| Document Title | Specification of Crypto Interface |
|---|---|
| **Document Category** | Software specification (SWS) |
| **Document UID** | 806 |
| **Document Status** | published |
| **Part of AUTOSAR Standard** | Classic Platform |
| **Part of Standard Release** | R26-11 |

| Document Change History | | | |
|---|---|---|---|
| **Date** | **Release** | **Changed by** | **Description** |
| 2026-11-26 | R26-11 | AUTOSAR Release Management | • Restructure of crypto channels and primitives across the whole crypto stack.<br>• Remove GetVersionInfo API and configuration parameters. |
| 2024-11-27 | R24-11 | AUTOSAR Release Management | • Add CRYPTO_E_KEY_EMPTY as a return value for CryIf_RandomSeed API<br>• Minor changes |
| 2023-11-23 | R23-11 | AUTOSAR Release Management | • Changed pubValueLengthPtr into publicValueLengthPtr<br>• Removed CRYPTO_E_QUEUE_FULL from SWS_CryIf_91003<br>• Minor changes |
| 2022-11-24 | R22-11 | AUTOSAR Release Management | • Added CRYPTO_CUSTOM service<br>• Updated CRYPTO_E_PARAM_HANDLE to CRYIF_E_PARAM_HANDLE<br>• Removed return values after reporting Det errors |
| 2021-11-25 | R21-11 | AUTOSAR Release Management | • CryIf_KeyGenerate() and CryIf_RandomSeed() are always synchronous |
| 2020-11-30 | R20-11 | AUTOSAR Release Management | • Improve structure of error handling<br>• Improve flexibility of crypto stack for multi core<br>• Clarifications on requirements, API and configuration parameters<br>• Add functionality for key status |



△

| | | | |
|---|---|---|---|
| 2019-11-28 | R19-11 | AUTOSAR Release Management | • Minor changes<br>• Clarify key ID handling<br>• Remove certificate handling<br>• Cleanup of DET and return errors<br>• Changed Document Status from Final to published |
| 2018-10-31 | 4.4.0 | AUTOSAR Release Management | • Remove secure counter<br>• Align return values of interface functions.<br>• Support source and destination buffers for crypto operations located in crypto driver.<br>• Support key management operation in asynchronous mode |
| 2017-12-08 | 4.3.1 | AUTOSAR Release Management | • minor corrections, clarifications and editorial changes; For details please refer to the ChangeDocumentation |
| 2016-11-30 | 4.3.0 | AUTOSAR Release Management | • Initial Release |

**Disclaimer**

This work (specification and/or software implementation) and the material contained in it, as released by AUTOSAR, is for the purpose of information only. AUTOSAR and the companies that have contributed to it shall not be liable for any use of the work.

The material contained in this work is protected by copyright and other types of intellectual property rights. The commercial exploitation of the material contained in this work requires a license to such intellectual property rights.

This work may be utilized or reproduced without any modification, in any form or by any means, for informational purposes only. For any other purpose, no part of the work may be utilized or reproduced, in any form or by any means, without permission in writing from the publisher.

The work has been developed for automotive applications only. It has neither been developed, nor tested for non-automotive applications.

The word AUTOSAR and the AUTOSAR logo are registered trademarks.

# Table of Contents

1 Introduction and functional overview 6
2 Acronyms and Abbreviations 7
3 Related documentation 9
3.1 Input documents & related standards and norms . . . 9
3.2 Related specification . . . 9
4 Constraints and assumptions 10
4.1 Limitations . . . 10
4.2 Applicability to car domains . . . 10
5 Dependencies to other modules 11
5.1 File structure . . . 11
5.1.1 Code file structure . . . 11
6 Requirements Tracing 12
7 Functional specification 14
7.1 Basics . . . 14
7.2 Multicore . . . 14
7.3 Error Classification . . . 15
7.3.1 Development Errors . . . 15
7.3.2 Runtime Errors . . . 15
7.3.3 Production Errors . . . 15
7.4 Error detection . . . 15
7.5 Security Events . . . 15
8 API specification 16
8.1 Imported types . . . 16
8.2 Type definitions . . . 16
8.2.1 Extension to Std_ReturnType . . . 16
8.2.2 CryIf_ConfigType . . . 17
8.3 Function definitions . . . 17
8.3.1 General API . . . 18
8.3.2 Job Processing Interface . . . 18
8.3.3 Job Cancellation Interface . . . 22
8.3.4 Key Management Interface . . . 25
8.3.4.1 Key Setting Interface . . . 25
8.3.4.2 Key Status Interface . . . 28
8.3.4.3 Key Extraction Interface . . . 29
8.3.4.4 Key Copying Interface . . . 30
8.3.4.5 Key Generation Interface . . . 35
8.3.4.6 Key Derivation Interface . . . 38
8.3.5 Custom Service Interface . . . 42

8.3.6 Callback notifications . . . 43
8.4 Expected interfaces . . . 44
8.4.1 Mandatory interfaces . . . 44
8.4.2 Optional interfaces . . . 44
9 Sequence diagrams 46
10 Configuration specification 47
10.1 How to read this chapter . . . 47
10.2 Containers and configuration parameters . . . 47
10.2.1 Variants . . . 47
10.2.2 CryIf . . . 47
10.2.3 CryIfGeneral . . . 48
10.2.4 CryIfChannels . . . 49
10.2.5 CryIfChannel . . . 49
10.2.6 CryIfKeys . . . 51
10.2.7 CryIfKey . . . 51
10.2.8 CryIfPrimitives . . . 52
10.2.9 CryIfPrimitive . . . 53
10.3 Published Information . . . 54
A Change history of AUTOSAR traceable items 55
A.1 Traceable item history of this document according to AUTOSAR Release R26-11 . . . 55
A.1.1 Added Specification Items in R26-11 . . . 55
A.1.2 Changed Specification Items in R26-11 . . . 55
A.1.3 Deleted Specification Items in R26-11 . . . 55
A.2 Traceable item history of this document according to AUTOSAR Release R25-11 . . . 55
A.2.1 Added Specification Items in R25-11 . . . 55
A.2.2 Changed Specification Items in R25-11 . . . 56
A.2.3 Deleted Specification Items in R25-11 . . . 56

# 1 Introduction and functional overview

This specification specifies the functionality, API and the configuration of the AUTOSAR Basic Software Module Crypto Interface (CRYIF).

The Crypto Interface module is located between the low level Crypto solutions (Crypto Driver [1] and SW-based CDD) and the upper service layer (Crypto Service Manager [2]). It represents the interface to the services of the Crypto Driver(s) for the upper service layer. An AUTOSAR Layered View can be found in 7.1.

The Crypto Interface module provides a unique interface to manage different Crypto HW and SW solutions like HSM, SHE or SW-based CDD. Thus, multiple underlying internal and external Crypto HW as well as SW solutions can be utilized by the Crypto Service Manager module based on a mapping scheme maintained by Crypto Interface.

# 2 Acronyms and Abbreviations

The glossary below includes acronyms and abbreviations relevant to the Crypto Interface module that are not included in the [3, AUTOSAR glossary].

| Abbreviation / Acronym: | Description: |
|---|---|
| CSM | Crypto Service Manager |
| CRYIF | Crypto Interface |
| CRYPTO | Crypto Driver |
| HSM | Hardware Security Module |
| SHE | Security Hardware Extension |

**Table 2.1: Acronyms and abbreviations used in the scope of this Document**

| *Terms:* | *Description:* | |
|---|---|---|
| Crypto Driver Object | A Crypto Driver Object is an instance of a crypto module (hardware or software), which is able to perform one or more different crypto operations. | |
| Key | A Key can be referenced by a job in the Csm.<br>In the Crypto Driver, the key references a specific key type. | |
| Key Type | A key type consists of references to key elements.<br>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.<br>It can also be used to configure the behavior of the key management functions. Key elements from different keys have different memory area (both NV and RAM area). | |
| Channel | A channel is the statically configured path of a Job provided by the Crypto Service Manager via the Crypto Interface to a specific Crypto Driver Object. | |
| Job | A Job is the statically configured combination of e.g. a primitive reference, a channel, a priority, an optional key, etc., accessible by the user via its id. | |
| Crypto Primitive | A crypto primitive is an instance of a configured cryptographic algorithm. | |
| Operation | An operation of a crypto primitive declares what part of the crypto primitive shall be performed. There are three different operations: | |
| | START | Operation indicates a new request of a crypto primitive, and it shall cancel all previous requests. |
| | UPDATE | Operation indicates, that the crypto primitive expect input data. |
| | FINISH | Operation 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. | |
| Primitive | A 'Primitive' is an instance of a configured cryptographic algorithm realized in a Crypto Driver Object. Among others it refers to a functionality provided by the CSM to the application, the concrete underlining 'algorithm family' (e.g. AES, MD5, RSA, ...), and a 'algorithmmode' (e.g. ECB, CBC, ...). | |
| 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. | |
| Processing | Indicates the kind of job processing. | |

△

| *Terms:* | *Description:* | |
|---|---|---|
| | Asynchronous | The job is not processed immediately when calling a corresponding function. Usually, the caller is informed via a callback function when the job has been finished. |
| | Synchronous | The job is processed immediately when calling a corresponding function. When the function returns, a result will be available. |
| Service | A 'Service' shall be understood as defined in the TR_Glossary document: A service is a type of operation that has a published specification of interface and behavior, involving a contract between the provider of the capability and the potential clients. | |

**Table 2.2: Definitions used in the scope of this Document**

# 3 Related documentation

## 3.1 Input documents & related standards and norms

[1] Specification of Crypto Driver
AUTOSAR_CP_SWS_CryptoDriver

[2] Specification of Crypto Service Manager
AUTOSAR_CP_SWS_CryptoServiceManager

[3] Glossary
AUTOSAR_FO_TR_Glossary

[4] General Specification of Basic Software Modules
AUTOSAR_CP_SWS_BSWGeneral

[5] General Requirements on Basic Software Modules
AUTOSAR_CP_RS_BSWGeneral

## 3.2 Related specification

AUTOSAR provides a General Specification on Basic Software modules [4, SWS BSW General], which is also valid for Crypto Interface.

Thus, the specification SWS BSW General shall be considered as additional and required specification for Crypto Interface.

# 4 Constraints and assumptions

## 4.1 Limitations

The Crypto Interface is specifically designed to operate with one or multiple underlying Crypto Drivers. Several Crypto Driver modules covering different HW processing units or cores are represented by just one generic interface as specified in the Crypto Driver specification [1].

Any software based Crypto Driver shall be implemented as a CDD represented by the same interface above.

## 4.2 Applicability to car domains

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

# 5 Dependencies to other modules

**[SWS_CryIf_00001] CryIf as an Interface BSW module**

*Upstream requirements:* SRS_BSW_00384

⌈The Crypto Interface (CRYIF) shall be able to be called by the Crypto Service Manager (CSM), and forward its service requests to the underlying Crypto Drivers.⌋

**[SWS_CryIf_00002] Access to the underlying Crypto Drivers**

*Upstream requirements:* SRS_BSW_00384

⌈The CRYIF shall be able to access the underlying Crypto Drivers to calculate results with their cryptographic services. These results shall be returned back to the CSM by the CRYIF.⌋

## 5.1 File structure

### 5.1.1 Code file structure

**[SWS_CryIf_00003] File Structure is not defined**

*Upstream requirements:* SRS_BSW_00346

⌈The code file structure shall not be defined within this specification completely.⌋

**[SWS_CryIf_00004] Only one source file**

*Upstream requirements:* SRS_BSW_00346

⌈The code file structure shall contain one source file CryIf.c, that contains the entire CRYIF code.⌋

# 6 Requirements Tracing

The following tables reference the requirements specified in [5] and links to the fulfillment of these. Please note that if column “Satisfied by” is empty for a specific requirement this means that this requirement is not fulfilled by this document.

| 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_CryIf_00015] [SWS_CryIf_91000] |
| **[SRS_BSW_00346]** | All AUTOSAR Basic Software Modules shall provide at least a basic set of module files | [SWS_CryIf_00003] [SWS_CryIf_00004] |
| **[SRS_BSW_00358]** | The return type of init() functions implemented by AUTOSAR Basic Software Modules shall be void | [SWS_CryIf_91000] |
| **[SRS_BSW_00359]** | Callback Function Return Types for AUTOSAR BSW | [SWS_CryIf_91013] |
| **[SRS_BSW_00360]** | AUTOSAR Basic Software Modules callback functions are allowed to have parameters | [SWS_CryIf_00109] [SWS_CryIf_91013] |
| **[SRS_BSW_00369]** | All AUTOSAR Basic Software Modules shall not return specific development error codes via the API | [SWS_CryIf_00014] [SWS_CryIf_00027] [SWS_CryIf_00028] [SWS_CryIf_00029] [SWS_CryIf_00049] [SWS_CryIf_00050] [SWS_CryIf_00052] [SWS_CryIf_00053] [SWS_CryIf_00056] [SWS_CryIf_00057] [SWS_CryIf_00059] [SWS_CryIf_00060] [SWS_CryIf_00062] [SWS_CryIf_00063] [SWS_CryIf_00064] [SWS_CryIf_00068] [SWS_CryIf_00069] [SWS_CryIf_00070] [SWS_CryIf_00071] [SWS_CryIf_00073] [SWS_CryIf_00074] [SWS_CryIf_00076] [SWS_CryIf_00077] [SWS_CryIf_00082] [SWS_CryIf_00083] [SWS_CryIf_00084] [SWS_CryIf_00085] [SWS_CryIf_00086] [SWS_CryIf_00090] [SWS_CryIf_00091] [SWS_CryIf_00092] [SWS_CryIf_00094] [SWS_CryIf_00107] [SWS_CryIf_00108] [SWS_CryIf_00110] [SWS_CryIf_00111] [SWS_CryIf_00112] [SWS_CryIf_00115] [SWS_CryIf_00116] [SWS_CryIf_00117] [SWS_CryIf_00118] [SWS_CryIf_00121] [SWS_CryIf_00122] [SWS_CryIf_00129] [SWS_CryIf_00130] [SWS_CryIf_00131] [SWS_CryIf_00137] [SWS_CryIf_00146] [SWS_CryIf_00150] |
| **[SRS_BSW_00384]** | The Basic Software Module specifications shall specify at least in the description which other modules they require | [SWS_CryIf_00001] [SWS_CryIf_00002] |
| **[SRS_BSW_00406]** | API handling in uninitialized state | [SWS_CryIf_00027] [SWS_CryIf_00049] [SWS_CryIf_00056] [SWS_CryIf_00059] [SWS_CryIf_00068] [SWS_CryIf_00073] [SWS_CryIf_00076] [SWS_CryIf_00082] [SWS_CryIf_00090] [SWS_CryIf_00110] [SWS_CryIf_00116] [SWS_CryIf_00129] [SWS_CryIf_00137] [SWS_CryIf_00146] [SWS_CryIf_00150] |

