECRYPT	0x8	
	AEAD_ENCRYPT	0x7	
	DECRYPT	0x6	
	ENCRYPT	0x5	
	HASH	0x9	
	MAC_GENERATE	0x2	
	MAC_VERIFY	0x1	
	RANDOM	0xA	
	SIGNATURE_GENERATE	0x4	
	SIGNATURE_VERIFY	0x3	
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


CryptoPrimitiveAlgorithmFamily: EcucEnumerationParamDef
lowerMultiplicity = 1
upperMultiplicity = 1
+literal
CRYPTO_ALGOFAM_AES: EcucEnumerationLiteralDef
CRYPTO_ALGOFAM_SIPHASH: EcucEnumerationLiteralDef
CRYPTO_ALGOFAM_ECIES: 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_PBKDF2: EcucEnumerationLiteralDef
CRYPTO_ALGOFAM_RNG: EcucEnumerationLiteralDef
CRYPTO_ALGOFAM_ECCANSI: EcucEnumerationLiteralDef
CRYPTO_ALGOFAM_ECCSEC: EcucEnumerationLiteralDef
CRYPTO_ALGOFAM_DRBG: EcucEnumerationLiteralDef
CRYPTO_ALGOFAM_FIPS186: EcucEnumerationLiteralDef
CRYPTO_ALGOFAM_PADDING_PKCS7: EcucEnumerationLiteralDef
CRYPTO_ALGOFAM_PADDING_ONEWITHZEROS: EcucEnumerationLiteralDef
CRYPTO_ALGOFAM_KDFX963: 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_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_3DES: EcucEnumerationLiteralDef
CRYPTO_ALGOFAM_DH: EcucEnumerationLiteralDef


CryptoPrimitiveAlgorithmSecondaryFamily: EcucEnumerationParamDef
lowerMultiplicity = 1
upperMultiplicity = 1
+literal
CRYPTO_ALGOFAM_SIPHASH: EcucEnumerationLiteralDef
CRYPTO_ALGOFAM_ECIES: 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_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 = 1
+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.