▽

△

| Requirement | Description | Satisfied by |
|---|---|---|
| **[SRS_BSW_00414]** | Init functions shall have a pointer to a configuration structure as single parameter | [SWS_CryIf_91000] [SWS_CryIf_91019] |

**Table 6.1: Requirements Tracing**

# 7 Functional specification

## 7.1 Basics

The Crypto Interface is located between the Crypto Service Manager and the underlying crypto drivers and is the unique interface to access cryptographic operations for all upper layers (BSW). The Crypto Interface is also the only user of the crypto drivers and provides a unique interface to manage different crypto hardware and software solutions. The Abstraction Layer encapsulates different mechanisms of hardware and software access, so the Crypto Interface implementation is independent from the underlying Crypto Drivers which can be realized in hardware or software.

Also it ensures the concurrent access to crypto services to enable the possibility to process multiple crypto tasks at the same time.

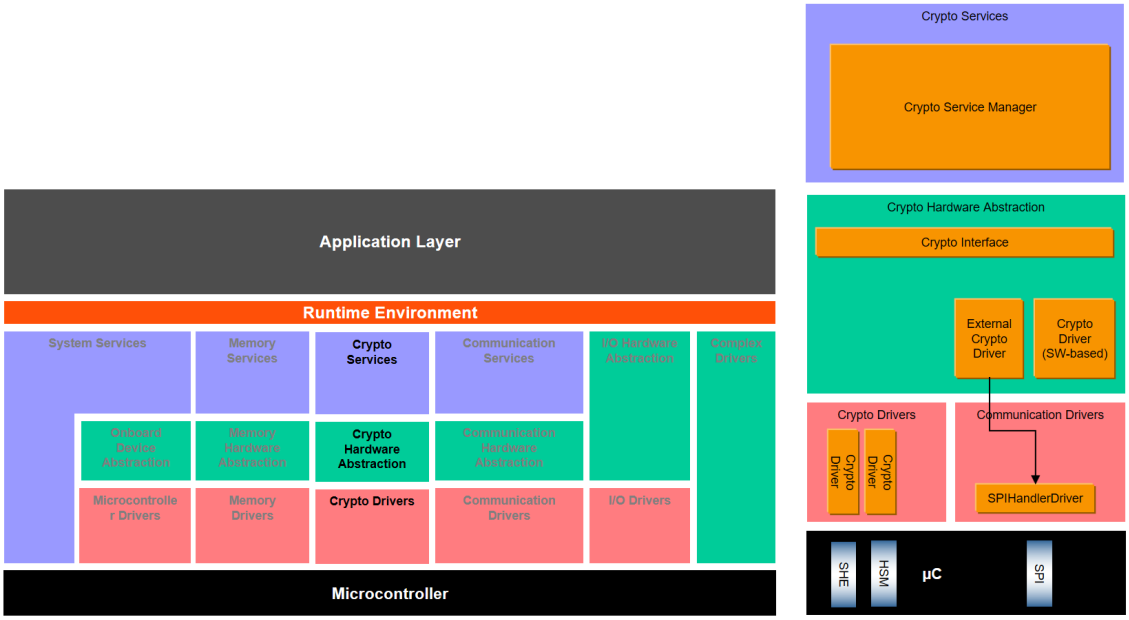



**Figure 7.1: AUTOSAR Layered View with Crypto-Interface)**

## 7.2 Multicore

In a setup with a distributed CSM and Crypto drivers assigned to different partitions, the CryIf APIs will be used in different partitions. Nevertheless, CryIf shall stay a pure forwarding component and not care about execution contexts anyway, means it simply forwards a request in the context of the original caller.

**[SWS_CryIf_00145] Context of call forwarding** ⌈The CryIf module shall forward a call in the context of the original caller.⌋

## 7.3 Error Classification

Chapter [4, General Specification of Basic Software Modules] 7.2 *"Error Handling"* describes the error handling of the Basic Software in detail. Above all, it constitutes a classification scheme consisting of five error types which may occur in BSW modules.

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

### 7.3.1 Development Errors

**[SWS_CryIf_00009] Definition of development errors in module CryIf** ⌈

| *Type of error* | *Related error code* | *Error value* |
|---|---|---|
| API request called before initialisation of CRYIF module. | CRYIF_E_UNINIT | 0x00 |
| Initialisation of CRYIF module failed. | CRYIF_E_INIT_FAILED | 0x01 |
| API request called with invalid parameter (null pointer). | CRYIF_E_PARAM_POINTER | 0x02 |
| API request called with invalid parameter (out of range). | CRYIF_E_PARAM_HANDLE | 0x03 |
| API request called with invalid parameter (invalid value). | CRYIF_E_PARAM_VALUE | 0x04 |
| Source key element size does not match the target key elements size. | CRYIF_E_KEY_SIZE_MISMATCH | 0x05 |

⌋

### 7.3.2 Runtime Errors

There are no runtime errors.

### 7.3.3 Production Errors

There are no production errors.

## 7.4 Error detection

There is no general error detection.

## 7.5 Security Events

The module does not report security events.

# 8 API specification

## 8.1 Imported types

In this chapter all types included from the following files are listed.

**[SWS_CryIf_91023] Definition of imported datatypes of module CryIf** ⌈

| *Module* | *Header File* | *Imported Type* |
|---|---|---|
| Csm | Rte_Csm_Type.h | Crypto_DataInputType |
| | Rte_Csm_Type.h | Crypto_DataOutputType |
| | Rte_Csm_Type.h | Crypto_DataRedirectionType |
| | Rte_Csm_Type.h | Crypto_JobActionType |
| | Rte_Csm_Type.h | Crypto_JobConfigType |
| | Rte_Csm_Type.h | Crypto_JobType |
| | Rte_Csm_Type.h | Crypto_KeyStatusType |
| | Rte_Csm_Type.h | Crypto_OperationModeType |
| | Rte_Csm_Type.h | Crypto_ProcessingType |
| | Rte_Csm_Type.h | Crypto_ResultType |
| | Rte_Csm_Type.h | Crypto_ServiceInfoType |
| | Rte_Csm_Type.h | Crypto_VerifyResultType |
| Std | Std_Types.h | Std_ReturnType |

⌋

It should be noted, that the Crypto Stack API uses the key element index definition from the CSM module (see [SWS_Csm_01022]).

## 8.2 Type definitions

### 8.2.1 Extension to Std_ReturnType

**[SWS_CryIf_91020] Definition of Std_ReturnType-extension for module CryIf** ⌈

| *Range* | CRYPTO_E_BUSY | 0x02 | The service request failed because the service is still busy |
|---|---|---|---|
| | CRYPTO_E_ENTROPY_EXHAUSTED | 0x04 | The service request failed because the entropy of the random number generator is exhausted |
| | CRYPTO_E_KEY_READ_FAIL | 0x06 | The service request failed because read access was denied |
| | CRYPTO_E_KEY_WRITE_FAIL | 0x07 | The service request failed because the writing access failed |
| | CRYPTO_E_KEY_NOT_AVAILABLE | 0x08 | The service request failed because at least one required key element is not available. |

▽

△

| | | | |
|---|---|---|---|
| | CRYPTO_E_KEY_NOT_VALID | 0x09 | The service request failed because the key is invalid. |
| | CRYPTO_E_KEY_SIZE_MISMATCH | 0x0A | The service request failed because the key size does not match. |
| | CRYPTO_E_JOB_CANCELED | 0x0C | The service request failed because the Job has been canceled. |
| | CRYPTO_E_KEY_EMPTY | 0x0D | The service request failed because of uninitialized source key element. |
| | CRYPTO_E_CUSTOM_ERROR | 0x0E | Custom processing failed. |
| ***Description*** | – | | |
| ***Available via*** | Crypto_GeneralTypes.h | | |

⌋

Notes:

- 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".)

### 8.2.2 CryIf_ConfigType

**[SWS_CryIf_91118] Definition of datatype CryIf_ConfigType** ⌈

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

⌋

## 8.3 Function definitions

This chapter contains all APIs provided by the module.

### 8.3.1 General API

**[SWS_CryIf_91000] Definition of API function CryIf_Init**

*Upstream requirements:* SRS_BSW_00101, SRS_BSW_00358, SRS_BSW_00414

⌈

| | | |
|---|---|---|
| ***Service Name*** | CryIf_Init | |
| ***Syntax*** | `void CryIf_Init ( const CryIf_ConfigType* configPtr )` | |
| ***Service ID [hex]*** | 0x00 | |
| ***Sync/Async*** | Synchronous | |
| ***Reentrancy*** | Non Reentrant | |
| ***Parameters (in)*** | configPtr | Pointer to a selected configuration structure |
| ***Parameters (inout)*** | None | |
| ***Parameters (out)*** | None | |
| ***Return value*** | None | |
| ***Description*** | Initializes the CRYIF module. | |
| ***Available via*** | CryIf.h | |

⌋

**[SWS_CryIf_91019] Configuration Pointer is always null**

*Upstream requirements:* SRS_BSW_00414

⌈The Configuration pointer `configPtr` shall always have a null pointer value.⌋

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

**[SWS_CryIf_00014] Report CRYIF_E_INIT_FAILED error**

*Upstream requirements:* SRS_BSW_00369

⌈If the initialization of the CRYIF module fails, the CRYIF shall report `CRYIF_E_INIT_FAILED` to the DET.⌋

**[SWS_CryIf_00015] Module Initialization Process**

*Upstream requirements:* SRS_BSW_00101

⌈The service `CryIf_Init` shall initialize the global variables and data structures of the CRYIF including flags and buffers.⌋

### 8.3.2 Job Processing Interface

To unite a single call function and a streaming approach for the crypto services, there is one interface `CryIf_ProcessJob`. Its job parameter `job` contains a `Crypto_OperationModeType` flag field (job->action->jobOperationMode), which can be set as "START", "UPDATE", "FINISH" or combination of them. It declares explicitly what

operation shall be performed. These operation modes can be mixed, and execute multiple operations at once.

To process a crypto service with a single call with `Crypto_ProcessJob` the operation mode is a disjunction of the 3 modes "START|UPDATE|FINISH".

**[SWS_CryIf_91003] Definition of API function CryIf_ProcessJob** ⌈

| | | |
|---|---|---|
| ***Service Name*** | CryIf_ProcessJob | |
| ***Syntax*** | `Std_ReturnType CryIf_ProcessJob ( uint32 channelId, Crypto_JobType* job )` | |
| ***Service ID [hex]*** | 0x03 | |
| ***Sync/Async*** | Depends on configuration | |
| ***Reentrancy*** | Reentrant | |
| ***Parameters (in)*** | channelId | Holds the identifier of the crypto channel. |
| ***Parameters (inout)*** | job | Pointer to the configuration of the job. Contains structures with user and primitive relevant information. |
| ***Parameters (out)*** | None | |
| ***Return value*** | Std_ReturnType | `E_OK`: Request successful<br>`E_NOT_OK`: Request failed<br>`CRYPTO_E_BUSY`: Request failed, Crypro Driver Object is busy<br>`CRYPTO_E_KEY_NOT_VALID`: Request failed, the key is not valid<br>`CRYPTO_E_KEY_SIZE_MISMATCH`: Request failed, a key element has the wrong size<br>`CRYPTO_E_KEY_READ_FAIL`: The service request failed, because key element extraction is not allowed<br>`CRYPTO_E_KEY_WRITE_FAIL`: The service request failed because the writing access failed<br>`CRYPTO_E_KEY_NOT_AVAILABLE`: The service request failed because the key is not available<br>`CRYPTO_E_JOB_CANCELED`: The service request failed because the synchronous Job has been canceled<br>`CRYPTO_E_KEY_EMPTY`: Request failed because of uninitialized source key element<br>`CRYPTO_E_ENTROPY_EXHAUSTED`<br>`CRYPTO_E_CUSTOM_ERROR`: Remote processing failed |
| ***Description*** | This interface dispatches the received jobs to the configured crypto driver object. | |
| ***Available via*** | CryIf.h | |

⌋

**[SWS_CryIf_00027] CryIf_ProcessJob API: Report CRYIF_E_UNINIT error**

*Upstream requirements:* SRS_BSW_00369, SRS_BSW_00406

⌈If development error detection for the CRYIF is enabled: The function `CryIf_ProcessJob` shall report `CRYIF_E_UNINIT` to the DET if the module is not yet initialized.⌋

**[SWS_CryIf_00028] CryIf_ProcessJob API: Report CRYIF_E_PARAM_HANDLE error**

*Upstream requirements:* SRS_BSW_00369

⌈If development error detection for the CRYIF is enabled: The function `CryIf_ProcessJob` shall report `CRYIF_E_PARAM_HANDLE` to the DET if the parameter channelId is out or range.⌋

**[SWS_CryIf_00029] CryIf_ProcessJob API: Report CRYIF_E_PARAM_POINTER error**

*Upstream requirements:* SRS_BSW_00369

⌈If development error detection for the CRYIF is enabled: The function `CryIf_ProcessJob` shall report `CRYIF_E_PARAM_POINTER` to the DET if the parameter job is a null pointer.⌋

**[SWS_CryIf_00161] CryIf_ProcessJob API: Report CRYIF_E_PARAM_HANDLE error** ⌈If development error detection for the CRYIF is enabled: The function `CryIf_ProcessJob` shall report `CRYIF_E_PARAM_HANDLE` to the DET if the parameter job->config->jobPrimitiveId is not in a valid range.⌋

**[SWS_CryIf_00162] CryIf_ProcessJob API: Report CRYIF_E_PARAM_HANDLE error** ⌈If development error detection for the CRYIF is enabled: The function `CryIf_ProcessJob` shall report `CRYIF_E_PARAM_HANDLE` to the DET if the parameter job->config->jobKeyId is not in a valid range and not equal to CRYPTO_NO_ID.⌋

**[SWS_CryIf_00163] CryIf_ProcessJob API: Report CRYIF_E_PARAM_HANDLE error** ⌈If development error detection for the CRYIF is enabled: The function `CryIf_ProcessJob` shall report `CRYIF_E_PARAM_HANDLE` to the DET if the parameter job->config->jobInputsCount is greater than CRYPTO_MAX_INPUTS.⌋

**[SWS_CryIf_00164] CryIf_ProcessJob API: Report CRYIF_E_PARAM_HANDLE error** ⌈If development error detection for the CRYIF is enabled: The function `CryIf_ProcessJob` shall report `CRYIF_E_PARAM_HANDLE` to the DET if the parameter job->config->jobInputRedirectionsPtr is not NULL_PTR and one of the members keyId of the parameters job->config->jobInputRedirectionsPtr[Idx], with Idx < job->config->jobInputsCount, is not in a valid range and not equal to CRYPTO_NO_ID.⌋

**[SWS_CryIf_00165] CryIf_ProcessJob API: Report CRYIF_E_PARAM_HANDLE error** ⌈If development error detection for the CRYIF is enabled: The function `CryIf_ProcessJob` shall report `CRYIF_E_PARAM_HANDLE` to the DET if the parameter job->config->jobOutputsCount is greater than CRYPTO_MAX_OUTPUTS.⌋

**[SWS_CryIf_00166] CryIf_ProcessJob API: Report CRYIF_E_PARAM_HANDLE error** ⌈If development error detection for the CRYIF is enabled: The function `CryIf_ProcessJob` shall report `CRYIF_E_PARAM_HANDLE` to the DET if the parameter job->config->jobOutputRedirectionsPtr is not NULL_PTR and one of the members keyId of the parameters job->config->jobOutputRedirectionsPtr[Idx], with Idx < job->config->jobOutputsCount, is not in a valid range and not equal to CRYPTO_NO_ID.⌋

**[SWS_CryIf_00167] CryIf_ProcessJob API: Report CRYIF_E_PARAM_HANDLE error** ⌈If development error detection for the CRYIF is enabled: The function `CryIf_ProcessJob` shall report `CRYIF_E_PARAM_HANDLE` to the DET if the parameter job->action->jobTargetKeyId is not in a valid range and not equal to CRYPTO_NO_ID.⌋

**[SWS_CryIf_00168] CryIf_ProcessJob API: Set the primitive ID** ⌈If the parameter job->config->jobPrimitiveId is in a valid range, then the function `CryIf_ProcessJob` shall set job->action->jobPrimitiveId with the primitive ID of the corresponding crypto driver.⌋

**[SWS_CryIf_00169] CryIf_ProcessJob API: Set the key ID** ⌈If the parameter job->config->jobKeyId is in a valid range and not equal to CRYPTO_NO_ID, then the function `CryIf_ProcessJob` shall set job->action->jobKeyId with the key ID of the corresponding crypto driver.⌋

**[SWS_CryIf_00189] CryIf_ProcessJob API: Set the key ID** ⌈If the parameter job->config->jobKeyId is equal to CRYPTO_NO_ID, then the function `CryIf_ProcessJob` shall set job->action->jobKeyId with CRYPTO_NO_ID.⌋

**[SWS_CryIf_00170] CryIf_ProcessJob API: Set the Job InputsPtr to valid key ID** ⌈If job->config->jobInputRedirectionsPtr is not NULL_PTR, then the function `CryIf_ProcessJob` shall set the parameters job->action->jobInputsPtr[Idx].keyId to the corresponding crypto driver keyId for all parameters job->config->jobInputRedirectionsPtr[Idx], with Idx < job->config->jobInputsCount, whose members keyId are in a valid range.⌋

**[SWS_CryIf_00171] CryIf_ProcessJob API: Set the Job InputsPtr to EMPTY key ID** ⌈If job->config->jobInputRedirectionsPtr is not NULL_PTR, then the function `CryIf_ProcessJob` shall set the parameters job->action->jobInputsPtr[Idx].keyId to CRYPTO_NO_ID for all parameters job->config->jobInputRedirectionsPtr[Idx], with Idx < job->config->jobInputsCount, whose members keyId have the value CRYPTO_NO_ID.⌋

**[SWS_CryIf_00172] CryIf_ProcessJob API: Set the Job OutputsPtr to valid key ID** ⌈If job->config->jobOutputRedirectionsPtr is not NULL_PTR, then the function `CryIf_ProcessJob` shall set the parameters job->action->jobOutputsPtr[Idx].keyId to the corresponding crypto driver keyId for all parameters job->config->jobOutputRedirectionsPtr[Idx], with Idx < job->config->jobOutputsCount, whose members keyId are in a valid range.⌋

**[SWS_CryIf_00173] CryIf_ProcessJob API: Set the Job OutputsPtr to EMPTY key ID** ⌈If job->config->jobOutputRedirectionsPtr is not NULL_PTR, then the function `CryIf_ProcessJob` shall set the parameters job->action->jobOutputsPtr[Idx].keyId to CRYPTO_NO_ID for all parameters job->config->jobOutputRedirectionsPtr[Idx], with Idx < job->config->jobOutputsCount, whose members keyId have the value CRYPTO_NO_ID.⌋

**[SWS_CryIf_00174] CryIf_ProcessJob API: Set the Job TargetKeyId to valid key ID** ⌈If the parameter job->action->jobTargetKeyId is in a valid range and not equal to CRYPTO_NO_ID, then the function `CryIf_ProcessJob` shall overwrite job->action->jobTargetKeyId with the key ID of the corresponding crypto driver.⌋

**[SWS_CryIf_00044] CryIf_ProcessJob API: Forward crypto job to the mapped driver** ⌈If no errors are detected by CRYIF, the service `CryIf_ProcessJob` shall call Crypto_<vi>_<ai>_ProcessJob for the driver configuration mapped to the service and pass on the return value.⌋

### 8.3.3 Job Cancellation Interface

**[SWS_CryIf_91014] Definition of API function CryIf_CancelJob** ⌈

| | | |
|---|---|---|
| ***Service Name*** | CryIf_CancelJob | |
| ***Syntax*** | `Std_ReturnType CryIf_CancelJob ( uint32 channelId, Crypto_JobType* job )` | |
| ***Service ID [hex]*** | 0x0e | |
| ***Sync/Async*** | Synchronous | |
| ***Reentrancy*** | Reentrant | |
| ***Parameters (in)*** | channelId | Holds the identifier of the crypto channel. |
| ***Parameters (inout)*** | job | Pointer to the configuration of the job. Contains structures with user and primitive relevant information. |
| ***Parameters (out)*** | None | |
| ***Return value*** | Std_ReturnType | `E_OK`: Request successful, job has been removed<br>`E_NOT_OK`: Request failed, job couldn't be removed<br>`CRYPTO_E_JOB_CANCELED` |
| ***Description*** | This interface dispatches the job cancellation function to the configured crypto driver object. | |
| ***Available via*** | CryIf.h | |

⌋

**[SWS_CryIf_00129] CryIf_CancelJob API: Report CRYIF_E_UNINIT error**

*Upstream requirements:* SRS_BSW_00369, SRS_BSW_00406

⌈If development error detection for the CRYIF is enabled: The function `CryIf_CancelJob` shall report `CRYIF_E_UNINIT` to the DET if the module is not yet initialized.⌋

**[SWS_CryIf_00130] CryIf_CancelJob API: Report CRYIF_E_PARAM_HANDLE error**

*Upstream requirements:* SRS_BSW_00369

⌈If development error detection for the CRYIF is enabled: The function `CryIf_CancelJob` shall report `CRYIF_E_PARAM_HANDLE` to the DET if the parameter `channelId` is out of range.⌋

**[SWS_CryIf_00131] CryIf_CancelJob API: Report CRYIF_E_PARAM_POINTER error**

*Upstream requirements:* SRS_BSW_00369

⌈If development error detection for the CRYIF is enabled: The function `CryIf_CancelJob` shall report `CRYIF_E_PARAM_POINTER` to the DET if the parameter `job` is a `null` pointer.⌋

**[SWS_CryIf_00175] CryIf_CancelJob API: Report CRYIF_E_PARAM_HANDLE error** ⌈If development error detection for the CRYIF is enabled: The function `CryIf_CancelJob` shall report `CRYIF_E_PARAM_HANDLE` to the DET if the parameter job->config->jobPrimitiveId is not in a valid range.⌋

**[SWS_CryIf_00176] CryIf_CancelJob API: Report CRYIF_E_PARAM_HANDLE error** ⌈If development error detection for the CRYIF is enabled: The function `CryIf_CancelJob` shall report `CRYIF_E_PARAM_HANDLE` to the DET if the parameter job->config->jobKeyId is not in a valid range and not equal to CRYPTO_NO_ID.⌋

**[SWS_CryIf_00177] CryIf_CancelJob API: Report CRYIF_E_PARAM_HANDLE error** ⌈If development error detection for the CRYIF is enabled: The function `CryIf_CancelJob` shall report `CRYIF_E_PARAM_HANDLE` to the DET if the parameter job->config->jobInputsCount is greater than CRYPTO_MAX_INPUTS.⌋

**[SWS_CryIf_00178] CryIf_CancelJob API: Report CRYIF_E_PARAM_HANDLE error** ⌈If development error detection for the CRYIF is enabled: The function `CryIf_CancelJob` shall report `CRYIF_E_PARAM_HANDLE` to the DET if the parameter job->config->jobInputRedirectionsPtr is not NULL_PTR and one of the members keyId of the parameters job->config->jobInputRedirectionsPtr[Idx] with Idx < job->config->jobInputsCount, is not in a valid range and not equal to CRYPTO_NO_ID.⌋

**[SWS_CryIf_00179] CryIf_CancelJob API: Report CRYIF_E_PARAM_HANDLE error** ⌈If development error detection for the CRYIF is enabled: The function `CryIf_CancelJob` shall report `CRYIF_E_PARAM_HANDLE` to the DET if the parameter job->config->jobOutputsCount is greater than CRYPTO_MAX_OUTPUTS.⌋

**[SWS_CryIf_00180] CryIf_CancelJob API: Report CRYIF_E_PARAM_HANDLE error** ⌈If development error detection for the CRYIF is enabled: The function `CryIf_CancelJob` shall report `CRYIF_E_PARAM_HANDLE` to the DET if the parameter job->config->jobOutputRedirectionsPtr is not NULL_PTR and one of the members keyId of the parameters job->config->jobOutputRedirectionsPtr[Idx] with Idx < job->config->jobOutputsCount, is not in a valid range and not equal to CRYPTO_NO_ID.⌋

**[SWS_CryIf_00181] CryIf_CancelJob API: Report CRYIF_E_PARAM_HANDLE error** ⌈If development error detection for the CRYIF is enabled: The function `CryIf_CancelJob` shall report `CRYIF_E_PARAM_HANDLE` to the DET if the parameter job->action->jobTargetKeyId is not in a valid range and not equal to CRYPTO_NO_ID.⌋

**[SWS_CryIf_00182] CryIf_CancelJob API: Set the primitive ID** ⌈If the parameter job->config->jobPrimitiveId is in a valid range, then the function `CryIf_CancelJob` shall set job->action->jobPrimitiveId with the primitive ID of the corresponding crypto driver.⌋

**[SWS_CryIf_00183] CryIf_CancelJob API: Set the key ID** ⌈If the parameter job->config->jobKeyId is in a valid range and not equal to CRYPTO_NO_ID, then the function `CryIf_CancelJob` shall set job->action->jobKeyId with the key ID of the corresponding crypto driver.⌋

**[SWS_CryIf_00190] CryIf_CancelJob API: Set the key ID** ⌈If the parameter job->config->jobKeyId is equal to CRYPTO_NO_ID, then the function `CryIf_CancelJob` shall set job->action->jobKeyId with CRYPTO_NO_ID.⌋

**[SWS_CryIf_00184] CryIf_CancelJob API: Set the Job Input Redirections with valid Key IDs** ⌈If job->config->jobInputRedirectionsPtr is not NULL_PTR, then the function `CryIf_CancelJob` shall set the parameters job->action->jobInputsPtr[Idx].keyId to the corresponding crypto driver keyId for all parameters job->config->jobInputRedirectionsPtr[Idx] with Idx < job->config->jobInputsCount whose members keyId are in a valid range.⌋

**[SWS_CryIf_00185] CryIf_CancelJob API: Set the Job Input Redirections with valid CRYPTO_NO_ID** ⌈If job->config->jobInputRedirectionsPtr is not NULL_PTR, then the function `CryIf_CancelJob` shall set the parameters job->action->jobInputsPtr[Idx].keyId to CRYPTO_NO_ID for all parameters job->config->jobInputRedirectionsPtr[Idx] with Idx < job->config->jobInputsCount whose members keyId have the value CRYPTO_NO_ID.⌋

**[SWS_CryIf_00186] CryIf_CancelJob API: Set the Job Output Redirections with valid Key IDs** ⌈If job->config->jobOutputRedirectionsPtr is not NULL_PTR, then the function `CryIf_CancelJob` shall set the parameters job->action->jobOutputsPtr[Idx].keyId to the corresponding crypto driver keyId for all parameters job->config->jobOutputRedirectionsPtr[Idx] with Idx < job->config->jobOutputsCount whose members keyId are in a valid range.⌋

**[SWS_CryIf_00187] CryIf_CancelJob API: Set the Job Output Redirections with valid CRYPTO_NO_ID** ⌈If job->config->jobOutputRedirectionsPtr is not NULL_PTR, then the function `CryIf_CancelJob` shall set the parameters job->action->jobOutputsPtr[Idx].keyId to CRYPTO_NO_ID for all parameters job->config->jobOutputRedirectionsPtr[Idx] with Idx < job->config->jobOutputsCount whose members keyId have the value CRYPTO_NO_ID.⌋

**[SWS_CryIf_00188] CryIf_CancelJob API: Set the Job TargetKeyId to valid key ID** ⌈If the parameter job->action->jobTargetKeyId is in a valid range and not equal to CRYPTO_NO_ID, then the function `CryIf_CancelJob` shall overwrite job->action->jobTargetKeyId with the key ID of the corresponding crypto driver.⌋

**[SWS_CryIf_00132] CryIf_CancelJob API: Forward job cancel to the mapped driver** ⌈If no errors are detected by CRYIF, the service `CryIf_CancelJob` shall call Crypto_<vi>_<ai>_CancelJob for the driver configuration mapped to the service and pass on the return value.⌋

### 8.3.4 Key Management Interface

#### 8.3.4.1 Key Setting Interface

**[SWS_CryIf_91004] Definition of API function CryIf_KeyElementSet** ⌈

| | | |
|---|---|---|
| ***Service Name*** | CryIf_KeyElementSet | |
| ***Syntax*** | `Std_ReturnType CryIf_KeyElementSet ( uint32 cryIfKeyId, uint32 keyElementId, const uint8* keyPtr, uint32 keyLength )` | |
| ***Service ID [hex]*** | 0x04 | |
| ***Sync/Async*** | Synchronous | |
| ***Reentrancy*** | Non Reentrant | |
| ***Parameters (in)*** | cryIfKeyId | 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<br>E_NOT_OK: Request failed<br>CRYPTO_E_BUSY: Request failed, Crypto Driver Object is busy<br>CRYPTO_E_KEY_WRITE_FAIL:Request failed because write access was denied<br>CRYPTO_E_KEY_NOT_AVAILABLE: Request failed because the key is not available<br>CRYPTO_E_KEY_SIZE_MISMATCH: Request failed, key element size does not match size of provided data |
| ***Description*** | This function shall dispatch the set key element function to the configured crypto driver object. | |
| ***Available via*** | CryIf.h | |

⌋

**[SWS_CryIf_00049] CryIf_KeyElementSet API: Report CRYIF_E_PARAM_POINTER error**

*Upstream requirements:* SRS_BSW_00369, SRS_BSW_00406

⌈If development error detection for the CRYIF module is enabled: The function `CryIf_KeyElementSet` shall report `CRYIF_E_UNINIT` to the DET if the module is not yet initialized.⌋

**[SWS_CryIf_00050] CryIf_KeyElementSet API: Report CRYIF_E_PARAM_HANDLE error**

*Upstream requirements:* SRS_BSW_00369

⌈If development error detection for the CRYIF module is enabled: The function `CryIf_KeyElementSet` shall report `CRYIF_E_PARAM_HANDLE` to the DET if the parameter `cryIfKeyId` is out of range.⌋

**[SWS_CryIf_00052] CryIf_KeyElementSet API: Report CRYIF_E_PARAM_POINTER error**

*Upstream requirements:* SRS_BSW_00369

⌈If development error detection for the CRYIF module is enabled: The function `CryIf_KeyElementSet` shall report `CRYIF_E_PARAM_POINTER` to the DET if the parameter `keyPtr` is a `null` pointer.⌋

**[SWS_CryIf_00053] CryIf_KeyElementSet API: Report CRYIF_E_PARAM_VALUE error**

*Upstream requirements:* SRS_BSW_00369

⌈If development error detection for the CRYIF is enabled: The function `CryIf_KeyElementSet` shall report `CRYIF_E_PARAM_VALUE` to the DET if `keyLength` is zero.⌋

**[SWS_CryIf_00055] CryIf_KeyElementSet API: Forward key job to the mapped driver** ⌈If no errors are detected by CRYIF, the service `CryIf_KeyElementSet` shall call Crypto_<vi>_<ai>_KeyElementSet for the driver configuration mapped to the service and pass on the return value.⌋

**[SWS_CryIf_91005] Definition of API function CryIf_KeySetValid** ⌈

| | | |
|---|---|---|
| ***Service Name*** | CryIf_KeySetValid | |
| ***Syntax*** | `Std_ReturnType CryIf_KeySetValid ( uint32 cryIfKeyId )` | |
| ***Service ID [hex]*** | 0x05 | |
| ***Sync/Async*** | Synchronous | |
| ***Reentrancy*** | Non Reentrant | |
| ***Parameters (in)*** | cryIfKeyId | Holds the identifier of the key whose key elements shall be set to valid. |
| ***Parameters (inout)*** | None | |
| ***Parameters (out)*** | None | |
| ***Return value*** | Std_ReturnType | `E_OK`: Request successful<br>`E_NOT_OK`: Request failed<br>`CRYPTO_E_BUSY`: Request failed, Crypro Driver Object is busy |
| ***Description*** | This function shall dispatch the set key valid function to the configured crypto driver object. | |
| ***Available via*** | CryIf.h | |

⌋

**[SWS_CryIf_00056] CryIf_KeySetValid API: Report CRYIF_E_UNINIT error**

*Upstream requirements:* SRS_BSW_00369, SRS_BSW_00406

⌈If development error detection for the CRYIF module is enabled: The function `CryIf_KeySetValid` shall report `CRYIF_E_UNINIT` to the DET if the module is not yet initialized.⌋

**[SWS_CryIf_00057] CryIf_KeySetValid API: Report CRYIF_E_PARAM_HANDLE error**

*Upstream requirements:* SRS_BSW_00369

⌈If development error detection for the CRYIF module is enabled: The function `CryIf_KeySetValid` shall report `CRYIF_E_PARAM_HANDLE` to the DET if the parameter `cryIfKeyId` is out of range.⌋

**[SWS_CryIf_00058] CryIf_KeySetValid API: Forward key job to the mapped driver** ⌈If no errors are detected by CRYIF, the service `CryIf_KeySetValid` shall call Crypto_<vi>_<ai>_KeySetValid for the driver configuration mapped to the service and pass on the return value.⌋

**[SWS_CryIf_91021] Definition of API function CryIf_KeySetInvalid** ⌈

| | | |
|---|---|---|
| ***Service Name*** | CryIf_KeySetInvalid | |
| ***Syntax*** | `Std_ReturnType CryIf_KeySetInvalid ( uint32 cryIfKeyId )` | |
| ***Service ID [hex]*** | 0x14 | |
| ***Sync/Async*** | Synchronous | |
| ***Reentrancy*** | Non Reentrant | |
| ***Parameters (in)*** | cryIfKeyId | Holds the identifier of the key for which the status shall be set to invalid. |
| ***Parameters (inout)*** | None | |
| ***Parameters (out)*** | None | |
| ***Return value*** | Std_ReturnType | `E_OK`: Request successful<br>`E_NOT_OK`: Request failed<br>`CRYPTO_E_BUSY`: Request failed, Crypro Driver Object is busy |
| ***Description*** | Sets invalid for the status of the key identified by cryIfKeyId. | |
| ***Available via*** | CryIf.h | |

⌋

**[SWS_CryIf_00150] CryIf_KeySetInvalid API: Report CRYIF_E_UNINIT error**

*Upstream requirements:* SRS_BSW_00369, SRS_BSW_00406

⌈If development error detection for the CryIf module is enabled, the function `CryIf_KeySetInvalid` shall report the error `CRYIF_E_UNINIT` if the module is not yet initialized.⌋

**[SWS_CryIf_00151] CryIf_KeySetInvalid API: Report CRYIF_E_PARAM_HANDLE error** ⌈If development error detection for the CryIf module is enabled, the function `CryIf_KeySetInvalid` shall report the error `CRYIF_E_PARAM_HANDLE` if the parameter `cryIfKeyId` is out of range.⌋

**[SWS_CryIf_00152] CryIf_KeySetInvalid API: Forward key job to the mapped driver** ⌈If no errors are detected by CryIf, the service `CryIf_KeySetInvalid` shall call Crypto_<vi>_<ai>_KeySetInvalid for the driver configuration mapped to the service and pass on the return value.⌋

#### 8.3.4.2 Key Status Interface

**[SWS_CryIf_91012] Definition of API function CryIf_KeyGetStatus** ⌈

| | | |
|---|---|---|
| ***Service Name*** | CryIf_KeyGetStatus | |
| ***Syntax*** | `Std_ReturnType CryIf_KeyGetStatus ( uint32 cryIfKeyId, Crypto_KeyStatusType* keyStatusPtr )` | |
| ***Service ID [hex]*** | 0x13 | |
| ***Sync/Async*** | Synchronous | |
| ***Reentrancy*** | Non Reentrant | |
| ***Parameters (in)*** | cryIfKeyId | Holds the identifier of the key for which the key state shall be returned. |
| ***Parameters (inout)*** | None | |
| ***Parameters (out)*** | keyStatusPtr | Contains the pointer to the data where the status of the key shall be stored. |
| ***Return value*** | Std_ReturnType | `E_OK`: Request successful<br>`E_NOT_OK`: Request failed |
| ***Description*** | Returns the key state of the key identified by cryIfKeyId. | |
| ***Available via*** | CryIf.h | |

⌋

**[SWS_CryIf_00146] CryIf_KeyGetStatus API: Report CRYIF_E_UNINIT error**

*Upstream requirements:* SRS_BSW_00369, SRS_BSW_00406

⌈If development error detection for the CryIf module is enabled, the function `CryIf_KeyGetStatus` shall report `CRYIF_E_UNINIT` to the DET if the module is not yet initialized.⌋

**[SWS_CryIf_00147] CryIf_KeyGetStatus API: Report CRYIF_E_PARAM_HANDLE error** ⌈If development error detection for the CryIf module is enabled, the function `CryIf_KeyGetStatus` shall report `CRYIF_E_PARAM_HANDLE` to the DET if the parameter `cryIfKeyId` is out of range.⌋

**[SWS_CryIf_00148] CryIf_KeyGetStatus API: Report CRYIF_E_PARAM_POINTER error** ⌈If development error detection for the CryIf module is enabled, the function `CryIf_KeyGetStatus` shall report `CRYIF_E_PARAM_POINTER` to the DET if the parameter `keyStatusPtr` is a `null` pointer.⌋

**[SWS_CryIf_00149] CryIf_KeyGetStatus API: Forward key job to the mapped driver** ⌈If no errors are detected by CryIf, the service `CryIf_KeyGetStatus` shall call Crypto_<vi>_<ai>_KeyGetStatus for the driver configuration mapped to the service and pass on the return value⌋

#### 8.3.4.3 Key Extraction Interface

**[SWS_CryIf_91006] Definition of API function CryIf_KeyElementGet ⌈**

| | | |
|---|---|---|
| ***Service Name*** | CryIf_KeyElementGet | |
| ***Syntax*** | `Std_ReturnType CryIf_KeyElementGet ( uint32 cryIfKeyId, uint32 keyElementId, uint8* resultPtr, uint32* resultLengthPtr )` | |
| ***Service ID [hex]*** | 0x06 | |
| ***Sync/Async*** | Synchronous | |
| ***Reentrancy*** | Reentrant | |
| ***Parameters (in)*** | cryIfKeyId | 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<br>`E_NOT_OK`: Request failed<br>`CRYPTO_E_BUSY`: Request failed, Crypto Driver Object is busy<br>`CRYPTO_E_KEY_NOT_AVAILABLE`: Request failed, the requested key element is not available<br>`CRYPTO_E_KEY_READ_FAIL`: Request failed because read access was denied<br>`CRYPTO_E_KEY_EMPTY`: Request failed because of uninitialized source key element |
| ***Description*** | This function shall dispatch the get key element function to the configured crypto driver object. | |
| ***Available via*** | CryIf.h | |

⌋

**[SWS_CryIf_00059] CryIf_KeyElementGet API: Report CRYIF_E_UNINIT error**

*Upstream requirements:* SRS_BSW_00369, SRS_BSW_00406

⌈If development error detection for the CRYIF module is enabled: The function `CryIf_KeyElementGet` shall report `CRYIF_E_UNINIT` to the DET if the module is not yet initialized.⌋

**[SWS_CryIf_00060] CryIf_KeyElementGet API: Report CRYIF_E_PARAM_HANDLE error**

*Upstream requirements:* SRS_BSW_00369

⌈If development error detection for the CRYIF module is enabled: The function `CryIf_KeyElementGet` shall report `CRYIF_E_PARAM_HANDLE` to the DET if the parameter `cryIfKeyId` is out of range.⌋

**[SWS_CryIf_00062] CryIf_KeyElementGet API: Report CRYIF_E_PARAM_POINTER error for resultPtr**

*Upstream requirements:* SRS_BSW_00369

⌈If development error detection for the CRYIF module is enabled: The function `CryIf_KeyElementGet` shall report `CRYIF_E_PARAM_POINTER` to the DET if the parameter `resultPtr` is a `null` pointer.⌋

**[SWS_CryIf_00063] CryIf_KeyElementGet API: Report CRYIF_E_PARAM_POINTER error for resultLengthPtr**

*Upstream requirements:* SRS_BSW_00369

⌈If development error detection for the CRYIF module is enabled: The function `CryIf_KeyElementGet` shall report `CRYIF_E_PARAM_POINTER` to the DET if the parameter `resultLengthPtr` is a `null` pointer.⌋

**[SWS_CryIf_00064] CryIf_KeyElementGet API: Report CRYIF_E_PARAM_VALUE error**

*Upstream requirements:* SRS_BSW_00369

⌈If development error detection for the CRYIF module is enabled: The function `CryIf_KeyElementGet` shall report `CRYIF_E_PARAM_VALUE` to the DET if the value, which is pointed by `resultLengthPtr`, is zero.⌋

**[SWS_CryIf_00065] CryIf_KeyElementGet API: Forward key job to the mapped driver** ⌈If no errors are detected by CRYIF, the service `CryIf_KeyElementGet` shall call Crypto_<vi>_<ai>_KeyElementGet for the driver configuration mapped to the service and pass on the return value.⌋

### 8.3.4.4 Key Copying Interface

**[SWS_CryIf_91015] Definition of API function CryIf_KeyElementCopy** ⌈

| | | |
|---|---|---|
| ***Service Name*** | CryIf_KeyElementCopy | |
| ***Syntax*** | `Std_ReturnType CryIf_KeyElementCopy ( uint32 cryIfKeyId, uint32 keyElementId, uint32 targetCryIfKeyId, uint32 targetKeyElementId )` | |
| ***Service ID [hex]*** | 0x0f | |
| ***Sync/Async*** | Synchronous | |
| ***Reentrancy*** | Reentrant, but not for the same cryIfKeyId | |
| ***Parameters (in)*** | cryIfKeyId | 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. |

▽

△

| | | |
|---|---|---|
| | targetCryIfKeyId | 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<br>`E_NOT_OK`: Request failed<br>`CRYPTO_E_BUSY`: Request failed, Crypto Driver Object is busy<br>`CRYPTO_E_KEY_NOT_AVAILABLE`: Request failed, the requested key element is not available<br>`CRYPTO_E_KEY_READ_FAIL`: Request failed, not allowed to extract key element<br>`CRYPTO_E_KEY_WRITE_FAIL`: Request failed, not allowed to write key element<br>`CRYPTO_E_KEY_SIZE_MISMATCH`: Request failed, key element sizes are not compatible<br>`CRYPTO_E_KEY_EMPTY`: Request failed because of uninitialized source key element |
| ***Description*** | This function shall copy a key elements from one key to a target key. | |
| ***Available via*** | CryIf.h | |

⌋

**[SWS_CryIf_00110] CryIf_KeyElementCopy API: Report CRYIF_E_UNINIT error**

*Upstream requirements:* SRS_BSW_00369, SRS_BSW_00406

⌈If development error detection for the CRYIF is enabled: The function `CryIf_KeyElementCopy` shall report `CRYIF_E_UNINIT` to the DET if the module is not yet initialized.⌋

**[SWS_CryIf_00111] CryIf_KeyElementCopy API: Report CRYIF_E_PARAM_HANDLE error for cryIfKeyId parameter**

*Upstream requirements:* SRS_BSW_00369

⌈If development error detection for the CRYIF is enabled: The function `CryIf_KeyElementCopy` shall report `CRYIF_E_PARAM_HANDLE` to the DET if the parameter `cryIfKeyId` is out of range.⌋

**[SWS_CryIf_00112] CryIf_KeyElementCopy API: Report CRYIF_E_PARAM_HANDLE error for targetCryIfKeyId parameter**

*Upstream requirements:* SRS_BSW_00369

⌈If development error detection for the CRYIF is enabled: The function `CryIf_KeyElementCopy` shall report `CRYIF_E_PARAM_HANDLE` to the DET if the parameter `targetCryIfKeyId` is out of range.⌋

**[SWS_CryIf_00113] CryIf_KeyElementCopy API: Forward key job to the mapped driver in case of cryIfKeyId and targetCryIfKeyId are located in the same Crypto Driver** ⌈If no errors are detected by CRYIF and the cryIfKeyId and targetCryIfKeyId are located in the same Crypto Driver, the service `CryIf_KeyElementCopy` shall call Crypto_<vi>_<ai>_KeyElementCopy for the driver configuration mapped to the service and pass on the return value.⌋

**[SWS_CryIf_00114] CryIf_KeyElementCopy API: Forward key job to the mapped driver in case of cryIfKeyId and targetCryIfKeyId are located in the different Crypto Driver** ⌈If no errors are detected by CRYIF and the cryIfKeyId and targetCryIfKeyId are located in different Crypto Drivers, the service `CryIf_KeyElementCopy` shall copy the provided key element by getting the element with Crypto_<vi>_<ai>_KeyElementGet and setting the target key element via Crypto_<vi>_<ai>_KeyElementSet.⌋

**[SWS_CryIf_00115] CryIf_KeyElementCopy API: Report CRYIF_E_KEY_SIZE_MISMATCH error**

*Upstream requirements:* SRS_BSW_00369

⌈If development error detection for the CRYIF is enabled: If requested key element of `cryIfKeyId` is available in `targetCryIfKeyId`, and if the source element size does not match the target key elements size, `CryIf_KeyElementCopy` shall report `CRYIF_E_KEY_SIZE_MISMATCH` to the DET.⌋

**[SWS_CryIf_91018] Definition of API function CryIf_KeyElementCopyPartial** ⌈

| | | |
|---|---|---|
| ***Service Name*** | CryIf_KeyElementCopyPartial | |
| ***Syntax*** | `Std_ReturnType CryIf_KeyElementCopyPartial ( uint32 cryIfKeyId, uint32 keyElementId, uint32 keyElementSourceOffset, uint32 keyElementTargetOffset, uint32 keyElementCopyLength, uint32 targetCryIfKeyId, uint32 targetKeyElementId )` | |
| ***Service ID [hex]*** | 0x12 | |
| ***Sync/Async*** | Synchronous | |
| ***Reentrancy*** | Reentrant but not for the same cryIfKeyId | |
| ***Parameters (in)*** | cryIfKeyId | 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 source key element indicating the start index of the copy operation. |
| | keyElementTargetOffset | This is the offset of the target key element indicating the start index of the copy operation. |
| | keyElementCopyLength | Specifies the number of bytes that shall be copied. |
| | targetCryIfKeyId | 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<br>E_NOT_OK: Request failed<br>CRYPTO_E_BUSY: Request failed, Crypto Driver Object is busy<br>CRYPTO_E_KEY_NOT_AVAILABLE: Request failed, the requested key element is not available<br>CRYPTO_E_KEY_READ_FAIL: Request failed, not allowed to extract key element<br>CRYPTO_E_KEY_WRITE_FAIL: Request failed, not allowed to write key element<br>CRYPTO_E_KEY_SIZE_MISMATCH: Request failed, key element sizes are not compatible<br>CRYPTO_E_KEY_EMPTY: Request failed because of uninitialized source key element |
| ***Description*** | Copies a key element to another key element. The keyElementOffsets and keyElementCopy Length allows to copy just parts of the source key element into the destination key element. | |
| ***Available via*** | CryIf.h | |

⌋

**[SWS_CryIf_00137] CryIf_KeyElementCopyPartial API: Report CRYIF_E_UNINIT error**

*Upstream requirements:* SRS_BSW_00369, SRS_BSW_00406

⌈If the Crypto Interface is not yet initialized and if development error detection for the Crypto Interface is enabled, the function CryIf_KeyElementCopyPartial shall report CRYIF_E_UNINIT to the DET.⌋

**[SWS_CryIf_00138] CryIf_KeyElementCopyPartial API: Report CRYIF_E_PARAM_HANDLE error** ⌈If cryIfKeyId or targetCryIfKeyId is out of range and if development error detection for the Crypto Interface is enabled, the function CryIf_KeyElementCopyPartial shall report CRYIF_E_PARAM_HANDLE to the DET.⌋

**[SWS_CryIf_00139] CryIf_KeyElementCopyPartial API: Forward key job to the mapped driver in case of cryIfKeyId and targetCryIfKeyId are located in the same Crypto Driver** ⌈If no errors are detected by CRYIF and the cryIfKeyId and targetCryIfKeyId are located in the same Crypto Driver, the service CryIf_KeyElementCopyPartial shall call Crypto_<vi>_<ai>_KeyElementCopyPartial for the driver configuration mapped to the service and pass on the return value.⌋

**[SWS_CryIf_00140] CryIf_KeyElementCopyPartial API: Forward key job to the mapped driver in case of cryIfKeyId and targetCryIfKeyId are located in the different Crypto Driver** ⌈If no errors are detected by CRYIF and the cryIfKeyId and targetCryIfKeyId are located in different Crypto Drivers, the service CryIf_KeyElementCopyPartial shall copy the provided key element by getting the element with Crypto_<vi>_<ai>_KeyElementGet, copy the partial data to its destination and setting the target key element via Crypto_<vi>_<ai>_KeyElementSet.⌋

### [SWS_CryIf_91016] Definition of API function CryIf_KeyCopy ⌈

| | | |
|---|---|---|
| ***Service Name*** | CryIf_KeyCopy | |
| ***Syntax*** | `Std_ReturnType CryIf_KeyCopy ( uint32 cryIfKeyId, uint32 targetCryIfKeyId )` | |
| ***Service ID [hex]*** | 0x10 | |
| ***Sync/Async*** | Synchronous | |
| ***Reentrancy*** | Reentrant but not for the same cryIfKeyId | |
| ***Parameters (in)*** | cryIfKeyId | Holds the identifier of the key whose key element shall be the source element. |
| | targetCryIfKeyId | 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<br>`E_NOT_OK`: Request failed<br>`CRYPTO_E_BUSY`: Request failed, Crypto Driver Object is busy<br>`CRYPTO_E_KEY_NOT_AVAILABLE`: Request failed, the requested key element is not available<br>`CRYPTO_E_KEY_READ_FAIL`: Request failed, not allowed to extract key element<br>`CRYPTO_E_KEY_WRITE_FAIL`: Request failed, not allowed to write key element<br>`CRYPTO_E_KEY_SIZE_MISMATCH`: Request failed, key element sizes are not compatible<br>`CRYPTO_E_KEY_EMPTY`: Request failed because of uninitialized source key element |
| ***Description*** | This function shall copy all key elements from the source key to a target key. | |
| ***Available via*** | CryIf.h | |

⌋

### [SWS_CryIf_00116] CryIf_KeyCopy API: Report CRYIF_E_UNINIT error

*Upstream requirements:* SRS_BSW_00369, SRS_BSW_00406

⌈If development error detection for the CRYIF is enabled: The function `CryIf_KeyCopy` shall report `CRYIF_E_UNINIT` to the DET if the module is not yet initialized.⌋

### [SWS_CryIf_00117] CryIf_KeyCopy API: Report CRYIF_E_PARAM_HANDLE error for cryIfKeyId parameter

*Upstream requirements:* SRS_BSW_00369

⌈If development error detection for the CRYIF is enabled: The function `CryIf_KeyCopy` shall report `CRYIF_E_PARAM_HANDLE` to the DET if the parameter `cryIfKeyId` is out of range.⌋

### [SWS_CryIf_00118] CryIf_KeyCopy API: Report CRYIF_E_PARAM_HANDLE error for targetCryIfKeyId parameter

*Upstream requirements:* SRS_BSW_00369

⌈If development error detection for the CRYIF is enabled: The function `CryIf_KeyCopy` shall report `CRYIF_E_PARAM_HANDLE` to the DET if the parameter `targetCryIfKeyId` is out of range.⌋

**[SWS_CryIf_00119] CryIf_KeyCopy API: Forward key job to the mapped driver in case of cryIfKeyId and targetCryIfKeyId are located in the same Crypto Driver** ⌈If no errors are detected by CRYIF and the `cryIfKeyId` and `targetCryIfKeyId` are located in the same Crypto Driver, the service `CryIf_KeyCopy` shall call Crypto_<vi>_<ai>_KeyCopy for the driver configuration mapped to the service and pass on the return value.⌋

**[SWS_CryIf_00120] CryIf_KeyCopy API: Forward key job to the mapped driver in case of cryIfKeyId and targetCryIfKeyId are located in the different Crypto Driver** ⌈If no errors are detected by CRYIF and the `cryIfKeyId` and `targetCryIfKeyId` are located in different Crypto Drivers, the service `CryIf_KeyCopy` shall transfer the key elements of the source key to the target key. First, a list of key elements from `cryIfKeyId` and `targetCryIfKeyId` shall be read using the function Crypto_<vi>_<ai>_KeyElementIdsGet. All key elements from this list that are identical to each other shall be copied by reading each key element of `cryIfKeyId` with Crypto_<vi>_<ai>_KeyElementGet and setting the target key element of `targetCryIfKeyId` via Crypto_<vi>_<ai>_KeyElementSet.⌋

**[SWS_CryIf_00121] CryIf_KeyCopy API: Report CRYIF_E_KEY_SIZE_MISMATCH error**

*Upstream requirements:* SRS_BSW_00369

⌈If development error detection for the CRYIF is enabled: For all key elements of `cryIfKeyId` that are available in `targetCryIfKeyId`, if the source element size does not match the target key elements size, `CryIf_KeyCopy` shall report `CRYIF_E_KEY_SIZE_MISMATCH` to the DET.⌋

### 8.3.4.5 Key Generation Interface

**[SWS_CryIf_91007] Definition of API function CryIf_RandomSeed** ⌈

| | | |
|---|---|---|
| ***Service Name*** | CryIf_RandomSeed | |
| ***Syntax*** | `Std_ReturnType CryIf_RandomSeed ( uint32 cryIfKeyId, const uint8* seedPtr, uint32 seedLength )` | |
| ***Service ID [hex]*** | 0x07 | |
| ***Sync/Async*** | Synchronous | |
| ***Reentrancy*** | Reentrant | |
| ***Parameters (in)*** | cryIfKeyId | 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<br>E_NOT_OK: Request failed<br>CRYPTO_E_KEY_NOT_VALID: Request failed, the key state of the key identified by cryptoKeyId is "invalid".<br>CRYPTO_E_KEY_EMPTY: Request failed because of uninitialized source key element |
| ***Description*** | This function shall dispatch the random seed function to the configured crypto driver object. | |
| ***Available via*** | CryIf.h | |

⌋

**[SWS_CryIf_00068] CryIf_RandomSeed API: Report CRYIF_E_UNINIT error**

*Upstream requirements:* SRS_BSW_00369, SRS_BSW_00406

⌈If development error detection for the CRYIF is enabled: The function CryIf_RandomSeed shall report CRYIF_E_UNINIT to the DET if the module is not yet initialized.⌋

**[SWS_CryIf_00069] CryIf_RandomSeed API: Report CRYIF_E_PARAM_HANDLE error**

*Upstream requirements:* SRS_BSW_00369

⌈If development error detection for the CRYIF is enabled: The function CryIf_RandomSeed shall report CRYIF_E_PARAM_HANDLE to the DET if the parameter cryIfKeyId is out of range.⌋

**[SWS_CryIf_00070] CryIf_RandomSeed API: Report CRYIF_E_PARAM_POINTER error**

*Upstream requirements:* SRS_BSW_00369

⌈If development error detection for the CRYIF is enabled: The function CryIf_RandomSeed shall report CRYIF_E_PARAM_POINTER to the DET if the parameter seedPtr is a null pointer.⌋

**[SWS_CryIf_00071] CryIf_RandomSeed API: Report CRYIF_E_PARAM_VALUE error**

*Upstream requirements:* SRS_BSW_00369

⌈If development error detection for the CRYIF is enabled: The function CryIf_RandomSeed shall report CRYIF_E_PARAM_VALUE to the DET if seedLength is zero.⌋

**[SWS_CryIf_00072] CryIf_RandomSeed API: Forward random seed job to the mapped driver** ⌈If no errors are detected by CRYIF, the service CryIf_RandomSeed shall call Crypto_<vi>_<ai>_RandomSeed for the driver configuration mapped to the service and pass on the return value.⌋

**[SWS_CryIf_91008] Definition of API function CryIf_KeyGenerate** ⌈

| | | |
|---|---|---|
| ***Service Name*** | CryIf_KeyGenerate | |
| ***Syntax*** | `Std_ReturnType CryIf_KeyGenerate ( uint32 cryIfKeyId )` | |
| ***Service ID [hex]*** | 0x08 | |
| ***Sync/Async*** | Synchronous | |
| ***Reentrancy*** | Reentrant | |
| ***Parameters (in)*** | cryIfKeyId | 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<br>`E_NOT_OK`: Request failed<br>`CRYPTO_E_BUSY`: Request failed, Crypto Driver Object is busy<br>`CRYPTO_E_KEY_EMPTY`: Request failed because of uninitialized source key element<br>`CRYPTO_E_KEY_NOT_VALID`: Request failed, the key state of the key identified by cryptoKeyId is "invalid". |
| ***Description*** | This function shall dispatch the key generate function to the configured crypto driver object. | |
| ***Available via*** | CryIf.h | |

⌋

**[SWS_CryIf_00073] CryIf_KeyGenerate API: Report CRYIF_E_UNINIT error**

*Upstream requirements:* SRS_BSW_00369, SRS_BSW_00406

⌈If development error detection for the CRYIF module is enabled: The function `CryIf_KeyGenerate` shall report `CRYIF_E_UNINIT` to the DET if the module is not yet initialized.⌋

**[SWS_CryIf_00074] CryIf_KeyGenerate API: Report CRYIF_E_PARAM_HANDLE error**

*Upstream requirements:* SRS_BSW_00369

⌈If development error detection for the CRYIF module is enabled: The function `CryIf_KeyGenerate` shall report `CRYIF_E_PARAM_HANDLE` to the DET if the parameter `cryIfKeyId` is out of range.⌋

**[SWS_CryIf_00075] CryIf_KeyGenerate API: Forward key job to the mapped driver** ⌈If no errors are detected by CRYIF, the service `CryIf_KeyGenerate` shall call Crypto_<vi>_<ai>_KeyGenerate for the driver configuration mapped to the service and pass on the return value.⌋

#### 8.3.4.6 Key Derivation Interface

**[SWS_CryIf_91009] Definition of API function CryIf_KeyDerive** ⌈

| | | |
|---|---|---|
| ***Service Name*** | CryIf_KeyDerive | |
| ***Syntax*** | `Std_ReturnType CryIf_KeyDerive ( uint32 cryIfKeyId, uint32 targetCryIfKeyId )` | |
| ***Service ID [hex]*** | 0x09 | |
| ***Sync/Async*** | Synchronous | |
| ***Reentrancy*** | Reentrant | |
| ***Parameters (in)*** | cryIfKeyId | Holds the identifier of the key which is used for key derivation. |
| | targetCryIfKeyId | 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<br>`E_NOT_OK`: Request failed<br>`CRYPTO_E_KEY_EMPTY`: Request failed because of uninitialized source key element<br>`CRYPTO_E_BUSY`: Crypto Driver Object has returned `CRYPTO_E_BUSY`.<br>`CRYPTO_E_KEY_NOT_VALID`: Request failed, the key state of the key identified by cryptoKeyId is "invalid". |
| ***Description*** | This function shall dispatch the key derive function to the configured crypto driver object. | |
| ***Available via*** | CryIf.h | |

⌋

**[SWS_CryIf_00076] CryIf_KeyDerive API: Report CRYIF_E_UNINIT error**

*Upstream requirements:* SRS_BSW_00369, SRS_BSW_00406

⌈If development error detection for the CRYIF module is enabled: The function `CryIf_KeyDerive` shall report `CRYIF_E_UNINIT` to the DET if the module is not yet initialized.⌋

**[SWS_CryIf_00077] CryIf_KeyDerive API: Report CRYIF_E_PARAM_HANDLE error for cryIfKeyId parameter**

*Upstream requirements:* SRS_BSW_00369

⌈If development error detection for the CRYIF module is enabled: The function `CryIf_KeyDerive` shall report `CRYIF_E_PARAM_HANDLE` to the DET if the parameter `cryIfKeyId` is out of range.⌋

**[SWS_CryIf_00122] CryIf_KeyDerive API: Report CRYIF_E_PARAM_HANDLE error for targetCryIfKeyId parameter**

*Upstream requirements:* SRS_BSW_00369

⌈If development error detection for the CRYIF module is enabled: The function `CryIf_KeyDerive` shall report `CRYIF_E_PARAM_HANDLE` to the DET if the parameter `targetCryIfKeyId` is out of range.⌋

**[SWS_CryIf_00081] CryIf_KeyDerive API: Forward key job to the mapped driver** ⌈If no errors are detected by CRYIF, the service `CryIf_KeyDerive` shall call Crypto_<vi>_<ai>_KeyDerive for the driver configuration mapped to the service and pass on the return value.⌋

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 `cryIf-KeyId`.

**[SWS_CryIf_91010] Definition of API function CryIf_KeyExchangeCalcPubVal** ⌈

| | | |
|---|---|---|
| ***Service Name*** | CryIf_KeyExchangeCalcPubVal | |
| ***Syntax*** | `Std_ReturnType CryIf_KeyExchangeCalcPubVal ( uint32 cryIfKeyId, uint8* publicValuePtr, uint32* publicValueLengthPtr )` | |
| ***Service ID [hex]*** | 0x0a | |
| ***Sync/Async*** | Synchronous | |
| ***Reentrancy*** | Reentrant | |
| ***Parameters (in)*** | cryIfKeyId | 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 public ValuePtr. 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<br>`E_NOT_OK`: Request failed<br>`CRYPTO_E_BUSY`: Request failed, Crypto Driver Object is busy<br>`CRYPTO_E_KEY_EMPTY`: Request failed because of uninitialized source key element<br>`CRYPTO_E_KEY_NOT_VALID`: Request failed, the key state of the key identified by cryptoKeyId is "invalid". |
| ***Description*** | This function shall dispatch the key exchange public value calculation function to the configured crypto driver object. | |
| ***Available via*** | CryIf.h | |

⌋

**[SWS_CryIf_00082] CryIf_KeyExchangeCalcPubVal API: Report CRYIF_E_UNINIT error**

*Upstream requirements:* SRS_BSW_00369, SRS_BSW_00406

⌈If development error detection for the CRYIF module is enabled: The function `CryIf_KeyExchangeCalcPubVal` shall report `CRYIF_E_UNINIT` to the DET if the module is not yet initialized.⌋

**[SWS_CryIf_00083] CryIf_KeyExchangeCalcPubVal API: Report CRYIF_E_PARAM_HANDLE error**

*Upstream requirements:* SRS_BSW_00369

⌈If development error detection for the CRYIF module is enabled: The function `CryIf_KeyExchangeCalcPubVal` shall report `CRYIF_E_PARAM_HANDLE` to the DET if the parameter `cryIfKeyId` is out of range.⌋

**[SWS_CryIf_00084] CryIf_KeyExchangeCalcPubVal API: Report CRYIF_E_PARAM_POINTER error for publicValuePtr**

*Upstream requirements:* SRS_BSW_00369

⌈If development error detection for the CRYIF module is enabled: The function `CryIf_KeyExchangeCalcPubVal` shall report `CRYIF_E_PARAM_POINTER` to the DET if the parameter `publicValuePtr` is a `null` pointer.⌋

**[SWS_CryIf_00085] CryIf_KeyExchangeCalcPubVal API: Report CRYIF_E_PARAM_POINTER error for publicValueLengthPtr**

*Upstream requirements:* SRS_BSW_00369

⌈If development error detection for the CRYIF module is enabled: The function `CryIf_KeyExchangeCalcPubVal` shall report `CRYIF_E_PARAM_POINTER` to the DET if the parameter `publicValueLengthPtr` is a `null` pointer.⌋

**[SWS_CryIf_00086] CryIf_KeyExchangeCalcPubVal API: Report CRYIF_E_PARAM_VALUE error**

*Upstream requirements:* SRS_BSW_00369

⌈If development error detection for the CRYIF module is enabled: The function `CryIf_KeyExchangeCalcPubVal` shall report `CRYIF_E_PARAM_VALUE` to the DET if the value, which is pointed by `publicValueLengthPtr`, is zero.⌋

**[SWS_CryIf_00087] CryIf_KeyExchangeCalcPubVal API: Forward key job to the mapped driver** ⌈If no errors are detected by CRYIF, the service `CryIf_KeyExchangeCalcPubVal` shall call Crypto_<vi>_<ai>_KeyExchangeCalcPubVal for the driver configuration mapped to the service and pass on the return value.⌋

**[SWS_CryIf_91011] Definition of API function CryIf_KeyExchangeCalcSecret** ⌈

| | |
|---|---|
| ***Service Name*** | CryIf_KeyExchangeCalcSecret |
| ***Syntax*** | `Std_ReturnType CryIf_KeyExchangeCalcSecret ( uint32 cryIfKeyId, const uint8* partnerPublicValuePtr, uint32 partnerPublicValueLength )` |
| ***Service ID [hex]*** | 0x0b |
| ***Sync/Async*** | Synchronous |
| ***Reentrancy*** | Reentrant |

▽

△

| | | |
|---|---|---|
| ***Parameters (in)*** | cryIfKeyId | 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<br>`E_NOT_OK`: Request failed<br>`CRYPTO_E_BUSY`: Request failed, Crypto Driver Object is busy<br>`CRYPTO_E_KEY_EMPTY`: Request failed because of uninitialized source key element<br>`CRYPTO_E_KEY_NOT_VALID`: Request failed, the key state of the key identified by cryIfKeyId is "invalid". |
| ***Description*** | This function shall dispatch the key exchange common shared secret calculation function to the configured crypto driver object. | |
| ***Available via*** | CryIf.h | |

⌋

**[SWS_CryIf_00090] CryIf_KeyExchangeCalcSecret API: Report CRYIF_E_UNINIT error**

*Upstream requirements:* SRS_BSW_00369, SRS_BSW_00406

⌈If development error detection for the CRYIF module is enabled: The function `CryIf_KeyExchangeCalcSecret` shall report `CRYIF_E_UNINIT` to the DET if the module is not yet initialized.⌋

**[SWS_CryIf_00091] CryIf_KeyExchangeCalcSecret API: Report CRYIF_E_PARAM_HANDLE error**

*Upstream requirements:* SRS_BSW_00369

⌈If development error detection for the CRYIF module is enabled: The function `CryIf_KeyExchangeCalcSecret` shall report `CRYIF_E_PARAM_HANDLE` to the DET if the parameter `cryIfKeyId` is out of range.⌋

**[SWS_CryIf_00092] CryIf_KeyExchangeCalcSecret API: Report CRYIF_E_PARAM_POINTER error**

*Upstream requirements:* SRS_BSW_00369

⌈If development error detection for the CRYIF module is enabled: The function `CryIf_KeyExchangeCalcSecret` shall report `CRYIF_E_PARAM_POINTER` to the DET if the parameter `partnerPublicValuePtr` is a `null` pointer.⌋

**[SWS_CryIf_00094] CryIf_KeyExchangeCalcSecret API: Report CRYIF_E_PARAM_VALUE error**

*Upstream requirements:* SRS_BSW_00369

⌈If development error detection for the CRYIF module is enabled: The function `CryIf_KeyExchangeCalcSecret` shall report `CRYIF_E_PARAM_VALUE` to the DET if `partnerPublicValueLength` is zero.⌋

**[SWS_CryIf_00095] CryIf_KeyExchangeCalcSecret API: Forward key job to the mapped driver** ⌈If no errors are detected by CRYIF, the service `CryIf_KeyExchangeCalcSecret` shall call Crypto_<vi>_<ai>_KeyExchangeCalcSecret for the driver configuration mapped to the service and pass on the return value.⌋

### 8.3.5 Custom Service Interface

**[SWS_CryIf_91022] Definition of API function CryIf_CustomSync** ⌈

| | | |
|---|---|---|
| ***Service Name*** | CryIf_CustomSync | |
| ***Syntax*** | `Std_ReturnType CryIf_CustomSync ( uint32 dispatchId, uint32 keyId, uint32 keyElementId, uint32 targetKeyId, uint32 targetKeyElementId, const uint8* inputPtr, uint32 inputLength, uint8* outputPtr, uint32* outputLengthPtr, uint8* secondaryOutputPtr, uint32* secondaryOutputLengthPtr )` | |
| ***Service ID [hex]*** | 0x15 | |
| ***Sync/Async*** | Synchronous | |
| ***Reentrancy*** | Non Reentrant | |
| ***Parameters (in)*** | dispatchId | unique id to identify the request |
| | keyId | key Id of the key the certificate is stored |
| | keyElementId | key element id |
| | targetKeyId | Holds the target key id |
| | targetKeyElementId | – |
| | inputPtr | Pointer to the input data. |
| | inputLength | Contains the input length in bytes. |
| ***Parameters (inout)*** | None | |
| ***Parameters (out)*** | outputPtr | Pointer to the output data. |
| | outputLengthPtr | Contains the output length in bytes. |
| | secondaryOutputPtr | Pointer to the secondary output data. |
| | secondaryOutputLength Ptr | Contains the secondary output length in bytes. |
| ***Return value*** | Std_ReturnType | `E_OK`: Request successful<br>`E_NOT_OK`: Request failed<br>`CRYPTO_E_BUSY`: The service request failed because the service is still busy<br>`CRYPTO_E_CUSTOM_ERROR`: Custom processing failed |
| ***Description*** | Requests the execution of a function that is specified by the given dispatch id. | |
| ***Available via*** | CryIf.h | |

⌋

**[SWS_CryIf_91002] CryIf_CustomSync API: Forward CustomSync job to the mapped driver** ⌈If no errors are detected by CRYIF, the service `CryIf_CustomSync` shall call `Crypto_CustomSync` for the driver configuration mapped to the service and pass on the return value.⌋

### 8.3.6 Callback notifications

This is a list of functions provided for other modules.

**[SWS_CryIf_91013] Definition of API function CryIf_CallbackNotification**

*Upstream requirements:* SRS_BSW_00359, SRS_BSW_00360

⌈

| | | |
|---|---|---|
| ***Service Name*** | CryIf_CallbackNotification | |
| ***Syntax*** | `void CryIf_CallbackNotification ( Crypto_JobType* job, Crypto_ResultType result )` | |
| ***Service ID [hex]*** | 0x0d | |
| ***Sync/Async*** | Synchronous | |
| ***Reentrancy*** | Non Reentrant | |
| ***Parameters (in)*** | job | Points to the completed job's information structure. It contains a callbackID to identify which job is finished. |
| | result | Contains the result of the cryptographic operation. |
| ***Parameters (inout)*** | None | |
| ***Parameters (out)*** | None | |
| ***Return value*** | void | -- |
| ***Description*** | Notifies the CRYIF about the completion of the request with the result of the cryptographic operation. | |
| ***Available via*** | CryIf.h | |

⌋

**[SWS_CryIf_00107] CryIf_CallbackNotification API: Report CRYIF_E_UNINITR error**

*Upstream requirements:* SRS_BSW_00369

⌈If development error detection for the CRYIF module is enabled: The function `CryIf_CallbackNotification` shall report `CRYIF_E_UNINIT` to the DET if the module is not yet initialized.⌋

**[SWS_CryIf_00108] CryIf_CallbackNotification API: Report CRYIF_E_PARAM_POINTER error**

*Upstream requirements:* SRS_BSW_00369

⌈If development error detection for the CRYIF module is enabled: The function `CryIf_CallbackNotification` shall report `CRYIF_E_PARAM_POINTER` to the DET if the parameter `job` is a `null` pointer.⌋

**[SWS_CryIf_00109] Forward CryIf CallbackNotification to Csm**

*Upstream requirements:* SRS_BSW_00360

⌈If no errors are detected by CRYIF, the service `CryIf_CallbackNotification` shall call `Csm_CallbackNotification` and pass on the result.⌋

## 8.4 Expected interfaces

In this chapter all interfaces required from other modules are listed.

### 8.4.1 Mandatory interfaces

Note: This section defines all interfaces, which are required to fulfill the core functionality of the module.

**[SWS_CryIf_91100] Definition of mandatory interfaces required by module CryIf** ⌈

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

⌋

### 8.4.2 Optional interfaces

This section defines all interfaces, which are required to fulfill an optional functionality of the module.

**[SWS_CryIf_91101] Definition of optional interfaces requested by module CryIf** ⌈

| *API Function* | *Header File* | *Description* |
|---|---|---|
| Crypto_CancelJob | Crypto.h | This interface removes the provided job from the queue and cancels the processing of the job if possible. |
| Crypto_CustomSync | Crypto.h | Requests the execution of a function that is specified by the given dispatch id. |
| Crypto_KeyCopy | Crypto.h | 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) |
| Crypto_KeyDerive | Crypto.h | 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. |
| Crypto_KeyElementCopy | Crypto.h | 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) |

▽

△

| API Function | Header File | Description |
|---|---|---|
| Crypto_KeyElementCopyPartial | Crypto.h | 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.<br>Note: If the actual key element is directly mapped to flash memory, there could be a bigger delay when calling this function (synchronous operation). |
| Crypto_KeyElementGet | Crypto.h | 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). |
| Crypto_KeyElementIdsGet | Crypto.h | Used to retrieve information which key elements are available in a given key. |
| Crypto_KeyElementSet | Crypto.h | Sets the given key element bytes to the key identified by cryptoKeyId. |
| Crypto_KeyExchangeCalcPubVal | Crypto.h | Calculates the public value for the key exchange and stores the public key in the memory location pointed by the public value pointer. |
| Crypto_KeyExchangeCalcSecret | Crypto.h | 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. |
| Crypto_KeyGenerate | Crypto.h | Generates new key material store it in the key identified by cryptoKeyId. |
| Crypto_KeyGetStatus | Crypto.h | Returns the key state of the key identified by crypto KeyId. |
| Crypto_KeySetInvalid | Crypto.h | Sets invalid for the status of the key identified by cryptoKeyId. |
| Crypto_KeySetValid | Crypto.h | Sets the key state of the key identified by cryptoKey Id to valid. |
| Crypto_ProcessJob | Crypto.h | Performs the crypto primitive, that is configured in the job parameter. |
| Crypto_RandomSeed | Crypto.h | 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 |
| Csm_CallbackNotification | Csm.h | Notifies the CSM that a job has finished. This function is used by the underlying layer (CRYIF). |
| Det_ReportError | Det.h | Service to report development errors. |
| Det_ReportRuntimeError | Det.h | Service to report runtime errors. If a callout has been configured then this callout shall be called. |

⌋

# 9 Sequence diagrams

N/A.

# 10 Configuration specification

In general, this chapter defines configuration parameters and their clustering into containers. In order to support the specification Chapter 10.1 describes fundamentals. It also specifies a template (table) you shall use for the parameter specification. We intend to leave Chapter 10.1 in the specification to guarantee comprehension.

Chapter 10.2 specifies the structure (containers) and the parameters of the module CryIf.

Chapter 10.3 specifies published information of the module CryIf.

## 10.1 How to read this chapter

For details refer to [4] Chapter 10.1 *"Introduction to configuration specification"*.

## 10.2 Containers and configuration parameters

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

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

### 10.2.1 Variants

For details refer to the chapter 10.1.2 "Variants" in [4].

### 10.2.2 CryIf

**[ECUC_CryIf_00001] Definition of EcucModuleDef CryIf** ⌈

| | |
|---|---|
| **Module Name** | CryIf |
| **Description** | Configuration of the Crypto Interface. |
| **Post-Build Selectable Support** | false |
| **Post-Build Loadable Support** | false |

| **Included Containers** | | |
|---|---|---|
| **Container Name** | **Multiplicity** | **Description** |
| CryIfChannels | 0..1 | Container for CryIf channel configurations. |
| CryIfGeneral | 1 | Container for common configuration options. |
| CryIfKeys | 0..1 | Container for CryIf key configurations. |
| CryIfPrimitives | 0..1 | Container for CryIf primitive configurations. |

⌋

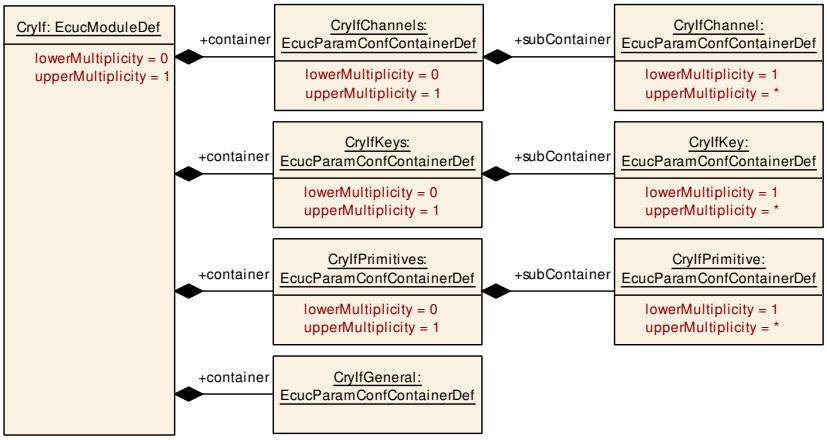



**Figure 10.1: Crypto Interface Overview**

### 10.2.3 CryIfGeneral

**[ECUC_CryIf_00009] Definition of EcucParamConfContainerDef CryIfGeneral** ⌈

| | |
|---|---|
| **Container Name** | CryIfGeneral |
| **Parent Container** | CryIf |
| **Upstream mapping defined** | – |
| **Description** | Container for common configuration options. |
| **Multiplicity** | 1 |
| **Pre-Compile only** | true |
| **Configuration Parameters** | |

| **Included Parameters** | | |
|---|---|---|
| **Parameter Name** | **Multiplicity** | **ECUC ID** |
| CryIfDevErrorDetect | 1 | [ECUC_CryIf_00010] |

| **No Included Containers** |
|---|

⌋

**[ECUC_CryIf_00010] Definition of EcucBooleanParamDef CryIfDevErrorDetect** ⌈

| | |
|---|---|
| **Parameter Name** | CryIfDevErrorDetect |
| **Parent Container** | CryIfGeneral |
| **Upstream mapping defined** | – |
| **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 |
| **Pre-Compile only** | true |

⌋

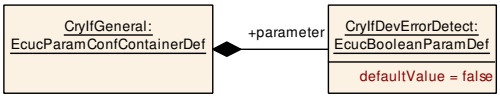



**Figure 10.2: CryIf general configuration overview**

### 10.2.4 CryIfChannels

### [ECUC_CryIf_00013] Definition of EcucParamConfContainerDef CryIfChannels ⌈

| | |
|---|---|
| **Container Name** | CryIfChannels |
| **Parent Container** | CryIf |
| **Upstream mapping defined** | – |
| **Description** | Container for CryIf channel configurations. |
| **Multiplicity** | 0..1 |
| **Pre-Compile only** | true |
| **Configuration Parameters** | |

| |
|---|
| **No Included Parameters** |

| **Included Containers** | | |
|---|---|---|
| **Container Name** | **Multiplicity** | **Description** |
| CryIfChannel | 1..* | Container for configuration of a CryIf channel. The container name serves as a symbolic name for the identifier of a channel configuration. |

⌋

### 10.2.5 CryIfChannel

### [ECUC_CryIf_00002] Definition of EcucParamConfContainerDef CryIfChannel ⌈

| | |
|---|---|
| **Container Name** | CryIfChannel |
| **Parent Container** | CryIfChannels |
| **Upstream mapping defined** | – |
| **Description** | Container for configuration of a CryIf channel. The container name serves as a symbolic name for the identifier of a channel configuration. |
| **Multiplicity** | 1..* |
| **Pre-Compile only** | true |
| **Configuration Parameters** | |

| Included Parameters | | |
|---|---|---|
| **Parameter Name** | **Multiplicity** | **ECUC ID** |
| CryIfChannelId | 1 | [ECUC_CryIf_00004] |
| CryIfDriverObjectRef | 1 | [ECUC_CryIf_00005] |

| No Included Containers |
|---|

⌋

## [ECUC_CryIf_00004] Definition of EcucIntegerParamDef CryIfChannelId ⌈

| **Parameter Name** | CryIfChannelId |
|---|---|
| **Parent Container** | CryIfChannel |
| **Upstream mapping defined** | – |
| **Description** | Identifier of the cryif channel.<br>Dependency: The range is limited to 0xFFFFFFFE, to have the value 0xFFFFFFFF (CRYPTO_NO_ID) reserved for stack internal use. This is common for Crypto stack "Id" parameters. |
| **Multiplicity** | 1 |
| **Type** | EcucIntegerParamDef (Symbolic Name generated for this parameter) |
| **Range** | 0 .. 4294967294 |
| **Default value** | – |
| **Pre-Compile only** | true |

⌋

## [ECUC_CryIf_00005] Definition of EcucReferenceDef CryIfDriverObjectRef ⌈

| **Parameter Name** | CryIfDriverObjectRef |
|---|---|
| **Parent Container** | CryIfChannel |
| **Upstream mapping defined** | – |
| **Description** | This parameter refers to a Crypto Driver Object. Specifies to which Crypto Driver Object the CryIf channel is connected to |
| **Multiplicity** | 1 |
| **Type** | Symbolic name reference to CryptoDriverObject |
| **Pre-Compile only** | true |

⌋

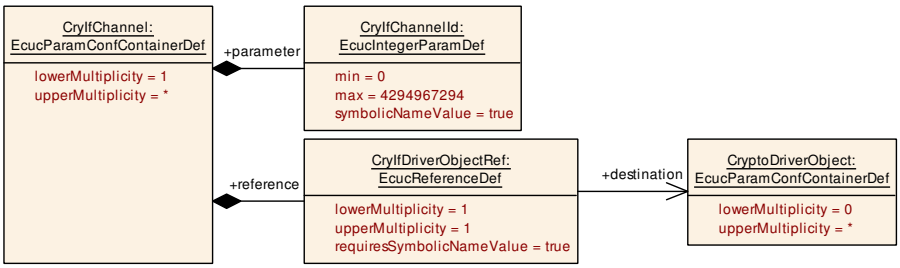



**Figure 10.3: CryIf channel configuration overview**

### 10.2.6 CryIfKeys

**[ECUC_CryIf_00014] Definition of EcucParamConfContainerDef CryIfKeys** ⌈

| | |
|---|---|
| **Container Name** | CryIfKeys |
| **Parent Container** | CryIf |
| **Upstream mapping defined** | – |
| **Description** | Container for CryIf key configurations. |
| **Multiplicity** | 0..1 |
| **Pre-Compile only** | true |
| **Configuration Parameters** | |

| **No Included Parameters** |
|---|

| **Included Containers** | | |
|---|---|---|
| **Container Name** | **Multiplicity** | **Description** |
| CryIfKey | 1..* | Container for configuration of a CryIf key. The container name serves as a symbolic name for the identifier of a key configuration. |

⌋

### 10.2.7 CryIfKey

**[ECUC_CryIf_00003] Definition of EcucParamConfContainerDef CryIfKey** ⌈

| | |
|---|---|
| **Container Name** | CryIfKey |
| **Parent Container** | CryIfKeys |
| **Upstream mapping defined** | – |
| **Description** | Container for configuration of a CryIf key. The container name serves as a symbolic name for the identifier of a key configuration. |
| **Multiplicity** | 1..* |
| **Pre-Compile only** | true |
| **Configuration Parameters** | |

| **Included Parameters** | | |
|---|---|---|
| **Parameter Name** | **Multiplicity** | **ECUC ID** |
| CryIfKeyId | 1 | [ECUC_CryIf_00007] |
| CryIfKeyRef | 1 | [ECUC_CryIf_00008] |

| **No Included Containers** |
|---|

⌋

### [ECUC_CryIf_00007] Definition of EcucIntegerParamDef CryIfKeyId ⌈

| Parameter Name | CryIfKeyId |
|---|---|
| Parent Container | CryIfKey |
| Upstream mapping defined | – |
| Description | Identifier of the CryIf key.<br>Dependency: The range is limited to 0xFFFFFFFE, to have the value 0xFFFFFFFF (CRYPTO_NO_ID) reserved for stack internal use. This is common for Crypto stack "Id" parameters. |
| Multiplicity | 1 |
| Type | EcucIntegerParamDef (Symbolic Name generated for this parameter) |
| Range | 0 .. 4294967294 |
| Default value | – |
| Pre-Compile only | true |

⌋

### [ECUC_CryIf_00008] Definition of EcucReferenceDef CryIfKeyRef ⌈

| Parameter Name | CryIfKeyRef |
|---|---|
| Parent Container | CryIfKey |
| Upstream mapping defined | – |
| Description | This parameter refers to the Crypto driver key. Specifies to which Crypto driver key the CryIf key is mapped to. |
| Multiplicity | 1 |
| Type | Symbolic name reference to CryptoKey |
| Pre-Compile only | true |

⌋

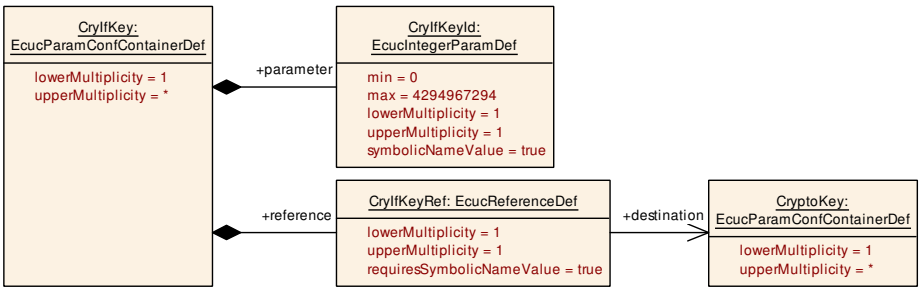



**Figure 10.4: CryIf key configuration overview**

## 10.2.8 CryIfPrimitives

### [ECUC_CryIf_00015] Definition of EcucParamConfContainerDef CryIfPrimitives ⌈

| Container Name | CryIfPrimitives |
|---|---|
| Parent Container | CryIf |
| Upstream mapping defined | – |

▽

△

| | |
|---|---|
| **Description** | Container for CryIf primitive configurations. |
| **Multiplicity** | 0..1 |
| **Pre-Compile only** | true |
| **Configuration Parameters** | |

| **No Included Parameters** |
|---|

| **Included Containers** | | |
|---|---|---|
| **Container Name** | **Multiplicity** | **Description** |
| CryIfPrimitive | 1..* | Container for configuration of a CryIf primitive. The container name serves as a symbolic name for the identifier of a primitive configuration. |

⌋

### 10.2.9 CryIfPrimitive

**[ECUC_CryIf_00016] Definition of EcucParamConfContainerDef CryIfPrimitive** ⌈

| | |
|---|---|
| **Container Name** | CryIfPrimitive |
| **Parent Container** | CryIfPrimitives |
| **Upstream mapping defined** | – |
| **Description** | Container for configuration of a CryIf primitive. The container name serves as a symbolic name for the identifier of a primitive configuration. |
| **Multiplicity** | 1..* |
| **Pre-Compile only** | true |
| **Configuration Parameters** | |

| **Included Parameters** | | |
|---|---|---|
| **Parameter Name** | **Multiplicity** | **ECUC ID** |
| CryIfPrimitiveId | 1 | [ECUC_CryIf_00017] |
| CryIfPrimitiveRef | 1 | [ECUC_CryIf_00018] |

| **No Included Containers** |
|---|

⌋

**[ECUC_CryIf_00017] Definition of EcucIntegerParamDef CryIfPrimitiveId** ⌈

| | |
|---|---|
| **Parameter Name** | CryIfPrimitiveId |
| **Parent Container** | CryIfPrimitive |
| **Upstream mapping defined** | – |
| **Description** | Identifier of the CryIf primitive.<br>Dependency: The range is limited to 0xFFFFFFFE, to have the value 0xFFFFFFFF (CRYPTO_NO_ID) reserved for stack internal use. This is common for Crypto stack "Id" parameters. |

▽

△

| | |
|---|---|
| **Multiplicity** | 1 |
| **Type** | EcucIntegerParamDef (Symbolic Name generated for this parameter) |
| **Range** | 0 .. 4294967294 |
| **Default value** | – |
| **Pre-Compile only** | true |

⌋

### [ECUC_CryIf_00018] Definition of EcucReferenceDef CryIfPrimitiveRef ⌈

| | |
|---|---|
| **Parameter Name** | CryIfPrimitiveRef |
| **Parent Container** | CryIfPrimitive |
| **Upstream mapping defined** | – |
| **Description** | This parameter refers to a crypto primitive. Specifies to which Crypto driver primitive the CryIf primitive is mapped to. |
| **Multiplicity** | 1 |
| **Type** | Symbolic name reference to CryptoPrimitive |
| **Pre-Compile only** | true |

⌋

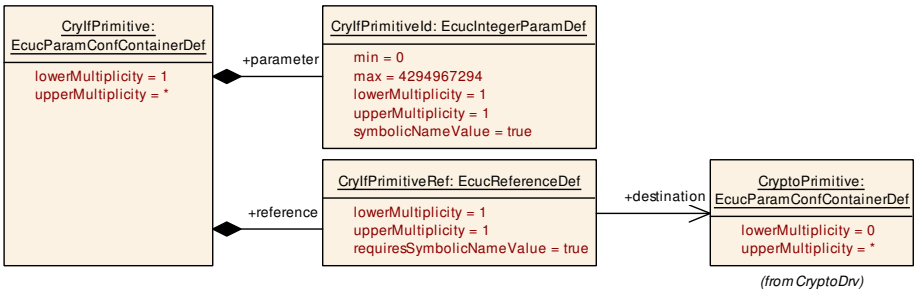



**Figure 10.5: CryIf primitive configuration overview**

## 10.3 Published Information

For details refer to [4] Chapter 10.3 *"Published Information"*.

# A Change history of AUTOSAR traceable items

Please note that the lists in this chapter also include traceable items that have been removed from the specification in a later version. These items do not appear as hyperlinks in the document.

## A.1 Traceable item history of this document according to AUTOSAR Release R26-11

### A.1.1 Added Specification Items in R26-11

[ECUC_CryIf_00013] [ECUC_CryIf_00014] [ECUC_CryIf_00015] [ECUC_CryIf_00016] [ECUC_CryIf_00017] [ECUC_CryIf_00018] [SWS_CryIf_00161] [SWS_CryIf_00162] [SWS_CryIf_00163] [SWS_CryIf_00164] [SWS_CryIf_00165] [SWS_CryIf_00166] [SWS_CryIf_00167] [SWS_CryIf_00168] [SWS_CryIf_00169] [SWS_CryIf_00170] [SWS_CryIf_00171] [SWS_CryIf_00172] [SWS_CryIf_00173] [SWS_CryIf_00174] [SWS_CryIf_00175] [SWS_CryIf_00176] [SWS_CryIf_00177] [SWS_CryIf_00178] [SWS_CryIf_00179] [SWS_CryIf_00180] [SWS_CryIf_00181] [SWS_CryIf_00182] [SWS_CryIf_00183] [SWS_CryIf_00184] [SWS_CryIf_00185] [SWS_CryIf_00186] [SWS_CryIf_00187] [SWS_CryIf_00188] [SWS_CryIf_00189] [SWS_CryIf_00190]

### A.1.2 Changed Specification Items in R26-11

[ECUC_CryIf_00001] [ECUC_CryIf_00002] [ECUC_CryIf_00003] [ECUC_CryIf_00004] [ECUC_CryIf_00007] [ECUC_CryIf_00009] [SWS_CryIf_91023]

### A.1.3 Deleted Specification Items in R26-11

[ECUC_CryIf_00011] [SWS_CryIf_00017] [SWS_CryIf_00133] [SWS_CryIf_00134] [SWS_CryIf_00135] [SWS_CryIf_00136] [SWS_CryIf_00141] [SWS_CryIf_00142] [SWS_CryIf_00143] [SWS_CryIf_00144] [SWS_CryIf_91001]

## A.2 Traceable item history of this document according to AUTOSAR Release R25-11

### A.2.1 Added Specification Items in R25-11

none

### A.2.2 Changed Specification Items in R25-11

none

### A.2.3 Deleted Specification Items in R25-11

none